Hochschule Offenburg
offenburg.university

Vergleichende Untersuchung von „Platform as a Service“-Angeboten für das „Internet of Things“

Master Thesis

Medien und Kommunikation

Von
Matthias Karow

Sommersemester 2016

Erstbetreuer: Prof. Dr. Tom Rüdebusch
Zweitbetreuer: Prof. Dr. Volker Sänger

Eidesstattliche Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Master-Thesis von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Offenburg, 02.05.2016,

Ort, Datum

Inhaltsverzeichnis

1 Einleitung 9
2 Definitionen 11
2.1 Internet of Things 11
2.2 Platform as a Service 11
2.2.1 Abgrenzung zu Iaas & SaaS 12
2.3 Verbindungsprotokolle 12
2.3.1 HTTP 13
2.3.2 MQTT 13
3 Untersuchung der PaaS-Angebote 15
3.1 Untersuchungskriterien 15
3.1.1 Quantitativ 15
3.1.1.1 Verbreitung 15
3.1.1.2 Preis 15
3.1.1.3 Verbindungsrate 15
3.1.1.4 Datenmenge 16
3.1.1.5 Schnittstellen 16
3.1.2 Qualitativ 16
3.1.2.1 API 16
3.1.2.2 Visualisierung 16
3.1.2.3 Weiterverarbeitung 16
3.1.2.4 Dokumentation 16
3.1.2.5 Usability 16
3.1.2.6 Sicherheit 17
3.2 Plattformspezifische Untersuchung 17
3.2.1 ThingSpeak 17
3.2.1.1 Verbreitung 17
3.2.1.2 Preis 17
3.2.1.3 Verbindungsrate 17
3.2.1.4 Datenmenge 17
3.2.1.5 Schnittstellen 17
3.2.1.6 API 18
3.2.1.7 Visualisierung 19
3.2.1.8 Weiterverarbeitung 19
3.2.1.9 Dokumentation 19
3.2.1.10 Usability 20
3.2.1.11 Sicherheit 20
3.2.2 data.sparkfun.com 21
3.2.2.1 Verbreitung 21
3.2.2.2 Preis 21

3.2.2.3 Verbindungsrate 21
3.2.2.4 Datenmenge....21
3.2.2.5 Schnittstellen 21
3.2.2.6 API 21
3.2.2.7 Visualisierung....21
3.2.2.8 Weiterverarbeitung 21
3.2.2.9 Dokumentation 22
3.2.2.10 Usability....22
3.2.2.11 Sicherheit 22
3.2.3 RunAbove IoT Lab....23
3.2.3.1 Verbreitung 23
3.2.3.2 Preis....23
3.2.3.3 Verbindungsrate 23
3.2.3.4 Datenmenge....23
3.2.3.5 Schnittstellen 23
3.2.3.6 API 23
3.2.3.7 Visualisierung....24
3.2.3.8 Weiterverarbeitung 24
3.2.3.9 Dokumentation 24
3.2.3.10 Usability....24
3.2.3.11 Sicherheit 25
3.2.4 flowthings.io....25
3.2.4.1 Verbreitung 25
3.2.4.2 Preis....25
3.2.4.3 Verbindungsrate 25
3.2.4.4 Datenmenge....25
3.2.4.5 Schnittstellen 25
3.2.4.6 API 26
3.2.4.7 Visualisierung....26
3.2.4.8 Weiterverarbeitung 27
3.2.4.9 Dokumentation 27
3.2.4.10 Usability....27
3.2.4.11 Sicherheit 28
3.2.5 Carriots....28
3.2.5.1 Verbreitung 28
3.2.5.2 Preis....28
3.2.5.3 Verbindungsrate 28
3.2.5.4 Datenmenge....29
3.2.5.5 Schnittstellen 29
3.2.5.6 API 29
3.2.5.7 Visualisierung....30
3.2.5.8 Weiterverarbeitung 30
3.2.5.9 Dokumentation 30
3.2.5.10 Usability....31
3.2.5.11 Sicherheit 31
3.2.6 Ubidots....31

3.2.6.1 Verbreitung 32
3.2.6.2 Preis 32
3.2.6.3 Verbindungsrate 32
3.2.6.4 Datenmenge 32
3.2.6.5 Schnittstellen 32
3.2.6.6 API 32
3.2.6.7 Visualisierung 33
3.2.6.8 Weiterverarbeitung 34
3.2.6.9 Dokumentation 34
3.2.6.10 Usability 34
3.2.6.11 Sicherheit 35
3.2.7 GroveStreams 35
3.2.7.1 Verbreitung 35
3.2.7.2 Preis 35
3.2.7.3 Verbindungsrate 35
3.2.7.4 Datenmenge 36
3.2.7.5 Schnittstellen 36
3.2.7.6 API 36
3.2.7.7 Visualisierung 37
3.2.7.8 Weiterverarbeitung 37
3.2.7.9 Dokumentation 37
3.2.7.10 Usability 38
3.2.7.11 Sicherheit 38
3.2.8 Exosite 38
3.2.8.1 Verbreitung 39
3.2.8.2 Preis 39
3.2.8.3 Datenrate 39
3.2.8.4 Datenmenge 39
3.2.8.5 Schnittstellen 39
3.2.8.6 API 39
3.2.8.7 Visualisierung 40
3.2.8.8 Weiterverarbeitung 41
3.2.8.9 Dokumentation 41
3.2.8.10 Usability 41
3.2.8.11 Sicherheit 42
3.2.9 Beebotte 42
3.2.9.1 Verbreitung 42
3.2.9.2 Preis 42
3.2.9.3 Verbindungsrate 42
3.2.9.4 Datenmenge 42
3.2.9.5 Schnittstellen 43
3.2.9.6 API 43
3.2.9.7 Visualisierung 43
3.2.9.8 Weiterverarbeitung 43
3.2.9.9 Dokumentation 43
3.2.9.10 Usability 44

3.2.9.11 Sicherheit 44
3.2.10 MODE 44
3.2.10.1 Verbreitung 45
3.2.10.2 Preis 45
3.2.10.3 Verbindungsrate 45
3.2.10.4 Datenmenge 45
3.2.10.5 Schnittstellen 45
3.2.10.6 API 45
3.2.10.7 Visualisierung 46
3.2.10.8 Weiterverarbeitung 46
3.2.10.9 Dokumentation 46
3.2.10.10 Usability 46
3.2.10.11 Sicherheit 47
3.2.11 Initial State 47
3.2.11.1 Verbreitung 47
3.2.11.2 Preis 47
3.2.11.3 Verbindungsrate 47
3.2.11.4 Datenmenge 47
3.2.11.5 Schnittstellen 47
3.2.11.6 API 48
3.2.11.7 Visualisierung 48
3.2.11.8 Weiterverarbeitung 49
3.2.11.9 Dokumentation 49
3.2.11.10 Usability 50
3.2.11.11 Sicherheit 50
3.2.12 Temboo 50
3.2.12.1 Verbreitung 50
3.2.12.2 Preis 51
3.2.12.3 Verbindungsrate 51
3.2.12.4 Datenmenge 51
3.2.12.5 Schnittstellen 51
3.2.12.6 API 51
3.2.12.7 Visualisierung 52
3.2.12.8 Weiterverarbeitung 52
3.2.12.9 Dokumentation 52
3.2.12.10 Usability 52
3.2.12.11 Sicherheit 53
3.2.13 Oracle Cloud 53
3.2.13.1 Verbreitung 53
3.2.13.2 Preis 53
3.2.13.3 Verbindungsrate 53
3.2.13.4 Datenmenge 53
3.2.13.5 Schnittstellen 53
3.2.13.6 API 53
3.2.13.7 Visualisierung 54
3.2.13.8 Weiterverarbeitung 55

3.2.13.9 Dokumentation 55
3.2.13.10 Usability 55
3.2.13.11 Sicherheit 56
3.2.14 IBM Watson Internet of Things 56
3.2.14.1 Verbreitung 56
3.2.14.2 Preis 56
3.2.14.3 Verbindungsrate 56
3.2.14.4 Datenmenge 56
3.2.14.5 Schnittstellen 56
3.2.14.6 API 57
3.2.14.7 Visualisierung 57
3.2.14.8 Weiterverarbeitung 58
3.2.14.9 Dokumentation 58
3.2.14.10 Usability 58
3.2.14.11 Sicherheit 58
3.2.15 Microsoft Azure IoT Hub 59
3.2.15.1 Verbreitung 59
3.2.15.2 Preis 59
3.2.15.3 Verbindungsrate 59
3.2.15.4 Datenmenge 59
3.2.15.5 Schnittstellen 59
3.2.15.6 API 59
3.2.15.7 Visualisierung 60
3.2.15.8 Weiterverarbeitung 60
3.2.15.9 Dokumentation 61
3.2.15.10 Usability 61
3.2.15.11 Sicherheit 61
3.2.16 AWS IoT 62
3.2.16.1 Verbreitung 62
3.2.16.2 Preis 62
3.2.16.3 Verbindungsrate 62
3.2.16.4 Datenmenge 62
3.2.16.5 Schnittstellen 62
3.2.16.6 API 62
3.2.16.7 Visualisierung 63
3.2.16.8 Weiterverarbeitung 63
3.2.16.9 Dokumentation 63
3.2.16.10 Usability 63
3.2.16.11 Sicherheit 64
3.2.17 Google Cloud Platform 64
3.2.17.1 Verbreitung 65
3.2.17.2 Preis 65
3.2.17.3 Verbindungsrate 65
3.2.17.4 Datenmenge 65
3.2.17.5 Schnittstellen 65
3.2.17.6 API 65

3.2.17.7 Visualisierung 66
3.2.17.8 Weiterverarbeitung 66
3.2.17.9 Dokumentation 66
3.2.17.10 Usability 67
3.2.17.11 Sicherheit 67
4 Vergleich der Plattformen 69
4.1 Verbreitung 69
4.2 Preis 69
4.3 Verbindungsrate 69
4.4 Datenmenge 70
4.5 Schnittstellen 70
4.6 API 71
4.7 Visualisierung 71
4.8 Weiterverarbeitung 71
4.9 Dokumentation 72
4.10 Usability 72
4.11 Sicherheit 72
4.12 Auswertung des Plattformvergleichs 73
5 Vergleichende Implementierung einer IoT-Anwendung 75
5.1 Anwendungsbeschreibung 75
5.1.1 Anforderungen 75
5.1.2 Konzept 75
5.1.3 Funktionsweise 76
5.2 Hardware 76
5.2.1 Arduino 76
5.2.2 Shields 76
5.3 Software 77
5.4 Auswahl der Plattformen 77
5.5 Umsetzung 77
5.5.1 Arduino 77
5.5.1.1 Aufbau der Hardware 78
5.5.1.2 Programmieren von WLAN und NFC Shield 78
5.5.2 Plattformübergreifender Teil der Webanwendung 82
5.5.3 ThingSpeak 83
5.5.3.1 Einrichtung der Plattform 83
5.5.3.2 Kommunikation mit dem Arduino 84
5.5.3.3 Webanwendung 88

5.5.3.4 Senden des Tweets ... 89
5.5.4 Flowthings.io ... 90
5.5.4.1 Einrichtung der Plattform ... 90
5.5.4.2 Kommunikation mit dem Arduino ... 91
5.5.4.3 Webanwendung ... 93
5.5.4.4 Senden des Tweets ... 93
5.5.5 IBM Watson IoT ... 94
5.5.5.1 Einrichtung der Plattform ... 94
5.5.5.2 Kommunikation mit dem Arduino ... 95
5.5.5.3 Webanwendung ... 98
5.5.5.4 Senden des Tweets ... 100
6 Auswertung ... 103
7 Zusammenfassung ... 105
8 Literaturverzeichnis ... 107
9 Anhang ... 117
9.1 Tabelle Plattformvergleich ... 117
9.2 Arduino Sketches ... 123
9.2.1 ThingSpeak ... 123
9.2.2 Flowthings.io ... 129
9.2.3 IBM Watson Internet of Things ... 135
9.3 Webanwendung ... 139
9.3.1 ThingSpeak ... 139
9.3.2 Flowthings.io ... 141
9.3.3 IBM Watson Internet of Things ... 143

1 Einleitung

Der Einsatz von Cloudlösungen hat spätestens mit der Einführung von Office 365 und der Adobe Creative Cloud Einzug in den Alltag von IT-Nutzern gehalten. Inzwischen wird nicht nur Software, sondern auch Hardware in der Cloud angeboten.

Zwischen diesen Angeboten liegt die Platform as a Service, welche die Ressourcen zur Anwendungsentwicklung in der Cloud anbietet. Dies ist besonders für ein weiteres neues Technologiefeld interessant, dem Internet of Things. Hier müssen gerade im Bereich von Industrie 4.0 oft riesige Mengen an Endgeräten verwaltet, ihre Daten analysiert und weiterverarbeitet werden. Dies ist mit klassischen Computing-Modellen oft sehr aufwändig und teuer. Platform as a Service-Angebote bieten genau diese Fähigkeiten in einer skalierbaren Umgebung, die sich dem Anwendungsfall perfekt anpassen lässt.

Im Rahmen dieser Master Thesis soll der aktuelle Stand dieser Platform as a Service-Angebote für das Internet of Things recherchiert werden. Die erhobenen Daten werden kategorisiert und die Plattformen anhand qualitativer und quantitativer Kriterien miteinander verglichen. Anhand des Ergebnisses werden drei der Plattformen ausgewählt. Für diese wird eine Beispielanwendung implementiert, welche die Fähigkeiten der Plattformen hinsichtlich der Integration von Geräten, Webanwendungen und Services, der Kommunikation und der Analyse- und Aktionsmöglichkeiten testen und tiefer untersuchen soll.

Aufgrund dieser Implementierung findet eine Auswertung der Implementierung und eine abschließende Zusammenfassung zum Stand der Dinge bei Platform as a Service-Angeboten für das Internet of Things statt.

2 Definitionen

2.1 Internet of Things

Das Internet of Things (IoT) stellt ein weitreichendes Konzept für die Zukunft der Informations- und Kommunikationstechnologien dar. Es bietet eine globale Infrastruktur für die moderne Informationsgesellschaft. Dies wird durch die Verknüpfung von physischen und virtuellen Objekten mit dem Internet erreicht. Während sich die klassische Kommunikation in der Gesellschaft bisher hauptsächlich auf die Kommunikation zu jeder Zeit und an jedem Ort bezog, integriert das IoT nun Objekte in diese Umgebung. [1]

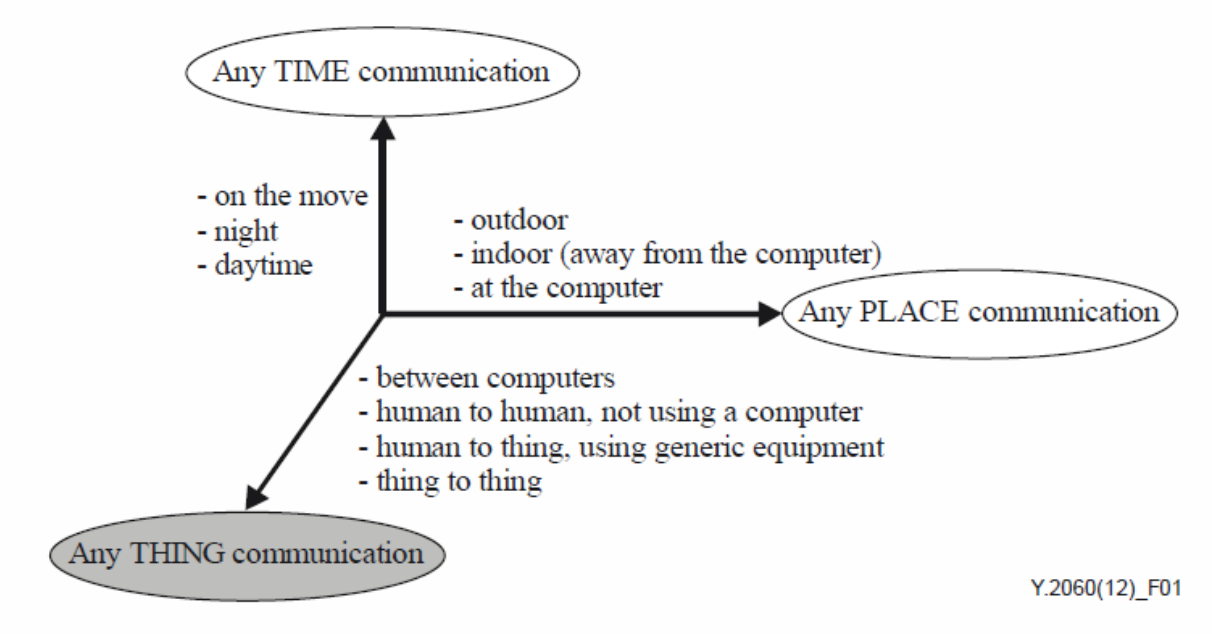

Abbildung 1: Die Erweiterung von Informations- und Kommunikationstechnologien durch das IoT [1]

Auf technischer Ebene ermöglicht dies neue Kommunikationsstrukturen, die so vorher nicht möglich waren. Diese erstrecken sich über sämtliche Netzwerktypen und gewährleisten eine nahtlose Datenübertragung zwischen selbigen. Im Detail bestehen diese aus

- BAN (Body Area Network)
- LAN (Local Area Network)
- WAN (Wide Area Network)
- VWAN (Very Wide Area Network)

Daraus ergeben sich vielfältige Anwendungsmöglichkeiten, von Wearables, über Smarthomes, bis zu Smart Cities. [2] Vor allem auf dem Konsumermarkt steigt die Zahl der vernetzen IoT-Geräte ständig. Nach den niedrigsten Schätzungen wird 2016 die Zahl dieser Devices auf 4 Mrd. geschätzt, bis 2020 soll sie auf über 13,5 Mrd. steigen. Für den gesamten Markt steigt die Anzahl der Geräte in diesem Zeitraum von 6,4 Mrd. auf 20,8 Mrd. [3]

2.2 Platform as a Service

Platform as a Service (PaaS) Systeme sind webbasierte Entwicklungsplattformen, die in der Cloud zur Verfügung gestellt werden. Diese haben den Vorteil, dass sowohl Hardware,

als auch Software nicht selbst beschafft und administriert werden müssen. Somit sinkt für den Entwickler der monetäre und zeitliche Aufwand. Die Plattformen bieten standardisierte Schnittstellen zu den auf ihnen verfügbaren Diensten. Diese Dienste können nicht nur Tools zur Anwendungsentwicklung, sondern auch Datenbankzugriff, Zugangskontrolle, Datenvisualisierung bis hin zur fertigen Anwendungen bereitstellen. [4] Bei PaaS-Systemen handelt es sich um virtuelle Umgebungen. Durch die Virtualisierung sind diese Systeme in Bezug auf Speicher- und Rechenleistung skalierbar, da sie auf einer bestehenden Cloud-Infrastruktur aufbauen. Dies bezeichnet man im Cloud-Computing als Cloud Stack. [5]

2.2.1 Abgrenzung zu Iaas & SaaS

Im Cloud Stack liegt die PaaS in der mittleren Schicht. Sie basiert auf der Infrastructure as a Service (IaaS) und dient als Basis für Software as a Service (SaaS) Angebote.

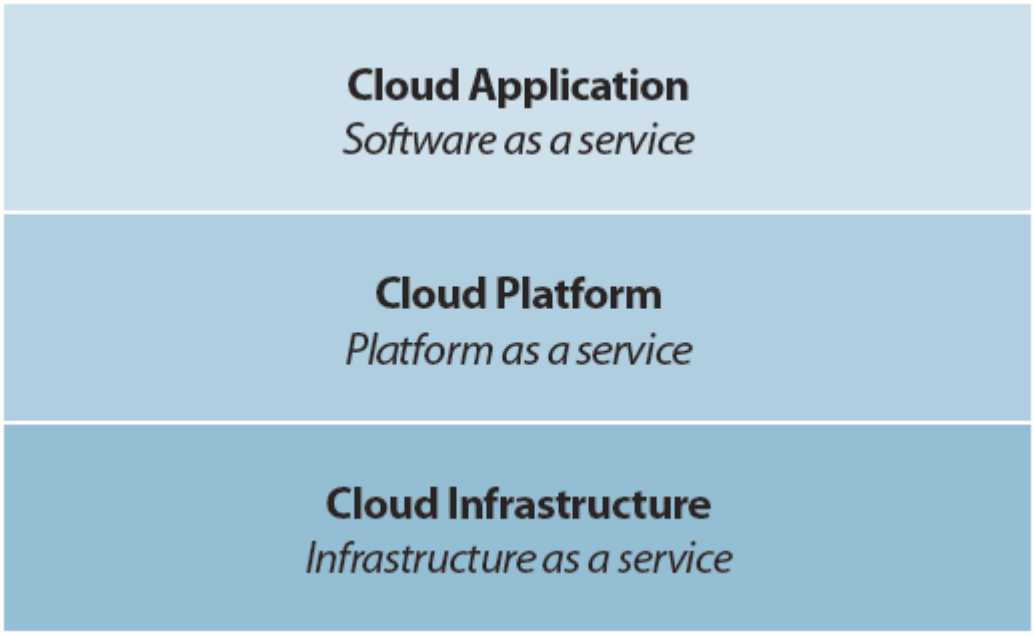

Abbildung 2: PaaS im Cloudstack [5]

Die IaaS bietet grundlegende Ressourcen wie Rechenleistung, Speicher und Netzwerkfunktionalitäten. Dadurch lassen sich normalerweise statische Hardwareumgebungen je nach Anforderung und Nutzungslast dynamisch skalieren. Der Nutzer hat bei IaaS die volle Kontrolle über das zur Verfügung gestellte System. SaaS-Angebote wiederum setzen auf der PaaS auf. Sie umfassen sämtliche Anwendungen, die in einer Cloud lauffähig sind. Im Gegensatz zur PaaS hat der Nutzer hier allerdings keine Kontrolle über die Softwareinstallation selbst, er benutzt diese lediglich. Bekannte Beispiele für SaaS-Angebote sind Google Docs, Microsoft Office Online oder Salesforce.[6]

2.3 Verbindungsprotokolle

Für die Kommunikation zwischen IoT-Device und dem PaaS-Angebot sind standardisierte Nachrichtenprotokolle nötig. Die populärsten Vertreter für das Internet of Things sind hierbei HTTP und MQTT.

2.3.1 HTTP

Das Hypertext Transfer Protocol (HTTP) wurde 1991 eingeführt und dient als zustandsloses Protokoll zur Datenübertragung auf der Anwendungsschicht. Es basiert auf einem Request/Response Pattern. Hierbei sendet der Client eine Anfrage (Request) an den Zielserver und erhält daraufhin eine Antwort (Response). Dieses Pattern erlaubt lediglich eine 1 zu 1 Kommunikation und ist immer unidirektional. Somit können keine Nachrichten an mehrere Empfänger gleichzeitig versendet werden. Auch Push-Nachrichten sind aufgrund dieser Beschränkungen nicht möglich. Ein weiterer Nachteil ist der große Overhead, der durch den HTTP-Header entsteht, der bei jedem Request übertragen wird. Der große Vorteil von HTTP ist die weite Verbreitung. Es existieren für fast alle Programmiersprachen Bibliotheken, die eine Datenübertragung per HTTP gewährleisten. [7]

2.3.2 MQTT

Das Message Queue Telemetry Transport (MQTT) Protokoll ist ein effizientes Nachrichtenprotokoll, welches speziell für Machine to Machine (M2M) Kommunikation entwickelt wurde. Es basiert auf einem Publish/Subscribe Pattern. Der Client kann hierbei per Publish Nachrichten an den Message Broker senden, per Subscribe öffnet der Client einen Kommunikationskanal für eingehende Nachrichten. Hierbei ist die Kommunikation von der Serverseite aus mit bis zu mehreren hunderttausend Clients möglich. Auch Push-Nachrichten sind kein Problem. Ein weiterer Vorteil ist der minimale Protokolloverhead, somit ist eine Implementierung auch auf ressourcenbeschränkten Geräten möglich. Für das Protokoll sind inzwischen Bibliotheken für alle gängigen Programmiersprachen verfügbar. [8]

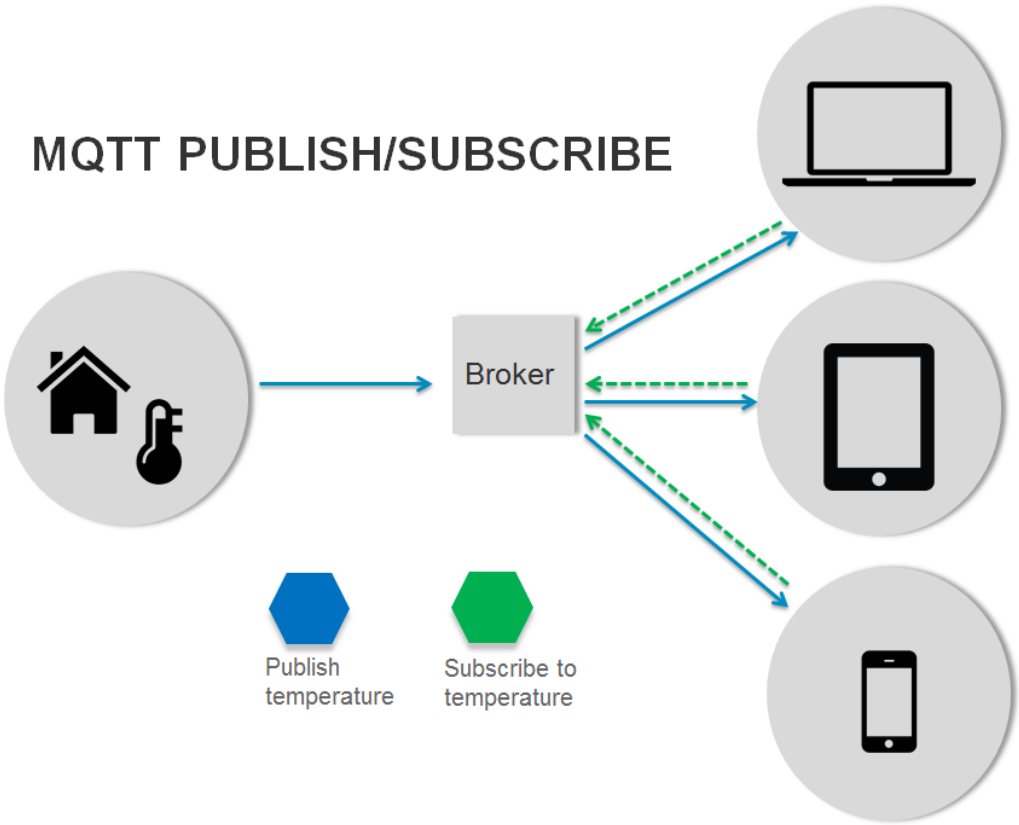

Abbildung 3: Beispiel eines MQTT Publish/Subscribe Modells [9]

3 Untersuchung der PaaS-Angebote

Für den Vergleich der PaaS-Angebote wurden 17 verschiedene Plattformen untersucht. Diese wurden aufgrund ihrer Relevanz bzw. ihres Umfangs ausgewählt. Diese Auswahl umfasst keinesfalls alle zum Zeitpunkt der Untersuchung existierenden Plattformen. Es wurden keine geschlossenen Systeme untersucht, zu denen es keinen kostenfreien Testzugang gibt oder welche nur für Unternehmen zur Verfügung stehen. Folgende Plattformen wurden nicht in die Untersuchung mit einbezogen, da für sie kein kostenloser Testzugang existiert:

- Bosch IoT Suite (https://www.bosch-si.com/de/produkte/bosch-iot-suite/platform-as-service/paas.html)
- Cisco (http://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html)
- WISE-PaaS (http://www.advantech.com/embedded-boards-design-in-services/wisepaas)
- Xively (https://xively.com/)
- Solair (https://www.solaircorporate.com/en/)

Folgende Plattformen wurden nicht in die Untersuchung mit einbezogen, da ausschließlich für Unternehmen zur Verfügung stehen:

- IoT-X Platform (http://www.stream-technologies.com/iotx/)
- ThingWorx (http://www.thingworx.com/)

3.1 Untersuchungskriterien

Um eine Vergleichbarkeit der Untersuchungsergebnisse zu gewährleisten, werden im Folgenden die Untersuchungskriterien festgelegt, auf welche hin die PaaS-Angebote untersucht werden. Diese sind in quantitative und qualitative Kriterien unterteilt.

3.1.1 Quantitativ

Quantitative Kriterien sind alle Kriterien, welche die quantifizierbaren Eigenschaften eines PaaS-Angebots betreffen, d.h. alle messbaren oder in Zahlenwerten darstellbaren Größen.

3.1.1.1 Verbreitung

Bewertet die Verbreitung der Plattform in Bezug auf die Aktivität und den Umfang der User-Community. Hierzu werden die Aktivität im plattformeigenen Forum (falls vorhanden) und dem Entwicklerforum Stack Overflow (http://stackoverflow.com/), sowie die Anzahl der Google-Treffer untersucht. Gesucht wird hierbei auf Stack Overflow und Google nach dem Plattformnamen.

3.1.1.2 Preis

Der Preis umfasst alle Kosten, die durch die Nutzung eines Angebots entstehen. Dies umfasst sowohl Verbindungs- als auch Hardwarekosten.

3.1.1.3 Verbindungsrate

Die Anzahl von erlaubten Datenstreams zwischen Device und Server und die erlaubte Aktualisierungsrate.

3.1.1.4 Datenmenge

Die maximal mögliche Datenmenge in MB, die zwischen Device und Plattform übertragen und dort gespeichert werden kann.

3.1.1.5 Schnittstellen

Anzahl der Schnittstellen zu anderen Services, externen Ressourcen und Programmiersprachen. Hierzu zählen auch SDK's, Libraries und plattformeigene Anwendungen.

3.1.2 Qualitativ

Qualitative Kriterien sind alle Kriterien, welche sich zwar eindeutig in Kategorien einteilen lassen, jedoch keinen quantifizierbaren, mathematischen Wert annehmen können.

3.1.2.1 API

Dieses Kriterium bewertet den Funktionsumfang und die Nutzbarkeit der bereitgestellten API, die verwendet wird, um eine Verbindung zwischen Device und Plattform herzustellen. Zur Vergleichbarkeit wird hierzu als kleinster gemeinsamer Nenner die REST-API verwendet. Die API wird hierbei auf folgende Fähigkeiten untersucht:

- Datenübertragung: Übertragung von Daten von und zur Plattform
- Datenverwaltung: Verwalten und Organisation der Daten
- Devicemanagement: Verwalten der verbundenen Devices
- Diagnostik: Überprüfen des Device-Status
- Eventverwaltung: Verwaltung von Prozessen und Events
- Benutzerverwaltung: Verwaltung der Plattformbenutzer
- Plattformverwaltung: Verwaltung von administrativen Konfigurationen der Plattform
- Anwendungsverwaltung: Verwaltung von Anwendungen der Plattform
- Visualisierung: Erstellen und Anzeigen von Visualisierungen
- Authentifizierung: Rechtemanagement und Authentifizierung

Als Vergleich für die Nutzbarkeit wird ein Beispielaufruf angeführt, mit dem Daten vom Device an die Plattform gesendet werden können.

3.1.2.2 Visualisierung

Untersucht die Möglichkeiten zur Datenvisualisierung der Plattform.

3.1.2.3 Weiterverarbeitung

Möglichkeiten zur weiteren Verarbeitung der gesammelten Daten auf der Plattform.

3.1.2.4 Dokumentation

Bewertet Struktur, Umfang und Benutzerfreundlichkeit der Dokumentation zur Nutzung aller Funktionen des PaaS-Angebotes.

3.1.2.5 Usability

Untersucht die Übersichtlichkeit und Benutzerfreundlichkeit der Benutzeroberfläche und der Funktionen der Plattform. Dies geschieht in Anlehnung an die Grundsätze der Dialoggestaltung der DIN EN ISO 9241-110. [10]

- Aufgabenangemessenheit: Geeignete Funktionalität für die Aufgabe
- Selbstbeschreibungsfähigkeit: Verständlichkeit der Aktionen durch Hilfen/Rückmeldungen

- Erwartungskonformität: Einheitliche Darstellung, Konventionalität
- Steuerbarkeit: Steuerung des Dialogs durch den Benutzer
- Fehlertoleranz: Möglichkeit der leichten Fehlerkorrektur
- Individualisierbarkeit: Anpassbarkeit an den Benutzer

3.1.2.6 Sicherheit

Dieses Kriterium dient der Bewertung der Sicherheit sowohl von ein- und ausgehenden Datenverbindungen, als auch von auf der Plattform gespeicherten und verarbeiteten Daten.

3.2 Plattformspezifische Untersuchung

Im Folgenden werden die einzelnen Plattformen nach den festgelegten Kriterien untersucht.

3.2.1 ThingSpeak

ThingSpeak ist eine offene Datenplattform und API für das Internet of Things. Sie erlaubt es dem Nutzer, Daten von Sensoren oder Aktoren wie z.B. Arduino und Raspberry Pi zu sammeln, speichern, analysieren, visualisieren und diese weiter zu verarbeiten oder entsprechende Aktionen auszulösen. Dies geschieht bei ThingSpeak mit Hilfe von sogenannten Channels. Diese enthalten alle verfügbaren Datenfelder, die mit Sensordaten beschrieben werden können. [12]

Abbildung 4: ThingSpeak Logo [11]

https://thingspeak.com/

3.2.1.1 Verbreitung

ThingSpeak verfügt über ein eigenes Forum mit aktiver Community. Insgesamt sind dort 4589 Mitglieder registriert. Es existieren 2859 Posts zu 803 Themen. [13] Auf Stack Overflow ergab die Suche 60 Treffer. Eine Google Suchanfrage lieferte ca. 261.000 Treffer.

3.2.1.2 Preis

ThingSpeak ist ein kostenloses Angebot. Dies ist Einschränkungen in der Verbindungsrate verbunden.

3.2.1.3 Verbindungsrate

ThingSpeak erlaubt einen eingehenden Datenstream alle 15 Sekunden. [14]

3.2.1.4 Datenmenge

Es gibt zum Stand der Untersuchung keine Beschränkung der übertragbaren und speicherbaren Datenmenge.

3.2.1.5 Schnittstellen

ThingSpeak verfügt über mehrere Schnittstellen zu internen Apps, die wiederum als Schnittstellen für externe Anwendungen dienen. Diese sind:

- ThingTweet App
  Eine Schnittstelle, über die Nachrichten an Twitter gesendet werden können. [15]
- TweetControl App
  Ermöglicht die Steuerung eines Devices per Tweet. [16]

- TalkBack App
  Ermöglicht das Speichern von Aktionsketten, die vom verbundenen Device nacheinander ausgeführt werden sollen. [17]
- ThingHTTP App
  Löst auf einen HTTP-Request hin eine vorher gespeicherte Aktion aus oder sendet einen eigenen Request. [18]
- MATLAB Visualizations App
  Bietet Visualisierungsmöglichkeiten und Schaubilder für die Datenvisualisierung. [19]
- TimeControl App
  Ermöglicht zeitgesteuerte Aktionen sowohl für Devices, als auch externe Webseiten und Services. [20]
- Plugins App
  Ermöglicht die individuelle Anpassung des Channel-Backends per HTML, CSS und JavaScript. [21]

3.2.1.6 API

Die REST-API von ThingSpeak verfügt über folgenden Funktionsumfang:

- Datenübertragung
- Datenverwaltung
- Visualisierung

Über diese GET- und POST-Requests können alle essentiellen Funktionen der Plattform gesteuert werden. [22] Um Daten von einem Device an einen Channel zu senden, muss ein HTTP GET- oder POST-Request an https://api.thingspeak.com/update gesendet werden. Dieser kann folgende Parameter enthalten: [23]

- **api_key** (string) - Write API Key für den Channel (required)
- **field1** (string) - Field 1 data (optional)
- **field2** (string) - Field 2 data (optional)
- **field3** (string) - Field 3 data (optional)
- **field4** (string) - Field 4 data (optional)
- **field5** (string) - Field 5 data (optional)
- **field6** (string) - Field 6 data (optional)
- **field7** (string) - Field 7 data (optional)
- **field8** (string) - Field 8 data (optional)
- **lat** (decimal) - Latitude in Grad (optional)
- **long** (decimal) - Longitude in Grad (optional)
- **elevation** (integer) - Elevation in Meter (optional)
- **status** (string) - Status Update Nachricht (optional)
- **twitter** (string) - Twitter Benutzername verlinkt mit ThingTweet (optional)
- **tweet** (string) - Twitter Status Update (optional)
- **created_at** (datetime) – Datum, an dem dieser Eintrag erstellt wurde, im ISO 8601 Format (optional)

Beispiel POST-Request: [23]

```
POST https://api.thingspeak.com/update
     api_key=XXXXXXXXXXXXXXXX
     field1=73
```

3.2.1.7 Visualisierung

ThingSpeak bietet mehrere Möglichkeiten zur Datenvisualisierung. Standardmäßig stehen mehrere Arten von Diagrammen zur Verfügung, mit denen die Daten eines Channels angezeigt werden können. Hierbei lassen sich in einem Chart jeweils die Werte eines Datenfeldes anzeigen. Als Diagrammtypen stehen Linien-, Balken-, Säulen- und Kurvendiagramm zur Verfügung. Es kann zusätzlich ausgewählt werden, ob sich das Diagramm automatisch aktualisiert. [24]

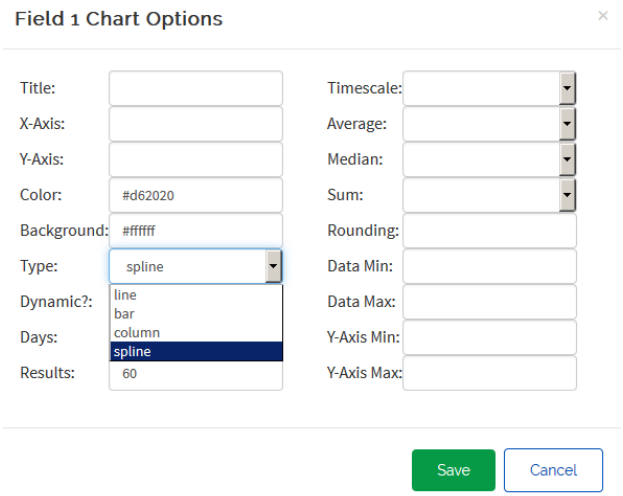

Abbildung 5: ThingSpeak Charts [24]

Für komplexere Visualisierungen bietet ThingSpeak die Mathlab Visualisierungs-App. In dieser lassen sich auch mehrere Datenfelder in einem Diagramm anzeigen. Hier steht außerdem eine größere Anzahl von Diagrammtypen zur Verfügung, die sich individuell anpassen lassen. Dies geschieht über eine Programmierschnittstelle, welche die nötigen Funktionen bereitstellt. [25]

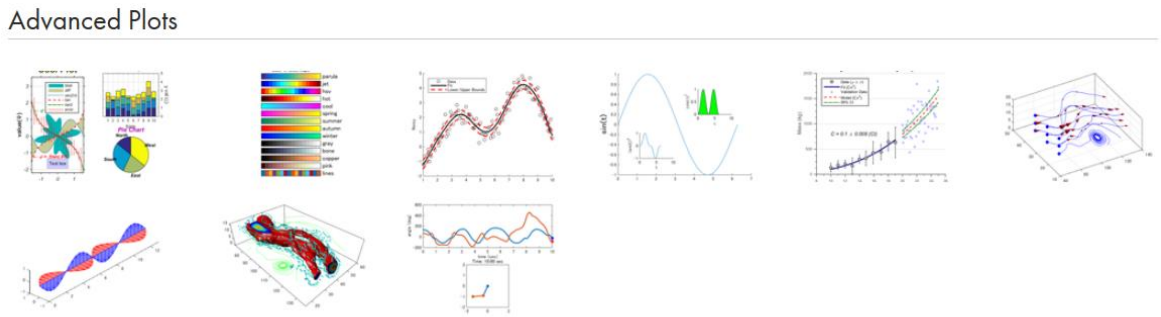

Abbildung 6: Mathlab Advanced Plots [26]

3.2.1.8 Weiterverarbeitung

Die Weiterverarbeitung der Daten bei ThingSpeak funktioniert über Plattformeigene Apps. Das Herzstück ist hierbei die React App. Diese ermöglicht das Ausführen von Aktionen, sobald die Daten eines Channels oder der Channel selbst gewisse Konditionen erfüllen. [27] Daraufhin lassen sich andere Apps aufrufen, welche die eigentliche Aktion ausführen. Dies sind die ThingTweet App, über die Nachrichten an Twitter gesendet werden können [15], die ThingHTTP App, welche das Senden von HTTP-Requests ermöglicht [18], und die MATHLAB Analysis App, welche mathematische Operationen an den Daten ausführen kann. [28]

3.2.1.9 Dokumentation

Die Dokumentation von ThingSpeak ist übersichtlich und klar strukturiert. Sie ist entsprechend des Funktionsumfangs wenig umfangreich, bietet aber einen guten Überblick über alle Funktionen. Sie umfasst unter anderem eine Übersicht mit

Schnelleinstieg, die API-Referenz, Informationen über die Benutzung der ThingSpeak Apps, Beispielanwendungen und Tutorials aus der User-Community und genauere Informationen über HTTP-Header und Fehlercodes. Beispiele und Erklärungen sind oft gut bebildert und verständlich erklärt. [29]

3.2.1.10 Usability

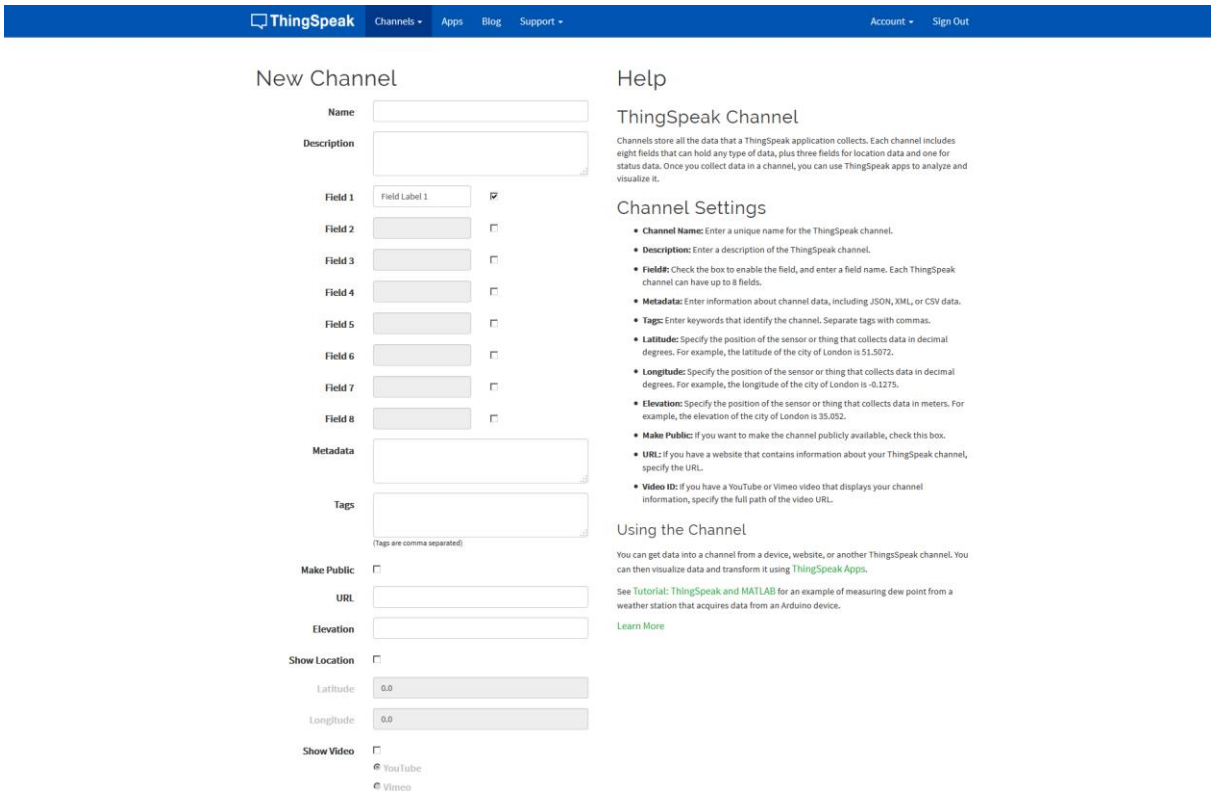

Abbildung 7: ThingSpeak User-Interface [30]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind verständlich gestaltet und bieten eine ausführliche Beschreibung.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist sehr fehlertolerant, fehlende Angaben werden automatisch mit Zufallswerten befüllt.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.1.11 Sicherheit

ThingSpeak nutzt zur sicheren Datenübertragung TLS. [31] Der Channelzugriff ist über einen Write API Key zum Schreiben von Daten und einem Read API Key zum Lesen von Daten abgesichert.

3.2.2 data.sparkfun.com

data.sparkfun.com

Abbildung 8: data.sparkfun.com Logo [32]

https://data.sparkfun.com/

Data.sparkfun.com ist eine IoT-Plattform des Online-Elektronikhändlers SparkFun. Die Plattform basiert auf einer eigens entwickelten Open-Source Software namens phant. [33] Sie soll einen einfachen Service für IoT-Projekte bieten. [32] Verbindungen zur Plattform sind in sogenannten Streams organisiert.

3.2.2.1 Verbreitung

Data.sparkfun.com verfügt über kein eigenes Forum. Die Suche auf Stack Overflow ergab 19 Treffer. Eine Google Suchanfrage lieferte ca. 13.500 Ergebnisse.

3.2.2.2 Preis

Data.sparkfun.com ist kostenlos verfügbar. [32]

3.2.2.3 Verbindungsrate

Die Plattform erlaubt 100 eingehende Datenstreams in einem Zeitfenster von 15 Minuten. Dadurch soll sichergestellt werden, dass auch sehr viele Daten auf einmal übertragen werden können. [32]

3.2.2.4 Datenmenge

Jeder Stream kann maximal 50MB an Daten speichern. Wird dieses Limit erreicht, werden die ältesten gespeicherten Daten gelöscht. [32]

3.2.2.5 Schnittstellen

Data.sparkfun.com verfügt über keine besonderen Schnittstellen zu externen oder internen Services oder Anwendungen. Die einzige Schnittstelle nach außen ist die HTTP REST-API. [34]

3.2.2.6 API

Die REST-API verfügt über folgenden Funktionsumfang: [35]

- Datenübertragung
- Datenverwaltung

Um Daten an einen Stream zu senden, muss ein HTTP GET- oder POST-Request an http://data.sparkfun.com/input/ gesendet werden. Als Parameter können nur Datenfelder verwendet werden, die zuvor auf der Plattform im Stream erstellt wurden.

Beispiel POST-Request: [36]

```
POST https://data.sparkfun.com/input/PUBLIC_KEY
     Phant-Private-Key: PRIVATE_KEY
     temp=91.4&humidity=86%25
```

3.2.2.7 Visualisierung

Data.sparkfun.com stellt auf seiner Plattform lediglich eine tabellarische Anzeige der erfassten Werte bereit.

3.2.2.8 Weiterverarbeitung

Data.sparkfun.com ermöglicht keine Weiterverarbeitung der Daten auf der Plattform.

3.2.2.9 Dokumentation

Die Dokumentation der Plattform ist übersichtlich und klar strukturiert. Sie ist wenig umfangreich und besteht aus der API-Referenz und jeweils einem zusätzlichen Punkt zur Stream-Erstellung und der Übertragungslimitierungen. Dies erscheint als sehr spärlich und ist dem geringen Funktionsumfang geschuldet. Es existieren Codebeispiele zu den API-Funktionen. [35]

3.2.2.10 Usability

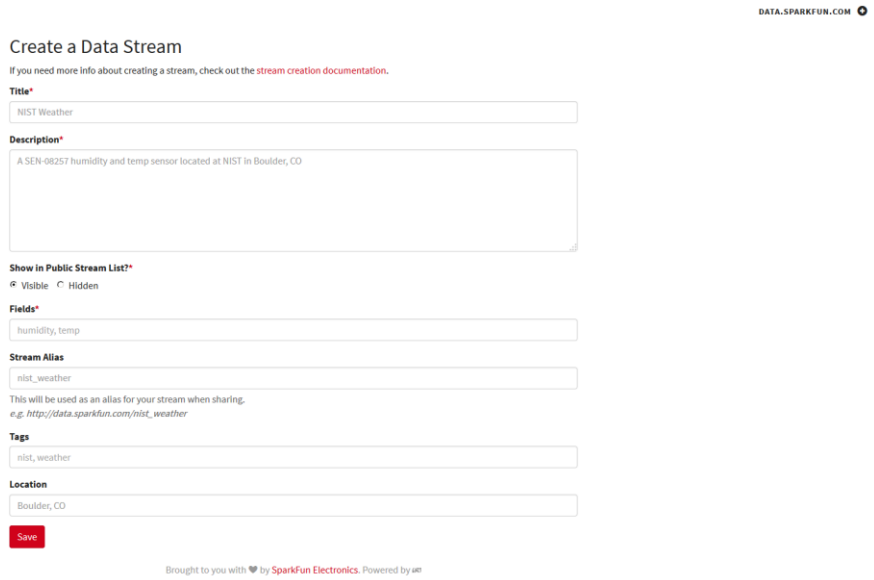

Abbildung 9: data.sparkfun.com User-Interface [37]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind verständlich gestaltet und bieten eine Beschreibung und Beispielwerte.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist sehr fehlertolerant, fehlende Angaben werden präzise gekennzeichnet und angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.2.11 Sicherheit

Data.sparkfun.com nutzt zur sicheren Datenübertragung TLS [32]. Der Streamzugriff kann allerdings nur zum Schreiben mittels eines Private-Keys abgesichert werden [36], auf alle Daten eines Stream kann von außen per öffentlich zugänglichem Public-Key zugegriffen werden. [34]

3.2.3 Runabove IoT Lab

Runabove IoT Lab ist ein Teil der RunAbove Cloud-Plattform. Die Plattform wurde konzipiert um Daten von IoT-Devices zu speichern, visualisieren und analysieren. Die Plattform befindet sich zur Zeit der Untersuchung im Beta-Stadium. [38]

Abbildung 10: RunAbove Logo [38]

https://www.runabove.com/iot-paas-timeseries.xml

3.2.3.1 Verbreitung

Runabove IoT Lab verfügt über ein eigenes Forum, welches allerdings nicht sehr aktiv ist. In 15 Themen finden sich lediglich 18 Posts. [39] Die Suche auf Stack Overflow ergab keine Treffer. Eine Google Suchanfrage lieferte nur 60 Ergebnisse.

3.2.3.2 Preis

Zum Zeitpunkt der Untersuchung ist die Nutzung des Angebots kostenlos.

3.2.3.3 Verbindungsrate

Zum Zeitpunkt der Untersuchung gibt es keine Angaben zu Limitierungen der Verbindungsrate.

3.2.3.4 Datenmenge

Zum Zeitpunkt der Untersuchung gibt es keine Angaben zu Limitierungen der Datenmenge.

3.2.3.5 Schnittstellen

Runabove IoT Lab verfügt über keine besonderen Schnittstellen zu externen oder internen Services oder Anwendungen. Die einzige Schnittstelle nach außen ist die HTTP REST-API. [40]

3.2.3.6 API

Die REST-API verfügt über folgende Funktionen:

- Datenübertragung
- Datenverwaltung
- Devicemanagement
- Authentifizierung

Implementierungsbeispiele zur Datenübertragung für die Programmiersprachen Bash, C, Golang, JQuery, Java, Node.js und Python werden von Runabove auf Github angeboten. [41]
Um Daten an einen Stream zu senden, muss ein HTTP POST-Request an https://opentsdb.iot.runabove.io/api/put gesendet werden. Folgende Parameter sind dabei zulässig: [42]

- **metric** (string) – Namensgebung des Datentyps (required)
- **timestamp** (numeric) - Zeitstempel im Unix Epoch Format in Sekunden oder Millisekunden (required)
- **value** (numeric) – Der eigentliche Datenwert (required)
- **tags** (object) – Zusätzliche Tags für die Daten (optional)

Die Parameter müssen hierbei in einem JSON-Request-Body übertragen werden.

Beispiel POST-Request mit JSON:

```
POST https://opentsdb.iot.runabove.io/api/put
     Authorization: Basic write_token_id:write_token_key

     {
         "metric":"app.test",
         "timestamp":1437591536,
         "value":1,
         "tags": {
                 "key1":"value1",
                 "key2":"value0",
                 }
     }
```

3.2.3.7 Visualisierung

Runabove IoT Lab stellt auf seiner Plattform keine Möglichkeiten zur Visualisierung bereit.

3.2.3.8 Weiterverarbeitung

Runabove IoT Lab ermöglicht keine Weiterverarbeitung der Daten auf der Plattform.

3.2.3.9 Dokumentation

Runabove IoT Lab verfügt über keine wirkliche Dokumentation. Es existieren lediglich 4 Tutorials in einer „Knowledge Base“, dementsprechend schlecht sind Struktur und Umfang. Die Tutorials sind zweckmäßig und teilweise mit Beispielen und Bildern angereichert. Ein wenig mehr Tiefe wäre gerade bei der Dokumentation der API wünschenswert. [43]

3.2.3.10 Usability

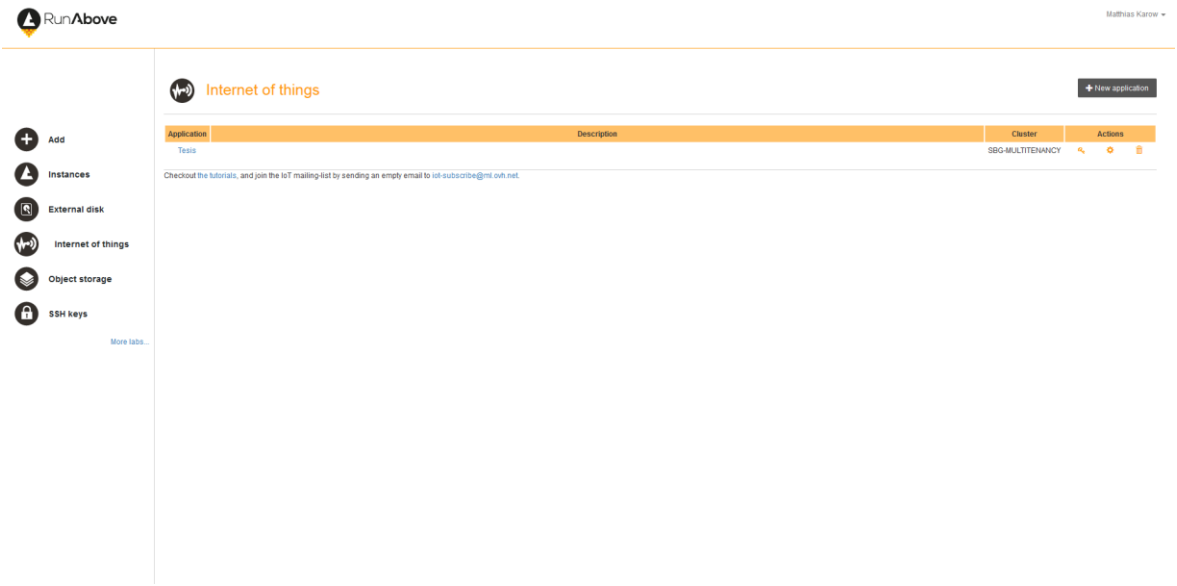

Abbildung 11: RunAbove IoT Lab User-Interface [44]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind verständlich gestaltet. Es existieren keine zusätzlichen Hilfestellungen oder Beschreibungen.

Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet. Einige Konfigurationsmöglichkeiten sind über gut erkennbare Icons erreichbar.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.3.11 Sicherheit

Alle Verbindungen von und zur Plattform sind mit TLS verschlüsselt. Der Zugriff wird per Read- und Write-Tokens authentifiziert. [42]

3.2.4 flowthings.io

Flowthings.io ist eine Cloud basierte Plattform, die das Entwickeln von Internet of Things Anwendungen ermöglichen soll. Sie legt hohen Wert auf die freie Anbindung zu externen Datenquellen, Diensten und Anwendungen. [45] Die Datenerfassung und Verarbeitung basiert auf 3 Bausteinen: Drops, Flows und Tracks. Drops sind hierbei die Daten, die an die Plattform gesendet werden. Flows bestehen aus den Echtzeit-Datenströmen von Drops. Tracks wiederum stellen eine Verbindung zwischen einzelnen Flows her.

Abbildung 12: flowthings.io Logo [45]

https://flowthings.io/

3.2.4.1 Verbreitung

Flowthings.io verfügt über ein eigenes Forum, welches nicht sehr aktiv genutzt wird. Es existieren 15 Themen mit insgesamt 58 Posts. [46] Die Suche auf Stack Overflow ergab keine Treffer. Auf Google lieferte die Suchanfrage ca. 2.130 Ergebnisse.

3.2.4.2 Preis

Die Plattform ist zum Zeitpunkt der Untersuchung kostenlos verfügbar. [45]

3.2.4.3 Verbindungsrate

Zur Verbindungsrate liegen keine Informationen vor.

3.2.4.4 Datenmenge

Zur Datenmenge selbst gibt es keine Aussage. Allerdings existiert eine Limitierung der möglichen zu speichernden Drops in einem Flow auf 100.000 Stück. [47]

3.2.4.5 Schnittstellen

Flowthings.io bietet mehrere Libraries für die einfache Verbindung zur Plattform. Zur Verfügung stehen: [48]

- Python Library
- Node Library
- Java Library
- Ruby Library
- C Library
- C# Library
- Browser Client (JavaScript)

- Angular Client (Angular.js)

Weitere Schnittstellen stehen über HTTP-REST, WebSockets, Webhooks und MQTT zur Verfügung. [49]

3.2.4.6 API

Die REST-API von flowthings.io verfügt über folgenden Funktionsumfang: [50]

- Datenübertragung
- Datenverwaltung
- Benutzerverwaltung
- Devicemanagement
- Authentifizierung

Um Daten an einen Stream zu senden, muss ein HTTP POST-Request an https://api.flowthings.io/v0.1/<account_id>/drop/ gesendet werden. Als Parameter sind sämtliche Werte zulässig, deren Datentypen von JSON unterstützt werden. Ist dies nicht der Fall, muss der Datentyp selbst definiert werden. Die Parameter müssen hierbei in einem JSON-Request-Body übertragen werden.

Beispiel POST-Request mit JSON: [51]

```
POST https://api.flowthings.io/v0.1/<account_id>/drop
     Content-Type:application/json
     X-Auth-Token:Authentication Token

     {
      "path": "/my_account_id/my_first_flow",
      "location": {
            "lat": 40.703285,
            "lon": -73.987852
      },
      "elems": {
            "name": "foo",
            "stats": {
                  "voltage": 15.4,
                  "temp": 30.4,
                  "uptime": 1992343.4
            }
      }
     }
```

3.2.4.7 Visualisierung

Flowthings.io bietet zur Datenvisualisierung 3 verschiedene Diagrammtypen. Diese sind das Kurven-, Linien- und Stufendiagramm. Es lassen sich pro Diagramm jeweils nur die Werte eines einzelnen Datenflows darstellen. Die Werte werden immer dynamisch aktualisiert. Hierzu können Sampleraten von 1, 3, 10 und 30 Sekunden gewählt werden.

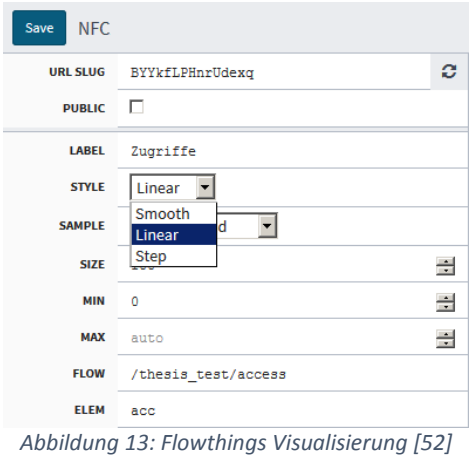

Abbildung 13: Flowthings Visualisierung [52]

3.2.4.8 Weiterverarbeitung

Die Weiterverarbeitung der Daten auf Flowthings.io wird über sogenannte Tracks realisiert. Tracks sind mit einem Flow verknüpft und werden ausgelöst, sobald ein neuer Drop erfasst wird. Tracks können hierbei mehrere Funktionen erfüllen. Sie können Drops an andere Flows weiterleiten, die Daten eines Drops selbst analysieren und daraufhin eine Aktion auslösen, oder per HTTP mit externen Services kommunizieren. [53] Tracks können sehr einfach über die plattformeigene Track-GUI erstellt und verwaltet werden.

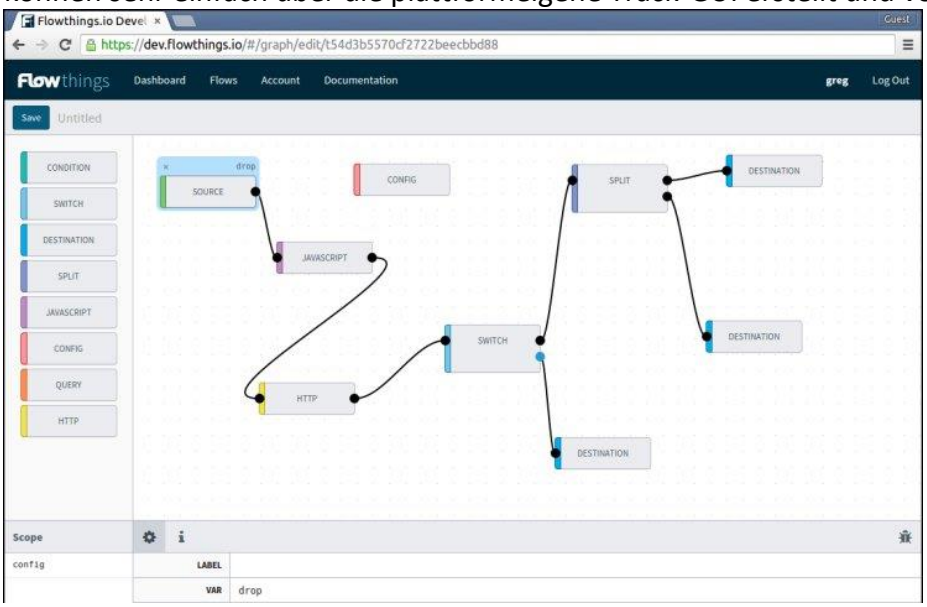

Abbildung 14: Flowthings.io Track GUI [54]

Diese ermöglicht es, einzelne Funktionsbausteine per Drag and Drop miteinander zu verknüpfen. Für komplexere Operationen können eigene Funktionen mit der serverseitigen JavaScript-Umgebung Flow.js erstellt werden. [55]

3.2.4.9 Dokumentation

Die Dokumentation ist sehr umfangreich und deckt alle Funktionen, die zum Betrieb der Plattform nötig sind ab. Die Kernkonzepte werden anschaulich und verständlich erklärt. Die Dokumentation ist übersichtlich strukturiert, so dass die gewünschten Informationen leicht zu finden sind. Für alle Funktionen der API existieren gut nachvollziehbare Code-Beispiele. Neben den APIs finden sich ein Schnelleinstieg, Informationen zur Authentifizierung und Rechteverwaltung und allgemeine Informationen zum Aufbau der Plattform. [47]

3.2.4.10 Usability

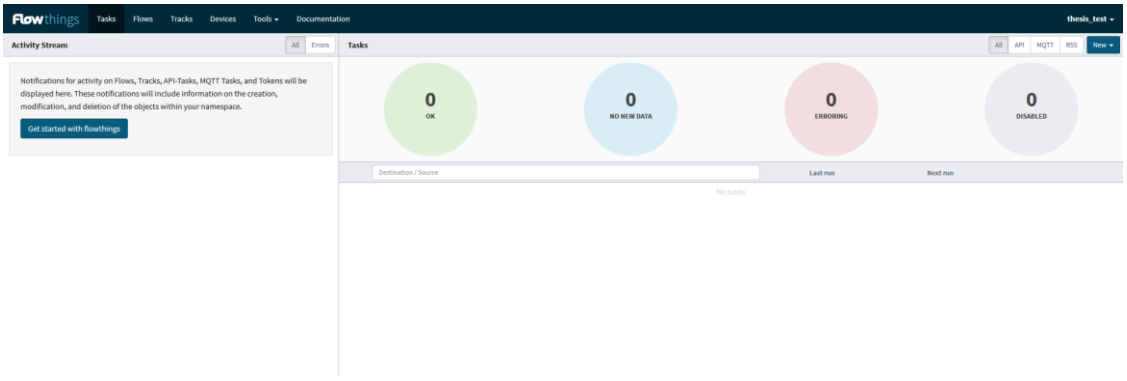

Abbildung 15: flowthings.io User-Interface [52]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind minimalistisch, es existieren keine zusätzlichen Hilfestellungen oder Beschreibungen.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet. Einige Funktionalitäten sind nur schwer zu erreichen.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist wenig fehlertolerant, bei fehlenden Angaben kann der Dialog nicht fortgesetzt werden.
Individualisierbarkeit: Die Benutzeroberfläche ist durch Dashboards anpassbar.

3.2.4.11 Sicherheit

Alle Verbindungen von und zur Plattform sind mit TLS verschlüsselt. Der Zugriff wird Tokens authentifiziert. Flowthings.io benutzt hierzu unterschiedliche Token-Arten. Entweder erfolgt der Zugriff über das Master Token, welches an den Account gebunden ist, oder über eigens erstellte Tokens. Dies hat den Vorteil, dass für diese Tokens limitierte Zugriffsrechte und ein Ablaufdatum festgelegt werden können. Des Weiteren haben sie nur Schreibzugriff auf Drops, jedoch nicht auf Flows oder Tracks. [49]

3.2.5 Carriots

Abbildung 16: Carriots Logo [56]

https://www.carriots.com/

Carriots ist ein Platform as a Service-Angebot für Internet of Things und Machine to Machine (M2M) Projekte. Es richtet sich vor allem an Unternehmen und Großprojekte. Die Plattform soll die Verbindung von Objekten mit dem Internet und die Anwendungsentwicklung für diese erleichtern. [57]

3.2.5.1 Verbreitung

Carriots verfügt über ein eigenes Forum mit aktiver Userbase. Insgesamt existieren 985 Posts zu 240 Themen. [58] Auf Stack Overflow ergab eine Suchanfrage lediglich 5 Treffer. Eine Google Suche lieferte ca. 17.300 Ergebnisse.

3.2.5.2 Preis

Carriots verfügt über mehrere Preismodelle. Für Testzwecke und Prototyping kann ein kostenloser Account erstellt werden. Dieser ist geeignet für ein Minimum von einem Device bis maximal 2 Devices. [59] Die nächste Stufe ist der Corporate Account mit einem Preis von 2€ pro Monat für jedes Device. Geeignet ist dieser für ein Minimum von 10 Devices, nach oben ist keine Grenze gesetzt. [59] Der Lite Account kostet pro Monat und Device 0,50€ und ist ab einer Anzahl von 100 verbundenen Devices geeignet. Auch hier gibt es nach oben keine Grenze. [59]

3.2.5.3 Verbindungsrate

Die Verbindungsrate richtet sich nach der Art des Accounts.

- **Free**: Maximal 500 Streams pro Tag und 10 Streams pro Minute
- **Corporate**: Maximal 1.500*[Anzahl Devices] Streams pro Tag und 50*[Anzahl Devices] Streams pro Minute

- **Lite**: Maximal 25*[Anzahl Devices] Streams pro Tag und 5*[Anzahl Devices] Streams pro Minute

3.2.5.4 Datenmenge

Die zulässige Datenmenge richtet sich nach der Art des Accounts.

- **Free**: Maximal 5KB pro Stream, 5000KB Speicherplatz pro Tag
- **Corporate**: Maximal 10KB pro Stream, 1MB*[Anzahl Devices] Speicherplatz pro Tag
- **Lite**: Maximal 5KB pro Stream, 100KB*[Anzahl Devices] Speicherplatz pro Tag

3.2.5.5 Schnittstellen

Carriots verfügt über eine eigene SDK, welche die Plattformerweiterung ermöglicht. Diese ist in Groovy geschrieben. [60] Weitere Schnittstellen von und zur Plattform stehen per eigener Twitter- und Dropbox-App, [61] sowie dem MQTT-Protokoll und einer HTTP REST-API zur Verfügung. [62]

3.2.5.6 API

Über die Carriots REST API kann die komplette Plattform gesteuert werden.
Die API verfügt über folgende Funktionen: [63]

- Datenübertragung
- Datenverwaltung
- Devicemanagement
- Eventmanagement

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an http://api.carriots.com/streams/ gesendet werden. Dieser kann folgende Parameter enthalten: [64]

- **protocol** (Text) – Verwendetes Kommunikationsprotokoll (required)
- **device** (Text) – Device Identifier (required)
- **data** (JSON) – Zu übertragende Daten (required)
- **at** (Unix Timestamp) – Zeitstempel (required)
- **checksum** (Text/SHA1) – Checksum control code (optional)
- **persist** (Boolean) – Stream Persistenz (optional)

Beispiel POST-Request: [64]

```
POST https://api.carriots.com/streams
     Content-Type:application/json
     Carriots.apiKey:xxxxxxxxxxxxxxxxxxxxxxxx

     {
       "protocol": "v1",
       "device": "defaultDevice@carriots",
       "at": "now",
       "persist": "true",
       "data": {
             "temperature": "20",
             "humidity": "50"
       }
     }
```

3.2.5.7 Visualisierung

Carriots bietet zur Visualisierung mehrere Diagrammtypen. Diese umfassen ein Linien-, Balken-, Punkte- und Kuchendiagramm. Es können pro Diagramm mehrere Datenquellen angezeigt werden. Eine dynamische Aktualisierung der Werte ist nicht möglich. Diese muss manuell ausgelöst werden. [65]

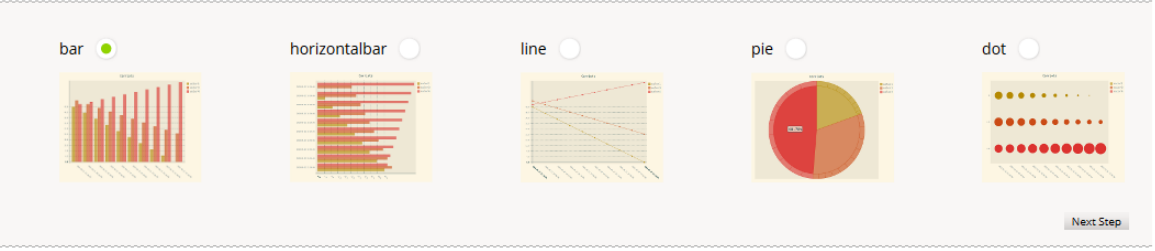

Abbildung 17: Carriots Graphs [66]

3.2.5.8 Weiterverarbeitung

Die Datenverarbeitung auf Carriots wird mit Hilfe sogenannter Listener und Rules ermöglicht. Listener lösen Rules (Aktionen) aus, sobald das verbundene Device oder die eintreffenden Daten sich ändern oder gleich bleiben.

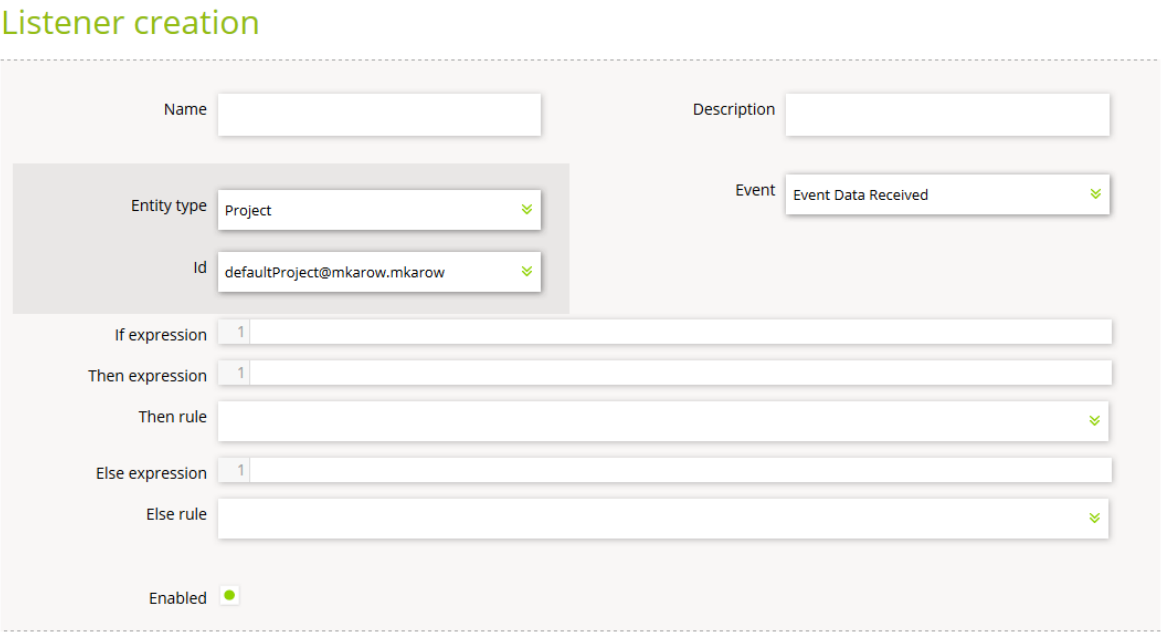

Abbildung 18: Carriots Listener [66]

Zur Programmierung der Bedingungen und Aktionen wird die Carriots SDK verwendet. Diese wurde mit der Programmiersprache Groovy entwickelt und bietet eine mächtige Anzahl an Funktionen und Objekten. [60]

3.2.5.9 Dokumentation

Die Dokumentation ist klar strukturiert, ausführlich und umfasst alle Funktionen der Plattform. Sie enthält einen Überblick über das Ökosystem von Carriots, die API-Referenz, Carriots SDK, MQTT, Einführungen in Arduino und Raspberry Pi, externe Apps, Visualisierung und ein FAQ. Die Dokumentation ist gut bebildert und mit Codebeispielen angereichert. Zum Testen der API existiert eine Entwicklerkonsole. [62]

3.2.5.10 Usability

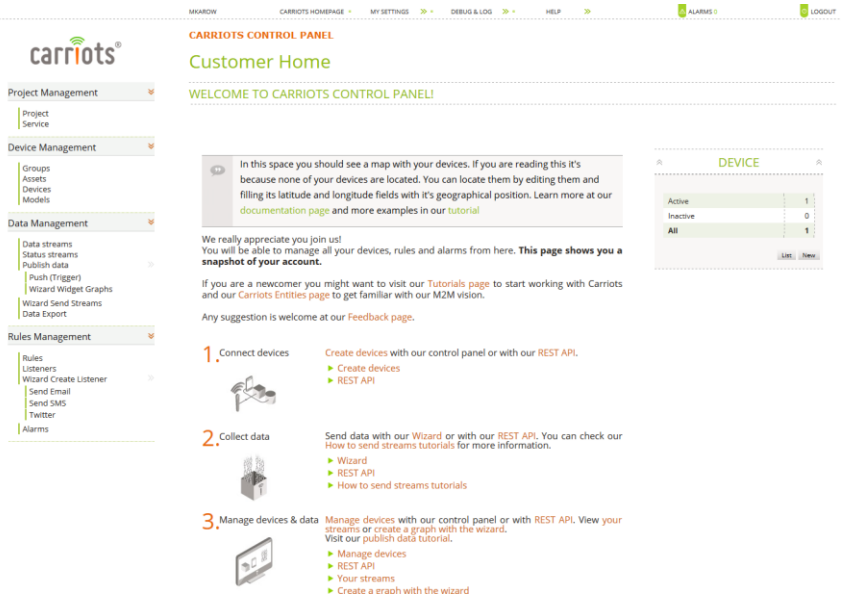

Abbildung 19: Carriots User-Interface [66]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind ausführlich und verständlich, es existieren zusätzliche Hilfestellungen durch vorkonfigurierte Auswahlmöglichkeiten.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden benannt und angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.5.11 Sicherheit

Carriots verwendet zur Übertragungssicherheit TLS über HTTP. Über API-Keys werden Sichtbarkeiten und Zugriff auf einzelne Komponenten der Plattform geregelt. Über einen HMAC-Hash kann zudem die Datenintegrität gewährleistet werden. Carriots unterstützt außerdem den Einsatz und die Implementierung eigener Verschlüsslungsmethoden. [67]

3.2.6 Ubidots

Ubidots ist ein Cloud Service für IoT-Projekte mit Fokus auf Datenanalyse und Projektentwicklung. [68] Die Plattform arbeitet mit 5 Grundbausteinen: Data Source, Variable, Value, Event und Widget. Data Source repräsentiert hierbei das verbundene Device und kann mehrere Variablen enthalten. Eine Variable bei Ubidots ist hierbei eine Serie von Daten, die sich über die Zeit ändert. Als Value wird der Wert einer Variablen zu einem festen Zeitpunkt bezeichnet. Events lösen Aktionen aus, sobald eine Variable eine gewisse Kondition erfüllt. Widgets dienen der Datenvisualisierung. [69]

Abbildung 20: Ubidots Logo [68]

http://ubidots.com/

3.2.6.1 Verbreitung

Ubidots verfügt über ein eigenes Forum mit aktiver Community. Insgesamt existieren 139 Themen mit 615 Posts. [70] Die Suche auf Stack Overflow lieferte 10 Ergebnisse. Eine Suchanfrage bei Google ergab ca. 15.200 Treffer.

3.2.6.2 Preis

Ubidots verfügt über mehrere Preismodelle, die sich grob in einen Basic- und einen Unternehmensbereich aufteilen lassen. Der Basicbereich beginnt bei einer kostenlosen Option, die auf 5 Sensorvariablen begrenzt ist. Darauf folgt die „Individual“ Option für 12$ pro Monat, begrenzt auf 15 Sensorvariablen. Die letzte Stufe in diesem Bereich ist die „Startup“ Option für 48$ pro Monat mit 65 Sensorvariablen. Der Unternehmensbereich zeichnet sich durch eine höhere Anzahl an nutzbaren Sensorvariablen und professionelle Features wie Usermanagement, Branding und individuellem Support aus. Er beginnt mit einer „Professional“ Option für 120$ pro Monat mit 180 Sensorvariablen. Die nächste Stufe ist die „Business“ Option für 360$ pro Monat, begrenzt auf 600 Sensorvariablen. Die letzte Stufe ist die „Enterprise“ Option, die nach oben frei skalierbar ist. Der Preis ist hierbei von den gewünschten Optionen abhängig und deshalb nur auf Anfrage verfügbar. [71]

3.2.6.3 Verbindungsrate

Die Verbindungsrate liegt im Basicbereich bei 0,5 Mio. Streams pro Monat. Im Unternehmensbereich beträgt die Verbindungsrate 1 Mio. Streams pro Monat, zusätzliches Volumen kann für 5$/Mio. Streams hinzugekauft werden. [71]

3.2.6.4 Datenmenge

Zur Datenmenge selbst liegen keine Daten vor. Nur die Speicherdauer der Datenstreams ist auf maximal einen Monat begrenzt. [71]

3.2.6.5 Schnittstellen

Ubidots stellt mehrere API-Clients zur Verfügung, um die Anwendungsentwicklung zu beschleunigen. Diese umfasst Libraries für folgende Sprachen: [72]

- Python
- Java
- C
- PHP
- Node.js
- Ruby
- LabVIEW

Zusätzlich steht eine HTTP REST-API zur Verfügung.

3.2.6.6 API

Die Ubidots REST API erlaubt die Interaktion mit den Datenquellen, den Variablen und den Datenwerten. Die API verfügt über folgende Funktionen: [73]

- Datenübertragung
- Datenverwaltung
- Devicemanagement
- Authentifizierung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an https://things.ubidots.com/api/v1.6/collections/values gesendet werden. Dieser kann folgende Parameter enthalten:

- **variable** (ID) – ID der zu beschreibenden Variable (required)
- **value** (Decimal) – Zu übertragender Wert (required)
- **Timestamp** (Timestamp/POSIX) – Zeit in Millisekunden (optional)

Beispiel POST-Request: [74]

```
POST http://things.ubidots.com/api/v1.6/collections/values
     Content-Type:application/json
     X-Auth-Token:xxxxxxxxxxxxxxxxxxxxxxxx

     {
      "variable": "527e656f73d1513793b2d053"
      "value":2,
      "timestamp":1383497090000
     },
     {
      "variable": "527e656f73d1513793b2d053"
      "value":32,
      "timestamp":1383497090001
     }
```

3.2.6.7 Visualisierung

Ubidots bietet zur Visualisierung mehrere Widgets an. Als klassische Diagrammtypen stehen ein Linien- und ein Streudiagramm zur Verfügung. Auch metrische Werte können ausgegeben werden, diese lassen sich nach verschiedenen Kriterien anzeigen. Es lassen sich Durchschnitts-, Maximal- und Minimalwerte, Summen, Anzahl und der letzte erfasste Wert ausgeben. Für sämtliche Werte lässt sich der Zeitraum festlegen, über welchen diese erfasst wurden. Die Abstufungen sind hierbei heute, gestern, letzte Woche und letzter Monat. Ein weiterer Visualisierungstyp ist die Karte, auf welchem sich die GPS-Daten von einem oder mehreren Devices anzeigen lassen. Als genauere Visualisierungsart stellt die Plattform auch eine tabellarische Anzeige der Werte zur Verfügung. Als Letztes steht eine Anzeige zur Verfügung, welche entweder binär „an“ oder „aus“ anzeigt, oder einen Wert auf einer Messanzeige darstellen kann. [75]

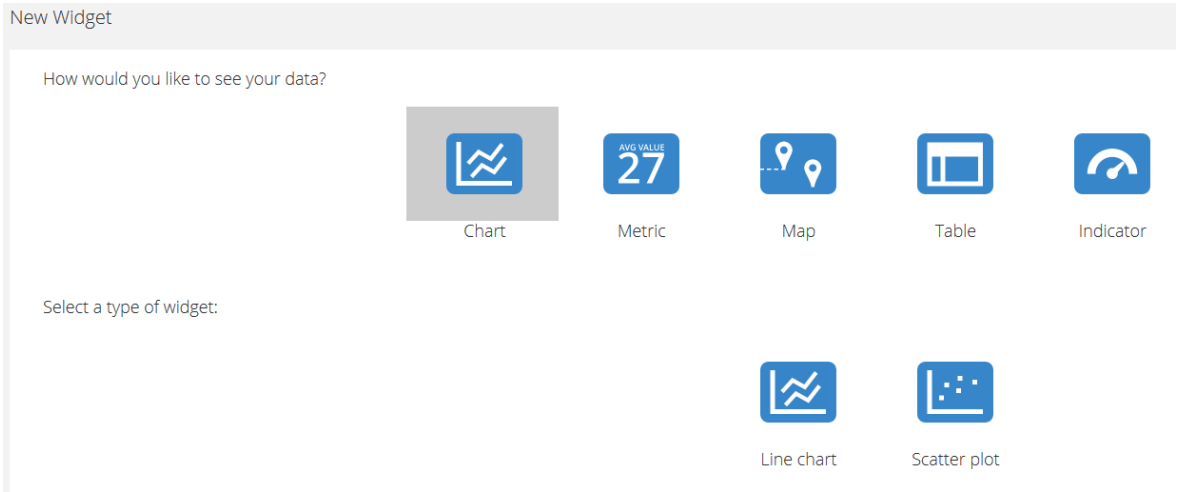

Abbildung 21: Ubidots Widgets [75]

3.2.6.8 Weiterverarbeitung

Die Weiterverarbeitung der Daten auf Ubidots geschieht mit Hilfe von Events. Diese überwachen einzelne Variablen und lösen beim Erreichen einer Bedingung Aktionen aus. Die Erstellung eines Events erfolgt rein visuell über eine Schritt für Schritt Anleitung. Als Operatoren für die Bedingung stehen <, >, <=, >= und = zur Verfügung. Als Aktion ausgelöst werden können das Versenden eines HTTP-Requests, einer E-Mail, einer SMS oder das setzen einer Variablen auf der Plattform. [76]

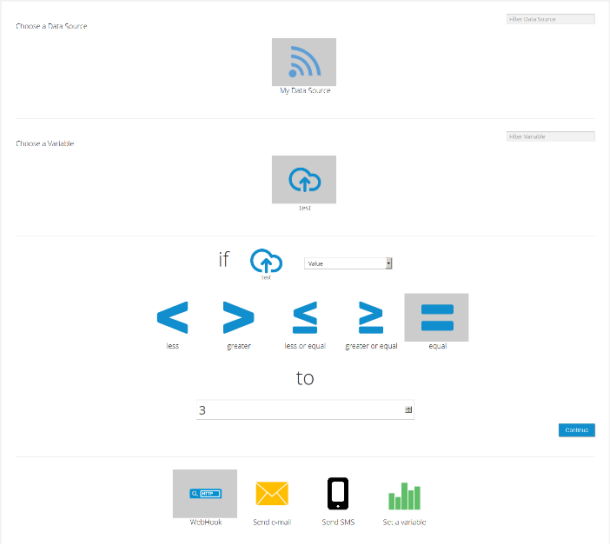

Abbildung 22: Ubidots Events [76]

3.2.6.9 Dokumentation

Die Dokumentation von Ubidots ist sehr übersichtlich, umfangreich und gut strukturiert. Sie umfasst einen Schnelleinstieg, Tutorials für unterschiedliche Devices, API-Libraries und API-Referenz, Community-Projekte, sowie ein Community-Forum. Sämtliche Anleitungen sind ausführlich bebildert und mit Codebeispielen versehen. [77]

3.2.6.10 Usability

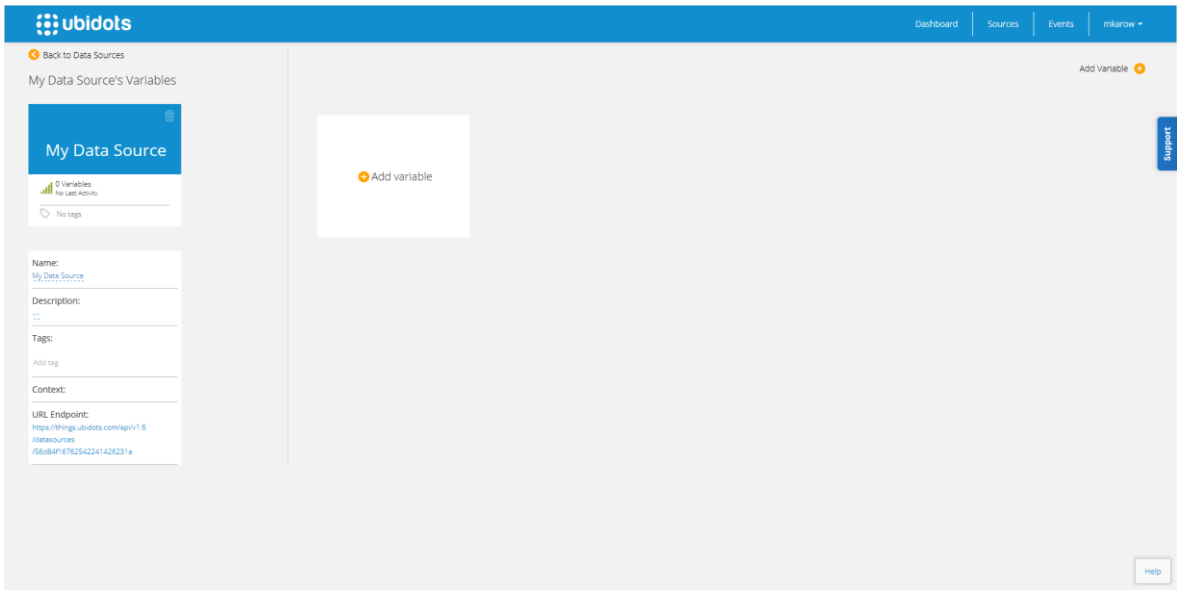

Abbildung 23: Ubidots User-Interface [78]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind ausführlich und verständlich, es existieren zusätzliche Hilfestellungen durch vorkonfigurierte Auswahlmöglichkeiten und Beschreibungen.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.

Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, eine Schrittweise Steuerung ist möglich. Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden angezeigt. Das beenden eines Dialogs ist nur mit allen erforderlichen Angaben möglich.
Individualisierbarkeit: Die Benutzeroberfläche ist durch Widgets anpassbar.

3.2.6.11 Sicherheit

Sämtliche Datenübertragungen von und zu Ubidots sind per HTTPS/TLS verschlüsselt. Zugriffsberechtigungen werden per tokenbasierter Authentifizierung geregelt. [79]

3.2.7 GroveStreams

GoveStreams ist eine Cloudbasierte IoT-Plattform, welche die Nutzbarkeit von verschiedensten Devices und externen Plattformen erleichtern soll. Die Plattform bedient mehrere Zielgruppen, vom Anwendungsentwickler, über Produktmanager und Ingenieure bis hin zu Elektronikenthusiasten. GrooveStreams soll eine skalierbare Plattform bieten, die sämtliche Nutzungsfälle von einem Device bis zu einer Millionen Devices unterstützt. [81] GroveStreams strukturiert seine Daten folgendermaßen: Komponenten organisieren jeweils einen oder mehrere Streams. Streams wiederum enthalten die eigentlichen Datensätze. Einzelne Datensätze werden von GroveStreams Samples genannt.

Abbildung 24: GroveStreams Logo [80]

https://grovestreams.com/

3.2.7.1 Verbreitung

GroveStreams verfügt über ein eigenes Forum, welches allerdings nicht sehr aktiv ist. Es existieren 377 Themen mit 2.148 Posts. [82] Die Suche bei Stack Overflow ergab keine Treffer. Eine Google Suchanfrage lieferte ca. 3.340 Ergebnisse.

3.2.7.2 Preis

GroveStreams ist mit vier verschiedenen Preismodellen verfügbar. Diese unterscheiden sich vor allem in der Anzahl der erlaubten Input/Output-Datenmenge pro Monat und der unterstützen Organisations- und Benutzerzahl. Für Benutzeraccounts selbst entstehen hierbei keine Kosten. Das Beginner Abonnement ist kostenlos und ermöglicht das Erstellen von einer Organisation mit bis zu zwei Benutzern. Das Personal Abonnement kostet 5$ pro Monat und unterstützt bis zu 4 Organisationen mit jeweils bis zu 10 Benutzern. Das Business Essentials Abonnement kostet 95$ pro Monat und unterstützt bis zu 40 Organisationen mit jeweils bis zu 75 Benutzern. Das Business Pro Abonnement kostet 495$ und setzt keine Begrenzung für das Erstellen von Organisationen und Benutzern. [83]

3.2.7.3 Verbindungsrate

GroveStreams erlaubt 150 unauthentifizierte Streams pro Stunde. Streams, welche mit Tokens authentifiziert sind, haben eine Begrenzung auf 350 Requests pro Stunde. Ausgenommen davon sind GET-Requests an den Streamfeed, für diese gibt es keine Begrenzung der Verbindungsrate. Für PUT-Requests existiert ein Zeitlimit von 10 Sekunden zwischen den einzelnen Anfragen. [84]

3.2.7.4 Datenmenge

Pro Datenstream können auf einmal maximal 200MB übertragen werden. Die Datenmenge ist allerdings durch das gewählte Abonnement begrenzt. Es gelten folgende Limitierungen: [84]

- Beginner: maximal 5MB pro Monat inklusive, nicht erweiterbar
- Personal: maximal 20MB pro Monat inklusive, 0,50$ pro zusätzlichem MB
- Business Essentials: maximal 200 MB pro Monat inklusive, 0,40$ pro zusätzlichem MB
- Business Pro: maximal 1GB pro Monat inklusive, 0,30$ pro zusätzlichem MB

3.2.7.5 Schnittstellen

GroveStreams verfügt über keine besonderen Schnittstellen zu externen oder internen Services oder Anwendungen. Die einzige Schnittstelle nach außen ist die HTTP REST-API. [85]

3.2.7.6 API

GroveStreams verfügt über eine einfache und eine erweiterte REST-API. Die einfache API erlaubt die grundlegende Steuerung eines Datenstreams. [86] Die erweiterte API bietet zusätzlich die Steuerung der gesamten Plattform, von der Benutzer- und Organisationsverwaltung bis hin zur Konfiguration des Dashboards. [87]
Die API verfügt über folgende Funktionen: [86]

- Datenübertragung
- Datenverwaltung
- Benutzerverwaltung
- Eventverwaltung
- Authentifizierung
- Plattformverwaltung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP PUT-Request an https://grovestreams.com/api/feed gesendet werden. Dieser kann folgende Parameter enthalten: [86]

- **compId** – ID der Komponente, die den Stream enthält (required)
- **compTmplId** – ID der Komponenten-Vorlage (optional)
- **compName** – Name der neuen Komponente, die aus einer Vorlage erstellt wird (optional)
- **time** – Zeitstempel (optional)
- **timeEl** – Vergangene Zeit zwischen 2 Datensamples (optional)
- **freq** – Frequenz der Datensamples (optional)
- **data** – Wert des Datensamples (optional)
- **stream id parameters** – Stream-ID zur Zuordnung von Samples (optional)
- **rsid** – Gibt den letzten Wert eines Streams als Response zurück (optional)
- **delim** – Trennzeichen zwischen Daten (optional)
- **folder** – Ordner, in dem eine neu erstellte Komponente abgelegt werden soll (optional)
- **dtId** – ID eines Komponenten-Templates, welches einen Stream enthält, der als Vorlage für den neuen Stream verwendet werden soll. (optional)
- **dsId** – ID des Streams im Komponenten-Template, welcher als Template für den neuen Stream verwendet werden soll. (optional)

Beispiel POST-Request: [86]

```
POST https:// grovestreams.com:80/api/feed
     api_key=0ccc2159-2697-3dcd-a211-588e9ccbed71
     compTmplId=compTmpl1
     compId=comp1
     compName=Weather
     freq=60000
     data=1,2
     data=3.0,4.0
```

3.2.7.7 Visualisierung

Abbildung 25: GroveStreams Observation Studio [88]

GroveStreams bietet eine umfassende Menge an Visualisierungsmöglichkeiten. Standardmäßig stehen mehrere Diagrammtypen, wie das Linien-, Balken-, Säulen- und Kuchendiagramm zur Verfügung. Des Weiteren gibt es eine Tabellarische Ansicht und mehrere Mess- sowie binäre Anzeigen. Für die Ortsanzeige stehen zwei Kartentypen zur Verfügung. Eine davon ist statisch und somit nur für Standortdaten geeignet. Die andere erlaubt das dynamische Anzeigen von Bewegungsdaten zum genaueren Tracking. [88]

3.2.7.8 Weiterverarbeitung

Die Weiterverarbeitung der Daten auf GroveStreams erfolgt mit Hilfe von Events und Action Packages. Ein Event wird hierbei mit einem Datenstream verknüpft. Das Event enthält eine oder mehrere Bedingungen, die bei Erfüllung ein Action Package starten. Das Action Package enthält die eigentlichen Aktionen, die ausgeführt werden sollen. Diese können entweder eine Benachrichtigung direkt auf der Plattform, eine E-Mail, eine SMS, oder ein HTTP-Request sein.

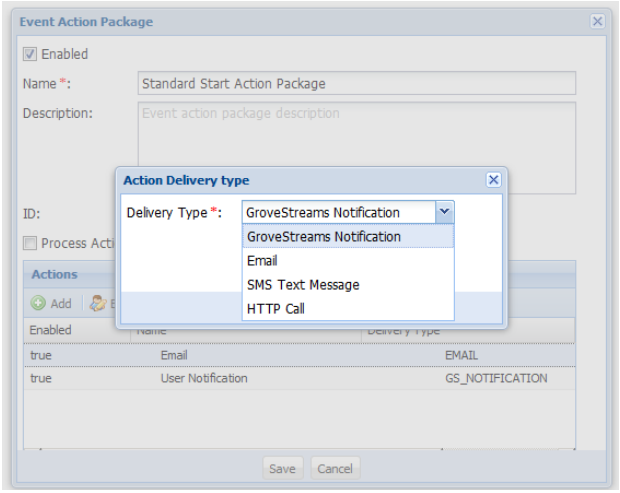

Abbildung 26: GroveStreams Action Package [88]

3.2.7.9 Dokumentation

Die Dokumentation von GroveStreams ist wenig übersichtlich, eher schlecht strukturiert und nicht sehr umfangreich. Sie besteht aus den API-Referenzen zur einfachen und erweiterten API, jeweils einer Seite zu den API-Konventionen und den API-Limitierungen, sowie mehreren Schnelleinstiegen bzw. Tutorials für verschiedene Mikrocontroller und

Programmiersprachen. Anleitungen sind gut und ausführlich bebildert und mit Codebeispielen versehen. [85]

3.2.7.10 Usability

Abbildung 27: GroveStreams User-Interface [88]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind ausführlich, es existieren zusätzliche Hilfestellungen durch vorkonfigurierte Auswahlmöglichkeiten.
Erwartungskonformität: Die Bedienung ist inkonsistent und schwer zu erfassen. Während ein Teil der Menüs per Linksklick bedienbar ist, sind wichtige Funktionen nur über einen Rechtsklick zu erreichen. Diese Funktionalität wird allerdings nicht beschrieben oder angezeigt.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden benannt und angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist über Dashboards anpassbar.

3.2.7.11 Sicherheit

Sämtlicher Datenverkehr von und zu GroveStreams lässt sich durch die Wahl des Endpunktes per TLS verschlüsseln. Für Zugriffsberechtigungen kommen Tokens zur Authentifizierung zum Einsatz. [89]

3.2.8 Exosite

Abbildung 28: Exosite Logo [90]

https://exosite.com/

Exosite ist eine Internet of Things-Plattform mit einem Fokus auf Unternehmen. Sie ist modular aufgebaut und soll den Nutzern die Entwicklung von angepassten Lösungen für das IoT erleichtern. Die Plattform bietet dabei Sicherheit und Skalierbarkeit um verbundene Produkte und Services zuverlässig zur Verfügung zu stellen. [90]

3.2.8.1 Verbreitung

Exosite verfügt über ein eigenes Forum mit 121 registrierten Mitgliedern. Es existieren 383 Posts zu 121 Themen. [91] Die Suche bei Stack Overflow ergab 7 Treffer. Eine Suchanfrage bei Google lieferte ca. 170.000 Ergebnisse.

3.2.8.2 Preis

Exosite bietet mehrere Accounttypen zu gestuften Preisen.
Als Einstiegsaccount bietet die Plattform einen kostenlosen Community-Account, der auf 2 Devices und einen Benutzer begrenzt ist. Die nächste Stufe ist ein Bronze Account für 100$ pro Monat, mit 20 Benutzern inklusive. Der Silver Account ist für 1000$ pro Monat und unterstützt maximal 1000 Benutzer, eigene URLs und Markenbranding. Als höchste Stufe gibt es den Custom Account, dessen Preis auf Anfrage verfügbar ist und sich nach den gewünschten Kapazitäten richtet. [92]

3.2.8.3 Datenrate

Daten können maximal einmal pro Sekunde übertragen werden. [93]

3.2.8.4 Datenmenge

Pro Device können maximal 10MB an Daten auf der Plattform gespeichert werden. Diese werden für zwei Jahr vorgehalten.
Für die Datenübertragung gelten folgende Limits der übertragbaren Datenmenge: [94]

- String: 64 KB
- Integer: 64 Bit
- Float: 8 Exponentenbits und 24 Bit Mantissenlänge
- Binär: 64 KB

3.2.8.5 Schnittstellen

Exosite stellt Libraries für folgende Programmiersprachen und Plattformen zur Verfügung: [95]

- Go
- Python
- Cocoa
- Node.js
- Java
- .NET
- C++
- C
- XMPP

Zusätzlich existieren APIs für HTTP, CoAP, RPC und WebSockets.

3.2.8.6 API

Die Exosite REST API erlaubt die Interaktion mit den Daten und Devices der Plattform.
Die API verfügt über folgende Funktionen: [93]

- Datenübertragung
- Datenverwaltung
- Devicemanagement
- Benutzerverwaltung
- Plattformverwaltung

- Authentifizierung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an http://m2.exosite.com/onep:v1/stack/alias gesendet werden. Dieser kann folgende Parameter enthalten: [93]

- **alias** (ID) – Alias der zu beschreibenden Variable (required)
- **CIK** (ID) – Device ID (required)
- **value** – Datenwert (required)

Beispiel POST-Request:

```
POST http://things.ubidots.com/api/v1.6/collections/values
     X-Exosite-CIK: <CIK>

     alias 1=value 1
     alias 2=value 2
```

3.2.8.7 Visualisierung

Exosite bietet zur Visualisierung mehrere Widgets. Diese reichen von der einfachen Anzeige einzelner Werte, über Messanzeigen und Tabellen bis hin zu Linien- und Balkendiagramm sowie statischen und dynamischen Karten. [96] Zusätzlich lassen sich über die Custom Widget API mit jQuery eigene Visualisierungen generieren. [97]

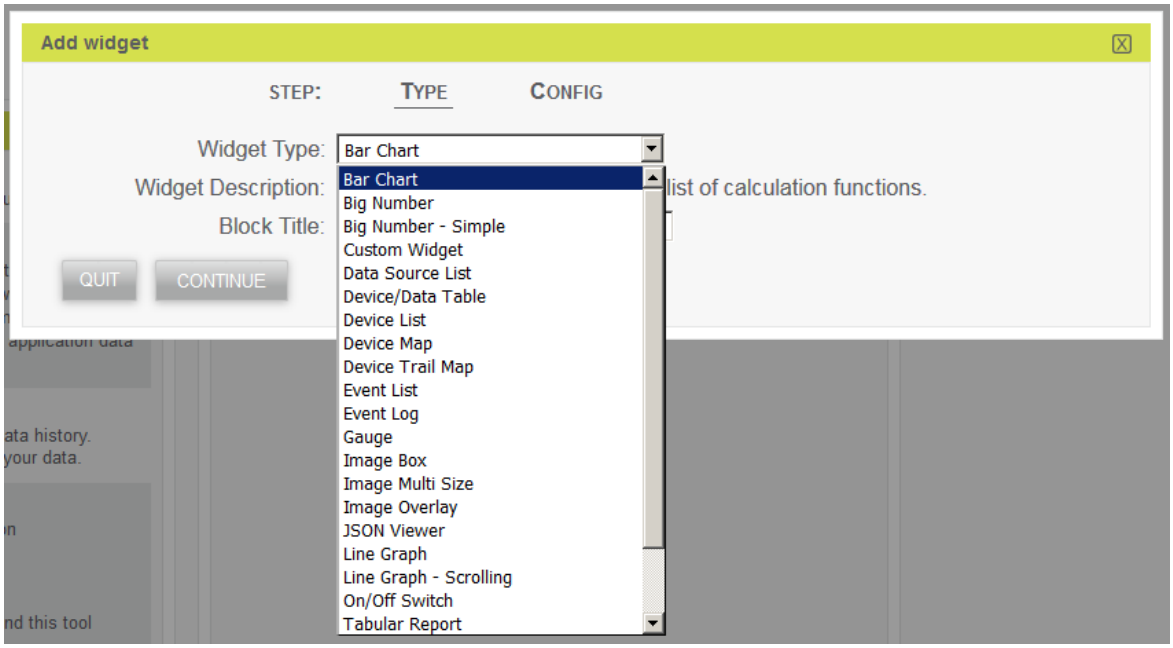

Abbildung 29: Exosite Widgets [96]

3.2.8.8 Weiterverarbeitung

Exosite bietet zwei Möglichkeiten zur Weiterverarbeitung der Daten. Zum einen gibt es die Möglichkeit, Events zu erstellen. Hierbei lassen sich über eine grafische Oberfläche die Datenquelle und die Bedingung auswählen. Als Aktion lässt sich hier allerdings nur ein Alert auslösen, der aus einer E-Mail mit zuvor gewählten Inhalt und Empfänger besteht.

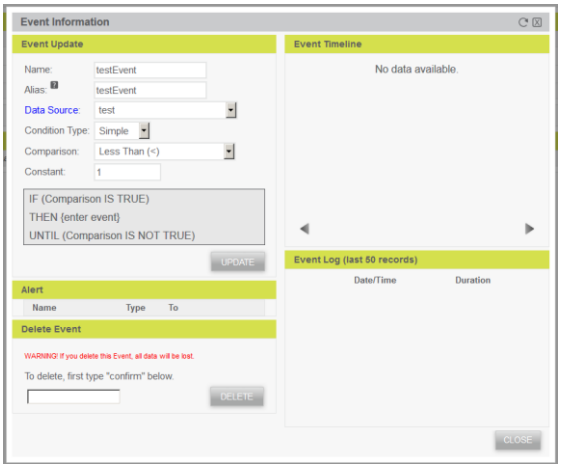

Abbildung 30: Exosite Event [98]

Eine weitaus komplexere Datenverarbeitung ist mit Hilfe von Scripts möglich. Diese werden in der Scriptsprache Lua verfasst, welche ein mächtiges Set an Funktionen mit sich bringt. Hiermit lassen sich Aktionen wie z.B. das Senden eines Tweets oder das Parsen von Daten bewerkstelligen. [99]

3.2.8.9 Dokumentation

Exosite verfügt über eine übersichtliche, klar strukturierte und sehr ausführliche Dokumentation, welche Tutorials und die Funktionsweise der Plattform inclusive SDK und APIs enthält. Die Tutorials sind ausführlich bebildert, zu einigen Themen existiert sogar Videomaterial. Sämtliche Programmierressourcen enthalten anschaulichen Beispielcode. [100]

3.2.8.10 Usability

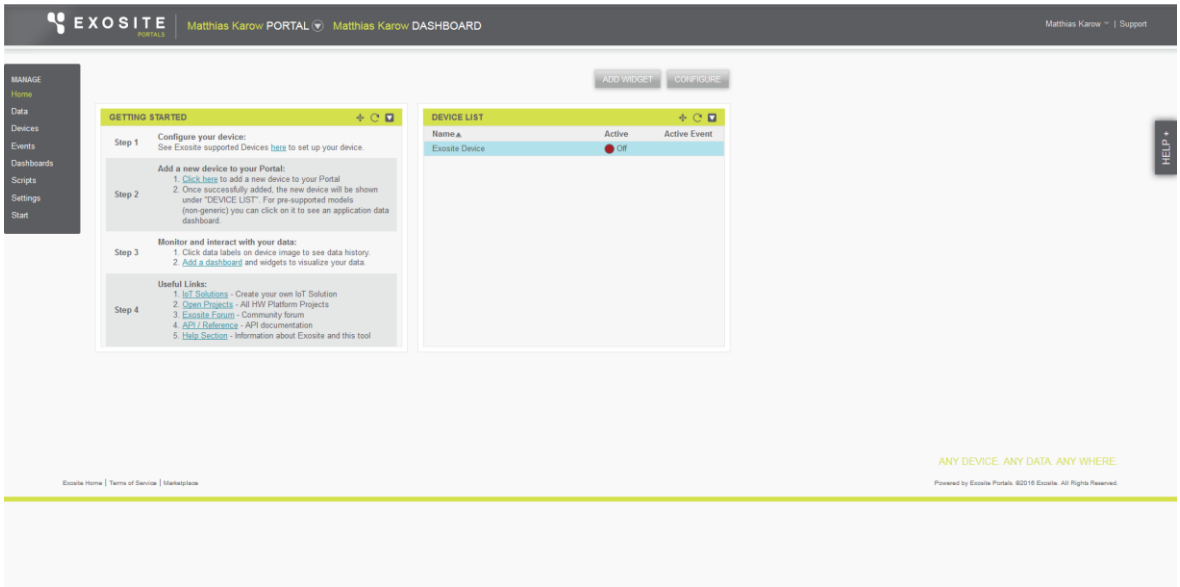

Abbildung 31: Exosite User-Interface [96]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind ausführlich und verständlich, es existieren zusätzliche Hilfestellungen durch vorkonfigurierte Auswahlmöglichkeiten und ausführliche Beschreibungen.

Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet, abgebrochen und schrittweise gesteuert werden. Daten werden zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden benannt und angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist durch Widgets anpassbar.

3.2.8.11 Sicherheit

Exosite verwendet für seine Datenübertragung TLS Verschlüsselung. Die Rechteverwaltung erfolgt über Authentifizierungstokens, sowohl für Benutzer als auch Devices. [101]

3.2.9 Beebotte

Abbildung 32: Beebotte Logo [102]

https://beebotte.com/

Beebotte ist eine Cloud Plattform für in Echtzeit verbundene Objekte. Sie soll die Vernetzung und Kommunikation zwischen tausenden von Objekten ermöglichen und für die unterschiedlichsten Anwendungen nutzbar sein. Beebotte nutzt Amazons AWS, um skalierbar zu sein und eine hohe Verfügbarkeit zu gewährleisten. [102]

3.2.9.1 Verbreitung

Beebotte verfügt nicht über ein eigenes Forum. Die Suche bei Stack Overflow lieferte lediglich ein Ergebnis. Eine Google Suchanfrage ergab ca. 2.440 Treffer.

3.2.9.2 Preis

Beebotte bietet vier unterschiedliche Abonnements, die sich jeweils in ihrem Leistungsumfang unterscheiden. Die Unterschiede liegen vor allem in der Datenrate und werden dementsprechend unter diesem Punkt behandelt. Alle Abonnements setzen dabei kein Limit, wie viele Channels erstellt werden können.
Die angebotenen Abonnements sind: [103]

- XS: kostenlos
- Small: 10$ pro Monat
- Medium: 30$ pro Monat
- Large: 120$ pro Monat

3.2.9.3 Verbindungsrate

Bei der Verbindungsrate unterscheidet Beebotte zwischen zwei Nachrichtenarten: normale Nachrichten und persistente Nachrichten. Der Unterschied zwischen diesen beiden Arten ist, dass persistente Nachrichten auf der Plattform gespeichert werden, normale Nachrichten nicht. Als Nachricht gilt dabei jedes 1 KB Chunk an übertragenen Daten. Für die unterschiedlichen Abonnements gelten folgende Datenraten: [103]

- XS: 50.000 normale Nachrichten und 5.000 persistente Nachrichten pro Tag
- Small: 200.000 normale Nachrichten und 15.000 persistente Nachrichten pro Tag
- Medium: 1 Mio. normale Nachrichten und 50.000 persistente Nachrichten pro Tag
- Large: 5 Mio. normale Nachrichten und 200.000 persistente Nachrichten pro Tag

3.2.9.4 Datenmenge

Beebotte erlaubt bei der Übertragung eine maximale Datenmenge von 16 KB.

3.2.9.5 Schnittstellen

Die Plattform stellt mehrere Libraries für die einfache Integration von verschiedenen Programmiersprachen und Plattformen zur Verfügung. Diese sind: [104]

- JavaScript
- Node.js
- .NET
- Python
- PHP

Zusätzlich stehen APIs für REST, WebSockets und MQTT zur Verfügung. [102]

3.2.9.6 API

Die Beebotte REST API erlaubt die Interaktion mit den Daten und Devices, sowie das Verwalten von Channels und Ressourcen der Plattform.
Die API verfügt über folgende Funktionen:

- Datenübertragung
- Datenverwaltung
- Devicemanagement

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an http://api.beebotte.com/v1/data/write/ gesendet werden. Dieser kann folgende Parameter enthalten: [105]

- **channel** (string) – Channel Name (required)
- **resource** (string) – Name der zu beschreibenden Ressource (required)
- **data** (json) – Datenwert (required)
- **ts** (int) – Timestamp (optional)

Beispiel POST-Request:

```
POST http://api.beebotte.com/v1/data/write/channel/resource
     Content-Type:application/json
     X-Auth-Token:xxxxxxxxxxxxxxxxxxxxxxxx
     {
       "data": "value"
     }
```

3.2.9.7 Visualisierung

Beebotte bietet zur Visualisierung mehrere Widgets. Diese umfassen eine einfache Anzeige einzelner Werte, eine Messanzeige, ein dynamisches Liniendiagramm, ein Liniendiagramm für mehrere Datenquellen, eine Tabelle und eine Heatmap. [107]

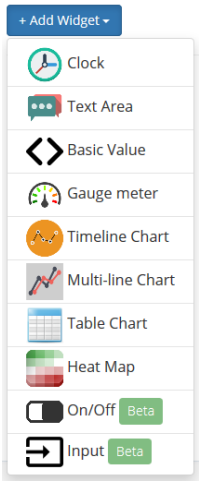

Abbildung 33: Beebotte Dashhboard Widgets [106]

3.2.9.8 Weiterverarbeitung

Beebotte ermöglicht keine Weiterverarbeitung der Daten auf der Plattform.

3.2.9.9 Dokumentation

Beebotte verfügt über eine sehr übersichtliche, gut strukturierte und ausführliche

Dokumentation, die Erklärungen zu grundlegenden Konzepten der Plattform, der Authentifizierung, Daten- und Managementoperationen und der API bietet. Zusätzlich existiert eine Reihe an Tutorials. Konzepte sind mit Schaubildern erklärend ergänzt und die API enthält Codebeispiele für unterschiedliche Libraries. [108] Für die API existiert außerdem eine eigene Konsole, auf der sich die Unterschiedlichen POST und GET Requests testen lassen. [105]

3.2.9.10 Usability

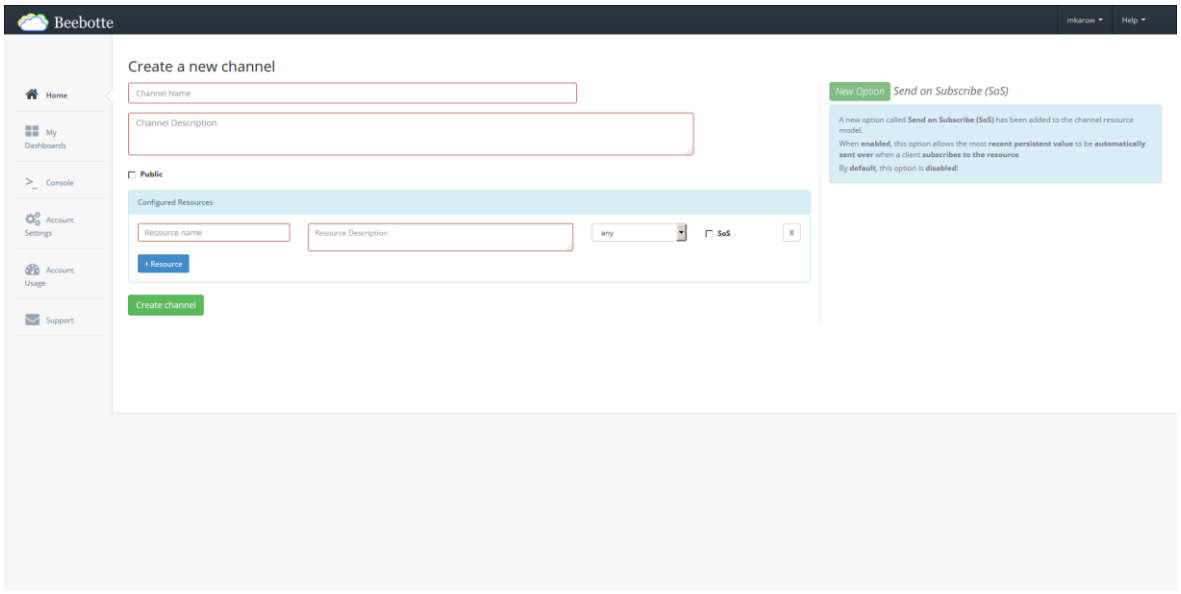

Abbildung 34: Beebotte User-Interface [109]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind ausführlich und verständlich, es existieren zusätzliche Hilfestellungen durch vorkonfigurierte Auswahlmöglichkeiten.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden benannt und angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist durch Dashboards anpassbar.

3.2.9.11 Sicherheit

Beebotte benutzt TLS zur Transport- und AES zur Datenverschlüsselung. Das Zugriffsmanagement erfolgt über Tokens. [108]

3.2.10 MODE

Abbildung 35: MODE Logo [110]

http://www.tinkermode.com/

MODE ist eine Cloud-Plattform für das Internet of Things, welche die Entwicklung von schlauen Produkten für die Zukunft ermöglichen soll. Sie bietet ein umfangreiches Instrumentarium um diese Produkte zu entwickeln, in den Markt einzuführen und zu verwalten. [110]

3.2.10.1 Verbreitung

MODE verfügt über ein eigenes Forum mit lediglich 55 Posts in 21 Themen. [111] Die Suche bei Stack Overflow ergab lediglich einen Treffer. Eine Suchanfrage bei Google lieferte ca. 2.150 Ergebnisse.

3.2.10.2 Preis

MODE bietet zwei Preismodelle. Das Prototyper Abonnement ist kostenlos und hat eine Limitierung auf 20 Devices und 10 Benutzer. Das Enterprise Abonnement ist nur auf Anfrage erhältlich, der Preis berechnet sich hier anhand des gewünschten Umfangs. [112]

3.2.10.3 Verbindungsrate

Zum Zeitpunkt der Untersuchung gibt es keine Angaben zu Limitierungen der Verbindungsrate.

3.2.10.4 Datenmenge

Zum Zeitpunkt der Untersuchung gibt es keine Angaben zu Limitierungen der Datenmenge.

3.2.10.5 Schnittstellen

MODE bietet mehrere SKDs für Hardware- und Mobileplattformen.
Hardware SDKs: [113]

- Broadcom WICED
- Texas Instruments CC3200
- Linux/Embedded Linux
- Node.js für Embedded Linux
- C++ für Embedded Linux

Mobile SDKs: [114]

- iOS
- Android

Zusätzlich existieren eine HTTP REST-API und WebSocket Unterstützung. [115]

3.2.10.6 API

Die MODE REST-API erlaubt die Interaktion und Verwaltung sämtlicher Plattformressourcen. Dies schließt die Daten- und Nutzerverwaltung mit ein.
Die API verfügt über folgende Funktionen: [115]

- Datenübertragung
- Datenverwaltung
- Benutzerverwaltung
- Devicemanagement
- Authentifizierung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP PUT-Request an https://api.tinkermode.com/devices/{deviceId}/event gesendet werden. Dieser kann folgende Parameter enthalten: [116]

- **eventType** (string) – ID des Eventtyps (required)
- **eventData** (json) – zu übertragende Daten (optional)
- **originDeviceId** (int) – ID des Devices (optional)

Beispiel POST-Request:

```
POST https://api.tinkermode.com/devices/{deviceId}/event
     Content-Type: application/json
     Authorisation: ModeCloud <AUTH_CODE>

    {
        "eventType": "sprinkler-status",
        "eventData": {
                "zones": [0,0,0,0,0,0,0,1],
                "serial": 12,
                "timestamp": 1402313424,
                "appSerial": 14,
                "appTimestamp": 14243423423
        }
    }
```

3.2.10.7 Visualisierung

MODE stellt auf seiner Plattform keine Möglichkeiten zur Visualisierung bereit.

3.2.10.8 Weiterverarbeitung

MODE ermöglicht keine Weiterverarbeitung der Daten auf der Plattform.

3.2.10.9 Dokumentation

Die Dokumentation von MODE ist übersichtlich, klar strukturiert und sehr umfangreich. Sie umfasst eine Übersicht über die Funktionsweise der Plattform, einen Schnelleinstieg, Tutorials für unterschiedliche Devices und Services, SDKs und die API-Referenz. Sämtliche Anleitungen sind ausführlich bebildert und mit Codebeispielen versehen.

3.2.10.10 Usability

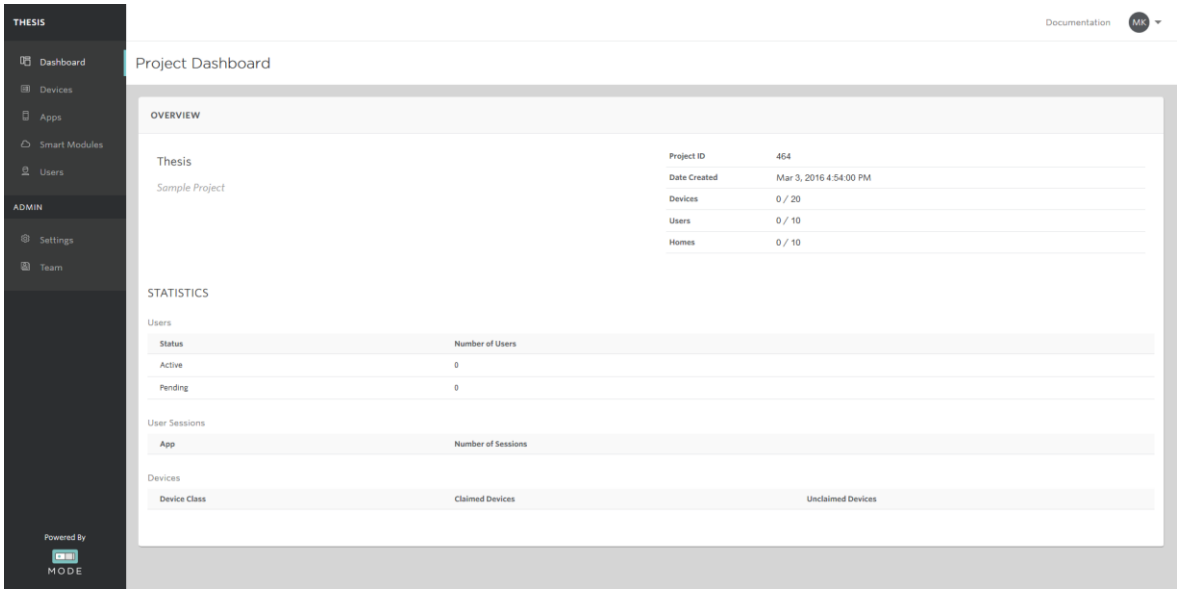

Abbildung 36: MODE User-Interface [117]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind ausführlich und verständlich, es existieren zusätzliche Hilfestellungen durch Beispiele und ausführliche Beschreibungen.

Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, bei fehlende Angaben kann ein Dialog nicht abgeschlossen werden.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.10.11 Sicherheit

MODE verwendet für ein- und ausgehende Verbindungen TLS zur Transportverschlüsselung. Die Authentifizierung erfolgt über Tokens. [118]

3.2.11 Initial State

Initial State ist eine IoT-Plattform mit Fokus auf Datenvisualisierung. Die Plattform bietet eine große Anzahl von Integrations-möglichkeiten mit anderen Web-Services. [119] Initial State organisiert seine Daten in sogenannte Buckets. Diese sind Sammlungen von Events, welche einzelne Datenpunkte mit Zeitstempel repräsentieren. [120]

Abbildung 37: Initial State Logo [119]

https://www.initialstate.com/

3.2.11.1 Verbreitung

Initial State verfügt über ein eigenes Forum mit 60 Posts in 45 Themen. [121] Die Suche bei Stack Overflow ergab einen Treffer. Eine Google Suchanfrage lieferte ca. 3.230 Ergebnisse.

3.2.11.2 Preis

Initial State ist in drei verschiedenen Abonnements verfügbar. Für die Anzahl der inklusiven Devices gibt es hierbei bei keinem der Abonnements ein Limit. Das Free-Abonnement ist kostenlos und speichert übertragene Daten für einen Tag.
Das Standard-Abonnement hat kein Limit bei der Datenspeicherung und kostet 5$ pro Monat bei jährlicher bzw. 6,50$ bei monatlicher Abrechnung. Das Pro-Abonnement hat zusätzlich zu den Leistungen des Standard-Abonnements persönlichen Support, öffentliches Teilen von Inhalten und die Möglichkeit Dateien zu importieren. Es kostet 25$ pro Monat bei jährlicher bzw. 30$ bei monatlicher Abrechnung. [119]

3.2.11.3 Verbindungsrate

Es können für alle Abonnements pro Sekunde 5 Datenstreams übertragen werden. [122] Das Free-Abonnement hat ein Übertragungslimit von 25.000 Streams pro Monat. Für die anderen beiden Abonnements existiert keine Begrenzung. [119]

3.2.11.4 Datenmenge

Jeder Datenstream kann maximal 1MB an Daten übertragen. [119]

3.2.11.5 Schnittstellen

Initial State stellt Libraries und SDKs für unterschiedliche Programmiersprachen und Plattformen zur Verfügung. Diese sind:

- C Library
- Java SDK

- Node.js SDK
- Python Library

Zusätzlich existiert eine HTTP REST-API. [123]

3.2.11.6 API

Die Initial State REST-API bietet einen sehr begrenzten Funktionsumfang. Allerdings lassen sich alle Funktionen direkt in einer Konsole testen.
Die API verfügt über folgende Funktionen: [124]

- Datenübertragung
- Datenverwaltung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP GET-Request an https://groker.initialstate.com/api/events gesendet werden. Dieser kann folgende Parameter enthalten: [125]

- **accessKey** (string) – Key zur Authentifizierung (required)
- **bucketKey** (string) – Key des Buckets (required)
- **eventKey<x>** (string) – Key für den Event-Stream (optional)
- **eventValue<x>** (string) – Zu übertragende Werte (optional)

Beispiel GET-Request:

```
GET https://groker.initialstate.com/api/events
    accessKey=fakee8do2JQN3Eos8Ah2FS8uiFD3Ola2
    bucketKey=bucket1sfa
    eventKey0=eventValue0
```

3.2.11.7 Visualisierung

Initial State bietet sehr umfangreiche Visualisierungsmöglichkeiten. Diese sind in 4 unterschiedliche Data Views gegliedert.

Die Tiles-View erlaubt die Anzeige von Datenwerten in verschiedenen Kacheln. Zur Verfügung stehen die Anzeige des letzten Wertes, Liniendiagramme, Balkendiagramme, Kuchendiagramme, Messanzeigen, Histogramme und Karten für Standortdaten. [126]

Abbildung 38: Initial State Tiles [127]

Die Waves-View stellt Daten in Bezug auf die horizontale Zeitachse dar. Im Gegensatz zu einem Liniendiagramm werden hierbei verschiedene Datentypen unterschiedlich dargestellt. Ein Datenwert, der immer nur den gleichen Wert hat und sich wiederholt, wird als Flag angezeigt. Ein binärer Wert, der immer nur einen von zwei Werten hat, wird als Bit-Signal dargestellt. Eine Datenquelle, die nur numerische Werte enthält wird als normale Linie angezeigt. Daten, die nicht in eine der oberen Kategorien fallen, werden textuell über den Digital String Bus ausgegeben. [128]

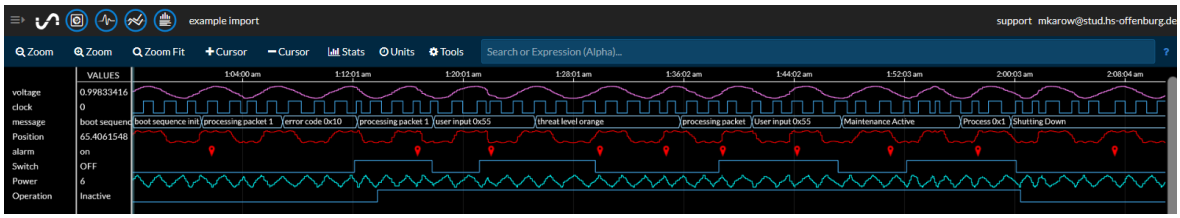

Abbildung 39: Initial State Waves [129]

Die Lines-View stellt Daten in einem Liniendiagramm dar. Hierbei können mehrere Datenquellen gleichzeitig angezeigt werden. [130]

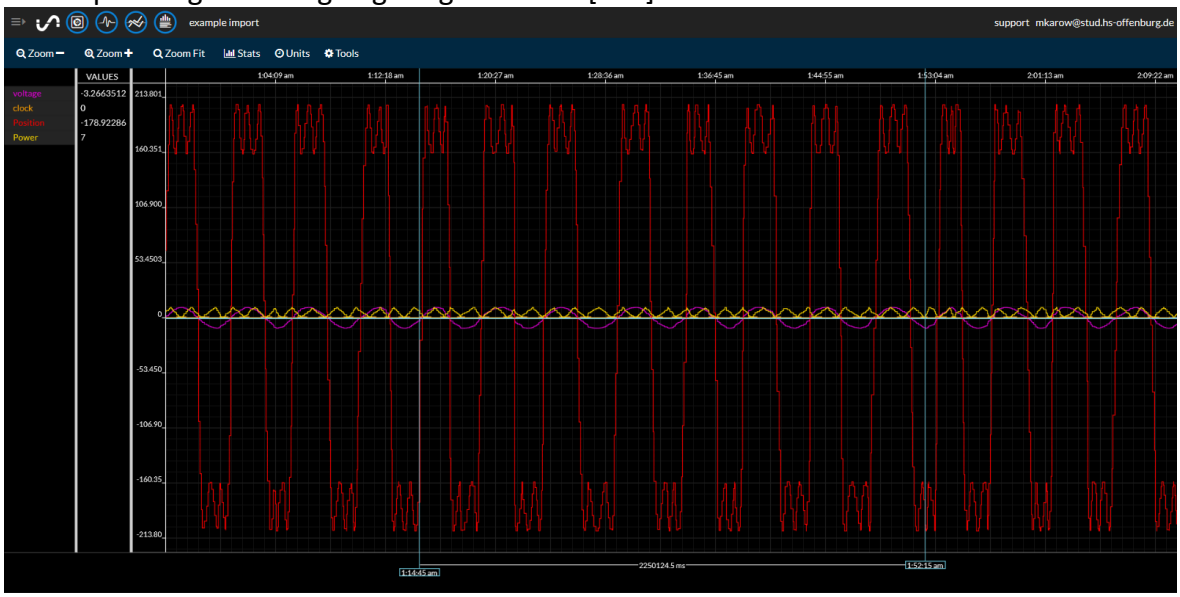

Abbildung 40: Initial State Lines [131]

Als letzte Möglichkeit steht die Anzeige der Rohdaten über die Source-View zur Verfügung. Diese dient hauptsächlich dem einfachen Datenexport aller Werte. [132]

3.2.11.8 Weiterverarbeitung

Initial State ermöglicht keine Weiterverarbeitung der Daten auf der Plattform.

3.2.11.9 Dokumentation

Die Dokumentation von Initial State ist wenig übersichtlich, eher schlecht strukturiert, aber sehr ausführlich. Sie bietet Informationen über die Plattform, einen Schnelleinstieg, Tutorials und Erklärungen der einzelnen Funktionen. Die Tutorials sind sehr ausführlich und decken einen großen Bereich an Devices und Web-Services ab. Sie sind gut bebildert und mit Codebeispielen versehen. [121]

3.2.11.10 Usability

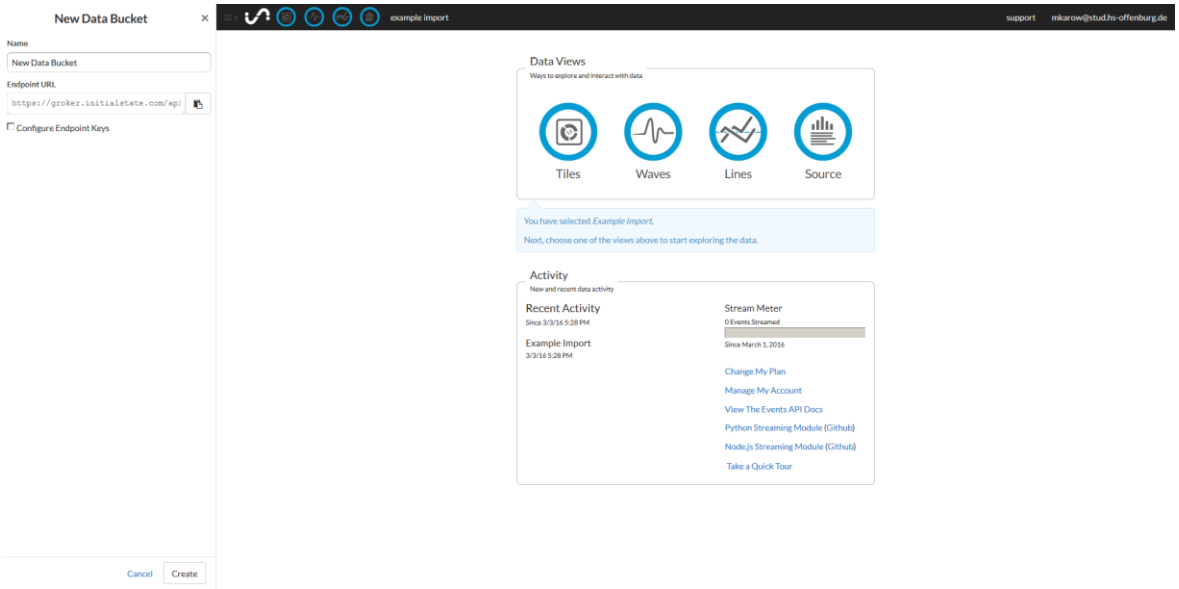

Abbildung 41: Initial State Use-Interface [133]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind minimalistisch aber verständlich, es existieren zusätzliche Hilfestellungen durch interaktive Anweisungen.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet und geschieht Großteils über aussagekräftige Symbole.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden farblich markiert.
Individualisierbarkeit: Die Benutzeroberfläche ist in Apps anpassbar.

3.2.11.11 Sicherheit

Initial State verwendet TLS zur Verschlüsselung der Datenübertragung. Die Sichtbarkeit und der Datenzugriff werden über Access-Tokens gesteuert. [134]

3.2.12 Temboo

Abbildung 42: Temboo Logo [135]

https://temboo.com/

Temboo ist eine IoT-Plattform, die sich auf die Verbindung von Devices mit externen Webservices spezialisiert. Sie bietet über 2000 vorkonfigurierte API-Prozesse und mehrere SDKs für verschiedene Programmiersprachen. [135] Temboo verwendet zur Verarbeitung der empfangenen Daten konfigurierbare Prozesse, sogenannte Choreos.

3.2.12.1 Verbreitung

Temboo verfügt über kein eigenes Forum. Die Suche bei Stack Overflow lieferte 148 Ergebnisse. Eine Suchanfrage bei Google ergab ca. 47.500 Treffer.

3.2.12.2 Preis

Temboo ist zu drei verschiedenen Abonnements verfügbar. Das Free-Abonnement ist kostenlos und enthält 3 auf einen Monat begrenzte Profile und einen monatlichen App Key. Das Individual-Abonnement kostet 7$ pro Monat und enthält 5 permanente Profile, einen permanenten App Key, sowie Standard Support. Das Enterprise-Abonnement ist ab 49$ pro Monat verfügbar und enthält 20 permanente Profile und unlimitierte App Keys. Zusätzlich enthalten sind M2M Messaging, Sensordaten Streaming, vorgefertigte IoT-Apps, Team Management und Premium Support. Weitere Kapazitäten können hinzu gekauft werden. [136]

3.2.12.3 Verbindungsrate

Die Verbindungraten sind je nach Abonnement unterschiedlich limitiert: [136]

- Free: 250 Datenstreams pro Monat
- Individual: 25.000 Datenstreams pro Monat
- Enterprise: 100.000 Datenstreams pro Monat

3.2.12.4 Datenmenge

Das Free- und Individual-Abonnement unterstützen eine übertragbare Datenmenge von 1GB pro Monat, das Enterprise-Abonnement 16GB pro Monat.

3.2.12.5 Schnittstellen

Temboo verfügt über Schnittstellen zu über 100 Webservices und Plattformen, wie z.B. Facebook, Youtube, Google, Microsoft und Youtube. Eine vollständige Liste ist unter https://temboo.com/library/ verfügbar. [137] Zusätzlich werden SDKs für Android, C#, iOS, Java, JavaScript, Node.js, PHP, Processing, Python und Ruby bereit gestellt. Als Standard-Schnittstelle steht eine REST-API zur Verfügung. [137]

3.2.12.6 API

Über die Temboo REST-API lassen sich die zuvor auf der Plattform erstellten Choreos steuern. Die API verfügt über folgende Funktionen: [138]

- Datenübertragung
- Datenverwaltung
- Eventverwaltung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an https://ACCOUNT_NAME.temboolive.com:443/temboo-api/1.0/choreos/myFolder/myChoreo gesendet werden. Die enthaltenen Parameter sind von der Art des Choreos abhängig und können per GET /choreos/<id> abgefragt werden. [138]

Beispiel GET-Request:

```
POST https://ACCOUNT_NAME.temboolive.com:443/temboo-
api/1.0/choreos/myFolder/myChoreo
     Content-Type: application/json
     APP_NAME: APPKEY

     {
        "inputs": [
        {
          "name": "input1",
          "value": "foo"
```

```
        },
        {
          "name": "input2",
          "value": "bar"
        }
        ]
    }
```

3.2.12.7 Visualisierung

Temboo stellt auf seiner Plattform keine Möglichkeiten zur Visualisierung bereit.

3.2.12.8 Weiterverarbeitung

Temboo ermöglicht keine Weiterverarbeitung der Daten auf der Plattform. Die Verarbeitung der Daten geschieht beim verbundenen Webservice.

3.2.12.9 Dokumentation

Temboo verfügt über keine gesammelte Dokumentation. Informationen zur Funktionsweise der Plattform und den APIs finden sich jeweils auf den Seiten zu den einzelnen Choreos und APIs. Diese sind mit Codebeispielen und Schritt für Schritt Anleitungen versehen. [139]

3.2.12.10 Usability

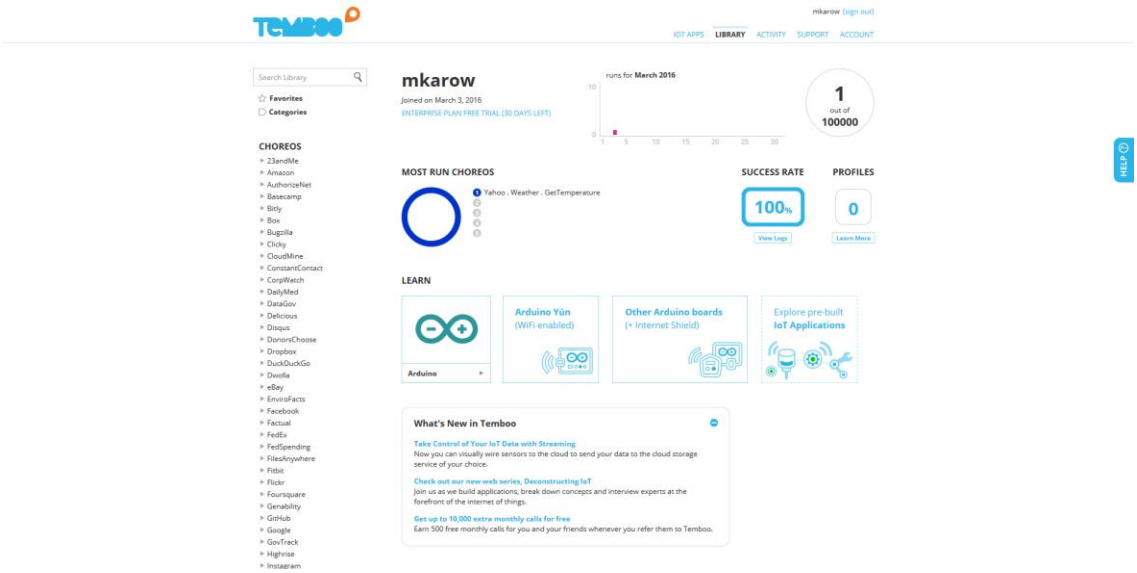

Abbildung 43: Temboo User-Interface [139]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind sehr ausführlich, es existieren umfassende Beschreibungen und Anleitungen zur Benutzung.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.12.11 Sicherheit

Temboo verwendet TLS zur Verschlüsselung der Datenübertragung. Alle gespeicherten Daten werden mit AES verschlüsselt. Zugriffsberechtigungen werden über Tokens geregelt. [140]

3.2.13 Oracle Cloud

Der Oracle Internet of Things Cloud Service ist Teil der Oracle Cloud. Diese bietet einen kompletten Cloud Stack mit IaaS, PaaS und SaaS Angeboten.
Die Plattform richtet sich an Unternehmen, die Gerätedaten verbinden, analysieren und in Geschäftsprozesse und –Anwendungen integrieren wollen. [142]

Abbildung 44: Oracle IoT Cloud Service Logo [141]

https://cloud.oracle.com/

3.2.13.1 Verbreitung

Die Plattform verfügt über kein eigenes Forum. Die Suche bei Stack Overflow ergab keinen Treffer. Eine Google Suchanfrage lieferte 2.240 Ergebnisse.

3.2.13.2 Preis

Die Preisstruktur der Plattform richtet sich nach den verbundenen Geräten. Es gelten folgende Preise pro Gerät: [143]

- Wearables: 0,40$ pro Monat für mindestens 100.000 Geräte
- Consumer-Geräte: 0,80$ pro Monat für mindestens 50.000 Geräte
- Telematikgeräte: 2,00$ pro Monat für mindestens 20.000 Geräte
- Gewerbliche/industrielle Geräte: 3,00$ pro Monat für mindestens 10.000 Geräte

3.2.13.3 Verbindungsrate

Auch die Verbindungsrate ist an die Geräte gebunden. Es gelten folgende Limitierungen: [143]

- Wearables: 1.500 Datenstreams pro Monat inklusive
- Consumer-Geräte: 15.000 Datenstreams pro Monat inklusive
- Telematikgeräte: 100.000 Datenstreams pro Monat inklusive
- Gewerbliche/industrielle Geräte: 100.000 Datenstreams pro Monat inklusive

3.2.13.4 Datenmenge

Zum Zeitpunkt der Untersuchung gibt es keine Angaben zu Limitierungen der Datenmenge.

3.2.13.5 Schnittstellen

Die Oracle Cloud bietet Schnittstellen über eine Java SE API, eine Java API und eine HTTP REST-API. [144]

3.2.13.6 API

Die REST API für den Oracle Internet of Things Cloud Service erlaubt die Interaktion mit den Datenquellen und die Konfiguration der Plattform.
Die API verfügt über folgende Funktionen: [145]

- Datenübertragung

- Datenverwaltung
- Authentifizierung
- Anwendungsverwaltung
- Plattformverwaltung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an https://iotserver/iot/api/v1/messages gesendet werden. Dieser kann folgende Parameter enthalten: [146]

- **id** (string) – ID der Nachricht (required)
- **clientId** (string) – ID für Nachrichten-Tracking (required)
- **source** (string) – ID des Clients (required)
- **destination** (string) – ID des Endpunktes, an den die Nachricht gesendet wird (optional)
- **sender** (string) – ID des Absenders (required)
- **eventTime** (string) – Zeitpunkt des Events (required)
- **type** (string) – Art der Nachricht
- **properties** (object) – Zusätzliche Metadaten (optional)
- **diagnostics** (object) – Zusätzliche Informationen zur Diagnostik (optional)
- **direction** (string) – Gibt an, ob die Nachricht vom Device oder an das Device gesendet wird (optional)

Beispiel POST-Request:

```
POST https://iotserver/iot/api/v1/messages
     Content-Type:application/json
     Authorisation:Bearer someRandomTokenFromOAuth

     [{
       "clientId":"12e08864-7616-4bf3-b70f-0cc137543aa1",
       "source":"0-YYNA",
       "destination":"",
       "eventTime":1442965615367,
       "sender":"",
       "type":"DATA",
       "properties":{},
       "payload":{
             "format":"urn:oracle:iot:device:data:thermometer",
             "data":{"temperature":70}
       }}]
```

3.2.13.7 Visualisierung

Die Oracle Internet of Things Cloud bietet standardmäßig nur eine Visualisierung per Liniendiagramm und Tabelle an. Erweiterte Visualisierungen können allerdings mit Hilfe des Data Visualization Dienstes (https://cloud.oracle.com/en_US/data_visualization) der Oracle Cloud realisiert werden. [147]

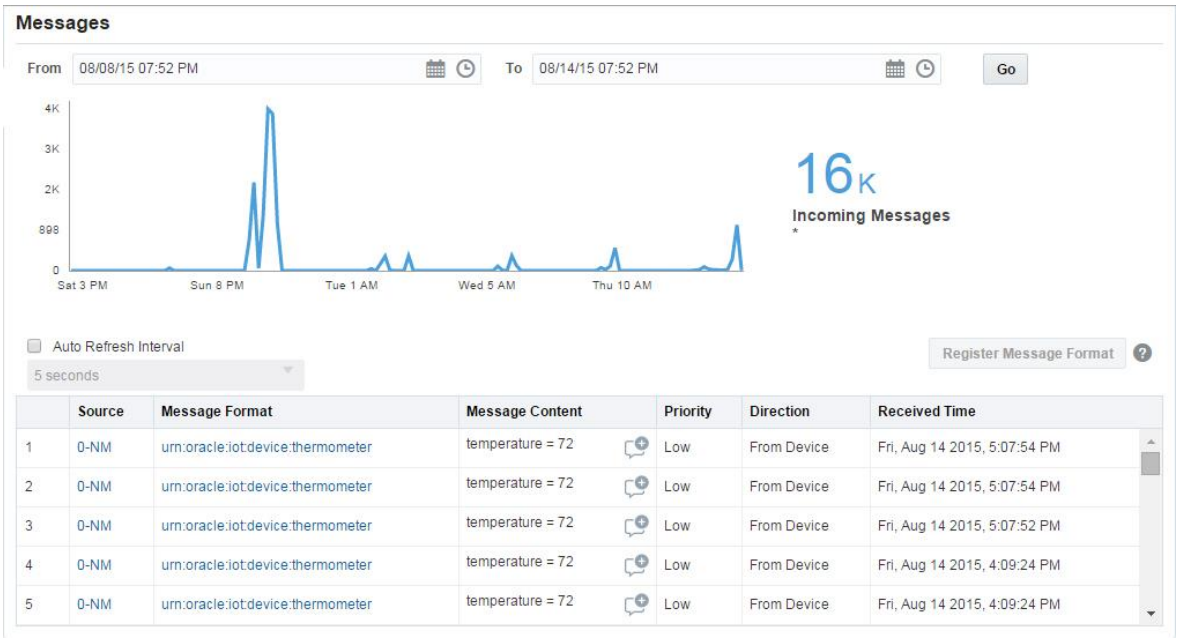

Abbildung 45: Oracle Cloud Visualisierung [148]

3.2.13.8 Weiterverarbeitung

Der Oracle Internet of Things Cloud Service ermöglicht keine Weiterverarbeitung der Daten direkt auf der Plattform. Dies liegt am Fokus der Plattform, der auf der Integration von externen Enterprise-Applications liegt.

3.2.13.9 Dokumentation

Der Oracle Internet of Things Cloud Service bietet eine übersichtliche, gut strukturierte und ausführliche Dokumentation. Diese enthält Informationen über den Aufbau der Plattform, die Funktionsweise der einzelnen Komponenten, Tutorials für die Benutzung so wie die API Referenz. Die Dokumentation ist sehr gut aufgebaut und bietet zahlreiche bebilderte Schritt für Schritt Anleitungen und Codebeispiele. [149]

3.2.13.10 Usability

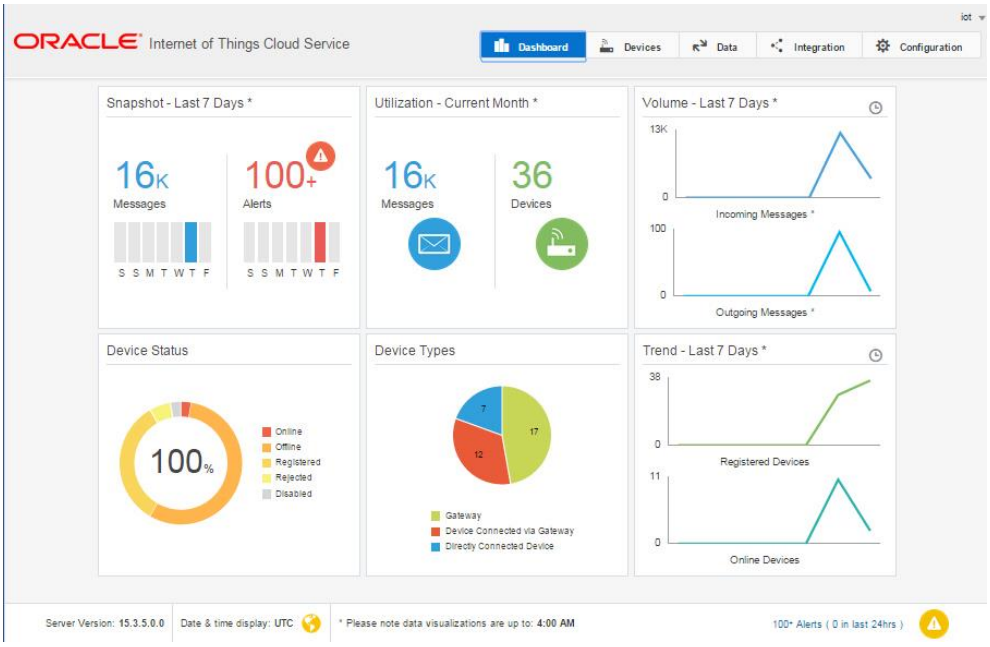

Abbildung 46: Oracle IoT Service User-Interface [150]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind gut gestaltet und bieten gute Beschreibungen.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist über Widgets anpassbar.

3.2.13.11 Sicherheit

Die Plattform verwendet TLS zur Verschlüsselung der Datenübertragung. Zusätzlich müssen Devices und Benutzer per OAuth Token Authentifiziert werden. [151]

3.2.14 IBM Watson Internet of Things

Abbildung 47: IBM Watson Internet of Things Logo [152]

https://www.ibm.com/internet-of-things/

IBM Watson Internet of Things ist ein Cloud Service, der für Gerätemanagement, schnelle und skalierbare Konnektivität, sichere Kommunikation und die Speicherung und den Zugriff auf Daten konzipiert wurde. Die Plattform ist Teil von IBM Bluemix, einer PaaS welche Entwicklern die Anwendungsentwicklung von mobilen Apps und Webanwendugnen erleichtern soll. [153]

3.2.14.1 Verbreitung

IBM Watson Internet of Things verfügt über ein eigenes Forum mit 512 Themen und insgesamt 957 Posts. [154] Die Suche auf Stack Overflow ergab 10 Treffer. Eine Google Suchanfrage lieferte ca. 145.000 Ergebnisse.

3.2.14.2 Preis

IBM Watson Internet of Things ist zu zwei verschiedenen Abonnements verfügbar. [155]

- Kostenlos: kostenlos, bis zu 20 aktive Geräte
- Standard: 0,0075€ pro MB, unlimitiert aktive Geräte

3.2.14.3 Verbindungsrate

Zum Zeitpunkt der Untersuchung gibt es keine Angaben zu Limitierungen der Verbindungsrate.

3.2.14.4 Datenmenge

Für alle Abonnements können pro Monat maximal 100MB an Daten übertragen, und 1GB gespeichert werden. Alle kostenpflichtigen Abonnements erlauben das Zukaufen von zusätzlichen Kapazitäten.

3.2.14.5 Schnittstellen

Die Plattform bietet mehrere Libraries für die Verwendung mit IBM Watson Internet of Things. Verfügbar sind diese für: [156]

- Python

- Embedded C
- Java
- Node.js
- C#

Zusätzlich existieren APIs für MQTT und HTTP-REST.

3.2.14.6 API

Die REST API für IBM Watson Internet of Things erlaubt die Interaktion mit den verbundenen Devices und Anwendungen.
Die API verfügt über folgende Funktionen: [157]

- Datenübertragung
- Datenverwaltung
- Devicemanagement
- Diagnostik

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an https://org_id.internetofthings.ibmcloud.com>/api/v0002/device/types/{DeviceType}/devices/{DeviceID}/events/{eventID} gesendet werden. Dieser kann folgende Parameter enthalten:

- **Event** (json/text/xml/bin) – Zu übertragende Daten

Beispiel POST-Request:

```
POST https://org_id.internetofthings.ibmcloud.com>/api/v0002/device/types/
     {DeviceType}/devices/{DeviceID}/events/{eventID}
     Content-Type:application/json
     Auth-Token:<AuthToken>

     {
       "data":"sample data"
     }
```

3.2.14.7 Visualisierung

Der IBM Watson Internet of Things Service bietet zur Datenvisualisierung Liniendiagramme, dynamische Liniendiagramme, Balkendiagramme, Kreisdiagramme, Messanzeigen und die einfache Anzeige eines Wertes. [155] Für eine erweiterte Visualisierung können allerdings weitere Services der IBM Bluemix Plattform genutzt werden, wie z.B. Monitoring and Analytics, dashDB, Geospatial Analytics, IBM Graph und IoT Real-Time Insights. [158]

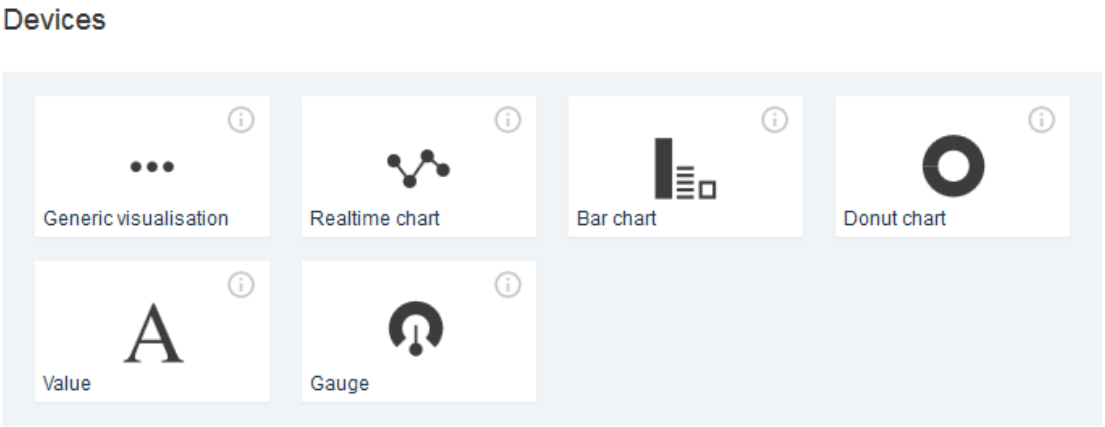

Abbildung 48: IBM Watson IoT Visualisierung [155]

3.2.14.8 Weiterverarbeitung

Der IBM Watson Internet of Things Service bietet selbst keine Möglichkeit zur Weiterverarbeitung der Daten an. Hierfür müssen andere Services der IBM Cloud zur Hilfe genommen werden.

3.2.14.9 Dokumentation

Die Dokumentation der Plattform ist übersichtlich, klar strukturiert, sehr ausführlich und deckt die grundlegende Funktionsweise und Struktur, die API, das Gerätemanagement, so wie externe Libraries ab. Erklärungen sind hierbei nicht bebildert und es existieren nur wenige Codebeispiele. [159]

3.2.14.10 Usability

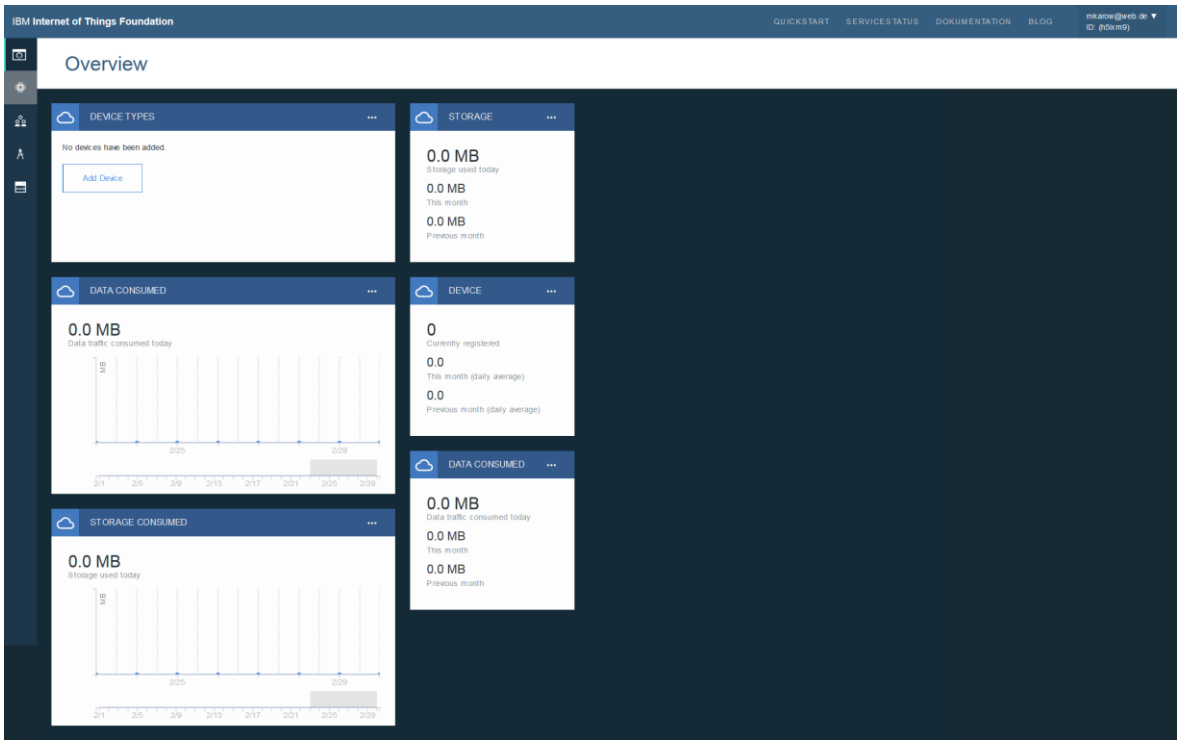

Abbildung 49: IBM Watson Internet of Things User-Interface [155]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sehr gut gestaltet, es existieren zusätzliche Hilfestellungen durch detaillierte Beschreibungen der Dialogschritte und vorkonfigurierte Optionen.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet und geschieht über eine Kombination aus Symbolen und Schrift.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, die Steuerung erfolgt meist schrittweise. Daten werden dabei zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist im Dashboard anpassbar.

3.2.14.11 Sicherheit

IBM Watson Internet of Things verwendet TLS zur Verschlüsselung der Datenübertragung. Die Authentifizierung erfolgt mit Hilfe von Tokens. [160]

3.2.15 Microsoft Azure IoT Hub

Das Azure IoT Hub ist Teil der Microsoft Azure Cloud. Die Plattform soll eine zuverlässige und sichere bidirektionale Kommunikation zwischen IoT-Geräten und die Verarbeitung und Auswertung der erfassten Daten ermöglichen. [162]

Abbildung 50: Microsoft Azure Logo [161]

https://www.microsoft.com/de-de/server-cloud/internet-of-things/

3.2.15.1 Verbreitung

Die Azure Plattform verfügt über ein Forum im Microsoft TechNet. Zum Azure IoT Hub gibt es dort 119 Posts zu 35 Themen. [163] Die Suche bei Stack Overflow ergab 218 Treffer. Eine Suchanfrage bei Google lieferte ca. 96.200 Ergebnisse.

3.2.15.2 Preis

Azure IoT Hub wird in drei Editionen angeboten: [164]

- Free: kostenlos, geeignet zum Testen des Services
- S1: 42,17€ pro Monat, geeignet für Anwendungen mit kleiner Datenmenge
- S2: 421,65€ pro Monat, geeignet für Anwendungen mit großer Datenmenge

3.2.15.3 Verbindungsrate

Die mögliche Verbindungsrate richtet sich nach der gewählten Edition: [165]

- Free: maximal 8.000 Datenstreams pro Tag
- S1: maximal 400.000 Datenstreams pro Tag
- S2: maximal 6.000.000 Datenstreams pro Tag

3.2.15.4 Datenmenge

Die mögliche übertragbare Datenmenge richtet sich nach der gewählten Edition: [164]

- Free: maximal 0,5 KB pro Nachricht
- S1: maximal 4 KB pro Nachricht
- S2: maximal 4 KB pro Nachricht

3.2.15.5 Schnittstellen

Die Plattform bietet mehrere SDKs für folgende Programmiersprachen und Services: [166]

- C SDK
- .NET Device SDK
- .NET Service SDK
- Node.js Device SDK
- Node.js Service SDK
- Java Device SDK
- Java Service SDK

Zusätzlich existiert eine HTTP REST-API.

3.2.15.6 API

Die REST API für das Azure IoT Hub erlaubt die Interaktion mit den Datenquellen und den Daten der Plattform.
Die API zur Datenübertragung verfügt über folgende Funktionen: [167]

- Datenübertragung

- Benutzerverwaltung
- Plattformverwaltung
- Devicemanagement
- Diagnostik

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an https://{IoTHubName}.azure-devices.net/devices/{deviceId}/messages/events gesendet werden. Dieser kann folgende Parameter enthalten:

- **IoTHubName** (string) – Name des IoT-Hubs (required)
- **deviceId** (string) – ID des Devices (required)
- **api-version** (string) – Verwendete Version der API (required)

Beispiel POST-Request:

```
POST https://{IoTHubName}.azure-devices.net/devices/{deviceId}/messages/
     Content-Type:ocet/stream

     {
       "data":"sample Data"
     }
```

3.2.15.7 Visualisierung

Das Azure IoT Hub stellt selbst keine Möglichkeiten zur Datenvisualisierung zur Verfügung. Hierzu soll die externe Microsoft Plattform Power BI (https://powerbi.microsoft.com/de-de/) verwendet werden. [168] Diese bietet zahlreiche Visualisierungstypen, vom einfachen Balken- und Säulendiagramm, über Flächen- und Kreisdiagramme, bis hin zu Treemap- und Wasserfalldiagramme. [169]

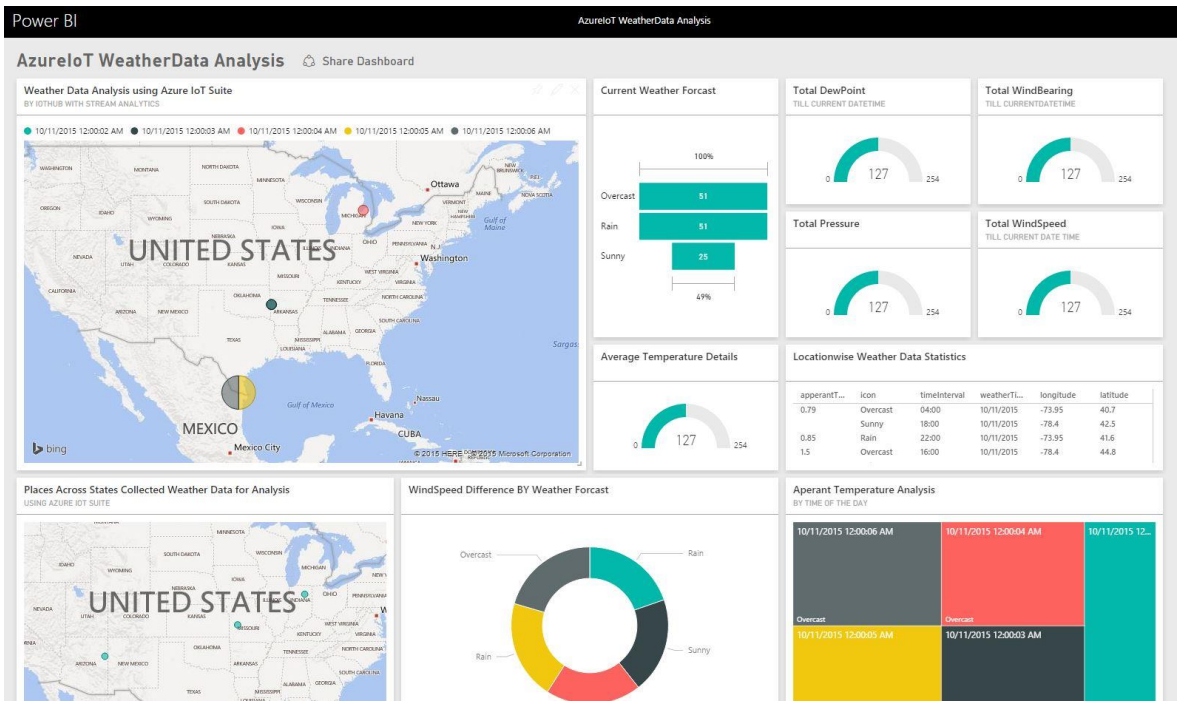

Abbildung 51: Azure IoT Hub Visualisierung über Power BI [170]

3.2.15.8 Weiterverarbeitung

Das Azure IoT Hub bietet selbst keine Möglichkeit zur Weiterverarbeitung der Daten an. Hierfür müssen andere Services von Azure zur Hilfe genommen werden.

3.2.15.9 Dokumentation

Die Dokumentation des Azure IoT Hub ist übersichtlich, klar strukturiert und sehr umfangreich. Sie bietet einen detaillierten Überblick für die Plattform. Sie umfasst die Struktur und Funktionsweise, den gesamten Anwendungs-entwicklungsprozess, sowie Links zu den API-Referenzen. Beispiele, Anleitungen und Tutorials sind teilweise gut bebildert. Es fehlen allerdings gerade bei der API-Referenz sinnvolle Codebeispiele. [171]

3.2.15.10 Usability

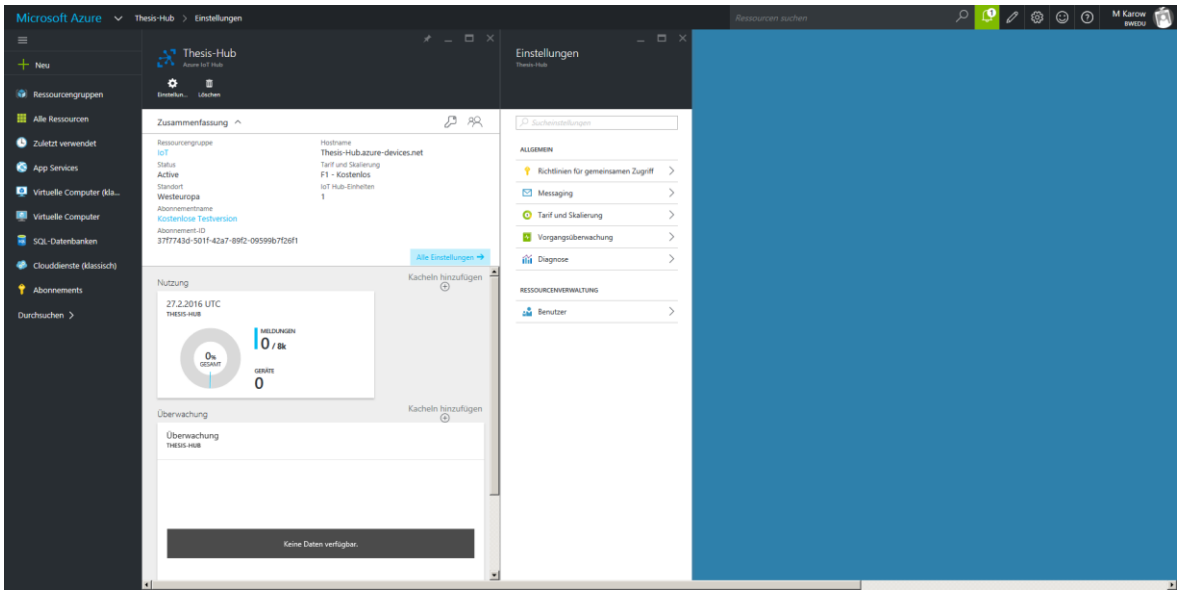

Abbildung 52: Azure IoT Hub User-Interface [172]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind ausführlich und bauen aufeinander auf, es existieren zusätzliche Hilfestellungen durch Beschreibungen der einzelnen Optionen.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent.
Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden farblich markiert und erklärt.
Individualisierbarkeit: Die Benutzeroberfläche ist komplett anpassbar, dies geschieht über ein Kachel-Design.

3.2.15.11 Sicherheit

Alle Datenübertragungen von und zu Azure IoT Hub sind mit TLS verschlüsselt. Die Authentifizierung von Benutzern und Devices geschieht über Tokens. [173]

3.2.16 AWS IoT

Abbildung 53: AWS Logo [174]

https://aws.amazon.com/de/iot/

AWS Iot ist Teil der Amazon Web Services (AWS). Die Plattform soll die Zusammenarbeit zwischen verbundenen Geräten und Cloud-Anwendungen ermöglichen und unterstützt Milliarden von Geräten und Billionen von Nachrichten. [174]

3.2.16.1 Verbreitung

AWS IoT verfügt über ein eigenes Forum mit insgesamt 1.296 Posts zu 325 Themen. [175] Stack Overflow liefert zur Suche 101 Ergebnisse. Eine Google Suchanfrage ergab ca. 194.000 Treffer.

3.2.16.2 Preis

AWS IoT arbeitet mit einem simplen Preismodell, welches sich nach der Anzahl der Übertragenen Nachrichten richtet. Die Kosten belaufen sich auf 5$ pro 1 Million Nachrichten, es wird dabei Nachrichtengenau abgerechnet. Als Nachricht gilt hierbei jeder 512 Byte Datenblock, der vom Service entweder veröffentlicht oder übertragen wurde. Für die Übertragungen zu den eigenen AWS-Services Amazon S3, Amazon DynamoDB, AWS Lambda, Amazon Kinesis, Amazon SNS und Amazon SQS fallen keine Kosten an. AWS IoT verfügt im ersten Jahr über ein kostenloses Kontingent von 250.000 Nachrichten pro Monat. [176]

3.2.16.3 Verbindungsrate

AWS IoT setzt ein Limit von maximal 100 gleichzeitig eingehenden Nachrichten. [177]

3.2.16.4 Datenmenge

Die maximal übertragbare Datenmenge pro Nachricht beträgt 128 KB. Eingehende Daten sind auf 512 KB/s limitiert, [177]

3.2.16.5 Schnittstellen

AWS IoT bietet SDKs für Embedded C, JavaScript (Node.js) und den Arduino Yún. [178] Des weiteren werden MQTT und HTTP-REST Schnittstellen bereitgestellt. [179]

3.2.16.6 API

Die AWS IoT REST-API erlaubt die Interaktion und Verwaltung sämtlicher Plattformressourcen. Dies schließt die Daten- und Zugriffsverwaltung mit ein.
Die REST-API verfügt über folgenden Funktionsumfang: [180]

- Datenübertragung
- Datenverwaltung
- Authentifizierung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an https://<endpoint>.iot.eu-west-1.amazonaws.com/things/<thingName> gesendet werden. Dieser kann folgende Parameter enthalten:

- **attributePayload** (json) – Zu übertragende Daten (required)
- **thingName** (string) – Name des Devices (required)

Beispiel POST-Request:

```
POST https://<endpoint>.iot.eu-west-1.amazonaws.com/things/<thingName>
     Content-Type: application/json

    {
      "attributePayload": {
            "attributes":
            {
               "string":"string"
            }
      },
      "thingName": "string"
    }
```

3.2.16.7 Visualisierung

AWS IoT bietet selbst keine Visualisierungen. Diese sollen über den AWS Dienst Amazon Kinesis (https://aws.amazon.com/de/kinesis/) realisiert werden. [181]

3.2.16.8 Weiterverarbeitung

AWS IoT bietet selbst keine Möglichkeit zur Weiterverarbeitung der Daten an. Hierfür müssen andere Services der Amazon Web Services zur Hilfe genommen werden.

3.2.16.9 Dokumentation

Die Dokumentation der AWS IoT Plattform ist sehr umfangreich und umfasst einen Developer Guide, die API-Referenz und weiterführende Themen zur Integration mit anderen AWS-APIs. Die Unterseiten sind übersichtlich und klar strukturiert. Bei Erklärungen werden viele Codebeispiele benutzt. Da die gesamte AWS-Plattform aus vielen einzelnen Services besteht, sind manche Themen schwer zu finden. [182]

3.2.16.10 Usability

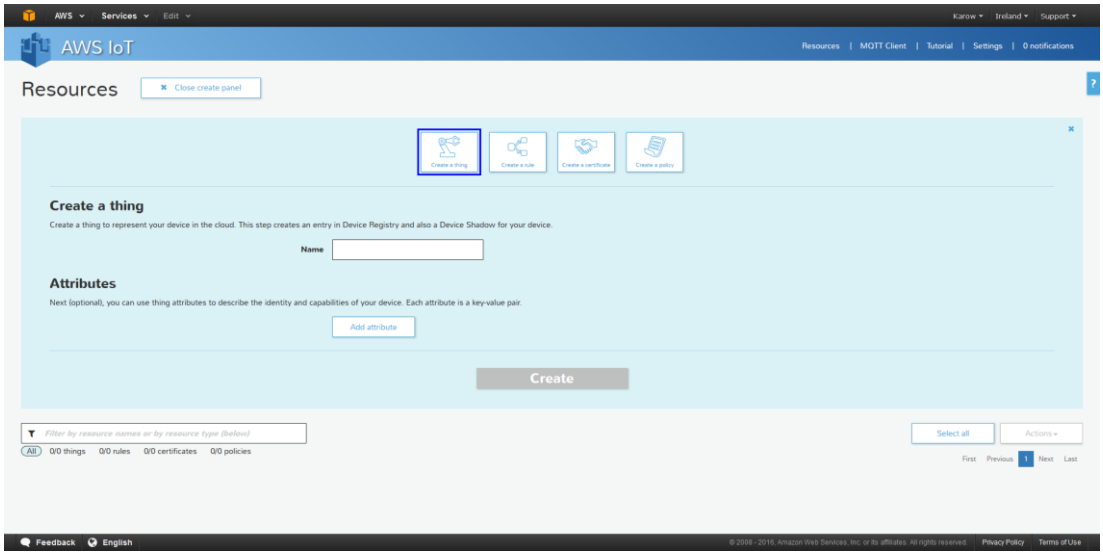

Abbildung 54: AWS IoT User-Interface [183]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind sehr verständlich, es existieren zusätzliche Hilfestellungen durch interaktive Anweisungen und Tutorials sowie gute Beschreibungen.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.

Steuerbarkeit: Dialoge könne jederzeit gestartet und abgebrochen werden, Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, bei fehlenden Angaben kann nicht fortgefahren werden.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.16.11 Sicherheit

Alle Übertragungen von und zur Plattform sind mit TLS verschlüsselt. Die Authentifizierung geschieht mit Hilfe von Zertifikaten, die vom Plattformeigenen Message-Broker verwaltet werden. Genauere Berechtigungen lassen sich per Rule- und Policy-Management vergeben. [184]

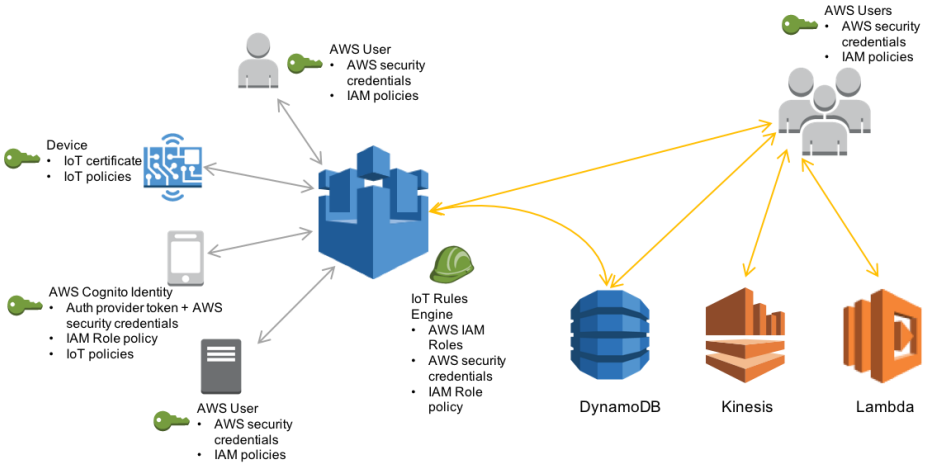

Abbildung 55: Sicherheitsmodell AWS IoT [184]

3.2.17 Google Cloud Platform

Abbildung 56: Google Cloud Platform Logo [185]

https://cloud.google.com/solutions/iot/

Die Google Cloud Platform bietet mehrere Lösungen für das Internet of Things. Die angebotenen Services decken von der Datenübertragung über Speicherung und Analyse, bis zur Anwendungsentwicklung alle wichtigen Bereiche ab. [186]

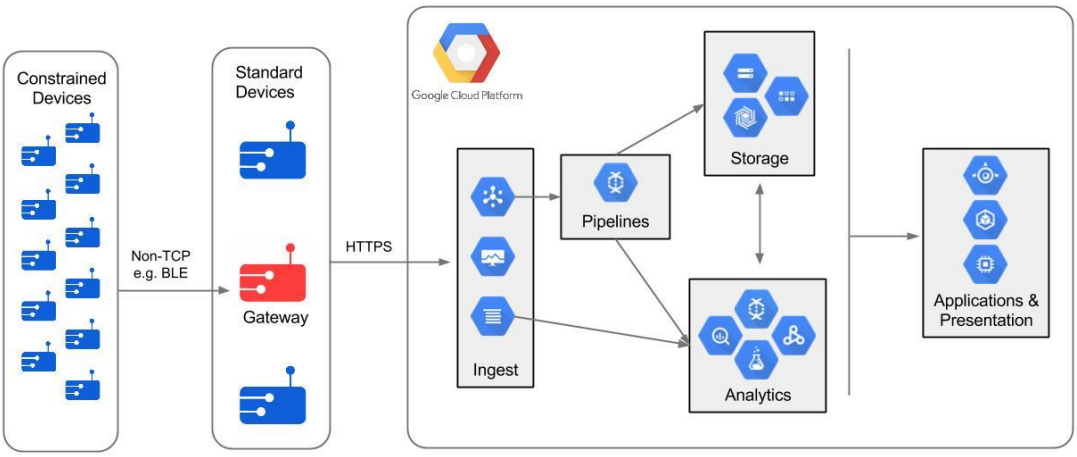

Abbildung 57: Architektur der Google Cloud Platform für das IoT [186]

Der Standarddienst für die Datenübertragung ist Cloud Pub/Sub. Dieser Dienst erlaubt den Austausch von Nachrichten in Echtzeit zwischen eigenständigen Devices, Anwendungen und Services. [187]

3.2.17.1 Verbreitung

Google Cloud Pub/Sub stellt sein Forum über Google Groups zur Verfügung. Es existieren dort 61 Themen mit insgesamt 223 Posts. [188] Bei Stack Overflow ergab eine Suche 200 Treffer. Eine Google Suchanfrage lieferte 7.980 Ergebnisse.

3.2.17.2 Preis

Google Cloud Pub/Sub berechnet seine Preise nach der Anzahl der ausgeführten Operationen pro Monat. Die Preise sind hierbei gestaffelt und sinken, je mehr Operationen ausgeführt werden. Als Operation zählt jeder API-Aufruf. Hierbei zählen alle 64KB einer Nachricht als eine Operation: [189]

- Bis 250 Millionen Operationen: 0,40$ pro Million
- Nächste 500 Millionen Operationen: 0,20$ pro Million
- Nächste 1.000 Millionen Operationen: 0,10$ pro Million
- Über 1.750 Millionen Operationen: 0,05$ pro Million

Die Kosten für zusätzliche Services können einfach über den Google Cloud Platform Preisrechner berechnet werden.[190]

3.2.17.3 Verbindungsrate

Für die Nachrichtenübertragung gilt standardmäßig eine Begrenzung von 10.000KB pro Sekunde. Diese kann auf Anfrage erhöht werden. [191]

3.2.17.4 Datenmenge

Es können pro Request maximal 10MB an Daten übertragen werden.

3.2.17.5 Schnittstellen

Die Google Cloud Platform bietet mehrere Libraries für den Plattformzugriff. Für den Cloud Pub/Sub Dienst stehen speziell angepasste Libraries für Go, Node.js und Python zur Verfügung. [192]
Zusätzlich gibt es automatisch generierte generische Libraries, die mehrere Services abdecken. Diese stehen zur Verfügung für: [192]

- Go
- Java
- JavaScript
- .NET
- Node.js
- Obj-C
- PHP
- Python
- Ruby

Für die allgemeine Verbindung existiert eine HTTP REST-API.

3.2.17.6 API

Die Pub/Sub REST-API erlaubt die Verwaltung der Nachrichtenschnittstelle. Nachrichten werden hierbei in sogenannten Topics gespeichert und organisiert.
Die API verfügt über folgende Funktionen: [193]

- Datenübertragung
- Datenverwaltung
- Authentifizierung

Um Daten von einem Device an die Plattform zu senden, muss ein HTTP POST-Request an https://pubsub.googleapis.com/v1/{topic=projects/*/topics/*}:publish gesendet werden. [194] Dieser kann folgende Parameter enthalten: [195]

- **data** (string) – Zu übertragende Daten (required)
- **attributes** (object) – Zusätzliche Attribute (optional)
- **messageId** (string) – Automatisch zugewiesene ID (optional)
- **publishTime** (string) – Automatischer Zeitstempel (optional)

Beispiel POST-Request: [196]

```
POST https://pubsub.googleapis.com/v1/projects/myproject/topics/
     mytopic:publish
     Content-Type:application/json
     key:<API-Key>

     {
       "messages": [
       {
              "attributes": [
              {
                     "key": "iana.org/language_tag",
                     "value": "en"
              }
              ],
              "data":"SGVsbG8gQ2xvdWQgUHViL1N1YiEgSGVyZSBpcyBteSBtZXNzYWdlIQ=="
       }
       ]
     }
```

3.2.17.7 Visualisierung

Google Cloud Pub/Sub bietet als Nachrichtendienst keine Visualisierungen an. Diese können über den Google Cloud Dienst Stackdriver (https://cloud.google.com/stackdriver/) realisiert werden. [197]

3.2.17.8 Weiterverarbeitung

Google Cloud Pub/Sub bietet selbst keine Möglichkeit zur Weiterverarbeitung der Daten an. Hierfür müssen andere Services der Google Cloud zur Hilfe genommen werden.

3.2.17.9 Dokumentation

Die Dokumentation der Plattform ist dem Umfang entsprechend sehr ausführlich. Sie ist übersichtlich und klar strukturiert und umfasst einen Schnelleinstieg, Anleitungen zur Konfiguration und Einrichtung, API- und Library-Referenzen sowie zusätzliche Informationen zu Preismodellen, Support und Troubleshooting. Anleitungen und Beispiele sind sinnvoll bebildert und mit Codebeispielen versehen, was zur guten Verständlichkeit beiträgt. API-Requests können direkt auf der Seite interaktiv getestet werden. [198]

3.2.17.10 Usability

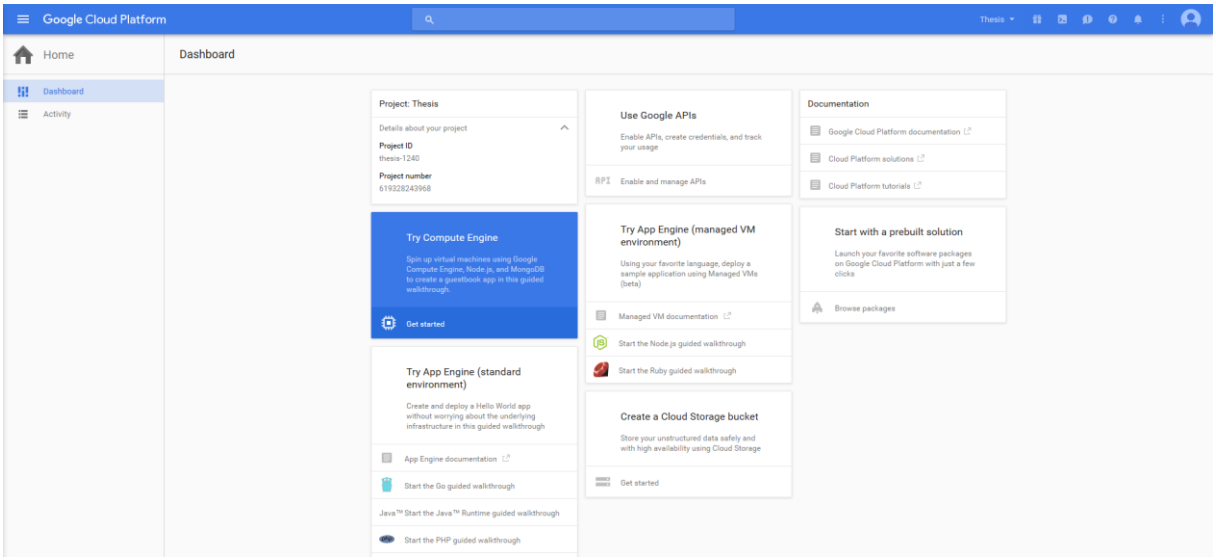

Abbildung 58: Google Cloud Platform User-Interface [199]

Aufgabenangemessenheit: Die Benutzeroberfläche bietet alle Funktionalitäten, die zur Konfiguration der Plattform nötig sind.
Selbstbeschreibungsfähigkeit: Dialoge sind minimalistisch aber verständlich, es existieren zusätzliche Hilfestellungen durch Beschreibungstexte.
Erwartungskonformität: Die Bedienung ist einheitlich und konsistent gestaltet.
Steuerbarkeit: Dialoge könne jederzeit gestartet aber teilweise nicht mehr abgebrochen werden. Sie finden oft schrittweise statt. Daten werden nicht zwischengespeichert.
Fehlertoleranz: Die Plattform ist fehlertolerant, fehlende Angaben werden farblich markiert und angezeigt.
Individualisierbarkeit: Die Benutzeroberfläche ist nicht anpassbar.

3.2.17.11 Sicherheit

Google Cloud Platform verwendet zur Verschlüsslung der Datenübertragung TLS. Die gespeicherten Daten sind per 256Bit mit AES verschlüsselt. Zur Authentifizierung wird OAuth verwendet. [200]

4 Vergleich der Plattformen

Im Folgenden werden die Plattformen anhand der in Kapitel 3.1 festgelegten Untersuchungskriterien verglichen. Zur besseren Übersicht findet sich im Anhang Kapitel 9.1 eine Tabelle, welche alle Plattformen und ihre Kriterien in ihrer Gesamtheit darstellt.

Die untersuchten Plattformen lassen sich anhand ihres Umfangs und ihrer Funktionen in 3 verschiedene Benutzerklassen einteilen. Die kostenlosen Plattformen ThingSpeak, data.sparkfun.com, Runabove IoT Lab und flowthings.io sind besonders für Einsteiger und Hobbyentwickler geeignet. Die Plattformen Carriots, Ubidots, GroveStreams, Exosite, Beebotte, MODE, Initial State, Temboo, Oracle Cloud, IBM Watson IoT und Azure IoT Hub richten sich vor allem an Unternehmen, die eine Benutzerverwaltung und erweiterten Support benötigen. AWS IoT und Google Pub/Sub richten sich an erfahrene Webentwickler, die ihre Anwendungen im Ökosystem der Amazon Web Services und Google Cloud Platform entwickeln.

4.1 Verbreitung

Nach Google Ergebnissen zählen die Plattformen ThingSpeak, Exosite, IBM Watson IoT und AWS IoT zu den populärsten Plattformen. Besonders von Entwicklern benutzt werden laut den Suchergebnissen auf Stack Overflow die Plattformen Temboo, Azure IoT Hub, AWS IoT und Google Pub/Sub. Über eine große Community mit aktiven Foren verfügen die Plattformen ThingSpeak, Carriots, Ubidots, GroveStreams, Exosite, IBM Watson IoT und AWS IoT. Als großer Softwareentwickler fällt Oracle besonders negativ auf, da es hier kein Forum und keine Treffer bei Stack Overflow gibt. Auch die Google Suchanfrage liefert ein sehr niedriges Ergebnis.

4.2 Preis

Bis auf die Oracle Cloud, AWS IoT und die Google Cloud Platform bieten alle Paas-Angebote einen kostenlosen Zugang an. Komplett kostenlos sind die Plattformen ThingSpeak, data.sparkfun.com, Runabove IoT Lab und flowthings.io verfügbar.

Die bezahlten Angebote sind auf größere Projekte und Unternehmen zugeschnitten. Dies stellt sich der Anzahl der unterstützten Devices, des Speicherplatzes und Premium-Features wie eine komplexe Benutzer- und Rechteverwaltung sowie direktem Support dar. Falls Unternehmensspezifische Features nicht benötigt werden, sind die Angebote von AWS IoT und Google Cloud Platform weitaus kostengünstiger als die der restlichen Plattformen, da hier eine Nachrichtengenaue Abrechnung zu niedrigen Preisen erfolgt. Eine Ausnahme bildet hier IBM Watson IoT, welche nach übertragenen Daten abrechnet.

4.3 Verbindungsrate

Die Verbindungsraten der Plattformen Runabove IoT Lab, flowthings.io, MODE und IBM Watson Internet of Things konnten nicht ermittelt werden.

Die Verbindungsrate der kostenlosen Plattformen ist erwartungsgemäß limitiert. Sie liegt bei 1 Stream/15 Sek. für ThingSpeak und bei 100Streams/15 Min. für data.sparkfun.com.

Die kostenlosen Angebote der restlichen Plattformen liegen fast alle im selben Rahmen. Deutlich niedriger ist die Verbindungsrate nur bei den Plattformen Carriots mit 500Streams/Tag und Temboo mit lediglich 250 Streams/Monat.

Negativ auffällig ist die Plattform GroveStreams, die selbst beim teuersten Angebot insgesamt maximal 500 Streams pro Stunde erlaubt.

Besonders großzügig ist die Verbindungsrate von Exosite, die bei allen Angeboten bei 1 Stream/Sek. liegt und keine Volumenbegrenzung hat. Sehr großzügige Limitierungen haben außerdem die großen Cloudplattformen AWS IoT und Google Cloud Platform, die Begrenzungen von maximal 100 simultanen Streams bzw. 160 Operationen/Sek. setzen.

4.4 Datenmenge

Limitierungen der Datenmenge konnten für die Plattformen ThingSpeak, Runabove IoT Lab, Ubidots, MODE und Oracle Cloud nicht ermittelt werden. Für data.sparkfun.com gibt es lediglich Angaben zum Speicherplatz, der bei 50MB liegt. Flowthings.io gibt lediglich die Anzahl der Speicherbaren Datenpunkte von maximal 100.000 an.

Die übertragbare Datenmenge unterscheidet sich zwischen den einzelnen Plattformen sehr stark. Sie reicht von wenigen Bytes bis mehreren MegaBytes pro Stream.
Am unteren Ende sind hier das Azure IoT Hub (max. 4KB/Stream), Carriots (max. 10KB/Stream), Beebotte (max. 16KB/Stream), Exosite (max. 64KB/Stream) und AWS IoT (max. 128KB/Stream) angesiedelt.

Besonders großzügig sind die Google Cloud Platform mit max. 10MB/Stream und GroveStreams mit max. 200MB/Stream. Für GroveStreams existiert allerdings zusätzlich eine maximal übertragbare Datenmenge von 1GB/Monat.

Die Plattformen IBM Watson Internet of Things (100MB/Monat) und Temboo (max. 16GB/Monat) begrenzen das Datenvolumen lediglich auf den Zeitraum eines Monats gesehen. Welche Plattform hier einen Vorteil hat ist Anwendungsabhängig.

4.5 Schnittstellen

Alle Plattformen bieten mindestens eine HTTP-Rest API. Bei den Plattformen data.sparkfun.com, Runabove IoT Lab, Ubidots, GroveStreams, Initial State, Temboo, Azure IoT Hub und Google Cloud Platform ist dies die einzige Kommunikationsschnittstelle.

Mehrere Plattformen bieten zusätzlich Libraries für gängige Programmiersprachen an, um die Implementierung zu erleichtern. Bis auf flowthings.io sind dies alle bezahlte Plattformen. Am oberen Ende finden sich hier Exosite und die Google Cloud Platform mit jeweils 9 Libraries.

Die Plattformen Carriots, MODE, Initial State, Temboo und AWS IoT bieten komplette SDKs für die Schnittstellenentwicklung zu Endgeräten an.

Besonders hervorzuheben ist die Plattform Temboo, welche mit über 100 vorkonfigurierten Schnittstellen zu externen Services und Plattformen aufwartet.

4.6 API

Der Umfang der API für die unterschiedlichen Plattformen hängt stark von der Zielgruppe des jeweiligen Angebots ab. Während die kostenlosen Plattformen meist nur die grundlegenden Funktionen zur Datenübertragung und Datenverwaltung enthalten, bieten die unternehmensorientierten Plattformen meist zusätzliche Funktionalitäten zur Plattformverwaltung und Benutzerverwaltung. Plattformen wie AWS IoT und Google Cloud Platform haben ebenfalls nur sehr grundlegende API Funktionalitäten, dies ist allerdings der Modularisierung des gesamten Cloud-Angebotes geschuldet.

Positiv fällt die Plattform flowthings.io auf, die als kostenlose Plattform eine sehr umfangreiche API bietet, welche die von bezahlten Plattformangeboten teilweise übertrifft. Ein weiterer Ausreißer ist Initial State, welche lediglich die Funktionalität zur Datenübertragung und –Verwaltung bietet. Dies ist hier dem Fokus der Plattform auf die Datenvisualisierung geschuldet.

4.7 Visualisierung

Bei den Visualisierungsmöglichkeiten unterscheiden sich die Plattformen stark. Bei vielen fehlt hier selbst die Möglichkeit zur grundlegenden Darstellung von Daten. Hierbei ist nicht ausschlaggebend, ob es sich um eine kostenlose oder bezahlte Plattform handelt.

Überhaupt keine Visualisierungsmöglichkeiten bieten die Plattformen Data.sparkfun.com, Runabove IoT Lab, MODE und Temboo. Die großen Cloud Anbieter Google Pub/Sub, AWS IoT, Azure IoT Hub und Oracle Internet of Things Cloud bieten eine ausführliche Visualisierung nur über die Integration anderer Services der jeweiligen Plattform. Besonders positiv aufgefallen sind die Plattformen ThingSpeak und Initial State. Diese bieten bei weitem die umfangreichsten Visualisierungsmöglichkeiten.

4.8 Weiterverarbeitung

Bei der Weiterverarbeitung der Daten zeichnet sich ein ähnliches Bild ab wie bei der Visualisierung. Keine Möglichkeiten zur Datenverarbeitung bieten die Plattformen Data.sparkfun.com, Runabove IoT Lab, Beebotte, MODE, Initial State und Temboo. Google Pub/Sub, AWS IoT, Azure IoT Hub und Oracle Internet of Things Cloud bieten wie bei der Visualisierung eine Weiterverarbeitung nur über zusätzliche Cloud Services an. Bei den restlichen Plattformen ist der Grad der Möglichkeiten zur Weiterverarbeitung sehr unterschiedlich. Während Plattformen wie ThingSpeak, Ubidots und GroveStreams nur sehr rudimentäre Bedingungen und vorgefertigte Aktionen zulassen, stellen Flowthings.io, Carriots und Exosite Schnittstellen zu serverseitigen Scriptsprachen zur Verfügung über die sich auch komplexe Operationen durchführen lassen.

4.9 Dokumentation

Die Qualität und der Umfang der Dokumentationen schwanken stark.
Positiv anzumerken ist, dass fast alle Plattformen in ihren Dokumentationen sinnvolle Illustrationen und Codebeispiele verwenden.

Aufgrund des geringen Funktionsumfangs sind die Dokumentationen von ThingSpeak und data.sparkfun.com wenig umfangreich, dafür aber sehr übersichtlich und gut strukturiert. Besonders hervorzuheben sind Carriots, Beebotte und die Google Cloud Platform, die eine API-Konsole zum Testen von API-Requests anbieten.

Negativ aufgefallen sind die Dokumentationen von Runabove IoT Lab, GroveStreams, Initial State und Temboo, die alle sehr unübersichtlich und wenig strukturiert sind. Besonders auffällig ist dies bei den Plattformen Runabove IoT Lab und GroveStreams, die zusätzlich nur wenig umfangreich sind.

4.10 Usability

Bis auf die Plattform GroveStreams sind die Benutzeroberflächen zeitgemäß und einfach zu bedienen. GroveStreams verfügt über ein veraltetes und inkonsistentes Bedienkonzept, welches die Nutzung der Plattform erschwert. Bei flowthings.io verstecken sich einige Funktionalitäten in schwer erreichbaren Untermenüs.
Die Dialogführung ist allgemein sehr gut und mit zeitgemäßen Hilfen wie Tooltips und Beschreibungen versehen.

Die Plattformen flowthings.io, Ubidots, GroveStreams, Exosite, Beebotte, Initial State, Oracle Cloud, IBM Watson Internet of Things und Azure IoT Hub lassen sich per Dashboard/Widgets an den Benutzer anpassen. Hier sticht besonders das Azure IoT Hub positiv hervor, welches sich per Kacheln individuell gestalten lässt.

Negativ fiel die Google Cloud Platform auf, da hier manche Dialoge nicht abgebrochen werden können und die Benutzeroberfläche blockieren.

Positiv aufgefallen ist das AWS IoT Hub, das zu zahlreichen Konfigurationen direkt in der Oberfläche interaktive Tutorials bietet.

4.11 Sicherheit

Alle Plattformen bieten Sicherheitsmechanismen, die dem Stand der aktuellen Technik entsprechen.

Die Übertragungssicherheit wird bei allen Plattformen per TLS gewährleistet.
Zur Authentifizierung kommen bis auf AWS IoT, Oracle Cloud und die Google Cloud Platform Tokens zum Einsatz. AWS IoT verwendet Zertifikate, welche von einem eigenen Message-Broker ausgegeben werden. Oracle Cloud und Google Cloud Platform verwenden ein OAuth-Verfahren.

Erwähnenswert sind hier die Plattformen von Carriots, welche eine Hash-Funktion zur Sicherstellung der Datenintegrität bietet und Beebotte, Temboo und Google Cloud Platform, welche AES-Verschlüsselung anbieten.

4.12 Auswertung des Plattformvergleichs

Der derzeitige Stand der PaaS- Angebote für das Internet of Things bietet insgesamt eine große Vielfalt. Die Unterschiede zwischen den einzelnen Plattformen sind vor allem durch ihre jeweilige Zielgruppe bestimmt.

Die Verbreitung von PaaS-Angeboten ist zum Stand der Untersuchung in großen Teilen noch sehr niedrig. Lediglich ThingSpeak, Exosite und die Angebote der großen Cloud Anbieter IBM und Amazon scheinen sich hier bis jetzt durchgesetzt zu haben.

Das durchgängige Angebot zur kostenlosen Nutzung ist vor allem für Einsteiger in das IoT sehr positiv, da hierdurch ohne finanziellen Aufwand Anwendungen erstellt und getestet werden können. Hierbei verfügen alle Plattformen über die nötigen Grundfunktionalitäten, einige kostenlose Plattformen können sogar mit bezahlten Angeboten konkurrieren.

Bezüglich der gebotenen Leistungen eignen sich die kostenlosen Plattformen vor allem für Einsteiger, welche mit wenigen Devices arbeiten. Die bezahlten Plattformen richten sich mit ihren Funktionalitäten eher an Unternehmen. Das schlägt sich darin nieder, dass die bezahlten Features meist nur eine erweiterte Benutzerverwaltung und professionellen Support beinhalten. Bezüglich der Grundfunktionalität der Datenübertragung zwischen Endgeräten, Anwendungen und dem Internet existieren oft nur wenige Vorteile. Für Entwickler sind vor allem AWS IoT und die Google Cloud Platform interessant, da diese sehr gut skalieren und mit den inzwischen weitläufig genutzten anderen Services der Plattformen integriert sind.

Als Standardschnittstelle für die Datenübertragung ist HTTP-REST der Stand der Dinge. Lediglich 5 der 17 Plattformen bieten zusätzlich das eigens für Machine-to-Machine (M2M) Kommunikation entwickelte MQTT Protokoll. Positiv ist, dass viele Plattformen die Integration und den Einstieg zusätzlich durch die Bereitstellung von Libraries und SDKs erleichtern.

Der Umfang der APIs ist stark von der Funktionalität der zugehörigen Plattform abhängig. Während Datenübertragung und –Verwaltung standardmäßig überall integriert sind, sind zusätzliche Funktionen auch hier wieder von der Zielgruppe der Plattform abhängig.
Der Stand der Dokumentationen ist bis auf wenige Ausnahmen zeitgemäß und der Plattformfunktionalität angepasst. Für einen schnellen Einstieg bieten die meisten Plattformen gute Tutorials und Erklärungen zum Konzept der Plattform. Auffällig ist, dass die Dokumentationen von bezahlten Angeboten überproportional schlechter sind, als die der kostenlosen Plattformen.

Bei Visualisierung und Weiterverarbeitung der Daten gibt es große Unterschiede zwischen den einzelnen Plattformen. Diese unterscheiden sich nicht nach der Benutzerklasse der Plattform. Die großen Cloud-Plattformen setzen hierbei auf die Nutzung Cloud eigener

Services. Bei den anderen Plattformen gibt es erstaunlich oft keine Möglichkeiten Daten überhaupt zu Visualisieren oder zu verarbeiten.

Die Usability der Plattformen entspricht bis auf einen Ausreißer dem Stand moderner Webtechnologien und stellt für den normalen Benutzer kein Hindernis dar. Dialoge sind fast durchgängig gut beschriftet und mit Hilfestellungen versehen, der Ablauf ist überall fehlertolerant. Die Individualisierung einiger Plattformen ist möglich und geschieht meist webüblich über Dashboards und Widgets.

Die Sicherheitsfunktionen der Plattformen entsprechen positiverweise allesamt dem Stand der Technik und bieten sowohl Übertragungssicherheit, als auch Zugriffssteuerung. Dies spiegelt das erstarkte Sicherheitsbewusstsein wieder, welches sich in den letzten Jahren im Bereich des Internets manifestiert hat.

Durch die Spezialisierung, die leichte Nutzbarkeit und flexible Preismodelle zeigt sich ein breitgefächertes Bild an PaaS-Angeboten, welches ein breites Spektrum an Benutzern abdeckt. Vom Kleinstprojekt bis zum Großprojekt mit mehreren Benutzern und Millionen von Devices sind durch Cloud-Technologie fast alle momentanen IoT-Anwendungen schnell, skalierbar und sicher umsetzbar.

5 Vergleichende Implementierung einer IoT-Anwendung

Zum Vergleich der Implementierung wird im Folgenden eine IoT-Anwendung für drei ausgewählte Plattformen (Kapitel 5.4) beschrieben und realisiert. Die Implementierung wird am Ende ausgewertet und zwischen den Plattformen verglichen.

5.1 Anwendungsbeschreibung

5.1.1 Anforderungen

Für die Anwendung wurden folgende Anforderungen gewählt:

- Geringe Latenz
- Einfache Bedienbarkeit
- Aktualisierbarkeit der Informationsdaten des Devices über die Plattform
- Darstellung der bidirektionalen Kommunikation zwischen Device und Plattform

Diese sollen dazu dienen, einen realen Anwendungsfall nachzustellen. Eine geringe Latenz ist für die Datenübertragung in vielen Industrien von großer Wichtigkeit, da es sich oft um zeitkritische Anwendungen handelt und zeitnahe entsprechende Aktionen ausgelöst werden sollen. Die einfache Bedienbarkeit spiegelt den aktuellen Stand von Benutzerführung und User-Experience wieder. Die Aktualisierbarkeit der Daten auf dem Device ist ein wichtiges Kriterium, da viele Devices oft räumlich sehr weit auseinander liegen oder in ihrer Menge nicht per direkten Zugriff aktualisierbar sind. Die Darstellung der bidirektionalen Kommunikation dient der Veranschaulichung der unterschiedlichen Kommunikationsmöglichkeiten zwischen Device und Plattform in beide Richtungen.

5.1.2 Konzept

Die Anwendung soll die Kommunikation zwischen einem Device, dem PaaS-Angebot, einer Webanwendung und einem Webservice realisieren. Hierfür wird die Funktionalität eines Infoterminals nachgestellt. Das Device stellt hierbei das Terminal dar und stellt die Informationen per NFC zur Verfügung. Bei jeder Verwendung meldet das Device dieses an die PaaS-Plattform. Diese sendet jeweils alle 10 Zugriffe einen Tweet über den verknüpften Twitter-Kanal. Die Webanwendung wiederum dient zur Administration der Inhalte, welche das Device anzeigt. Diese werden über ein Webinterface an die PaaS-Plattform gesendet. Das Device gleicht seinen Inhalt periodisch mit der PaaS-Plattform ab.

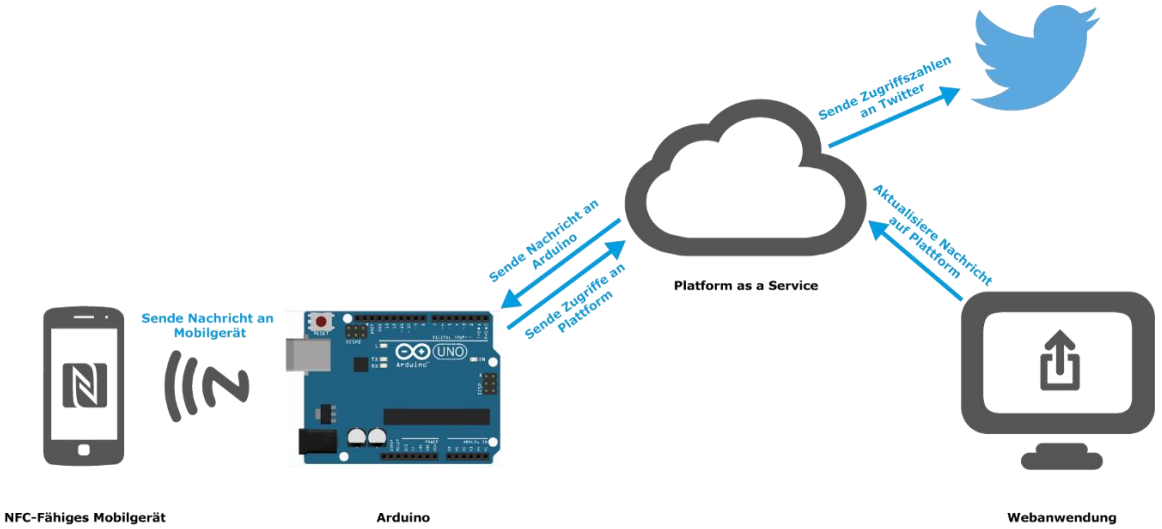

Abbildung 59: Schematische Darstellung der Anwendung

5.1.3 Funktionsweise

Device:
Das Device soll Informationen per NFC-Schnittstelle für Mobilgeräte bereitstellen. Zugriffe auf die Schnittstelle werden an die Plattform gesendet. Periodisch gleicht das Device seine Informationsdaten mit der Plattform ab. Sind aktuellere Daten vorhanden, ersetzen diese die auf dem Device gespeicherten Daten.

Plattform:
Die Plattform speichert die vom Device übertragenen Nutzungsdaten und stellt die Informationsdaten zur Verfügung. Sie dient als Schnittstelle zwischen Device und Webanwendung. Außerdem sendet sie Nutzungsdaten an Twitter, sobald eine Bedingung erreicht wird.

Webanwendung:
Die Webanwendung bietet ein Administrationsinterface über das die Informationsdaten auf der Plattform aktualisiert werden können.

5.2 Hardware

Im Folgenden wird die verwendete Hardware beschrieben.

5.2.1 Arduino

Abbildung 60: Arduino Uno Rev3

Zur Verwendung kommt ein Arduino Uno Rev3. Dieser ist ein Mikrokontroller, welcher auf dem ATmega328P basiert. Diese Hardwareplattform ist leicht über Module (Shields) erweiterbar und bietet eine Softwareumgebung für die einfache Programmierung.

5.2.2 Shields

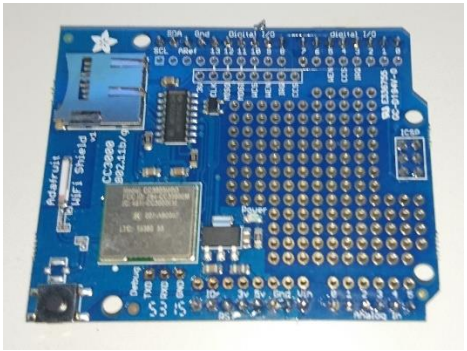

Abbildung 61: Adafruit CC3000 WiFi Shield

Wifi Shield:
Zur Verbindung mit dem Internet wird das Adafruit CC3000 WiFi Shield verwendet. Dieses verfügt über eine interne Antenne und unterstützt 802.11b/g, open/WEP/WPA/WPA2 security sowie TKIP & AES.

NFC Shield:
Als Schnittstelle für die Datenübertragung per NFC kommt das Seeed Studio NFC Shield V2.0 zum Einsatz. Dieses hat den Vorteil, dass es ein NFC-Tag emulieren kann. Somit können die zu übertragenden Daten geändert werden. Das Shield basiert auf dem PN532 Chipset und unterstützt die kabellose Datenübertragung bei einer Frequenz von 13,56MHz.

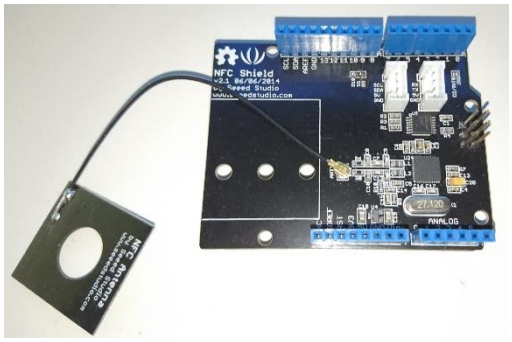

Abbildung 62: Seeed Studio NFC Shield V2.0

5.3 Software

Zur Programmierung des Arduinos kommt die Ardunio IDE zum Einsatz. Diese bietet mehrere SDKs zur schnellen und einfachen Entwicklung. Benötigte Funktionen können einfach über externe Libraries integriert werden.

5.4 Auswahl der Plattformen

Zur vergleichenden Implementierung wurden die Plattformen ThingSpeak, Flowthings.io und IBM Watson Internet of Things ausgewählt. ThingSpeak ist als bekannteste kostenlose Plattform für Hobbyentwickler sehr relevant und bietet einen einfachen Workflow. Flowthings.io wurde als Vertreter der kostenlosen Plattformen aufgrund des größten Funktionsumfangs unter diesen gewählt. IBM Watson IoT steht als Vertreter für die großen Cloud-Plattformen, die sich an Unternehmen richten und ihren Funktionsumfang durch die Modularisierung in einzelne Services erhalten.

Ursprünglich war anstatt für Flowthings.io eine Implementierung für AWS IoT geplant. Dies ließ sich allerdings nicht umsetzen, da diese Plattform genau wie sein Pendant die Google Cloud Platform für die Authentifizierung OAuth verwenden. OAuth verwendet SSL-Zertifikate, welche mit einer eigenen Crypto-Library erstellt werden müssen. Aufgrund der Speicherplatzbeschränkungen von 8-Bit Microcontrollern, zu denen auch der Arduino Uno gehört, lassen sich diese jedoch nicht implementieren. Allerdings arbeitet Amazon für seinen AWS IoT Service zurzeit an alternativen Methoden um eine sichere Datenübertragung und Authentifizierung auch für diese Devices bereit zu stellen. [201]

5.5 Umsetzung

Die Umsetzung gliedert sich in die Implementierung der allgemeinen NFC- und WIFI-Funktionalität für den Arduino und die plattformspezifische Umsetzung der Datenübertragung zwischen Device, Webanwendung und Plattform, sowie die Interaktion mit Twitter. Der komplette Quellcode der einzelnen Sketches und Webanwendungen ist im Anhang (Kapitel 9.2) verfügbar.

5.5.1 Arduino

Im Folgenden werden der Aufbau der Hardware sowie die Programmierung der Grundfunktionalität des Wifi Shields und des NFC Shields beschrieben.

5.5.1.1 Aufbau der Hardware

Auf der untersten Schicht ist der Arduino Uno selbst. Auf diesen wird das NFC Shield gesteckt, auf welches wiederum das Wifi Shield gesteckt wird.

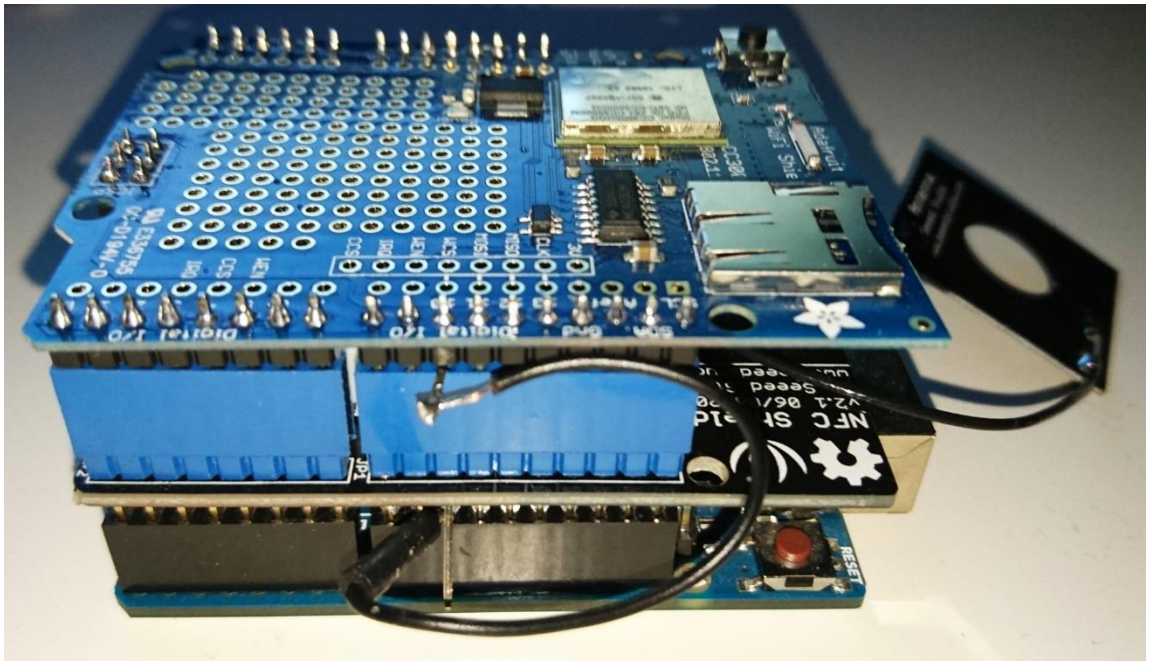

Abbildung 63: Arduino Shield Stack

Das NFC und das Wifi Shield verwenden in ihrer Standardkonfiguration den gleichen Slave- oder Chip-Select Pin, dieser ist standardmäßig Pin 10 auf dem Arduino Uno. Dies sorgt für einen Hardwarekonflikt. Glücklicherweise lässt sich dieser Pin für das NFC Shield softwareseitig auf Pin 9 umlegen. Das Shield verhindert jedoch dennoch die Verwendung des Pins durch das Wifi Shield. Dieses Problem lässt sich lösen, indem Pin 10 des NFC Shields vorsichtig nach außen gebogen wird und die Verbindung an Pin 10 zwischen Arduino und Wifi Shield mit einem Kabel überbrückt wird.

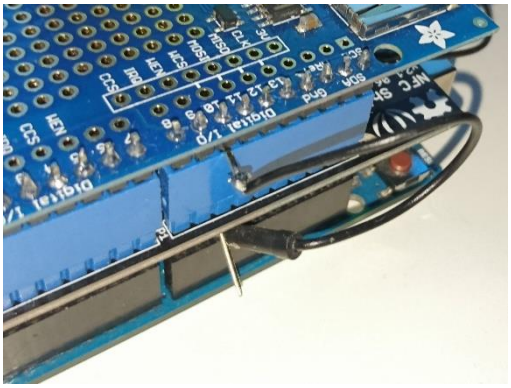

Abbildung 64: Überbrückung an Pin 10

5.5.1.2 Programmieren von WLAN und NFC Shield

Für die Programmierung müssen zuerst folgende Libraries heruntergeladen werden:

- Wifi Shield: https://github.com/adafruit/Adafruit_CC3000_Library
- NFC Shield: https://github.com/Seeed-Studio/PN532
- NDEF: https://github.com/don/NDEF
- MQTT: https://github.com/knolleary/pubsubclient

Die Libraries können entweder in der Arduino IDE im Menü über „Sketch“ -> „Bibliothek einbinden“ -> „.ZIP Bibliothek hinzufügen…“ oder durch das Entpacken der heruntergeladenen Archive in den Libraries Ordner hinzugefügt werden. Dieser befindet sich sich unter Windows standardmäßig unter „Documents\Arduino\libraries“ und unter

Mac und Linux unter „Documents/Arduino/libraries/“. Nach dem Hinzufügen sollte die IDE neu gestartet werden.

Das Einbinden von Libraries in ein Arduino Sketch erfolgt über den Befehl #include <Dateiname>. Für die Grundfunktionalität des Wifi und NFC Shields werden folgende Dateien eingebunden:

```
// Libraries für das Wifi Shield
#include <Adafruit_CC3000.h>
#include <SPI.h>
// Libraries für das NFC Shield
#include <PN532_SPI.h>
#include <snep.h>
#include <NdefMessage.h>
```

Danach erfolgt die Zuweisung der Pins der beiden Shields:

```
// Setze CS Pin für das NFC Shield
PN532_SPI pn532spi(SPI, 9);
// Erstelle ein NFC Objekt
SNEP nfc(pn532spi);

// Interrupt und Control Pins für das Wifi Shield
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
// Benutze Hardware SPI für die restlichen Pins
// Arduino UNO: SCK = 13, MISO = 12, and MOSI = 11
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIVIDER);
```

Für die spätere Verwendung werden SSID, Passwort und Verschlüsselung des WLANs gespeichert. Über den Befehl #define werden diese als statische Variablen im Programmspeicher gespeichert und belegen so keinen dynamischen Speicher:

```
// Setze SSID und WLAN-Passwort
#define WLAN_SSID  "<SSID>"  // darf nicht länger als 32 Zeichen sein!
#define WLAN_PASS  "<Passwort>"
// Setze WLAN-Verschlüsselung
#define WLAN_SECURITY   WLAN_SEC_WPA2
```

Danach erfolgt die Initialisierung der globalen Variablen:

```
// Initialisierung der Variablen
int messageSize;
uint8_t ndefBuf[128];
String msg;
int accessed;
```

In der Setup-Funktion erfolgt daraufhin die initiale Wertezuweisung an einzelne Variablen, das Erstellen der initialen NFC-Nachricht und der Start der WLAN-Verbindung.

Es ist dabei wichtig, dass vor dem Start des WLANs mindestens eine NFC-Nachricht gesendet wird, da ansonsten das NFC Shield im späteren Programmverlauf nicht mehr richtig erkannt wird:

```
// Setup
void setup(void)
{
  Serial.begin(115200); //Beginne serielle Übertragung
  Serial.println(F("Start"));
  accessed = 0;
  msg = "NFC initialisiert!";
  createMessage();
  nfc.write(ndefBuf, messageSize, 0); // Sende die NDEF Nachricht
  startWLAN();
}
```

Die erste aufgerufene Funktion im Setup ist createMessage(). Diese erstellt aus dem in der Variablen msg gespeicherten String einen gültigen NDEF-Record. Dieser besteht aus dem Type Name Format (TNF), dem Record Type und der Payload an zu übertragenden Daten. Der TNF gibt in diesem Anwendungsfall an, dass es sich beim Record Type um einen bekannten Typ handelt, der in der Record Type Definition des NDEF Formats festgelegt ist. Hier ist der Record Type ein Text-Record (T). Die Payload besteht aus einem Bytearray. Die ersten 3 Byte geben hierbei die Sprachkodierung an:

```
// Erstelle NDEF Nachricht
void createMessage(void)
{
  memset(ndefBuf, 0, sizeof(ndefBuf));
  NdefMessage message = NdefMessage();

  // Erstelle den NDEF Record manuell, damit der Speicher des Arduino
Uno nicht überläuft
  NdefRecord record = NdefRecord();
  record.setTnf(TNF_WELL_KNOWN);  // Setze Typ des NDEF Records

  uint8_t recordType[] = { 0x54 }; // "T" Text Record
  record.setType(recordType, sizeof(recordType));

  uint8_t payload[msg.length()+3];  // Berechne Länge der Payload

  for(int i = 0; i < msg.length(); i++)
  {
    payload[i+3] = msg.charAt(i); // Schreibe Nachricht byteweise in die
Payload
  }
  // Language encoding für TNF_WELL_KNOWN RTD_TEXT
  payload[0] = 0x02;
  payload[1] = 0x65; // e
  payload[2] = 0x6e; // n

  record.setPayload(payload, sizeof(payload));  // Setze die Payload
  message.addRecord(record);
```

```
  messageSize = message.getEncodedSize();
  if (messageSize > sizeof(ndefBuf)) {
    Serial.println(F("ndefBuf ist zu klein"));
    while (1) {
    }
  }
  message.encode(ndefBuf);  // kodiere die Nachricht
  message.print();
}
```

Der nächste Funktionsaufruf im Setup ist startWLAN(). Diese Funktion initialisiert zunächst das WLAN-Shield, löscht alle gespeicherten Verbindungsprofile und verbindet sich dann mit dem Netzwerk. Bei erfolgreicher Verbindung wird das Shield per DHCP automatisch für die Nutzung des Netwerks konfiguriert:

```
// Starte WLAN-Verbindung
void startWLAN(void)
{
  // Starte WLAN Shield
  Serial.println(F("Initialisiere CC3000"));
  if (!cc3000.begin())
  {
    Serial.println(F("\nKonnte CC3000 nicht initialisieren!"));
    while (1);
  }

  // Lösche alte Verbindungsprofile
  Serial.println(F("\nLösche alte Verbindungsprofile"));
  if (!cc3000.deleteProfiles()) {
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  // Verbinde mit WLAN
  Serial.print(F("\nVersuche mit Netzwerk zu verbinden"));
  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  Serial.println(F("\nVerbunden!"));

  // Warte auf DHCP
  Serial.println(F("\nFrage DHCP an"));
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }
}
```

Nach der Setup-Funktion folgt die Loop-Schleife. Diese ruft kontinuierlich die Funktion sendMessage() auf:

```
// Schleife
void loop(void) {
  Serial.println(F("\nSende NFC-Nachricht"));
  sendMessage();
}
```

Die sendMessage() Funktion dient dem versenden der NDEF-Nachricht über das NFC Shield an das Endgerät. Die Nachricht wird dabei genau 3 Sekunden bereitgestellt. Wird sie in diesem Zeitraum abgerufen, wird die Funktion sendRequest() aufgerufen. Diese unterscheidet sich von Plattform zu Plattform und wird in den nächsten Kapiteln behandelt:

```
// Sende NFC Nachricht
void sendMessage(void)
{
  if (nfc.write(ndefBuf, messageSize, 3000) >= 0) // Sende die NDEF
Nachricht für 3 Sekunden
  {
    sendRequest(); // Falls Übertragung erfolgreich rufe sendRequest auf
  }
}
```

5.5.2 Plattformübergreifender Teil der Webanwendung

Abbildung 65: Webinterface

Die Webanwendung wird mit Hilfe von HTML, CSS und JavaScript bzw. jQuery realisiert. Für die Gestaltung des Userinterfaces wird das CSS-Framework Pure (http://purecss.io/) verwendet. Die Anwendung besteht aus einem Formular mit einem Textfeld, in welches der Nachrichtentext eingegeben wird.

Das Submit-Event des Formulars wird in einer jQuery Funktion abgefangen, die daraufhin das Textfeld ausliest und die Nachricht an die jeweilige Plattform sendet.

```
<!DOCTYPE html>
<html>
<head>
      <meta charset="utf-8">
      <meta http-equiv="X-UA-Compatible" content="IE=edge">
      <title>Webinterface <Plattform></title>
      <link rel="stylesheet"
href="http://yui.yahooapis.com/pure/0.6.0/pure-min.css">
      <style>
      body {
```

```
                margin: auto;
                width: 380px;
                text-align: center;
            }
            textarea {
                width: 100%;
            }
    </style>
</head>
<body>
      <h2>Webinterface - <Plattform></h2>
      <form id="msg_form" class="pure-form pure-form-stacked">
          <fieldset>
              <textarea name="message" placeholder="Nachricht">
              </textarea>
              <button type="submit" class="pure-button pure-button-
primary">
                  Nachricht aktualisieren
               </button>
          </fieldset>
      </form>
      <div id="status"></div>

      <script src="https://code.jquery.com/jquery-2.2.3.min.js">
      </script>
      <script>
            $( document ).ready(function() {

                <Plattformspezifischer Teil>

            });
      </script>

</body>
</html>
```

5.5.3 ThingSpeak

Im Folgenden wird die plattformspezifische Implementierung der Anwendung für ThingSpeak beschrieben.

5.5.3.1 Einrichtung der Plattform

Im ersten Schritt muss ein neuer Channel auf ThingSpeak erstellt werden. Dieser benötigt einen Namen und zwei Datenfelder, um die Zugriffe und die Nachricht zu speichern.

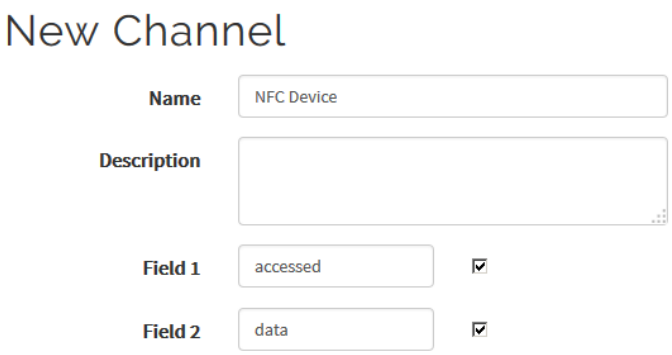

Abbildung 66: Neuer Kanal auf ThingSpeak [30]

Nach dem Erstellen folgt eine Weiterleitung auf die Seite des neuen Channels. Hier findet sich die Channel ID und unter dem Tab „API Keys“ der Write und der Read API Key. Diese Daten werden für die Kommunikation mit der Plattform über die HTTP REST Schnittstelle benötigt.

5.5.3.2 Kommunikation mit dem Arduino

Damit vom Arduino aus über das Wifi Shield HTTP-Requests versendet werden können, muss im Sketch zusätzlich die Datei ccspi.h eingebunden werden. Zusätzlich wird der Timeout für die TCP-Verbindung definiert:

```
…
#include <ccspi.h>
…
// Setze Timeout für die TCP-Verbindung
#define IDLE_TIMEOUT_MS  3000   // Wartezeit (in Millisekunden), bis die
Verbindung zum Server geschlossen wird falls keine Daten empfangen
werden
…
```

Als nächstes werden die Website und die Webpages für die HTTP-POST und –GET Requests festgelegt und die Variablen counter und ip erstellt:

```
…
// Website und Endpunkte der Requests
#define WEBSITE      "api.thingspeak.com"
#define WEBPAGE_POST "/update?key=<Write API Key>&field1="
#define WEBPAGE_GET "/channels/<Channel ID>/fields/2/last.json?key=<Read
API Key>"

int counter;
uint32_t ip;
…
```

Im Setup wird daraufhin zusätzlich die Variable counter initialisiert und die Funktionen resolveIP() und getMessage() aufgerufen:

```
// Setup
void setup(void)
{
  Serial.begin(115200); //Beginne serielle Übertragung
```

```
  Serial.println(F("Start"));
  counter = 0;
  accessed = 0;
  msg = "NFC initialisiert!";
  createMessage();
  nfc.write(ndefBuf, messageSize, 0); // Sende die NDEF Nachricht
  startWLAN();
  resolveIP();
  getMessage();
}
```

Die Funktion resolveIP() löst die IP für die oben definierte Website auf. Diese wird später für die TCP-Verbindung benötigt:

```
// Löse IP auf
void resolveIP(void)
{
  ip = 0;
  // IP Lookup
  Serial.print(WEBSITE); Serial.print(F(" -> "));
  while (ip == 0) {
    if (! cc3000.getHostByName(WEBSITE, &ip)) {
      Serial.println(F("IP konnte nicht aufgelöst werden!"));
    }
    delay(500);
  }

  cc3000.printIPdotsRev(ip);
}
```

Die Funktion getMessage() sendet einen GET-Request an ThingSpeak und liest das Feld aus, in dem die Nachricht gespeichert ist. Die empfangene Nachricht ist ein JSON Objekt. Aus diesem wird daraufhin der Wert des Feldes extrahiert und gespeichert. Daraufhin wird die Funktion createMessage() aufgerufen, um einen neuen NDEF Record mit der aktualisierten Nachricht zu erstellen (siehe Kapitel 5.5.1.2):

```
// Empfange Nachricht von Server
void getMessage(void)
{
  msg = "";
  boolean writeJson = false;
  /* Versuche eine Verbindung mit dem Server herzustellen.
     Das HTTP/1.1 Protokoll wird benutzt, um zu verhindern, dass der
     Server die Verbindung schließt bevor alle Daten übertragen wurden.
  */
  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80); // Starte TCP
Client
  if (www.connected()) {
    // GET Request über HTTP
    www.fastrprint(F("GET "));
    www.fastrprint(WEBPAGE_GET);
    www.fastrprint(F(" HTTP/1.1\r\n"));
```

```
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
www.fastrprint(F("\r\n"));
    www.fastrprint(F("\r\n"));
    www.println();
  } else {
    Serial.println(F("Verbindung fehlgeschlagen!"));
    return;
  }

  // Lese Daten bis die Verbindung geschlossen wird, oder der Timeout
erreicht wird
  unsigned long lastRead = millis();
  while (www.connected() && (millis() - lastRead < IDLE_TIMEOUT_MS)) {
    while (www.available()) {
      char c = www.read();
      Serial.print(c);
      // Extrahiere übertragenen String
      if (c == '{')
      {
        writeJson = true;
      }

      if (writeJson)
      {
        msg += c;
      }

      if (c == '}')
      {
        writeJson = false;
      }

      lastRead = millis();
    }
  }
  www.close();

  // Extrahiere Wert aus emfangenem String
  msg.remove(0, msg.indexOf("field2")+9);
  msg.remove(msg.length()-2);
  Serial.print(msg);
  // Erstelle aktualisierten NDEF Record
  createMessage();
}
```

Als nächstes wird die Loop-Schleife ausgeführt. In dieser wird die Variable counter bei jedem Durchgang um 1 hochgezählt. Erreicht counter den Wert 20 wird die Funktion getMessage() aufgerufen und counter auf 0 gesetzt. Mit dem Delay von 3 Sekunden bei der Bereitstellung der NDEF-Nachricht in der Funktion sendMessage() wird die Nachricht so jede Minute einmal aktualisiert, falls die Funktion sendRequest() nicht ausgelöst wurde. Am Ende des Loops wird die Funktion sendMessage() (siehe Kapitel 5.5.1.2) aufgerufen:

```
// Schleife
void loop(void) {
  counter++;
  Serial.println(F("\nSende NFC-Nachricht"));
  if (counter == 20)  // Rufe nach 20 Durchgängen getMessage auf
  {
    getMessage();
    counter = 0;
  }
  sendMessage();
}
```

Findet in der Funktion sendMessage() eine erfolgreiche Nachrichtenübertragung über NFC statt, wird die Funktion sendRequest() aufgerufen. In dieser wird die Variable accessed um 1 hochgezählt, um die Zugriffe auf das NFC-Shield zu zählen. Diese wird dann in einem POST-Request an ThingSpeak gesendet. Alle 10 Zugriffe wird an den Request das Query &status=sendTweet angehängt. Dies dient später dem auslösen des Tweet-Versands. Am Ende der Funktion wartet das Programm 15 Sekunden. Dies dient dem Einhalten des Zugriffslimits von einer Anfrage pro 15 Sekunden auf ThingSpeak:

```
// Sende Nachricht an Server
void sendRequest(void)
{
  accessed++;
  /* Versuche eine Verbindung mit dem Server herzustellen.
     Das HTTP/1.1 Protokoll wird benutzt, um zu verhindern, dass der
     Server die Verbindung schließt bevor alle Daten übertragen wurden.
  */
  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80); // Starte TCP
Client
  if (www.connected()) {  // Sende Request bei erfolgreicher Verbindung
    // POST Request über HTTP
    www.fastrprint(F("POST "));
    www.fastrprint(WEBPAGE_POST);
    www.print(accessed);
    if((accessed % 10) == 0)  // Alle 10 Zugriffe wird der Status
"sendTweet" angehängt
    {
      www.fastrprint(F("&status=sendTweet"));
    }
    www.fastrprint(F(" HTTP/1.1\r\n"));
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
www.fastrprint(F("\r\n"));
    www.fastrprint(F("\r\n"));
    www.println();
  } else {
    Serial.println(F("Verbindung fehlgeschlagen!"));
    return;
  }

  // Lese Daten bis die Verbindung geschlossen wird, oder der Timeout
erreicht wird
  unsigned long lastRead = millis();
```

```
  while (www.connected() && (millis() - lastRead < IDLE_TIMEOUT_MS)) {
    while (www.available()) {
      char c = www.read();
      Serial.print(c);
      lastRead = millis();
    }
  }
  www.close();  // Schließe die Verbindung
  delay(15000); // Warte 15 Sekunden
}
```

5.5.3.3 Webanwendung

Im plattformspezifischen Teil der Webanwendung wird zuerst eine Variable für den Request erzeugt. Danach folgt die Funktion, die das Submit-Event verarbeitet. In dieser werden zuerst alle noch laufenden Requests abgebrochen. Daraufhin wird das Formular in einem jQuery Objekt gespeichert und alle Formularfelder ausgewählt und in einer seperaten Variablen gespeichert. Über diese lassen sich später für die Dauer des Requests sämtliche Formulareingaben deaktivieren. Danach wird die Payload erstellt und über einen Ajax-Request an ThingSpeak gesendet. Erfolg oder Fehlschlag des Requests werden über die Status-Div ausgegeben.

```
$(document).ready(function() {
    // Variable für den Request
    var request ;

    // Binde das Submit Event an das Formular
    $("#msg_form").submit(function(event) {

        // Breche alle ausstehenden Requests ab
        if (request) {
            request.abort();
        }

        // Initialisiere lokale Variablen
        var $form = $(this);

        // Wähle alle Felder des Formulars und speichere sie
        var $inputs = $form.find("button, textarea");

        // Erstelle die Payload
        var payload = {
            'key' : '<Write API Key>' , // Write API Key
            'field2' : $('textarea[name=message]').val() // Hole den
Inhalt der Textarea
        } ;
        console.log(payload);

        // Deaktiviere alle Inputs für die Dauer des Ajax Requests
        $inputs.prop("disabled", true);

        // Sende den Request
        request = $.ajax( {
```

```
            url : "https://api.thingspeak.com/update",
            type : "post",
            data : payload
        });

        // Callback Handler bei Erfolg
        request.done(function(response, textStatus, jqXHR) {
            $("#status").text("Nachricht gesendet!"); // Setze Status
        });

        // Callback Handler bei Fehlschlag
        request.fail(function(jqXHR, textStatus, errorThrown) {
            $("#status").text("Fehler:" + textStatus, errorThrown); //
Setze Status
        });

        // Callback Handler der immer ausgelöst wird
        request.always(function() {
            // Reaktiviere alle Inputs
            $inputs.prop("disabled", false);
        });

        // Verhindere das Default-Submit des Formulars
        event.preventDefault();
    });
});
```

5.5.3.4 *Senden des Tweets*

Aktionen werden auf ThingSpeak über die plattformeignen Apps gesteuert. Für das Versenden des Tweets müssen eine React App und eine ThingTweet App erstellt werden. Dies erfolgt über den Menüpunkt App.

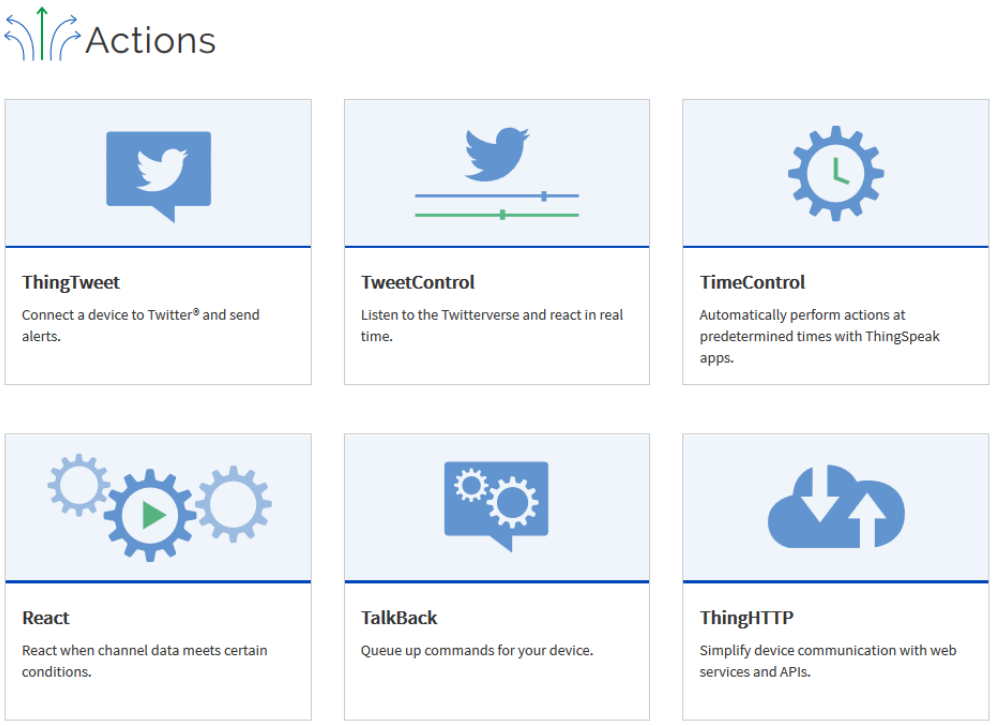

Abbildung 67: ThingSpeak Apps [202]

Zuerst muss die ThingTweet App konfiguriert werden, welche einen Twitter Account mit der App verknüpft. Dies geschieht per Klick auf den Button „Link Twitter Account“. Dieser startet den Authentifizierungsvorgang, bei dem der Benutzername und das Passwort des Accounts benötigt werden. Nach erfolgreicher Authentifizierung wird zurück auf ThingSpeak geleitet. Hier kann nun die React App erstellt werden. Nach der Appauswahl kann diese mit einem Klick auf „New React“ konfiguriert werden. Als Condition Type wird „Status“ ausgewählt. Dieser wird vom Arduino alle 10 Zugriffe übertragen. Als Testfrequenz wird „On Data Insertion“ gewählt, dies löst die Aktion sofort beim Eintreffen des Status aus. Als Kondition wird gesetzt, dass der Status des Channels den String „sendTweet“ enthält. Als Aktion kann nun die ThingTweet App ausgewählt werden. Als Tweet kann nun eine beliebige Nachricht gesetzt werden. Soll der Wert eines Feldes mit gesendet werden, kann dieser über die Syntax %%channel_<Channel ID>_field_<Feldnummer>%% eingefügt werden. Zuletzt muss nun noch der verknüpfte Twitter Account ausgewählt werden und ob die Aktion nur einmal oder bei jedem erfüllen der Bedingung ausgeführt werden soll.

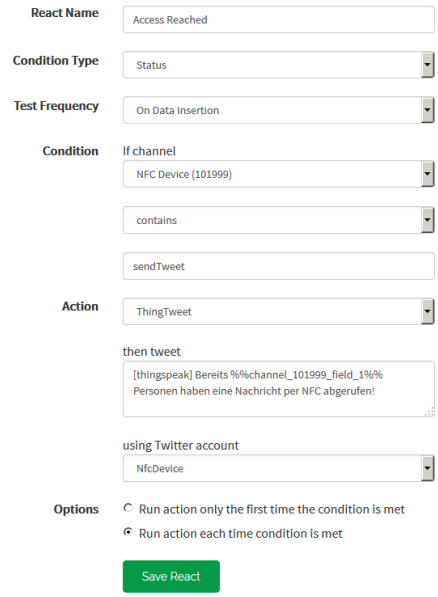

Abbildung 68: ThingSpeak React App [203]

5.5.4 Flowthings.io

Im Folgenden wird die plattformspezifische Implementierung der Anwendung für Flowthings.io beschrieben.

5.5.4.1 Einrichtung der Plattform

Im ersten Schritt müssen zwei neue Flows erstellt werden, einer für die Zugriffe und einer für die Nachrichten. Dies geschieht im Reiter Flows. Hier muss zuerst der Account ausgewählt werden. Auf der nächsten Seite kann mit Klick auf „New“ ein neuer Child-Flow erstellt werden. Die Namensgebung erfolgt hierbei über den Pfad.

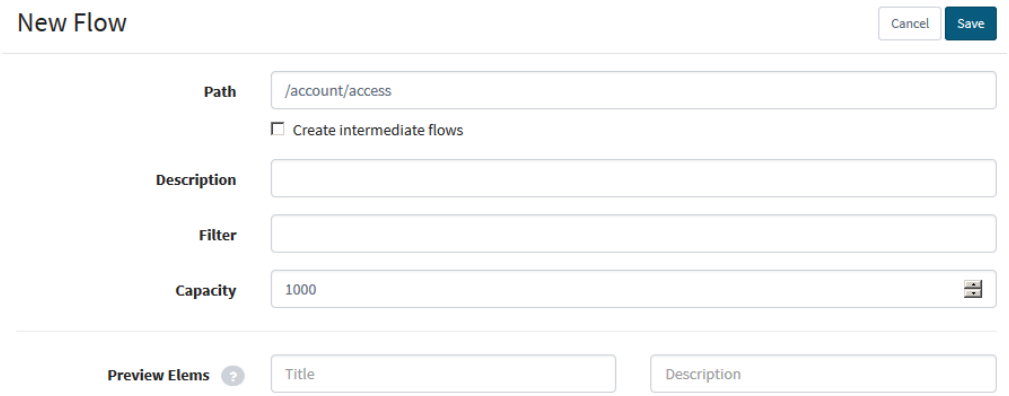

Abbildung 69: Flowthings.io Neuer Flow [52]

Nach dem Erstellen der Flows muss für jeden einzelnen Flow ein eigenes Access Token generiert werden. Dies ist über den Menüpunkt „Tokens“ im Accountmenü möglich.

5.5.4.2 Kommunikation mit dem Arduino

Da für die Kommunikation mit Flowthings.io wie bei ThingSpeak über eine HTTP-Schnittstelle läuft, kann das ThingSpeak Arduino Sketch größtenteils übernommen werden. Änderungen müssen in folgenden Teilen vorgenommen werden:

Initialisierung

```
// Website und Endpunkte der Requests
#define WEBSITE      "api.flowthings.io"
#define WEBPAGE_POST  "/v0.1/<Account>/drop/<Flow ID Zugriffe>"
#define WEBPAGE_GET   "/v0.1/<Account>/drop/<Flow ID Nachricht>?limit=1
&sort=elems.creationDate"
```

sendRequest()

Da die Datenübertragung beim POST-Request nicht wie bei ThingSpeak über Queries sondern in einem JSON-Objekt erfolgt, muss zuerst ein String mit der Nachricht erstellt werden. Im HTTP-Header muss das Access Token zur Authentifizierung mitgesendet werden. Als Content-Type wird application/json gesetzt, da ein JSON-Objekt mitgesendet wird. Zuletzt wird das JSON-Objekt als String gesendet. Im Gegensatz zu ThingSpeak wird am Ende kein Delay von 15 Sekunden benötigt, da Flowthings.io kein Zugriffslimit hat.

```
// Sende Nachricht an Server
void sendRequest(void)
{
  accessed++;
  /* Versuche eine Verbindung mit dem Server herzustellen.
     Das HTTP/1.1 Protokoll wird benutzt, um zu verhindern, dass der
     Server die Verbindung schließt bevor alle Daten übertragen wurden.
  */
  // Erstelle zu übertragenden String
  String data = "{\"elems\":{\"acc\":";
  data += accessed;
  data += "}}";

  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80); // Starte TCP
Client
  if (www.connected()) {
    // POST Request über HTTP
    www.fastrprint(F("POST "));
    www.fastrprint(WEBPAGE_POST);
    www.fastrprint(F(" HTTP/1.1\r\n"));
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("X-Auth-Token: <Access Token Zugriffe>"));
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("Content-Type: application/json"));
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("Content-Length: "));
    www.print(data.length());
```

```
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("\r\n"));
    Serial.print(F("Sending data ...\n"));
    www.print(data);
    www.fastrprint(F("\r\n"));
  } else {
    Serial.println(F("Verbindung fehlgeschlagen!"));
    return;
  }

  // Lese Daten bis die Verbindung geschlossen wird, oder der Timeout
erreicht wird
  unsigned long lastRead = millis();
  while (www.connected() && (millis() - lastRead < IDLE_TIMEOUT_MS)) {
    while (www.available()) {
      char c = www.read();
      Serial.print(c);
      lastRead = millis();
    }
  }
  www.close();  // Schließe die Verbindung
}
```

getMessage()

In dieser Funktion muss nur der HTTP-Header um das X-Auth-Token und den Content-Type erweitert werden. Zusätzlich gelten bei der Extrahierung der Nachricht andere Werte:

…

```
// GET Request über HTTP
www.fastrprint(F("GET "));
www.fastrprint(WEBPAGE_GET);
www.fastrprint(F(" HTTP/1.1\r\n"));
www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
www.fastrprint(F("\r\n"));
www.fastrprint(F("X-Auth-Token: <Access Token Nachricht>"));
www.fastrprint(F("\r\n"));
www.fastrprint(F("Content-Type: application/json"));
www.fastrprint(F("\r\n"));
www.println();
```

…

```
// Extrahiere Wert aus emfangenem String
msg.remove(0, msg.indexOf("value")+8);
msg.remove(msg.length()-2);
Serial.println();
Serial.print(msg);
```

…

5.5.4.3 Webanwendung

Auch die Webanwendung ist größtenteils mit ThingSpeak identisch. Anpassungen sind hier an folgenden Stellen nötig:

Erstellen der Payload

```
// Hole den Inhalt der Textarea
var $msg = $("textarea[name=message]").val();

// Erstelle die Payload
var payload = '{"elems":{"msg":"' + $msg + '"}}';
console.log(payload);
```

Senden des Requests
Aufgrund der Unterschiedlichen Authentifizierung muss hier ein X-Auth-Token Header hinzugefügt werden.

```
// Sende den Request
request = $.ajax( {
     url : "https://api.flowthings.io/v0.1/<Account>/drop/<Flow ID
Nachricht>",
     type : "post",
     data : payload,
     headers : {
             "X-Auth-Token" : "<Access Token Nachricht>"
     }
});
```

5.5.4.4 Senden des Tweets

Aktionen werden auf FlowThings.io über Tracks gesteuert. Ein neuer Track kann auf der Plattform unter dem Reiter „Tracks“ erstellt werden. Auf der folgenden grafischen Oberfläche können dann einzelne Logik und Aktionsbausteine miteinander verknüpft werden.

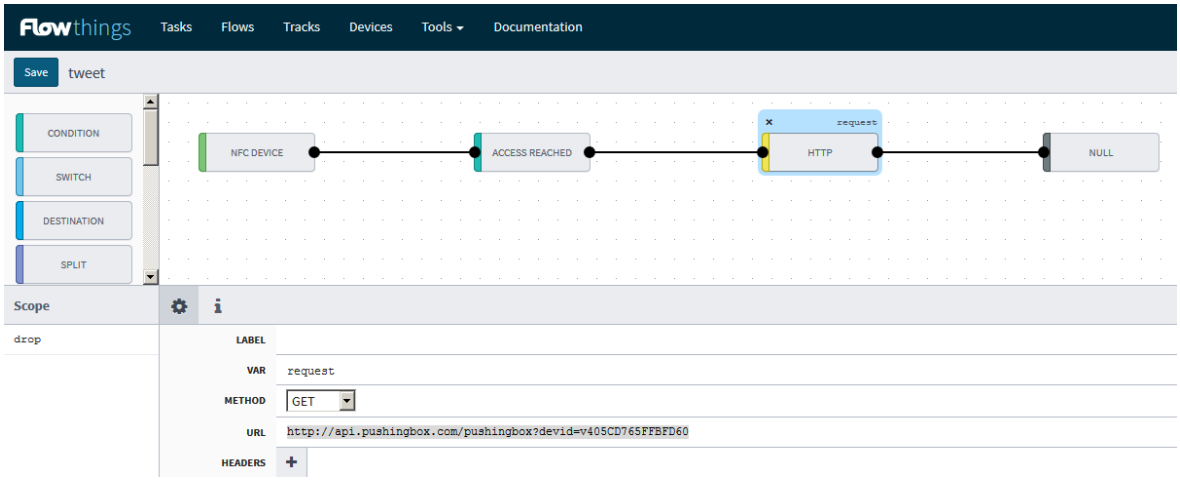

Abbildung 70: Flowthings.io Tracks [52]

Ausgangspunkt ist hier die Datenquelle. Für diese muss der Pfad des Flows angegeben werden, auf dessen Daten zugegriffen werden soll. Als nächstes folgt ein

Bedingungsblock. In diesem wird als Bedingung gesetzt, dass alle 10 Zugriffe die folgende Aktion ausgelöst werden soll. Die Bedingung hierfür lautet drop.elems.acc.value %10 == 0. Als Aktion soll daraufhin ein Tweet gesendet werden. Jedoch bietet Flowthings.io keine Möglichkeit, dies direkt über die Plattform zu tun. Allerdings kann hier der externe Service PushingBox (https://www.pushingbox.com/) verwendet werden. Dieser bietet eine einfache API, über welche sich das Versenden von Nachrichten triggern lässt. Hierfür wird auf PushingBox zuerst ein neuer Twitter Service angelegt. Dabei startet der gleiche Authentifizierungsablauf wie bereits bei ThingTweet. War die Authentifizierung erfolgreich, kann ein neues Szenario angelegt werden. Dieses löst Empfang eines HTTP-Requests Aktionen aus.

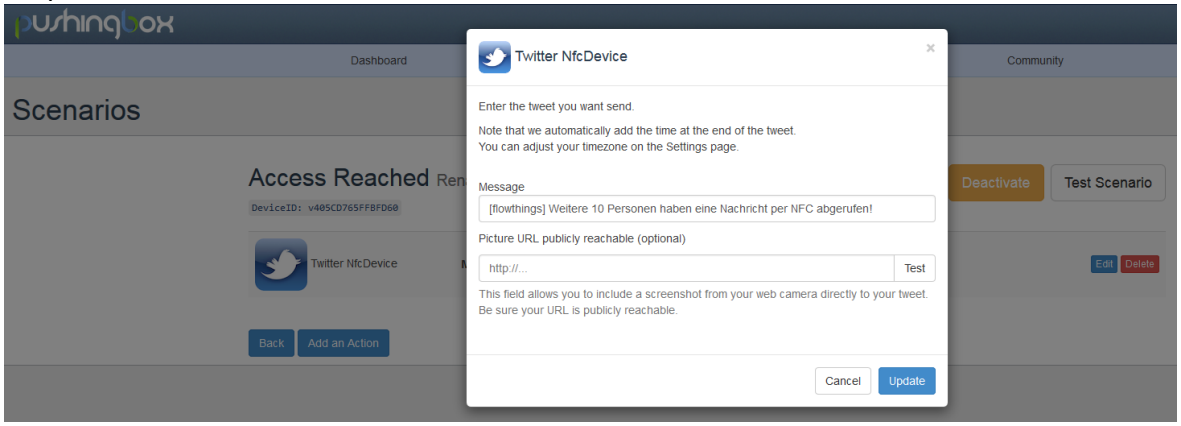

Abbildung 71: PushingBox Tweet Action [204]

Im Falle dieser Anwendung wird hier eine Aktion angelegt, die den zuvor registrierten Service benutzt um einen Tweet zu senden. Auf Flowthings.io kann nun als Aktion ein HTTP Block hinzugefügt werden, der einen GET-Request an http://api.pushingbox.com/pushingbox?devid=<DeviceID> sendet. Zum Abschluss wird ein NULL Block gesetzt.

5.5.5 IBM Watson IoT

Im Folgenden wird die plattformspezifische Implementierung der Anwendung für ThingSpeak beschrieben.

5.5.5.1 Einrichtung der Plattform

Der Startpunkt für die Einrichtung der Plattform ist die IBM Bluemix Konsole (https://console.eu-gb.bluemix.net/catalog/). Hier können im Reiter „Katalog“ die einzelnen Services ausgewählt werden. Für diese Anwendung wird der „Internet of Things Platform“-Dienst verwendet.

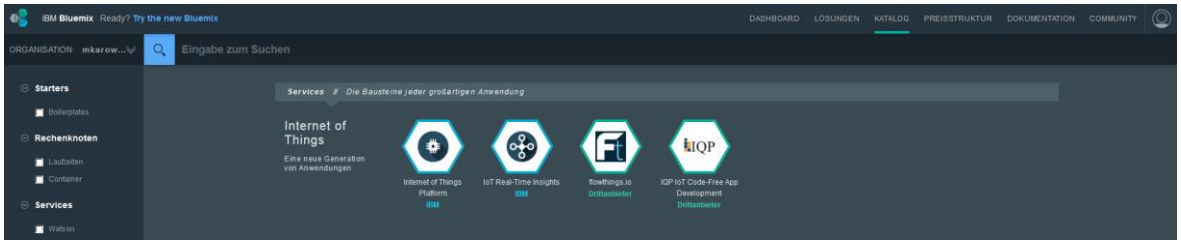

Abbildung 72: IBM Bluemix Katalog [158]

Nach der Bereitstellung ist dieser unter einer eigenen URL nach dem Schema <Organisation ID>.internetofthings.ibmcloud.com zu erreichen. Hier kann nun im Menü unter „Geräte“ ein neues Device hinzugefügt werden.

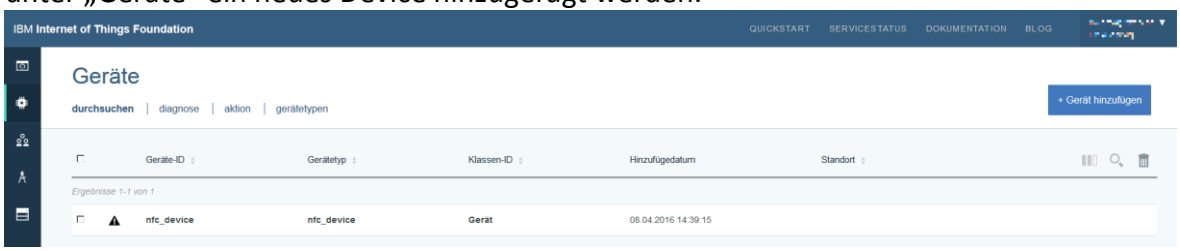

Abbildung 73: IBM Watson IoT Dashboard [155]

Beim Erstellen eines neuen Devices muss zunächst ein Gerätetyp erstellt werden. Dieser dient als Vorlage, falls weitere Geräte vom gleichen Typ hinzugefügt werden sollen. Für das Device selbst muss zwingend nur eine Geräte-ID festgelegt werden. Zusätzliche Informationen zur Hardware sowie Metadaten sind optional. Das automatisch erstellte Authentifizierungstoken muss unbedingt direkt nach der Erstellung an einem sicheren Ort gespeichert werden, da es danach nicht mehr einsehbar ist. Nun kann direkt auch im Menü „Zugriff“ unter dem Reiter „api-schlüssel“ ein Authentifizierungstoken für die Webanwendung generiert werden. Auch hier ist das Token nur direkt bei der Erstellung einsehbar und sollte direkt extern gespeichert werden.

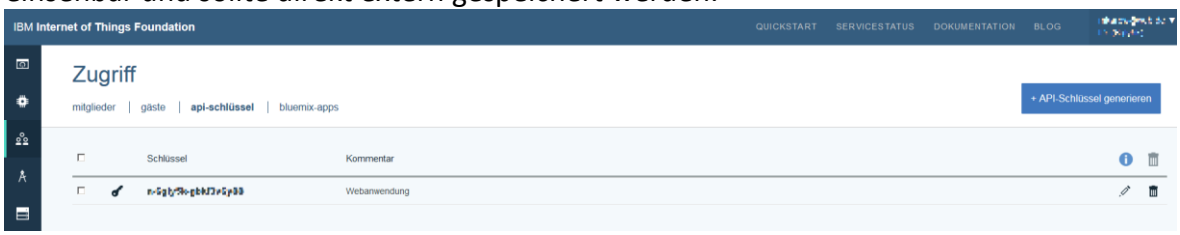

Abbildung 74: IBM Watson IoT API-Schlüssel [155]

5.5.5.2 Kommunikation mit dem Arduino

Im Gegensatz zu ThingSpeak und Flowthings.io wird die Nachrichtenübertragung für IBM Watson IoT nicht über HTTP, sondern über MQTT realisiert. Dies hat den Grund, dass die Webclient-Library des Adafruit CC3000 Wifi Shields keinen Basic-Auth Header unterstützt. Dieser wäre aber für die Authentifizierung über HTTP nötig. Dies hat zur Folge, dass einige grundlegende Änderungen am Sketch für den Arduino vorgenommen werden müssen. Dies beginnt beim Einbinden der Libraries. Hier muss zusätzlich die MQTT Library bereitgestellt werden:

```
// MQTT Library
#include <PubSubClient.h>
```

Des Weiteren müssen der MQTT-Host, der MQTT-Client, Benutzername und Passwort, sowie die Endpunkte des Message Brokers definiert und das Client-Objekt erstellt werden. Die Event ID und Command ID sind hier synonym zum Namen eines Datenstreams und können frei gewählt werden:

```
// Host
#define HOST "<Organisation ID>.messaging.internetofthings.ibmcloud.com"
// Client-ID
#define CLIENTID "d:<Organisation ID>:<Device Type>:<Device ID>"
// Benutzername & Passwort
```

```
#define USERNAME "use-token-auth"
#define PASSWORD "<Device Authentifizierungstoken>"
// Endpunkte
#define PUBPATH "iot-2/evt/<Event ID>/fmt/json"
#define SUBPATH "iot-2/cmd/<Command ID>/fmt/json"

// Erstelle MQTT Client
Adafruit_CC3000_Client c;
PubSubClient client(c);
```

In der Setup-Funktion wird nun der Zielserver auf dem der Message Broker gehostet ist gesetzt und die Callback-Funktion für den Datenempfang definiert. Nach Start des WLANs wird direkt die Funktion connectMqtt() aufgerufen, die den Client auf dem Arduino mit dem Server verbindet:

```
// Setup
void setup(void)
{
  Serial.begin(115200); //Beginne serielle Übertragung
  Serial.println(F("Start"));
  client.setServer(HOST,1883);  // Setze MQTT Broker
  client.setCallback(getMessage); // Setze Callback-Funktion für
empfangene Nachrichten
  accessed = 0;
  msg = "NFC initialisiert!";
  createMessage();
  nfc.write(ndefBuf, messageSize, 0); // Sende die NDEF Nachricht
  startWLAN();
  connectMqtt();
  msg = "Default Message";
  createMessage();
}
```

Die connectMqtt() Funktion stellt die Verbindung zum Message Broker her. Hierzu werden die ClientID, der Benutzername und das Passwort übergeben. Ist die Verbindung erfolgreich, verbindet sich der Client direkt mit dem Endpunkt, über den die Nachrichten an den Arduino gesendet werden.

```
// Verbinde mit MQTT
void connectMqtt(void)
{
  if(client.connect(CLIENTID,USERNAME,PASSWORD))
  {
    Serial.println(F("Verbunden mit MQTT\n"));
    if(client.subscribe(SUBPATH)) // Subscribe zu gewünschtem Topic
    {
      Serial.println(F("Erfolgreich subscribed!\n"));
    }
  }
}
```

In der Loop-Schleife müssen mit MQTT keine Durchgänge mehr gezählt werden, da die Nachrichten über die client.loop() Funktion bei erhalt direkt an die Callback-Funktion getMessage() geleitet werden:

```
// Schleife
void loop(void) {
  Serial.println(F("\nSende NFC-Nachricht"));
  sendMessage();
  client.loop();  // Prüfe, ob neue Nachricht verfügbar
}
```

Die Funktion getMessage() erhält als Eingangsparameter das Thema, die Daten und die Länge der Daten welche gesendet wurden. Zur Extrahierung der Nachricht wird die Payload vom Typ Byte in einen String umgewandelt und dann weiter verarbeitet:

```
// Empfange Nachricht von Server
void getMessage(char* topic, byte* payload, unsigned int length)
{
  msg = "";
  msg = (char*)payload; // konvertiere Nachricht in String
  // Extrahiere empfangene Nachricht
  msg.remove(0, msg.indexOf("message")+11);
  msg.remove(msg.indexOf("}")-1);
  Serial.println(F("Empfangene Payload:\n"));
  Serial.println(msg);
  // Erstelle aktualisierten NDEF Record
  createMessage();
}
```

Wie schon bei den anderen beiden Plattformen wird auch hier bei erfolgreichem Versenden einer NFC-Nachricht die sendRequest() Funktion aufgerufen. In dieser wird die Zugriffsvariable hochgezählt und die Payload erstellt. Diese ist als String in JSON formatiert und wird zur Übertragung in ein CharArray umgewandelt. Die Übertragung selbst geschieht über den Aufruf der Funktion client.publish(), welche als Parameter den Endpunkt der Zugriffsdaten und das CharArray mit der Payload enthält:

```
// Sende Nachricht an Server
void sendRequest(void)
{
  accessed++;

  // Erstelle zu übertragenden String
  String data = "{\"d\":{\"acc\": ";
  data += accessed;
  data += "}}";
  int str_len = data.length() + 1;
  char json[str_len];
  data.toCharArray(json, str_len);  // konvertiere String in CharArray

  if(client.publish(PUBPATH, json)) // Sende Nachricht an MQTT Broker
  {
    Serial.println(F("Nachricht published\n"));
```

```
  }
  else
  {
    Serial.println(F("Publishing Error!\n"));
  }
}
```

5.5.5.3 Webanwendung

Wie auf dem Arduino realisiert auch die Webanwendung die Datenübertragung über das MQTT-Protokoll. Hierzu wird der Paho MQTT JavaSript Client verwendet, der unter https://projects.eclipse.org/projects/technology.paho/downloads zur Verfügung steht. Dieser muss nach jQuery in die HTML-Datei eingebunden werden:

```
<script src="https://code.jquery.com/jquery-2.2.3.min.js"></script>
<script src="js/mqttws31.js"></script>
```

Beim Laden des HTML-Dokuments wird direkt ein MQTT-Client erstellt. Dieser erhält als Parameter den Host-Server, den Port über den der Message Broker erreichbar ist und die Client ID. Die Client ID ist frei wählbar. Für den Client werden als Optionen der Verbindungstimeout, der Benutzername und das Authentifizierungstoken gesetzt. Zusätzlich werden zwei Funktionen erstellt, die bei erfolgreicher oder fehlgeschlagener Verbindung einen Alert auslösen. Am Ende erfolgt mit client.connect() die Herstellung der Verbindung:

```
$(document).ready(function() {
    // Erstelle einen MQTT Client
    var client = new Paho.MQTT.Client(<Organisation
ID>.messaging.internetofthings.ibmcloud.com",
      Number(1883),
      "a:<Organisation ID>:<App ID>"
    );

    // Setze Verbindungsoptionen
    var options = {
        timeout : 3 , // Verbindungstimeout
        userName : "<API Schlüssel>",
        password : "<Authentifizierungstoken>",

        // Alert bei erfolgreicher Verbindung
        onSuccess : function() {
            alert("MQTT verbunden!");
        },

        // Alert bei fehlgeschlagener Verbindung
        onFailure : function(message) {
            alert("Verbindung fehlgeschlagen: "
            + message.errorMessage + " Bitte Seite neu laden!");
        }
    };
```

```
    // Versuche eine Verbindung herzustellen
    client.connect(options);

    …

});
```

In der Funktion, welche das Submit-Event abfängt wird zuerst wie schon bei den anderen Plattformen das Textfeld ausgelesen und die Payload generiert. Danach wird die Funktion sendData() aufgerufen:

```
// Binde das Submit Event an das Formular
$("#msg_form").submit(function(event) {

    // Initialisiere lokale Variablen
    var $form = $(this);

    // Wähle alle Felder des Formulars und speichere sie
    var $inputs = $form.find("button, textarea");

    // Hole den Inhalt der Textarea
    var $msg = $("textarea[name=message]").val () ;

    // Erstelle die Payload
    var payload = '{"d":{"message": "' + $msg + '"}}';
    console.log(payload);

    // Deaktiviere alle Inputs für die Dauer des Ajax Requests
    $inputs.prop("disabled", true);

    // Rufe sendData() auf
    sendData(payload);

    // Reaktiviere alle Inputs
    $inputs.prop("disabled", false);

    // Prevent default posting of form
    event.preventDefault();
});
```

In der Funktion sendData() wird die eigentliche MQTT-Nachricht erstellt. Danach wird der Endpunkt gesetzt, für den diese bestimmt ist. Nach dem Setzen des Quality of Service Levels wird die Nachricht dann versandt:

```
function sendData(payload) {
    var message = new Paho.MQTT.Message(payload); // Erstelle Nachricht
    message.destinationName = "iot-2/type/<Device Type>/id/<Device
ID>/cmd/<Command ID>/fmt/json"; // API Endpunkt
    message.qos = 0;
    client.send(message); // Sende Nachricht
    $("#status").text("Nachricht gesendet!"); // Setze Status
}
```

5.5.5.4 Senden des Tweets

Der Internet of Things Platform Service unterstützt selbst keine Weiterverarbeitung von Daten oder das Ausführen von Aktionen. Dies lässt sich über die Integration des Services in eine Plattformanwendung von IBM Bluemix. Wie bereits der IoT Platform Service lässt sich eine Anwendung über die IBM Bluemix Konsole erstellen. Für den schnellen Start stehen hier Boilerplates zur Verfügung.

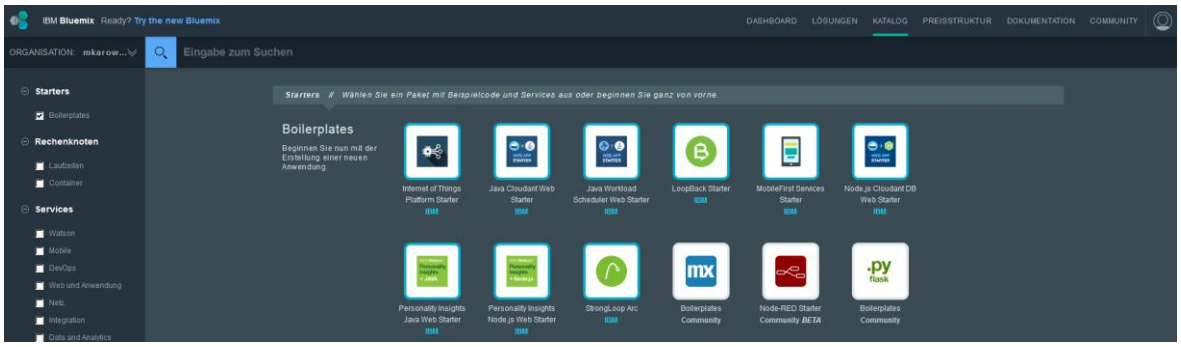

Abbildung 75: IBM Bluemix Boilerplates [158]

Für die Twitter-Anwendung wird das Node-RED Starter Paket verwendet. Dieses bietet eine komfortable Entwicklungsumgebung mit Flow-Editor auf Basis einer SDK für Node.js. Nach der Bereitstellung der Node-RED Anwendung muss im Dashboard der Internet of Things Platform Service eingebunden werden. Dies geschieht über den Button „Service oder API binden“. Im folgenden Dialog kann dann der Service ausgewählt werden.

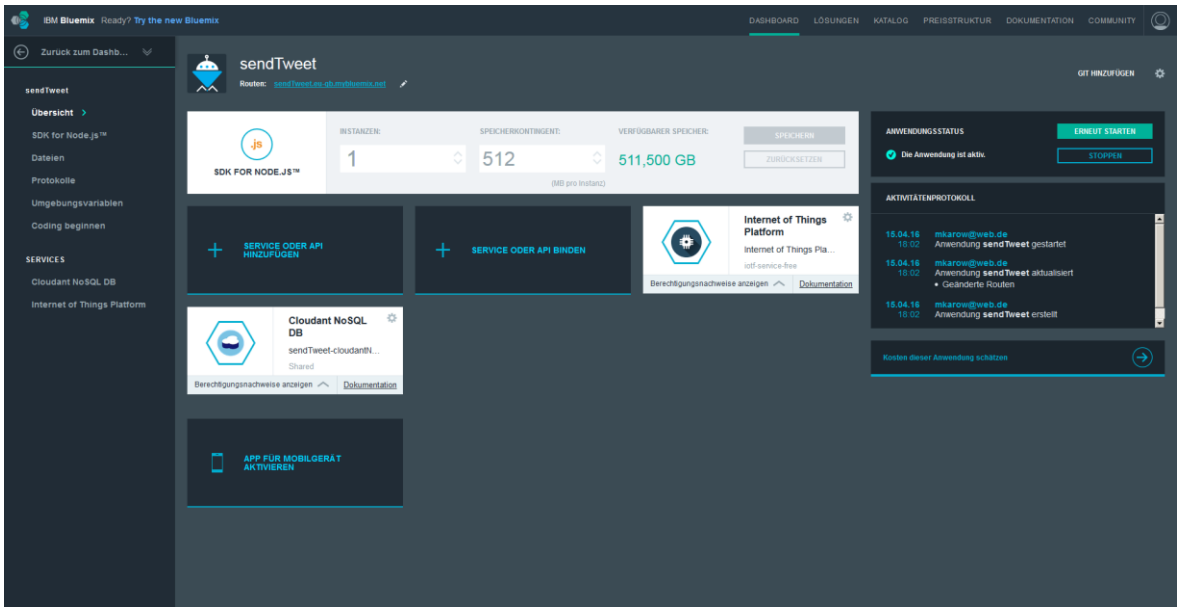

Abbildung 76: Node-RED Dashboard [158]

Nach dem Einbinden wird die Anwendung neu gestartet. Ab diesem Zeitpunkt steht die Entwicklungsumgebung unter der URL <Anwendungsname>.eu-gb.mybluemix.net zur Verfügung. Dort lässt sich der Flow-Editor aufrufen. Dieser bietet eine große Auswahl an Inputs, Outputs, Funktionen und Schnittstellen zu anderen Plattform- und Webservices.

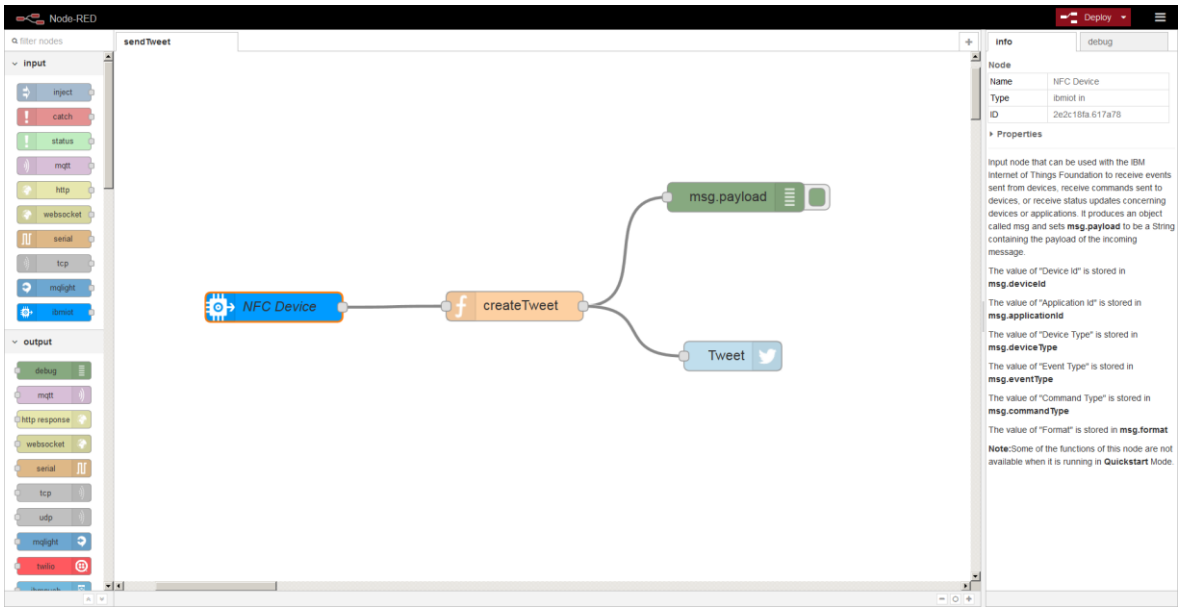

Abbildung 77: Node-RED Flow-Editor

Als Startpunkt und Datenquelle dient eine ibmiot-Node. In dieser werden das Device und das zugehörige Event konfiguriert, dessen Daten weiter verarbeitet werden sollen. Die Authentifizierung funktioniert hierbei automatisch. Als Input Type wird ein Device Event gewählt.

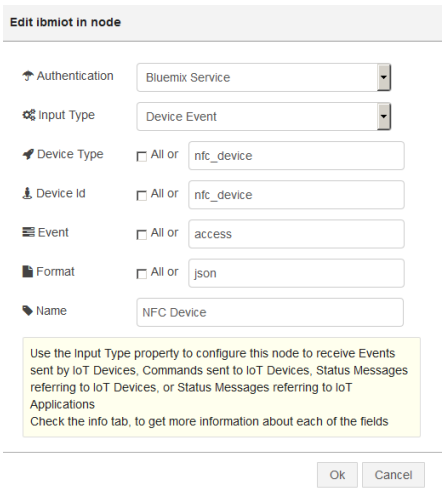

Abbildung 78: ibmiot-Node

Die empfangenen Daten werden in einer Function-Node verarbeitet. Programiersprache ist hierbei JavaScript. Die Funktion überprüft, ob die Anzahl der Zugriffe durch 10 teilbar ist. Ist diese Bedingung wahr, wird die Nachricht, welche getweetet werden soll in das Message-Object msg geschrieben und zurückgegeben. Somit leitet die Funktion alle 10 Zugriffe eine Twitternachricht weiter.

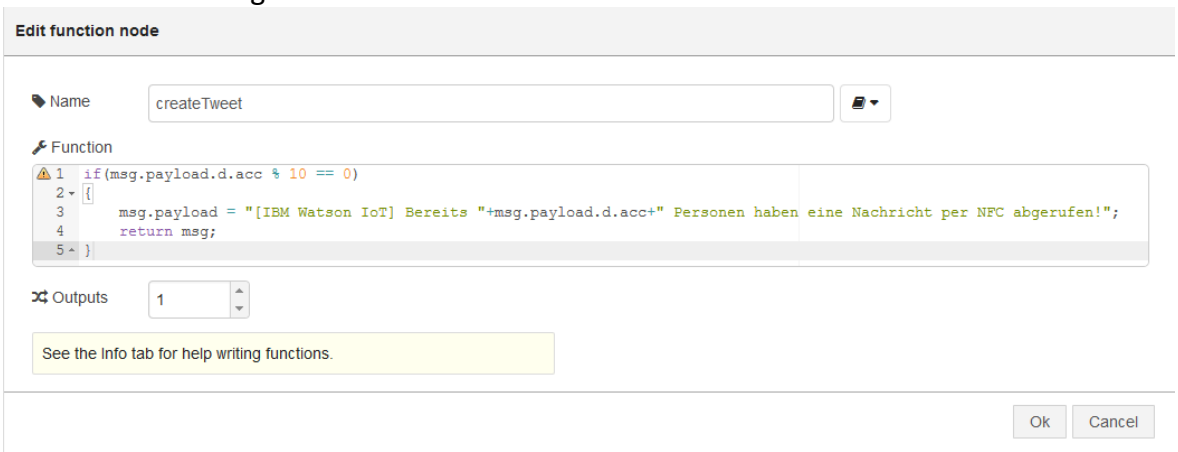

Abbildung 79: Function-Node

Die vollständige Bedingung hierfür lautet msg.payload.d.<Value ID> % 10 == 0.

Gibt die Function-Node einen Wert zurück, wird dieser von der twitter-Node empfangen. Diese verwendet den gleichen Authentifizierungsvorgang, der bereits bei den anderen beiden Plattformen zum Einsatz kam und einen Account mit der Plattform zu verknüpfen. Empfängt die twitter-Node eine Nachricht, wird der Inhalt der Payload automatisch mit dem verknüpften Account getweetet. Um die Payload vorher zu überprüfen, kann eine debug-Node verwendet werden, welche die gewünschten Daten in einem Debug-Tab ausgibt.

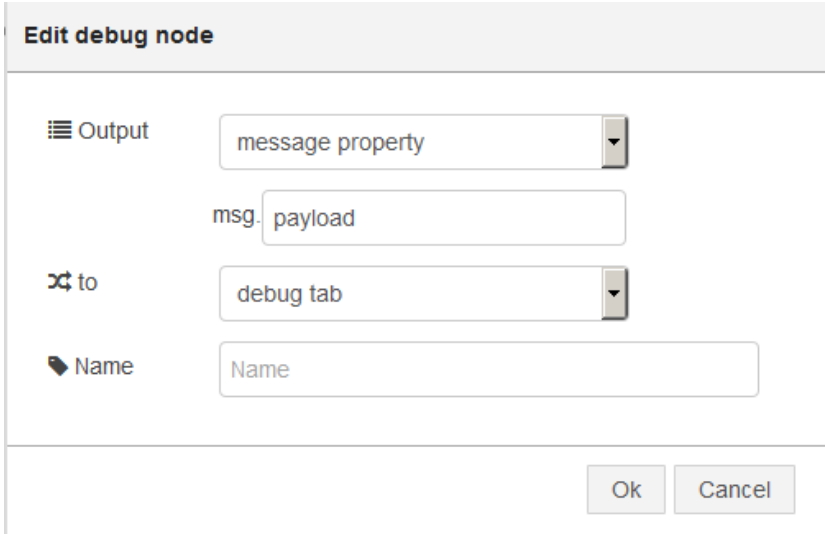

Abbildung 80: Debug-Node

6 Auswertung

Bei der Implementierung der IoT-Anwendung zeigt sich stark der Fokus auf die unterschiedlichen Zielgruppen der Plattformen. Als Anwender sollte man sich daher vorher bewusst machen, welche Funktionalitäten für die umzusetzende Anwendung benötigt werden.

ThingSpeak bietet als Plattform für Hobbyentwickler nur einen begrenzten Funktionsumfang bei für die Weiterverarbeitung von Daten und konkrete Aktionen. Jedoch ist die Umsetzung dieser Funktionalitäten gerade für Einsteiger sehr attraktiv. Die Plattform bietet mit seiner einfachen Handhabung und vielen Hilfestellungen eine geringe Einstiegshürde, die nicht zuletzt durch die kostenlose Nutzbarkeit noch weiter gesenkt wird. Auch die Integration von Devices und die Datenübertragung von und zu externen Webanwendungen gestaltet sich sehr einfach, da sämtliche Daten sowie die Authentifizierung über Queries im HTTP-Request übertragen werden. Durch die weite Verbreitung von ThingSpeak finden sich fast zu jeder erdenklichen Hardwarekombination Hilfestellungen und Tutorials, die bei der Entwicklung helfen. Etwas einschränkend ist die geringe Verbindungsrate, die schnelle Reaktionen bei der Datenübertragung über die API erschwert.

Flowthings.io positioniert sich als umfangreichste der kostenlosen Plattformen an der Grenze zwischen Hobbyentwicklern und professioneller Nutzung. Die Bedienung der Drops/Flow/Tracks Architektur kann am Anfang etwas verwirrend sein, mit kurzer Einarbeitungszeit und der gut geschriebenen Dokumentation stellt dies allerdings kein Hindernis dar. Die Integration von Devices und Webanwendungen gestaltet sich durch die Übertragung der Daten in JSON-Objekten und die verwendete Authentifizierung etwas komplizierter. Der Anwender sollte hier zumindest ein Grundwissen für den Aufbau eines HTTP-Headers mitbringen. Die Weiterverarbeitung der Daten gestaltet sich durch den Flow-Editor der Tracks-Komponente auch für weniger erfahrene Nutzer als einfach. Jedoch ist die Funktionalität im Bereich der möglichen Aktionen etwas eingeschränkt, da die einzige Schnittstelle nach außen über HTTP-Requests bereitgestellt wird. Insgesamt ist diese Plattform für ambitioniertere Hobbyentwickler und kleinere Projekte geeignet, da sie trotz der eingeschränkten Funktionalität immer noch mehr Features bietet, als viele der kostenpflichtigen Angebote.

Die IBM Watson Internet of Things Platform richtet sich klar an Unternehmen und erfahrene Entwickler, die eine starke und skalierbare Plattform für ihre Anwendungen benötigen. Der Einstieg ist durch den schieren Umfang der gesamten IBM Bluemix Plattform komplexer als bei den anderen beiden. Nach der Einarbeitungszeit gestaltet sich die Konfiguration und Entwicklung auf der Plattform jedoch als sehr angenehm, da unterschiedliche Services und Cloudanwendungen reibungslos miteinander zusammen arbeiten und sich schnell einbinden lassen. Für Entwickler ist zusätzlich die große Auswahl an Programmiersprachen zur Entwicklung von Plattformanwendungen von Vorteil. Durch den Einsatz von Boilerplates und vorgefertigten Lösungen ist mit der Plattform auch Rapid Prototyping kein Problem. Besonders interessant war bei dieser Implementierung die Benutzung des MQTT-Protokolls, die eigentlich nur durch den mangelnden Umfang der

Adafruit Wifi Shield CC3000 Library im Bereich der Authentifizierung entstand. Die gezielte Entwicklung von MQTT als Protokoll für die M2M-Kommunikation kann im Bereich des Internet of Things ganz klar seine Stärken ausspielen. Der geringe Protokolloverhead, die Geschwindigkeit, die einfache Implementierung und die Möglichkeit Nachrichten an ein Device oder eine Anwendung zu pushen sind der Datenübertragung per HTTP deutlich überlegen. Wenn möglich sollte das MQTT-Protokoll bei der Konzeption einer neuen IoT-Anwendung deshalb auf jeden Fall in Betracht gezogen werden.

7 Zusammenfassung

Im Rahmen der Master Thesis werden eine Auswahl von Platform as a Service-Angeboten für das Internet of Things untersucht und miteinander verglichen. Auf Grundlage der Untersuchung erfolgt die Implementierung einer IoT-Anwendung für drei der Plattformen. Durch die theoretische Einleitung in die Themen Platform as a Service und Internet of Things, sowie die für das IoT typischen Verbindungsprotokolle HTTP und MQTT wird ein grundlegendes Verständnis für das Sachgebiet vermittelt.

Nach den erklärenden Definitionen folgt die Untersuchung der PaaS-Angebote. Hier werden zunächst die Untersuchungskriterien vorgestellt und erklärt, nach denen die einzelnen Plattformen untersucht werden. Diese umfassen die quantitativen Kriterien Verbreitung, Preis, Verbindungsrate, Datenmenge und Schnittstellen, sowie die qualitativen Kriterien API, Visualisierung, Weiterverarbeitung, Dokumentation, Usability und Sicherheit. Danach erfolgt die detailierte Untersuchung der Plattformen nach den festgelegten Kriterien. Untersucht wurden hierbei die Plattformen ThingSpeak, data.sparkfun.com, RunAbove IoT Lab, flowthings.io, Carriots, Ubidots, GroveStreams, Exosite, Beebotte, MODE, Initial State, Temboo, Oracle Cloud, IBM Watson Internet of Things, Azure IoT Hub, AWS IoT und Google Pub/Sub.
Im Anschluss an die Untersuchung werden die Ergebnisse gegliedert nach den Untersuchungskriterien miteinander verglichen und abschließend ausgewertet. Das Ergebnis der Plattformuntersuchung bietet die Grundlage für die Implementierung einer Internet of Things Anwendung.

Für die Anwendung werden zunächst die Anforderungen beschrieben, welche die eines realitätsnahen Nutzungsfall nachbilden sollen. Das Konzept und die Funktionsweise stellt das Zusammenspiel der einzelnen Komponenten dar. Die Anwendung soll hierbei die Funktionalität eines Infoterminals nachstellen, welches über NFC Informationen an mobile Endgeräte bereitstellt. Dieses sendet die Anzahl der Zugriffe an das PaaS-Angebot, welches wiederum als Schnittstelle zu Twitter und einer externen Webanwendung dient. Über die Webanwendung kann die Nachricht des Infoterminals aktualisiert werden. Im Folgenden werden die zum Einsatz kommende Hardware und Software beschrieben. Es handelt sich hierbei um einem Arduino Uno Rev3, welcher über ein Adafruit CC3000 Wifi Shield und ein Seeed Studio NFC Shield V2.0 die benötigten Schnittstellen zur Verfügung stellt.

Anschließend erfolgt die Auswahl der drei für die Implementierung verwendeten Plattformen anhand der zuvor durchgeführten Untersuchung. Hierbei wurden als Vertreter verschiedener Benutzergruppen ThingSpeak, Flowthings.io und IBM Watson Internet of Things gewählt.

Im Anschluss an die Plattformauswahl wird die konkrete Umsetzung beschrieben. Zuerst erfolgt der Aufbau und die Programmierung der Grundfunktionalitäten zur Datenübertragung des Arduinos. Danach wird im Detail auf den allgemeinen Teil der Webanwendung eingegangen, welche mit HTML, CSS und jQuery realisiert wurde. Danach erfolgt die Beschreibung der plattformspezifischen Umsetzung der jeweiligen Kommunikation zwischen Arduino und Plattform, sowie der Webanwendung. Hierbei

belaufen sich die Unterschiede hauptsächlich auf die verschiedene Implementierung der Authentifizierung, sowie den verwendeten Übertragungsprotokollen HTTP und MQTT. Nach der Implementierung wird diese ausgewertet um die konkreten Herausforderungen, sowie Vor- und Nachteile der Plattformen, der verwendeten Hardware und Protokolle herauszustellen.

Im letzten Kapitel wird die Arbeit zusammengefasst und abschließend anhand aller gesammelten Erkenntnisse bewertet.

Zusammenfassend bleibt zu sagen, dass das Angebot an PaaS-Plattformen für das Internet of Things sehr durchwachsen ist. Die große Zahl an unterschiedlichen Preismodellen und Funktionen machen die Auswahl einer Plattform ohne vorherige Anforderungsanalyse so gut wie unmöglich. Positiv ist, dass alle Angebote mindestens einen zeitlich begrenzten kostenlosen Test-Account oder Freikontingente zur Verfügung stellen. Auch die Bedienung und Benutzerführung ist bei einem Großteil der Plattformen auf dem aktuellen Stand der Technik. Vom Funktionsumfang her liegen die meisten kostenpflichtigen Plattformen klar vor den kostenlosen Angeboten. Jedoch gibt es auch hier Ausnahmen, wie zum Beispiel Plattformen wie Initial State, welche sich rein auf die Datenvisualisierung konzentriert oder Temboo, die als Schnittstelle zu mehreren hundert Webservices dient. Bei den Plattformen der großen Cloud Anbieter Oracle, IBM, Microsoft, Amazon und Google zeigt sich ein starker Fokus auf erfahrene Entwickler und die Integration von externen Enterprise-Anwendungen. Gerade AWS IoT und Google Pub/Sub sind ohne Kenntnisse der jeweiligen Cloud-Plattform kaum nutzbar. In Hinsicht auf Sicherheit, was ein immer stärkeres Thema wird, ergibt sich ein positives Bild. Alle Plattformen bieten unterschiedliche Authentifizierungsmethoden und TLS als Verbindungssicherheit. Jedoch sind komplexere Sicherheitsmethoden die sich Verschlüsslungsalgorithmen bedienen auf schwächeren Endgeräten nicht implementierbar. Hier bleibt abzuwarten, ob dieses Problem in Zukunft hardwareseitig oder durch neue Sicherheitsfunktionen direkt auf Protokollebene gelöst wird.

8 Literaturverzeichnis

[1] ITU-T, “ITU-T Rec. Y.2060 (06/2012) Overview of the Internet of things,” https://www.itu.int/rec/T-REC-Y.2060-201206-I, 2012.

[2] *The Internet of Things | the internet of things.* Available: http://www.theinternetofthings.eu/what-is-the-internet-of-things (2016, Feb. 14).

[3] *Gartner Says 6.4 Billion Connected.* Available: http://www.gartner.com/newsroom/id/3165317 (2016, Feb. 16).

[4] G. Lawton, “Developing Software Online With Platform-as-a-Service Technology,” *Computer*, vol. 41, no. 6, pp. 13–15, http://dx.doi.org/10.1109/MC.2008.185, 2008.

[5] G. Raines and L. Pizette, “Systems Engineering at MITRE - Cloud Computing Series: Platform as a Service: a 2010 Marketplace Analysis,” 2010.

[6] *BSI - Cloud Computing Grundlagen.* Available: https://www.bsi.bund.de/DE/Themen/DigitaleGesellschaft/CloudComputing/Grundlagen/Grundlagen_node.html (2016, Mar. 04).

[7] r. v. 1. 118, Leach, P. J, Berners-Lee, Tim, Mogul, J. C, Masinter, Larry, Fielding, R. T, Gettys, and James, *Hypertext Transfer Protocol -- HTTP/1.1.* Available: https://tools.ietf.org/html/rfc2616 (2016, Apr. 22).

[8] *MQTT Version 3.1.1.* Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (2016, Apr. 22).

[9] *What is MQTT (MQ Telemetry Transport)? - Definition from WhatIs.com.* Available: http://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport (2016, Apr. 22).

[10] Normenausschuss Ergonomie, “DIN EN ISO 9241-110,” http://perinorm-fr.redi-bw.de/volltexte/CD21DE04/1464024/1464024.pdf, 2012.

[11] *Internet Of Things - ThingSpeak.* Available: https://thingspeak.com/ (2016, Mar. 04).

[12] *Getting Started with ThingSpeak - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/getting-started-with-thingspeak.html (2016, Feb. 21).

[13] *Forum.* Available: http://community.thingspeak.com/forum/ (2016, Apr. 24).

[14] *Channel Configurations - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/channel-settings.html (2016, Feb. 21).

[15] *ThingTweet App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/thingtweet-app.html (2016, Mar. 05).

[16] *TweetControl App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/tweetcontrol-app.html (2016, Mar. 05).

[17] *TalkBack App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/talkback-app.html (2016, Mar. 05).

[18] *ThingHTTP App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/thinghttp-app.html (2016, Mar. 05).

[19] *MATLAB Visualizations App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/matlab-visualizations-app.html (2016, Mar. 05).

[20] *TimeControl App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/timecontrol-app.html (2016, Mar. 05).

[21] *Plugins App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/plugins-app.html (2016, Mar. 05).

[22] *Channels and Charts API - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/channels-and-charts.html (2016, Mar. 05).
[23] *Update a Channel Feed - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/update-channel-feed.html (2016, Mar. 05).
[24] *Channels - ThingSpeak.* Available: https://thingspeak.com/channels (2016, Feb. 21).
[25] *Apps - ThingSpeak.* Available: https://thingspeak.com/apps/matlab_visualizations (2016, Apr. 23).
[26] *MATLAB Plot Gallery - MathWorks Deutschland.* Available: http://de.mathworks.com/products/matlab/plot-gallery.html#standardplots (2016, Apr. 22).
[27] *React App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/react-app.html (2016, Mar. 05).
[28] *MATLAB Analysis App - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/matlab-analysis-app.html (2016, Apr. 23).
[29] *ThingSpeak Documentation - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/ (2016, Feb. 21).
[30] *Channels - ThingSpeak.* Available: https://thingspeak.com/channels/new (2016, Mar. 03).
[31] *Channel Configurations - MATLAB & Simulink - MathWorks Deutschland.* Available: https://de.mathworks.com/help/thingspeak/channel-settings.html (2016, Feb. 21).
[32] *data.sparkfun.com.* Available: https://data.sparkfun.com/ (2016, Feb. 26).
[33] *sparkfun/phant.* Available: https://github.com/sparkfun/phant (2016, Mar. 06).
[34] *HTTP.* Available: http://phant.io/docs/output/http/ (2016, Feb. 26).
[35] *Documentation.* Available: http://phant.io/docs/ (2016, Feb. 26).
[36] *HTTP.* Available: http://phant.io/docs/input/http/ (2016, Feb. 26).
[37] *data.sparkfun.com - New Stream.* Available: https://data.sparkfun.com/streams/make (2016, Feb. 26).
[38] *IoT PaaS TimeSeries - RunAbove.* Available: https://www.runabove.com/iot-paas-timeseries.xml (2016, Mar. 01).
[39] *All Topics | All Topics | RunAbove community - feedback.* Available: https://community.runabove.com/share/topics/all/status/all/category/iot/sort/updated/page/1 (2016, Apr. 24).
[40] *How to get data from the RunAbove IoT metrics storage - RunAbove knowledge base.* Available: https://community.runabove.com/kb/en/iot/how-to-get-data-from-runabove-iot.html (2016, Mar. 01).
[41] *runabove/iot-push-examples.* Available: https://github.com/runabove/iot-push-examples (2016, Mar. 07).
[42] *How to push data to RunAbove IoT Lab - RunAbove knowledge base.* Available: https://community.runabove.com/kb/en/iot/how-to-push-data-to-runabove-iot.html (2016, Mar. 01).
[43] *iot - RunAbove knowledge base.* Available: https://community.runabove.com/kb/en/iot/ (2016, Mar. 01).
[44] Available: https://cloud.runabove.com/#/iot (2016, Mar. 01).
[45] *flowthings.io - Agile Intelligence for all your things.* Available: https://flowthings.io/ (2016, Mar. 01).

[46] *flowthings.io Support.* Available: https://flowthings.io/support#!/ (2016, Apr. 24).
[47] *Index.* Available: https://flowthings.io/docs/index (2016, Mar. 01).
[48] *Libraries.* Available: https://flowthings.io/developers/libraries (2016, Mar. 01).
[49] *Getting Started.* Available: https://flowthings.io/docs/getting-started (2016, Mar. 01).
[50] *Drop Object Overview.* Available: https://flowthings.io/docs/drop-object-overview (2016, Mar. 07).
[51] *Http Drop Create.* Available: https://flowthings.io/docs/http-drop-create (2016, Mar. 07).
[52] *flowthings.io Developers.* Available: https://dev.flowthings.io/#/ (2016, Mar. 01).
[53] *Example Tracks In Depth.* Available: https://flowthings.io/docs/example-tracks-in-depth (2016, Apr. 23).
[54] *Track Gui.* Available: https://flowthings.io/docs/track-gui (2016, Apr. 23).
[55] *Flow Javascript Processing Language.* Available: https://flowthings.io/docs/flow-javascript-processing-language (2016, Apr. 23).
[56] *Carriots - Internet of Things Platform | Home.* Available: https://www.carriots.com/ (2016, Mar. 04).
[57] *Carriots - Internet of Things Platform | what is carriots.* Available: https://www.carriots.com/what-is-carriots (2016, Mar. 01).
[58] *Carriots Forum.* Available: http://forum.carriots.com/ (2016, Apr. 24).
[59] *Carriots - Internet of Things Platform | pricing.* Available: https://www.carriots.com/pricing (2016, Mar. 02).
[60] *Carriots - Internet of Things Platform | documentation | sdk | carriots sdk.* Available: https://www.carriots.com/documentation/sdk/carriots_sdk (2016, Mar. 08).
[61] *Carriots - Internet of Things Platform | documentation | external apps.* Available: https://www.carriots.com/documentation/external_apps (2016, Mar. 02).
[62] *Carriots - Internet of Things Platform | documentation.* Available: https://www.carriots.com/documentation (2016, Mar. 02).
[63] *Carriots - Internet of Things Platform | documentation | api.* Available: https://www.carriots.com/documentation/api (2016, Mar. 02).
[64] *Carriots - Internet of Things Platform | documentation | api | data management.* Available: https://www.carriots.com/documentation/api/data_management#streams (2016, Mar. 08).
[65] *Carriots - Internet of Things Platform | documentation | graphs.* Available: https://www.carriots.com/documentation/graphs (2016, Apr. 22).
[66] *Carriots.* Available: https://cpanel.carriots.com/ (2016, Mar. 02).
[67] Alvaro Everlet, “CARRIOTS TECHNICAL PRESENTATION,” https://www.carriots.com/newFrontend/img-carriots/press_room/CARRIOTS_technical_presentation.pdf.
[68] *Internet of Things application development platform.* Available: http://ubidots.com/ (2016, Mar. 03).
[69] *Overview — Documentation Ubidots 1.2 documentation.* Available: http://ubidots.com/docs/get_started/overview.html (2016, Mar. 09).
[70] *Ubidots Community.* Available: http://community.ubidots.com/ (2016, Apr. 24).
[71] *Our cloud services plans - Ubidots.com.* Available: http://ubidots.com/pricing.html#price_details (2016, Mar. 03).

[72] *Libraries — Documentation Ubidots 1.2 documentation.* Available: http://ubidots.com/docs/libraries/index.html (2016, Mar. 03).
[73] *API Reference — Documentation Ubidots 1.2 documentation.* Available: http://ubidots.com/docs/api/index.html (2016, Mar. 03).
[74] *POST api/v1.6/collections/values — Documentation Ubidots 1.2 documentation.* Available: http://ubidots.com/docs/api/v1_6/collections/post_values.html#post-api-v1-6-collections-values (2016, Mar. 09).
[75] *Ubidots | Dashboard.* Available: https://app.ubidots.com/ubi/insights/#/list (2016, Apr. 22).
[76] *Ubidots | Events.* Available: https://app.ubidots.com/ubi/events/#/create (2016, Apr. 23).
[77] *Overview — Documentation Ubidots 1.2 documentation.* Available: http://ubidots.com/docs/index.html (2016, Mar. 03).
[78] *Ubidots | Data.* Available: https://app.ubidots.com/ubi/datasources/#/detail/56d84f16762542241426231a (2016, Mar. 03).
[79] *A Powerful API for Sensors - Ubidots.com.* Available: http://ubidots.com/features-powerful-api.html (2016, Mar. 03).
[80] *Welcome - Storage and Analytics for the Internet of Things.* Available: https://grovestreams.com/index.html (2016, Mar. 03).
[81] *Welcome to Grove Streams - Storage and Analytics for the Internet of Things - GroveStreams.* Available: https://grovestreams.com/aboutGS3.html (2016, Mar. 09).
[82] *GroveStreams.* Available: http://forum.grovestreams.com/ (2016, Apr. 24).
[83] *Pricing.* Available: https://grovestreams.com/pricing.html (2016, Mar. 03).
[84] *API Limits.* Available: https://grovestreams.com/developers/limits.html (2016, Mar. 03).
[85] *Developers.* Available: https://grovestreams.com/developers/developers.html (2016, Mar. 03).
[86] *REST API.* Available: https://grovestreams.com/developers/api.html (2016, Mar. 03).
[87] *REST API.* Available: https://grovestreams.com/developers/api_adv.html (2016, Mar. 09).
[88] *Observation Studio - GroveStreams.* Available: https://www.grovestreams.com/observationStudio.html?org=abe6e1b2-3e64-386a-befe-633f2e48ab6c (2016, Mar. 03).
[89] *REST API Conventions.* Available: https://grovestreams.com/developers/rest_api_conventions.html (2016, Mar. 03).
[90] *Internet of Things Platform for Connected Devices | Exosite.* Available: https://exosite.com/ (2016, Mar. 03).
[91] *Exosite Community.* Available: https://community.exosite.com/ (2016, Apr. 24).
[92] *Solutions - Thesis - Exosite Business Platform.* Available: https://exosite.com/business/solutions/i9kfn6g5xrfnu3di (2016, Mar. 03).
[93] *Exosite APIs.* Available: http://docs.exosite.com/http/ (2016, Mar. 10).
[94] *Data Limits and Retention.* Available: https://support.exosite.com/hc/en-us/articles/200845344-Data-Limits-and-Retention (2016, Mar. 03).
[95] Exosite, *Exosite Labs Github Page.* Available: https://exosite-labs.github.io/ (2016, Mar. 03).

[96] *Exosite Portals.* Available: https://portals.exosite.com/views/2809599258/2289502011 (2016, Mar. 03).
[97] *exosite/docs.* Available: https://github.com/exosite/docs/tree/master/widget (2016, Apr. 23).
[98] *Exosite Portals.* Available: https://portals.exosite.com/manage/events (2016, Apr. 23).
[99] *Exosite APIs.* Available: http://docs.exosite.com/scripting/ (2016, Apr. 23).
[100] *Exosite APIs.* Available: http://docs.exosite.com/ (2016, Mar. 03).
[101] *IoT Security Solutions Architecture | Exosite.* Available: https://exosite.com/architecture/security/ (2016, Mar. 03).
[102] *Beebotte.* Available: https://beebotte.com/ (2016, Mar. 03).
[103] *Beebotte.* Available: https://beebotte.com/plans (2016, Mar. 03).
[104] *Beebotte.* Available: https://beebotte.com/libs (2016, Mar. 03).
[105] *Beebotte v1 API API Documentation.* Available: https://beebotte.com/api/play (2016, Mar. 03).
[106] *Beebotte - mkarow.* Available: https://beebotte.com/mkarow/dashboards/create (2016, Apr. 23).
[107] *Beebotte - mkarow.* Available: https://beebotte.com/docs/dash#widgets (2016, Apr. 22).
[108] *Beebotte.* Available: https://beebotte.com/overview (2016, Mar. 03).
[109] *Beebotte - mkarow.* Available: https://beebotte.com/channel/create (2016, Mar. 03).
[110] *MODE - Cloud platform for IoT products.* Available: http://www.tinkermode.com/ (2016, Mar. 03).
[111] *MODE Community.* Available: https://community.tinkermode.com/ (2016, Apr. 24).
[112] *Pricing - MODE.* Available: http://www.tinkermode.com/pricing.html (2016, Mar. 03).
[113] *SDKs for Hardware Platforms.* Available: http://dev.tinkermode.com/docs/sdks_hardware.html (2016, Mar. 03).
[114] *SDKs for Mobile Platforms.* Available: http://dev.tinkermode.com/docs/sdks_mobile.html (2016, Mar. 03).
[115] *API Reference.* Available: http://dev.tinkermode.com/docs/api/ (2016, Mar. 03).
[116] *Data Models - API Reference.* Available: http://dev.tinkermode.com/docs/api/models.html#event (2016, Mar. 12).
[117] *Developer Console - MODE.* Available: https://console.tinkermode.com/console/projects/464 (2016, Mar. 03).
[118] *Features: Connectivity - MODE.* Available: http://www.tinkermode.com/features.html (2016, Mar. 03).
[119] *Initial State - Data Analytics for the Internet of Things.* Available: https://www.initialstate.com/ (2016, Mar. 03).
[120] Apiary, *Initial State Events Api · Apiary.* Available: http://docs.initialstateeventsapi.apiary.io/#reference/event-buckets (2016, Mar. 12).
[121] *Initial State Support Documentation.* Available: http://support.initialstate.com/ (2016, Mar. 03).
[122] *API: Rate and Bandwidth Limitations for Non-Enterprise Accounts – Initial State Support Documentation.* Available:

http://support.initialstate.com/knowledgebase/articles/803922-api-rate-and-bandwidth-limitations-for-non-enterp (2016, Mar. 12).
[123] *Initial State Works With.* Available: https://www.initialstate.com/workswith (2016, Mar. 03).
[124] Apiary, *Initial State Events Api · Apiary.* Available: http://docs.initialstateeventsapi.apiary.io/# (2016, Mar. 03).
[125] Apiary, *Initial State Events Api · Apiary.* Available: http://docs.initialstateeventsapi.apiary.io/#reference/event-data/events-no-json (2016, Mar. 12).
[126] *Tiles: Data View Options – Initial State Support Documentation.* Available: http://support.initialstate.com/knowledgebase/articles/670060-tiles-data-view-options (2016, Apr. 22).
[127] *Example Import.* Available: https://app.initialstate.com/#/tiles (2016, Apr. 22).
[128] *Waves: How Different Types of Data are Drawn – Initial State Support Documentation.* Available: http://support.initialstate.com/knowledgebase/articles/504317-waves-how-different-types-of-data-are-drawn (2016, Apr. 22).
[129] *Example Import.* Available: https://app.initialstate.com/#/waves (2016, Apr. 22).
[130] *Lines: Introduction – Initial State Support Documentation.* Available: http://support.initialstate.com/knowledgebase/articles/590178-lines-introduction (2016, Apr. 22).
[131] *Example Import.* Available: https://app.initialstate.com/#/lines (2016, Apr. 22).
[132] *Example Import.* Available: https://app.initialstate.com/#/source (2016, Apr. 23).
[133] *Example Import.* Available: https://www.initialstate.com/app#/ (2016, Mar. 03).
[134] *Introduction to Data Streaming – Initial State Support Documentation.* Available: http://support.initialstate.com/knowledgebase/articles/590100-introduction-to-data-streaming (2016, Mar. 03).
[135] *Temboo.* Available: https://temboo.com/ (2016, Mar. 03).
[136] *Pricing :: Temboo.* Available: https://temboo.com/pricing (2016, Mar. 03).
[137] *Library :: Temboo.* Available: https://temboo.com/library/ (2016, Mar. 03).
[138] *REST API :: Temboo.* Available: https://temboo.com/restapi/reference (2016, Mar. 03).
[139] *Library :: Temboo.* Available: https://temboo.com/library/ (2016, Mar. 03).
[140] *Trust :: Temboo.* Available: https://temboo.com/trust (2016, Mar. 03).
[141] Oracle, “Accelerate Innovation: With the Power of the Internet of Things,” 2015.
[142] *Internet of Things | Oracle Cloud.* Available: https://cloud.oracle.com/sites/Satellite?c=Page&cid=1448102143252&pagename=cloudUiSite%2FPage%2FwcsServiceLayout&siteid=de_DE (2016, Mar. 01).
[143] *Preise | Internet of Things | Oracle Cloud.* Available: https://cloud.oracle.com/sites/Satellite?c=Page&cid=1448102143252&pagename=cloudUiSite%2FPage%2FwcsServiceLayout&tabID=1385151694899&siteid=de_DE (2016, Mar. 01).
[144] *Oracle Internet of Things Cloud Service.* Available: http://docs.oracle.com/cloud/latest/iot/apidocs.htm (2016, Mar. 01).
[145] *REST API for Oracle Internet of Things Cloud Service.* Available: http://docs.oracle.com/cloud/latest/iot/IOTRP/index.html (2016, Mar. 13).

[146] *REST API for Oracle Internet of Things Cloud Service.* Available: http://docs.oracle.com/cloud/latest/iot/IOTRP/op-iot-api-v1-messages-post.html (2016, Mar. 13).
[147] *Data Visualization Service | Oracle Cloud.* Available: https://cloud.oracle.com/data_visualization# (2016, Apr. 22).
[148] *Managing Device Data Messages.* Available: http://docs.oracle.com/cloud/latest/iot/IOTGS/GUID-783E9F8E-8D90-42F2-A1AE-52894DF244D8.htm#IOTGS-GUID-783E9F8E-8D90-42F2-A1AE-52894DF244D8 (2016, Apr. 22).
[149] *Oracle Internet of Things Cloud Service.* Available: http://docs.oracle.com/cloud/latest/iot/index.html (2016, Mar. 01).
[150] *Accessing Oracle Internet of Things Cloud Service.* Available: https://docs.oracle.com/cloud/latest/iot/IOTGS/GUID-2D512408-B17E-46B7-B079-70C1B3DDD211.htm#IOTGS-GUID-2D512408-B17E-46B7-B079-70C1B3DDD211 (2016, Mar. 13).
[151] *REST API for Oracle Internet of Things Cloud Service.* Available: http://docs.oracle.com/cloud/latest/iot/IOTRP/Authentication.html (2016, Mar. 01).
[152] *IBM Watson Internet of Things (IoT).* Available: https://www.ibm.com/internet-of-things/ (2016, Apr. 22).
[153] *IBM - Explore IoT.* Available: http://discover-iot.eu-gb.mybluemix.net/#/play (2016, Apr. 22).
[154] *Questions in Internet of Things space - Internet of Things · dWAnswers.* Available: https://developer.ibm.com/answers/smartspace/internet-of-things/ (2016, Apr. 24).
[155] *Internet of Things Platform - IBM Bluemix.* Available: https://new-console.eu-gb.bluemix.net/catalog/services/internet-of-things-platform/ (2016, Mar. 01).
[156] *Programmierhandbücher — IBM IOT Foundation 1.0 Dokumentation.* Available: https://docs.internetofthings.ibmcloud.com/de/libraries/programmingguides.html (2016, Mar. 01).
[157] *Swagger UI.* Available: https://docs.internetofthings.ibmcloud.com/swagger/v0002.html (2016, Mar. 01).
[158] *Katalog - IBM Bluemix.* Available: https://console.eu-gb.bluemix.net/catalog/ (2016, Apr. 22).
[159] *Einführung — IBM IOT Foundation 1.0 Dokumentation.* Available: https://docs.internetofthings.ibmcloud.com/de/index.html (2016, Mar. 01).
[160] *IBM Internet of Things Foundation.* Available: https://internetofthings.ibmcloud.com/#/ (2016, Mar. 01).
[161] *Azure IoT Hub | Microsoft Azure.* Available: https://azure.microsoft.com/de-de/services/iot-hub/ (2016, Mar. 04).
[162] *Übersicht über Azure IoT Hub | Microsoft Azure.* Available: https://azure.microsoft.com/de-de/documentation/articles/iot-hub-what-is-iot-hub/ (2016, Feb. 27).
[163] *Technet forums.* Available: https://social.technet.microsoft.com/Forums/en-us/home (2016, Apr. 24).
[164] *Preise – IoT Hub | Microsoft Azure.* Available: https://azure.microsoft.com/de-de/pricing/details/iot-hub/ (2016, Feb. 27).

[165] *Microsoft Azure-Abonnements und Diensteinschränkungen, Kontingente und Einschränkungen.* Available: https://azure.microsoft.com/de-de/documentation/articles/azure-subscription-service-limits/#iot-hub-limits (2016, Feb. 27).
[166] *Microsoft Azure IoT Device SDKs.* Available: http://azure.github.io/azure-iot-sdks/ (2016, Feb. 27).
[167] *IoT-Hub REST.* Available: https://msdn.microsoft.com/library/mt548492.aspx (2016, Feb. 27).
[168] *Power BI | Interaktive BI-Tools für die Datenvisualisierung.* Available: https://powerbi.microsoft.com/de-de/ (2016, Apr. 22).
[169] *Visualisierungstypen in Power BI | Microsoft Power BI.* Available: https://powerbi.microsoft.com/de-de/documentation/powerbi-service-visualization-types-for-reports-and-q-and-a/ (2016, Apr. 22).
[170] *DZone.* Available: https://dzone.com/articles/exploring-the-microsoft-azure-iot-hub (2016, Apr. 22).
[171] *IoT Hub Documentation | Azure.* Available: https://azure.microsoft.com/de-de/documentation/services/iot-hub/ (2016, Feb. 27).
[172] *Microsoft Azure IoT.* Available: https://portal.azure.com/#create/Microsoft.IotHub (2016, Feb. 27).
[173] *Entwicklungsleitfaden für IoT Hub | Microsoft Azure.* Available: https://azure.microsoft.com/de-de/documentation/articles/iot-hub-devguide/#security (2016, Feb. 27).
[174] *AWS IoT – Amazon Web Services.* Available: https://aws.amazon.com/de/iot/ (2016, Feb. 26).
[175] *AWS Developer Forums: AWS IoT.* Available: https://forums.aws.amazon.com/forum.jspa?forumID=210 (2016, Apr. 24).
[176] *AWS IoT-Preise – Amazon Web Services.* Available: https://aws.amazon.com/de/iot/pricing/ (2016, Feb. 26).
[177] *AWS IoT Limits - AWS IoT.* Available: http://docs.aws.amazon.com/iot/latest/developerguide/iot-limits.html (2016, Feb. 26).
[178] *AWS IoT Device SDK – Amazon Web Services.* Available: https://aws.amazon.com/de/iot/sdk/ (2016, Mar. 14).
[179] *What Is AWS IoT? - AWS IoT.* Available: http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html (2016, Mar. 14).
[180] *Welcome - AWS IoT.* Available: http://docs.aws.amazon.com/iot/latest/apireference/Welcome.html (2016, Feb. 27).
[181] *AWS Real Time Analytics & Echtzeitdaten | Amazon Kinesis.* Available: https://aws.amazon.com/de/kinesis/?hp=tile (2016, Apr. 22).
[182] *AWS IoT Documentation.* Available: http://aws.amazon.com/de/documentation/iot/ (2016, Feb. 27).
[183] *AWS IoT.* Available: https://eu-west-1.console.aws.amazon.com/iot/home?region=eu-west-1#/dashboard?editor=thing (2016, Feb. 27).

[184] *Security and Identity for AWS IoT - AWS IoT.* Available: http://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html (2016, Feb. 26).
[185] Google, *Internet of Things.* Available: https://cloud.google.com/solutions/iot/ (2016, Mar. 04).
[186] Google, *Architecture: Real Time Stream Processing - Internet of Things.* Available: https://cloud.google.com/solutions/architecture/streamprocessing (2016, Feb. 26).
[187] Google, *What is Google Cloud Pub/Sub?* Available: https://cloud.google.com/pubsub/overview (2016, Mar. 04).
[188] *(61) Google Cloud Pub/Sub Discussions - Google Groups.* Available: https://groups.google.com/forum/#!forum/cloud-pubsub-discuss (2016, Apr. 24).
[189] Google, *Pricing.* Available: https://cloud.google.com/pubsub/pricing (2016, Feb. 26).
[190] Google, *Google Cloud Platform Pricing Calculator.* Available: https://cloud.google.com/products/calculator/ (2016, Feb. 26).
[191] Google, *Quotas.* Available: https://cloud.google.com/pubsub/quotas (2016, Mar. 14).
[192] Google, *Samples and Libraries.* Available: https://cloud.google.com/pubsub/libraries (2016, Mar. 04).
[193] Google, *Collection projects.topics.* Available: https://cloud.google.com/pubsub/reference/rest/v1/projects.topics (2016, Mar. 14).
[194] Google, *Method projects.topics.publish.* Available: https://cloud.google.com/pubsub/reference/rest/v1/projects.topics/publish (2016, Mar. 14).
[195] Google, *PubsubMessage.* Available: https://cloud.google.com/pubsub/reference/rest/v1/PubsubMessage (2016, Mar. 14).
[196] Google, *Publisher Guide.* Available: https://cloud.google.com/pubsub/publisher (2016, Mar. 14).
[197] Google, *Stackdriver Monitoring - Hybrid Cloud Monitoring.* Available: https://cloud.google.com/monitoring/ (2016, Apr. 22).
[198] Google, *Google Cloud Platform Documentation.* Available: https://cloud.google.com/docs/ (2016, Mar. 04).
[199] *Home - Thesis.* Available: https://console.cloud.google.com/home/dashboard?project=thesis-1240 (2016, Mar. 04).
[200] Google, *Security and Compliance on the Google Cloud Platform.* Available: https://cloud.google.com/security/ (2016, Mar. 14).
[201] *AWS Developer Forums: Lack of Arduino / ESP8266 / 8-bit …* Available: https://forums.aws.amazon.com/thread.jspa?threadID=225644&tstart=25 (2016, Apr. 24).
[202] *Apps - ThingSpeak.* Available: https://thingspeak.com/apps (2016, Apr. 25).
[203] *Apps - ThingSpeak.* Available: https://thingspeak.com/apps/reacts (2016, Apr. 25).
[204] *PushingBox - Notifications for your Internet of Things devices.* Available: https://www.pushingbox.com/scenarios.php (2016, Apr. 25).

9 Anhang

9.1 Tabelle Plattformvergleich

Plattform	3.2.1 ThingSpeak	3.2.2 data.sparkfun.com	3.2.3 Runabove IoT Lab
Verbreitung	Forum: 803 Themen/2.859 Posts Stack Overflow: 60 Treffer Google: ca. 261.000 Treffer	Stack Overflow: 19 Treffer Google: ca. 13.500 Treffer	Forum: 15 Themen/18 Posts Stack Overflow: 0 Treffer Google: ca. 60 Treffer
Preis	kostenlos	kostenlos	kostenlos
Verbindungsrate	1 Stream/15 Sek.	100 Streams/15 Min.	-
Datenmenge	-	50MB Speicherplatz	-
Schnittstellen	1 externe Schnittstellen (HTTP) 8 Apps	1 externe Schnittstelle (HTTP)	1 externe Schnittstelle (HTTP)
API	Datenübertragung Datenverwaltung Visualisierung	Datenübertragung Datenverwaltung	Datenübertragung Datenverwaltung Devicemanagement Authentifizierung
Visualisierung	Ausführliche Visualisierungsmöchlichkeiten mit allen gängigen Diagrammtypen, individuell konfigurierbar	Tabellarische Darstellung	-
Weiterverarbeitung	Plattformeigene Apps begrenzter Funktionsumfang einfache Bedienbarkeit	-	-
Dokumentation	Übersichtlich & klar strukturiert Geringer Umfang Illustrationen, Codebeispiele & Tutorials	Übersichtlich & klar strukturiert Geringer Umfang Codebeispiele	Schlechte Struktur Sehr geringer Umfang Illustrationen, Codebeispiele, Tutorials
Usability	Sehr einfache und leicht verständliche Bedienung	Sehr einfache und leicht verständliche Bedienung	Einfache und verständliche Bedienung
Sicherheit	TLS Authentifizierungstokens	TLS Authentifizierungstokens	TLS Authentifizierungstokens

Plattform	3.2.4 flowthings.io	3.2.5 Carriots	3.2.6 Ubidots
Verbreitung	Forum: 15 Themen/58 Posts Stack Overflow: 0 Treffer Google: ca. 2.130 Treffer	Forum: 240 Themen/985 Posts Stack Overflow: 5 Treffer Google: ca. 17.300 Treffer	Forum: 139 Themen/615 Posts Stack Overflow: 10 Treffer Google: ca. 15.200 Treffer
Preis	kostenlos	Free: kostenlos Corporate: 2€/Monat/Device Lite: 0,50€/Monat/Device	Free: kostenlos Individual: 12$/Monat Startup: 48$/Monat Professional: 120$/Monat Business: 360$/Monat Enterprise: auf Anfrage
Verbindungsrate	-	Free: 500 Streams/Tag, 10 Streams/Min. Corporate: 1.500*[Anz. Devices] Streams/Tag, 50*[Anz. Devices] Streams/Min. Lite: 25*[Anz. Devices] Streams/Tag, 5*[Anz. Devices] Streams/Min.	Free/Individual/Startup: 0,5 Mio. Streams/Monat Professional/Business/Enterprise: 1 Mio. Streams/Monat
Datenmenge	100.000 Datenpunkte	Free: 5KB/Stream, 5000KB Speicherplatz/Tag Corporate: 10KB/Stream, 1MB*[Anz. Devices] Speicherplatz/Tag Lite: 5KB/Stream, 100KB*[Anz. Devices] Speicherplatz/Tag	-
Schnittstellen	4 externe Schnittstellen (HTTP, WebSockets, Webhooks, MQTT) 8 Libraries	4 externe Schnittstellen (HTTP, MQTT, Twitter, Dropbox) 1 SDK	1 externe Schnittstelle (HTTP) 7 Libraries
API	Datenübertragung Datenverwaltung Benutzerverwaltung Devicemanagement Authentifizierung	Datenübertragung Datenverwaltung Devicemanagement Eventmanagement	Datenübertragung Datenverwaltung Devicemanagement Authentifizierung
Visualisierung	Kurvendiagramm Liniendiagramm Stufendiagramm	Gängige Diagrammtypen	Gängige Diagrammtypen Metrische Werte Karten Tabellarische Ansicht Binäre Anzeige Messanzeige
Weiterverarbeitung	Einfache Verarbeitung über visuellen Flow-Editor Wenig Schnittstellen nach Außen Eigene Funktionen mit JavaScript	Ausführliche Möglichkeiten zur Weiterverarbeitung über Bedingung -> Aktion Modell Eigene SDK in Groovy	Einfache Events begrenzter Funktionsumfang einfache Bedienbarkeit
Dokumentation	Übersichtlich & gut strukturiert Großer Umfang Codebeispiele	Übersichtlich & klar strukturiert Umfangreich Illustrationen, Codebeispiele & API Konsole	Übersichtlich & gut strukturiert Großer Umfang Illustrationen, Codebeispiele & Tutorials
Usability	Einfache Bedienung, einige Funktionalitäten schwer erreichbar Anpassbare Benutzeroberfläche durch Dashboard	Einfache und leicht verständliche Bedienung, viele Hilfestellungen, vorkonfigurierte Optionen	Sehr einfache und leicht verständliche Bedienung, gute Dialogführung, vorkonfigurierte Optionen Anpassbare Benutzeroberfläche durch Widgets
Sicherheit	TLS Authentifizierungstokens	TLS Authentifizierungstokens Hash-Funktion	TLS Authentifizierungstokens

Plattform	3.2.7 GroveStreams	3.2.8 Exosite	3.2.9 Beebotte
Verbreitung	Forum: 377 Themen/2.148 Posts Stack Overflow: 0 Treffer Google: ca. 3.340 Treffer	Forum: 121 Themen/383 Posts Stack Overflow: 7 Treffer Google: ca. 170.000 Treffer	Stack Overflow: 1 Treffer Google: ca. 2.440 Treffer
Preis	Beginner: kostenlos Personal: 5$/Monat Essentials: 95$/Monat Pro: 495$/Monat	Free: kostenlos Bronze: 100$/Monat Silver: 1000$/Monat Custom: auf Anfrage	XS: kostenlos Small: 10$/Monat Medium: 30$/Monat Large: 120$/Monat
Verbindungsrate	150 Streams/Std. (unauthentifiziert) 350 Streams/Std. (authentifiziert) max. 1 Stream/10 Sek.	1 Stream/Sek.	Normale Streams: XS: 50.000 Streams/Tag Small: 200.000 Streams/Tag Medium: 1 Mio. Streams/Tag Large: 5 Mio. Streams/Tag Persistene Streams: XS: 5.000 Streams/Tag Small: 15.000 Streams/Tag Medium: 50.000 Streams/Tag Large: 200.000 Streams/Tag
Datenmenge	max. 200MB/Stream Beginner: 5MB/Stream/Monat Personal: 20MB/Stream/Monat Essentials: 200MB/Stream/Monat Pro: 1GB/Stream/Monat	max. 10MB Speicherplatz/Device String: 64KB/Stream Integer: 64Bit/Stream Binär: 64KB/Stream Float: 8Exponentenbits+24Bit Mantisse	16KB/Stream
Schnittstellen	1 externe Schnittstelle (HTTP)	4 externe Schnittstellen (HTTP, CoAP, RPC, WebSockets) 9 Libraries	3 externe Schnittstellen (HTTP, WebSockets, MQTT) 5 Libraries
API	Datenübertragung Datenverwaltung Benutzerverwaltung Eventverwaltung Authentifizierung Plattformverwaltung	Datenübertragung Datenverwaltung Devicemanagement Benutzerverwaltung Plattformverwaltung Authentifizierung	Datenübertragung Datenverwaltung Devicemanagement
Visualisierung	Gängige Diagrammtypen Tabellarische Ansicht Karten Binäre Anzeige Messanzeige	Gängige Diagrammtypen Einzelne Werte Messanzeige Tabellarische Ansicht Karten eigene Visualisierungen über jQuery API	Einzelne Werte Messanzeige Liniendiagramm Tabellarische Ansicht Heatmap
Weiterverarbeitung	Möglichkeiten zur Weiterverarbeitung über Bedingung -> Aktion Modell Wenig Schnittstellen nach außen	Einfache Events zur Benachrichtigung per E-Mail Komplexe Aktionen über eigene Lua-Scripts	-
Dokumentation	Wenig Übersicht, schlechte Struktur Geringer Umfang Illustrationen, Codebeispiele & Tutorials	Übersichtlich & klar strukturiert Großer Umfang Illustrationen, Codebeispiele & Tutorials	Übersichtlich & gut strukturiert Großer Umfang Illustrationen, Codebeispiele, Tutorials & API Konsole
Usability	Inkonsistente Bedienung, einige Funktionalitäten schwer erreichbar Anbassbare Benutzeroberfläche durch Dashboards	Einfache und leicht verständliche Bedienung, viele Hilfestellungen, vorkonfigurierte Optionen Anpassbare Benutzeroberfläche durch Widgets	Einfache und leicht verständliche Bedienung, viele Hilfestellungen, vorkonfigurierte Optionen Anpassbare Benutzeroberfläche durch Dashboards
Sicherheit	TLS Authentifizierungstokens	TLS Authentifizierungstokens	TLS Authentifizierungstokens AES-Encryption

Plattform	3.2.10 MODE	3.2.11 Initial State	3.2.12 Temboo
Verbreitung	Forum: 21 Themen/55 Posts Stack Overflow: 1 Treffer Google: ca. 2.150 Treffer	Forum: 45 Themen/60 Posts Stack Overflow: 1 Treffer Google: ca. 3.230 Treffer	Stack Overflow: 148 Treffer Google: ca. 47.500 Treffer
Preis	Prototyper: kostenlos Enterprise: auf Anfrage	Free: kostenlos Standard: 5$/Monat Pro: 25$/Monat	Free: kostenlos Individual: 7$/Monat Enterprise: ab 40$/Monat
Verbindungsrate	-	5 Streams/Sek. Free: 25.000 Streams/Monat Standard/Pro: keine Begrenzung	Free: 250 Streams/Monat Individual: 25.000 Streams/Monat Enterprise: 100.000 Streams/Monat
Datenmenge	-	1MB/Stream	Free: 1GB/Monat Individual: 1GB/Monat Enterprise: 16GB/Monat
Schnittstellen	2 externe Schnittstellen (HTTP, WebSockets) 7 SDKs	1 externe Schnittstelle (HTTP) 1 Library 3 SDKs	1 externe Schnittstelle (HTTP) 10 SKDs 100+ externe Services & Plattformen
API	Datenübertragung Datenverwaltung Benutzerverwaltung Devicemanagement Authentifizierung	Datenübertragung Datenverwaltung	Datenübertragung Datenverwaltung Eventverwaltung
Visualisierung	-	Umfangreiche Visualisierungsmöglichkeiten in verschiedenen Views Einzelne Werte Gängige Diagrammtypen Messanzeige Histogramm Karten Hochanpassbare Wave- und Liniendiagramme in Echtzeit	-
Weiterverarbeitung	-	-	-
Dokumentation	Übersichtlich & klar strukturiert Großer Umfang Illustrationen, Codebeispiele & Tutorials	Unübersichtlich, wenig Struktur Großer Umfang Illustrationen, Codebeispiele & Tutorials	Dokumentation aufgesplittert auf Einzelmodule der Plattform, dadurch unübersichtlich & schlecht strukturiert Sehr Umfangreich Codebeispiele & Tutorials
Usability	Einfache und leicht verständliche Bedienung, viele Hilfestellungen	Einfache und verständliche Bedienung, viele Hilfestellungen Anpassbare Benutzeroberfläche durch Apps	Einfache und verständliche Bedienung, ausführliche Dialoge, viele Hilfestellungen
Sicherheit	TLS Authentifizierungstokens	TLS Authentifizierungstokens	TLS Authentifizierungstokens AES-Encryption

Plattform	3.2.13 Oracle Cloud	3.2.14 IBM Watson IoT	3.2.15 Azure IoT Hub
Verbreitung	Stack Overflow: 0 Treffer Google: ca. 2.240 Treffer	Forum: 512 Themen/957 Posts Stack Overflow: 10 Treffer Google: ca. 145.000 Treffer	Forum: 35 Themen/119 Posts Stack Overflow: 218 Treffer Google: ca. 92.200 Treffer
Preis	Wearables: 0,40$/Monat, mind. 100.000 Devices Consumer-Geräte: 0,80$/Monat, mind. 50.000 Devices Telematikgeräte: 2,00$/Monat, mind. 20.000 Geräte Gewerbl./industrielle Geräte: 3,00€/Monat, mind. 10.000 Geräte	Kostenlos: kostenlos, max. 20 Devices Standard: 0,0075€/MB, unlimitiert aktive Devices	Free: kostenlos S1: 42,17€/Monat S2: 421,65€/Monat
Verbindungsrate	Wearables: 1.500 Streams/Monat Consumer-Geräte: 15.000 Streams/Monat Telematikgeräte: 100.000 Streams/Monat Gewerbl./industrielle Geräte: 100.000 Streams/Monat	-	Free: 8.000 Streams/Tag S1: 400.000 Streams/Tag S2: 6 Mio. Streams/Tag
Datenmenge	-	100MB/Monat 1GB Speicherplatz	Free: 0,5KB/Stream S1: 4KB/Stream S2: 4KB/Stream
Schnittstellen	3 externe Schnittstellen (HTTP, Java, Java SE)	2 externe Schnittstellen (HTTP, MQTT) 5 Libraries	1 externe Schnittstelle (HTTP) 7 Libraries
API	Datenübertragung Datenverwaltung Authentifizierung Anwendungsverwaltung Plattformverwaltung	Datenübertragung Datenverwaltung Devicemanagement Diagnostik	Datenübertragung Benutzerverwaltung Plattformverwaltung Devicemanagement Diagnostik
Visualisierung	Liniendiagramm Tabellarische Ansicht	Gängige Diagrammtypen Einzelne Werte Messanzeige Erweiterte Visualisierungen über weitere Plattfrormservices möglich	Visualisierung über externe Microsoft Plattform Power BI
Weiterverarbeitung	Externe Enterprise Applications	Weiterverarbeitung nur über zusätzliche Plattformservices	Weiterverarbeitung nur über zusätzliche Plattformservices
Dokumentation	Übersichtlich & gut strukturiert Großer Umfang Illustrationen, Codebeispiele & Tutorials	Übersichtlich & klar strukturiert Großer Umfang Wenige Codebeispiele	Übersichtlich & klar strukturiert Sehr Umfangreich Illustrationen, Tutorials, wenig Codebeispiele
Usability	Einfache und leicht verständliche Bedienung, viele Hilfestellungen Anpassbare Benutzeroberfläche durch Widgets	Einfache und leicht verständliche Bedienung, viele Hilfestellungen, vorkonfigurierte Optionen Daten werden zwischengespeichert Anpassbare Benutzeroberfläche durch Dashboards	Einfache und verständliche Bedienung, viele Hilfestellungen Komplett anpassbare Benutzeroberfläche durch Kacheln
Sicherheit	TLS OAuth	TLS Authentifizierungstokens	TLS Authentifizierungstokens

Plattform	3.2.16 AWS IoT	3.2.17 Google Cloud Platform
Verbreitung	Forum: 325 Themen/1.296 Posts Stack Overflow: 101 Treffer Google: ca. 194.000 Treffer	Forum: 61 Themen/223 Posts Stack Overflow: 200 Treffer Google: ca. 7.980 Treffer
Preis	5$/1Mio. Nachrichten	Bis 250 Mio. Operationen: 0,40$/Million Nächste 500 Mio. Operationen: 0,20$/Million Nächste 1.000 Mio. Operationen: 0,10$/Million Über 1.750 Mio. Operationen: 0,05$/Million
Verbindungsrate	max. 100 simultane Streams	max. 10.000KB/Sek. 160 Operationen/Sek.
Datenmenge	128KB/Stream max. 512KB/Sek.	10MB/Stream
Schnittstellen	2 externe Schnittstellen (HTTP, MQTT) 3 SDKs	1 externe Schnittstelle (HTTP) 9 Libraries
API	Datenübertragung Datenverwaltung Authentifizierung	Datenübertragung Datenverwaltung Authentifizierung
Visualisierung	Visualisierung über Plattformservice Amazon Kinesis	Visualisierung über Plattformservice Google Stackdriver
Weiterverarbeitung	Weiterverarbeitung nur über zusätzliche Plattformservices	Weiterverarbeitung nur über zusätzliche Plattformservices
Dokumentation	Übersichtlich & klar strukturiert Sehr umfangreich Codebeispiele	Übersichtlich & klar strukturiert Sehr umfangreich Illustrationen, Codebeispiele & API Konsole
Usability	Sehr einfache und leicht verständliche Bedienung, viele Hilfestellungen, interaktive Anweisungen	Einfache und verständliche Bedienung, viele Hilfestellungen Dialoge können teilweise nicht abgebrochen werden
Sicherheit	TLS Authentifizierung per Zertifikat von Message-Broker	TLS OAuth AES-Encryption

9.2 Arduino Sketches

9.2.1 ThingSpeak

```
// Libraries für das Wifi Shield
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <ccspi.h>
// Libraries für das NFC Shield
#include <PN532_SPI.h>
#include <snep.h>
#include <NdefMessage.h>

// Setze CS Pin für das NFC Shield
PN532_SPI pn532spi(SPI, 9);
// Erstelle ein NFC Objekt
SNEP nfc(pn532spi);

// Interrupt und Control Pins für das Wifi Shield
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
// Benutze Hardware SPI für die restlichen Pins
// Arduino UNO: SCK = 13, MISO = 12, and MOSI = 11
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIVIDER);

// Setze SSID und WLAN-Passwort
#define WLAN_SSID       "SSID"        // darf nicht länger als 32 Zeichen
sein!
#define WLAN_PASS       "Passwort"
// Setze WLAN-Verschlüsselung
#define WLAN_SECURITY   WLAN_SEC_WPA2

// Setze Timeout für die TCP-Verbindung
#define IDLE_TIMEOUT_MS  3000      // Wartezeit (in Millisekunden), bis die
Verbindung zum Server geschlossen wird falls keine Daten empfangen werden

// Website und Endpunkte der Requests
#define WEBSITE      "api.thingspeak.com"
#define WEBPAGE_POST      "/update?key=<Write API Key>&field1="
#define WEBPAGE_GET      "/channels/<Channel ID>/fields/2/last.json?key=<Read
API Key>"

// Initialisierung der Variablen
int messageSize;
uint8_t ndefBuf[128];
String msg;
int accessed;
int counter;
uint32_t ip;

// Setup
```

```
void setup(void)
{
  Serial.begin(115200); //Beginne serielle Übertragung
  Serial.println(F("Start"));
  counter = 0;
  accessed = 0;
  msg = "NFC initialisiert!";
  createMessage();
  nfc.write(ndefBuf, messageSize, 0); // Sende die NDEF Nachricht
  startWLAN();
  resolveIP();
  getMessage();
}

// Schleife
void loop(void) {
  counter++;
  Serial.println(F("\nSende NFC-Nachricht"));
  if (counter == 20)  // Rufe nach 20 Durchgängen getMessage auf
  {
    getMessage();
    counter = 0;
  }
  sendMessage();
}

/* Funktionen */

// Erstelle NDEF Nachricht
void createMessage(void)
{
  memset(ndefBuf, 0, sizeof(ndefBuf));
  NdefMessage message = NdefMessage();

  // Erstelle den NDEF Record manuell, damit der Speicher des Arduino Uno
nicht überläuft
  NdefRecord record = NdefRecord();
  record.setTnf(TNF_WELL_KNOWN);  // Setze Typ des NDEF Records

  uint8_t recordType[] = { 0x54 }; // "T" Text Record
  record.setType(recordType, sizeof(recordType));

  uint8_t payload[msg.length()+3];  // Berechne Länge der Payload

  for(int i = 0; i < msg.length(); i++)
  {
    payload[i+3] = msg.charAt(i); // Schreibe Nachricht byteweise in die
Payload
  }
  // Language encoding für TNF_WELL_KNOWN RTD_TEXT
  payload[0] = 0x02;
  payload[1] = 0x65; // e
  payload[2] = 0x6e; // n
```

```
  record.setPayload(payload, sizeof(payload));  // Setze die Payload
  message.addRecord(record);

  messageSize = message.getEncodedSize();
  if (messageSize > sizeof(ndefBuf)) {
    Serial.println(F("ndefBuf ist zu klein"));
    while (1) {
    }
  }
  message.encode(ndefBuf);  // kodiere die Nachricht
  message.print();
}

// Sende NFC Nachricht
void sendMessage(void)
{
  if (nfc.write(ndefBuf, messageSize, 3000) >= 0) // Sende die NDEF Nachricht
für 3 Sekunden
  {
    sendRequest();  // Falls Übertragung erfolgreich rufe sendRequest auf
  }
}

// Sende Nachricht an Server
void sendRequest(void)
{
  accessed++;
  /* Versuche eine Verbindung mit dem Server herzustellen.
     Das HTTP/1.1 Protokoll wird benutzt, um zu verhindern, dass der
     Server die Verbindung schließt bevor alle Daten übertragen wurden.
  */
  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80); // Starte TCP
Client
  if (www.connected()) {  // Sende Request bei erfolgreicher Verbindung
    // POST Request über HTTP
    www.fastrprint(F("POST "));
    www.fastrprint(WEBPAGE_POST);
    www.print(accessed);
    if((accessed % 10) == 0)  // Alle 10 Zugriffe wird der Status "sendTweet"
angehängt
    {
      www.fastrprint(F("&status=sendTweet"));
    }
    www.fastrprint(F(" HTTP/1.1\r\n"));
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
www.fastrprint(F("\r\n"));
    www.fastrprint(F("\r\n"));
    www.println();
  } else {
    Serial.println(F("Verbindung fehlgeschlagen!"));
    return;
  }
```

```
  // Lese Daten bis die Verbindung geschlossen wird, oder der Timeout
erreicht wird
  unsigned long lastRead = millis();
  while (www.connected() && (millis() - lastRead < IDLE_TIMEOUT_MS)) {
    while (www.available()) {
      char c = www.read();
      Serial.print(c);
      lastRead = millis();
    }
  }
  www.close();  // Schließe die Verbindung
  delay(15000); // Warte 15 Sekunden
}

// Empfange Nachricht von Server
void getMessage(void)
{
  msg = "";
  boolean writeJson = false;
  /* Versuche eine Verbindung mit dem Server herzustellen.
     Das HTTP/1.1 Protokoll wird benutzt, um zu verhindern, dass der
     Server die Verbindung schließt bevor alle Daten übertragen wurden.
  */
  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80); // Starte TCP
Client
  if (www.connected()) {
    // GET Request über HTTP
    www.fastrprint(F("GET "));
    www.fastrprint(WEBPAGE_GET);
    www.fastrprint(F(" HTTP/1.1\r\n"));
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
www.fastrprint(F("\r\n"));
    www.fastrprint(F("\r\n"));
    www.println();
  } else {
    Serial.println(F("Verbindung fehlgeschlagen!"));
    return;
  }

  // Lese Daten bis die Verbindung geschlossen wird, oder der Timeout
erreicht wird
  unsigned long lastRead = millis();
  while (www.connected() && (millis() - lastRead < IDLE_TIMEOUT_MS)) {
    while (www.available()) {
      char c = www.read();
      Serial.print(c);
      // Extrahiere übertragenen String
      if (c == '{')
      {
        writeJson = true;
      }
```

```
      if (writeJson)
      {
        msg += c;
      }

      if (c == '}')
      {
        writeJson = false;
      }

      lastRead = millis();
    }
  }
  www.close();

  // Extrahiere Wert aus emfangenem String
  msg.remove(0, msg.indexOf("field2")+9);
  msg.remove(msg.length()-2);
  Serial.print(msg);
  // Erstelle aktualisierten NDEF Record
  createMessage();
}

// Starte WLAN-Verbindung
void startWLAN(void)
{
  // Starte WLAN Shield
  Serial.println(F("Initialisiere CC3000"));
  if (!cc3000.begin())
  {
    Serial.println(F("\nKonnte CC3000 nicht initialisieren!"));
    while (1);
  }

  // Lösche alte Verbindungsprofile
  Serial.println(F("\nLösche alte Verbindungsprofile"));
  if (!cc3000.deleteProfiles()) {
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  // Verbinde mit WLAN
  Serial.print(F("\nVersuche mit Netzwerk zu verbinden"));
  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  Serial.println(F("\nVerbunden!"));

  // Warte auf DHCP
  Serial.println(F("\nFrage DHCP an"));
  while (!cc3000.checkDHCP())
```

```
  {
    delay(100);
  }
}

// Löse IP auf
void resolveIP(void)
{
  ip = 0;
  // IP Lookup
  Serial.print(WEBSITE); Serial.print(F(" -> "));
  while (ip == 0) {
    if (! cc3000.getHostByName(WEBSITE, &ip)) {
      Serial.println(F("IP konnte nicht aufgelöst werden!"));
    }
    delay(500);
  }

  cc3000.printIPdotsRev(ip);
}
```

9.2.2 Flowthings.io

```
// Libraries für das Wifi Shield
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <ccspi.h>
// Libraries für das NFC Shield
#include <PN532_SPI.h>
#include <snep.h>
#include <NdefMessage.h>

// Setze CS Pin für das NFC Shield
PN532_SPI pn532spi(SPI, 9);
// Erstelle ein NFC Objekt
SNEP nfc(pn532spi);

// Interrupt und Control Pins für das Wifi Shield
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
// Benutze Hardware SPI für die restlichen Pins
// Arduino UNO: SCK = 13, MISO = 12, and MOSI = 11
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIVIDER);

// Setze SSID und WLAN-Passwort
#define WLAN_SSID       "SSID"        // darf nicht länger als 32 Zeichen
sein!
#define WLAN_PASS       "Passwort"
// Setze WLAN-Verschlüsselung
#define WLAN_SECURITY   WLAN_SEC_WPA2

// Setze Timeout für die TCP-Verbindung
#define IDLE_TIMEOUT_MS  3000      // Wartezeit (in Millisekunden), bis die
Verbindung zum Server geschlossen wird falls keine Daten empfangen werden

// Website und Endpunkte der Requests
#define WEBSITE      "api.flowthings.io"
#define WEBPAGE_POST      "/v0.1/<Account>/drop/<Flow ID Zugriffe>"
#define WEBPAGE_GET      "/v0.1/<Account>/drop/<Flow ID
Nachricht>?limit=1&sort=elems.creationDate"

// Initialisierung der Variablen
int messageSize;
uint8_t ndefBuf[128];
String msg;
int accessed;
int counter;
uint32_t ip;

// Setup
void setup(void)
{
```

```
  Serial.begin(115200); //Beginne serielle Übertragung
  Serial.println(F("Start"));
  counter = 0;
  accessed = 0;
  msg = "NFC initialisiert!";
  createMessage();
  nfc.write(ndefBuf, messageSize, 0); // Sende die NDEF Nachricht
  startWLAN();
  resolveIP();
  getMessage();
}

// Schleife
void loop(void) {
  counter++;
  Serial.println(F("\nSende NFC-Nachricht"));
  if (counter == 20)  // Rufe nach 20 Durchgängen getMessage auf
  {
    getMessage();
    counter = 0;
  }
  sendMessage();
}

/* Funktionen */

// Erstelle NDEF Nachricht
void createMessage(void)
{
  memset(ndefBuf, 0, sizeof(ndefBuf));
  NdefMessage message = NdefMessage();

  // Erstelle den NDEF Record manuell, damit der Speicher des Arduino Uno
nicht überläuft
  NdefRecord record = NdefRecord();
  record.setTnf(TNF_WELL_KNOWN);  // Setze Typ des NDEF Records

  uint8_t recordType[] = { 0x54 }; // "T" Text Record
  record.setType(recordType, sizeof(recordType));

  uint8_t payload[msg.length()+3];  // Berechne Länge der Payload

  for(int i = 0; i < msg.length(); i++)
  {
    payload[i+3] = msg.charAt(i); // Schreibe Nachricht byteweise in die
Payload
  }
  // Language encoding für TNF_WELL_KNOWN RTD_TEXT
  payload[0] = 0x02;
  payload[1] = 0x65; // e
  payload[2] = 0x6e; // n

  record.setPayload(payload, sizeof(payload));  // Setze die Payload
```

```
  message.addRecord(record);

  messageSize = message.getEncodedSize();
  if (messageSize > sizeof(ndefBuf)) {
    Serial.println(F("ndefBuf ist zu klein"));
    while (1) {
    }
  }
  message.encode(ndefBuf);  // kodiere die Nachricht
  message.print();
}

// Sende NFC Nachricht
void sendMessage(void)
{
  if (nfc.write(ndefBuf, messageSize, 3000) >= 0) // Sende die NDEF Nachricht
für 3 Sekunden
  {
    sendRequest();  // Falls Übertragung erfolgreich rufe sendRequest auf
  }
}

// Sende Nachricht an Server
void sendRequest(void)
{
  accessed++;
  /* Versuche eine Verbindung mit dem Server herzustellen.
     Das HTTP/1.1 Protokoll wird benutzt, um zu verhindern, dass der
     Server die Verbindung schließt bevor alle Daten übertragen wurden.
  */
  // Erstelle zu übertragenden String
  String data = "{\"elems\":{\"acc\":";
  data += accessed;
  data += "}}";

  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80); // Starte TCP
Client
  if (www.connected()) {
    // POST Request über HTTP
    www.fastrprint(F("POST "));
    www.fastrprint(WEBPAGE_POST);
    www.fastrprint(F(" HTTP/1.1\r\n"));
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("X-Auth-Token: <Access Token Zugriffe>"));
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("Content-Type: application/json"));
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("Content-Length: "));
    www.print(data.length());
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("\r\n"));
    Serial.print(F("Sending data ...\n"));
```

```
    www.print(data);
    www.fastrprint(F("\r\n"));
  } else {
    Serial.println(F("Verbindung fehlgeschlagen!"));
    return;
  }

  // Lese Daten bis die Verbindung geschlossen wird, oder der Timeout
erreicht wird
  unsigned long lastRead = millis();
  while (www.connected() && (millis() - lastRead < IDLE_TIMEOUT_MS)) {
    while (www.available()) {
      char c = www.read();
      Serial.print(c);
      lastRead = millis();
    }
  }
  www.close();  // Schließe die Verbindung
}

// Empfange Nachricht von Server
void getMessage(void)
{
  msg = "";
  boolean writeJson = false;
  /* Versuche eine Verbindung mit dem Server herzustellen.
     Das HTTP/1.1 Protokoll wird benutzt, um zu verhindern, dass der
     Server die Verbindung schließt bevor alle Daten übertragen wurden.
  */
  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80); // Starte TCP
Client
  if (www.connected()) {
    // GET Request über HTTP
    www.fastrprint(F("GET "));
    www.fastrprint(WEBPAGE_GET);
    www.fastrprint(F(" HTTP/1.1\r\n"));
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE);
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("X-Auth-Token: <Access Token Nachricht>"));
    www.fastrprint(F("\r\n"));
    www.fastrprint(F("Content-Type: application/json"));
    www.fastrprint(F("\r\n"));
    www.println();
  } else {
    Serial.println(F("Verbindung fehlgeschlagen!"));
    return;
  }

  // Lese Daten bis die Verbindung geschlossen wird, oder der Timeout
erreicht wird
  unsigned long lastRead = millis();
  while (www.connected() && (millis() - lastRead < IDLE_TIMEOUT_MS)) {
    while (www.available()) {
```

```
      char c = www.read();
      Serial.print(c);
      // Extrahiere übertragenen String
      if (c == '{')
      {
        writeJson = true;
      }

      if (writeJson)
      {
        msg += c;
      }

      if (c == '}')
      {
        writeJson = false;
      }

      lastRead = millis();
    }
  }
  www.close();

  // Extrahiere Wert aus emfangenem String
  msg.remove(0, msg.indexOf("value")+8);
  msg.remove(msg.length()-2);
  Serial.println();
  Serial.print(msg);
  // Erstelle aktualisierten NDEF Record
  createMessage();
}

// Starte WLAN-Verbindung
void startWLAN(void)
{
  // Starte WLAN Shield
  Serial.println(F("Initialisiere CC3000"));
  if (!cc3000.begin())
  {
    Serial.println(F("\nKonnte CC3000 nicht initialisieren!"));
    while (1);
  }

  // Lösche alte Verbindungsprofile
  Serial.println(F("\nLösche alte Verbindungsprofile"));
  if (!cc3000.deleteProfiles()) {
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  // Verbinde mit WLAN
  Serial.print(F("\nVersuche mit Netzwerk zu verbinden"));
  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
```

```
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  Serial.println(F("\nVerbunden!"));

  // Warte auf DHCP
  Serial.println(F("\nFrage DHCP an"));
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }
}

// Löse IP auf
void resolveIP(void)
{
  ip = 0;
  // IP Lookup
  Serial.print(WEBSITE); Serial.print(F(" -> "));
  while (ip == 0) {
    if (! cc3000.getHostByName(WEBSITE, &ip)) {
      Serial.println(F("IP konnte nicht aufgelöst werden!"));
    }
    delay(500);
  }

  cc3000.printIPdotsRev(ip);
}
```

9.2.3 IBM Watson Internet of Things

```
// Libraries für das Wifi Shield
#include <Adafruit_CC3000.h>
#include <SPI.h>
// MQTT Library
#include <PubSubClient.h>
// Libraries für das NFC Shield
#include <PN532_SPI.h>
#include <snep.h>
#include <NdefMessage.h>

// Setze CS Pin für das NFC Shield
PN532_SPI pn532spi(SPI, 9);
// Erstelle ein NFC Objekt
SNEP nfc(pn532spi);

// Interrupt und Control Pins für das Wifi Shield
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
// Benutze Hardware SPI für die restlichen Pins
// Arduino UNO: SCK = 13, MISO = 12, and MOSI = 11
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIVIDER);

// Setze SSID und WLAN-Passwort
#define WLAN_SSID       "thisislindenplatzOne_2.4G"        // darf nicht
länger als 32 Zeichen sein!
#define WLAN_PASS       "L1metr33pl@cE"
// Setze WLAN-Verschlüsselung
#define WLAN_SECURITY   WLAN_SEC_WPA2

// Host
#define HOST "<Organisation ID>.messaging.internetofthings.ibmcloud.com"
// Client-ID
#define CLIENTID "d:<Organisation ID>:<Device Type>:<Device ID>"
// Benutzername & Passwort
#define USERNAME "use-token-auth"
#define PASSWORD "<Device Authentifizierungstoken>"
// Endpunkte
#define PUBPATH "iot-2/evt/<Event ID>/fmt/json"
#define SUBPATH "iot-2/cmd/<Command ID>/fmt/json"

// Erstelle MQTT Client
Adafruit_CC3000_Client c;
PubSubClient client(c);

// Initialisierung der Variablen
int messageSize;
uint8_t ndefBuf[64];
String msg;
int accessed;
```

```
// Setup
void setup(void)
{
  Serial.begin(115200); //Beginne serielle Übertragung
  Serial.println(F("Start"));
  client.setServer(HOST,1883);  // Setze MQTT Broker
  client.setCallback(getMessage); // Setze Callback-Funktion für empfangene
Nachrichten
  accessed = 0;
  msg = "NFC initialisiert!";
  createMessage();
  nfc.write(ndefBuf, messageSize, 0); // Sende die NDEF Nachricht
  startWLAN();
  connectMqtt();
  msg = "Default Message";
  createMessage();
}

// Schleife
void loop(void) {
  Serial.println(F("\nSende NFC-Nachricht"));
  sendMessage();
  client.loop();  // Prüfe, ob neue Nachricht verfügbar
}

/* Funktionen */
// Erstelle NDEF Nachricht
void createMessage(void)
{
  memset(ndefBuf, 0, sizeof(ndefBuf));
  NdefMessage message = NdefMessage();

  // Erstelle den NDEF Record manuell, damit der Speicher des Arduino Uno
nicht überläuft
  NdefRecord record = NdefRecord();
  record.setTnf(TNF_WELL_KNOWN);  // Setze Typ des NDEF Records

  uint8_t recordType[] = { 0x54 }; // "T" Text Record
  record.setType(recordType, sizeof(recordType));

  uint8_t payload[msg.length()+3];  // Berechne Länge der Payload

  for(int i = 0; i < msg.length(); i++)
  {
    payload[i+3] = msg.charAt(i); // Schreibe Nachricht byteweise in die
Payload
  }
  // Language encoding für TNF_WELL_KNOWN RTD_TEXT
  payload[0] = 0x02;
  payload[1] = 0x65; // e
  payload[2] = 0x6e; // n
```

```
  record.setPayload(payload, sizeof(payload));  // Setze die Payload
  message.addRecord(record);

  messageSize = message.getEncodedSize();
  if (messageSize > sizeof(ndefBuf)) {
    Serial.println(F("ndefBuf ist zu klein"));
    while (1) {
    }
  }
  message.encode(ndefBuf);  // kodiere die Nachricht
  message.print();
}

// Sende NFC Nachricht
void sendMessage(void)
{
  if (nfc.write(ndefBuf, messageSize, 3000) >= 0) // Sende die NDEF Nachricht
für 3 Sekunden
  {
    sendRequest();  // Falls Übertragung erfolgreich rufe sendRequest auf
  }
}

// Sende Nachricht an Server
void sendRequest(void)
{
  accessed++;
  // Erstelle zu übertragenden String
  String data = "{\"d\":{\"acc\": ";
  data += accessed;
  data += "}}";
  int str_len = data.length() + 1;
  char json[str_len];
  data.toCharArray(json, str_len);  // konvertiere String in CharArray

  if(client.publish(PUBPATH, json)) // Sende Nachricht an MQTT Broker
  {
    Serial.println(F("Nachricht published\n"));
  }
  else
  {
    Serial.println(F("Publishing Error!\n"));
  }
}

// Empfange Nachricht von Server
void getMessage(char* topic, byte* payload, unsigned int length)
{
  msg = "";
  msg = (char*)payload; // konvertiere Nachricht in String
  // Extrahiere empfangene Nachricht
  msg.remove(0, msg.indexOf("message")+11);
  msg.remove(msg.indexOf("}")-1);
```

```
  Serial.println(F("Empfangene Payload:\n"));
  Serial.println(msg);
  // Erstelle aktualisierten NDEF Record
  createMessage();
}

// Starte WLAN-Verbindung
void startWLAN(void)
{
  // Starte WLAN Shield
  Serial.println(F("Initialisiere CC3000"));
  if (!cc3000.begin())
  {
    Serial.println(F("\nKonnte CC3000 nicht initialisieren!"));
    while (1);
  }

  // Lösche alte Verbindungsprofile
  Serial.println(F("\nLösche alte Verbindungsprofile"));
  if (!cc3000.deleteProfiles()) {
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  // Verbinde mit WLAN
  Serial.print(F("\nVersuche mit Netzwerk zu verbinden"));
  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
    Serial.println(F("\nFehlgeschlagen!"));
    while (1);
  }

  Serial.println(F("\nVerbunden!"));

  // Warte auf DHCP
  Serial.println(F("\nFrage DHCP an"));
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }
}

// Verbinde mit MQTT
void connectMqtt(void)
{
  if(client.connect(CLIENTID,USERNAME,PASSWORD))
  {
    Serial.println(F("Verbunden mit MQTT\n"));
    if(client.subscribe(SUBPATH)) // Subscribe zu gewünschtem Topic
    {
      Serial.println(F("Erfolgreich subscribed!\n"));
    }
  }
}
```

9.3 Webanwendung

9.3.1 ThingSpeak

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<meta http-equiv="X-UA-Compatible" content="IE=edge">
	<title>Webinterface Thingspeak</title>
	<link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.6.0/pure-
min.css">
	<style>
	body {
			margin: auto;
			width: 380px;
			text-align: center;
		}
		textarea {
			width: 100%;
		}
    </style>
</head>
<body>
	<h2>Webinterface - Thingspeak</h2>
	<form id="msg_form" class="pure-form pure-form-stacked">
	    <fieldset>
	        <textarea name="message" placeholder="Nachricht"></textarea>
	        <button type="submit" class="pure-button pure-button-
primary">Nachricht aktualisieren</button>
	    </fieldset>
	</form>
	<div id="status"></div>

	<script src="https://code.jquery.com/jquery-2.2.3.min.js"></script>
	<script>
$( document ).ready(function() {
	// Variable für den Request
	var request;

	// Binde das Submit Event an das Formular
	$("#msg_form").submit(function(event){

	    // Breche alle ausstehenden Requests ab
	    if (request) {
	        request.abort();
	    }

	    // Initialisiere lokale Variablen
	    var $form = $(this);

	    // Wähle alle Felder des Formulars und speichere sie
	    var $inputs = $form.find("button, textarea");
```

```
          // Erstelle die Payload
          var payload = {
            'key': '<Write API Key>',     // Write API Key
            'field2': $('textarea[name=message]').val() // Hole den Inhalt
der Textarea
        };
        console.log(payload);

          // Deaktiviere alle Inputs für die Dauer des Ajax Requests
          $inputs.prop("disabled", true);

          // Sende den Request
          request = $.ajax({
              url: "https://api.thingspeak.com/update",
              type: "post",
              data: payload
          });

          // Callback Handler bei Erfolg
          request.done(function (response, textStatus, jqXHR){
              $("#status").text("Nachricht gesendet!");    // Setze Status
          });

          // Callback Handler bei Fehlschlag
          request.fail(function (jqXHR, textStatus, errorThrown){
              $("#status").text("Fehler:"+textStatus, errorThrown);    //
Setze Status
          });

          // Callback Handler der immer ausgelöst wird
          request.always(function () {
              // Reaktiviere alle Inputs
              $inputs.prop("disabled", false);
          });

          // Verhindere das Default-Submit des Formulars
          event.preventDefault();
      });
});
      </script>

</body>
</html>
```

9.3.2 Flowthings.io

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<meta http-equiv="X-UA-Compatible" content="IE=edge">
	<title>Webinterface Flowthings</title>
	<link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.6.0/pure-
min.css">
	<style>
	body {
			margin: auto;
			width: 380px;
			text-align: center;
		}
		textarea {
			width: 100%;
		}
    </style>
</head>
<body>
	<h2>Webinterface - Flowthings</h2>
	<form id="msg_form" class="pure-form pure-form-stacked">
	    <fieldset>
	        <textarea name="message" placeholder="Nachricht"></textarea>
	        <button type="submit" class="pure-button pure-button-
primary">Nachricht aktualisieren</button>
	    </fieldset>
	</form>
	<div id="status"></div>

	<script src="https://code.jquery.com/jquery-2.2.3.min.js"></script>
	<script>
$( document ).ready(function() {
	// Variable für den Request
	var request;

	// Binde das Submit Event an das Formular
	$("#msg_form").submit(function(event){

	    // Breche alle ausstehenden Requests ab
	    if (request) {
	        request.abort();
	    }

	    // Initialisiere lokale Variablen
	    var $form = $(this);

	    // Wähle alle Felder des Formulars und speichere sie
	    var $inputs = $form.find("button, textarea");

	    // Hole den Inhalt der Textarea
```

```
          var $msg = $("textarea[name=message]").val();

          // Erstelle die Payload
          var payload = '{"elems":{"msg":"'+$msg+'"}}';
        console.log(payload);

          // Deaktiviere alle Inputs für die Dauer des Ajax Requests
          $inputs.prop("disabled", true);

          // Sende den Request
          request = $.ajax({
              url: "https://api.flowthings.io/v0.1/<Account>/drop/<Flow ID
Nachricht>",
              type: "post",
              data: payload,
              headers: {
                  "X-Auth-Token" : "<Access Token Nachricht>"
                }
          });

          // Callback Handler bei Erfolg
          request.done(function (response, textStatus, jqXHR){
              $("#status").text("Nachricht gesendet!");    // Setze Status
          });

          // Callback Handler bei Fehlschlag
          request.fail(function (jqXHR, textStatus, errorThrown){
              $("#status").text("Fehler:"+textStatus, errorThrown);    //
Setze Status
          });

          // Callback Handler der immer ausgelöst wird
          request.always(function () {
              // Reaktiviere alle Inputs
              $inputs.prop("disabled", false);
          });

          // Verhindere das Default-Submit des Formulars
          event.preventDefault();
});
      </script>

</body>
</html>
```

9.3.3 IBM Watson Internet of Things

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<meta http-equiv="X-UA-Compatible" content="IE=edge">
	<title>Webinterface IBM Watson IoT</title>
	<link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.6.0/pure-
min.css">
	<style>
		body {
			margin: auto;
			width: 380px;
			text-align: center;
		}
		textarea {
			width: 100%;
		}
	</style>
</head>
<body>
	<h2>Webinterface - IBM Watson IoT</h2>
	<form id="msg_form" class="pure-form pure-form-stacked">
		<fieldset>
			<textarea name="message"
placeholder="Nachricht"></textarea>
			<button type="submit" class="pure-button pure-button-
primary">Nachricht aktualisieren</button>
		</fieldset>
	</form>
	<div id="status"></div>

	<script src="https://code.jquery.com/jquery-2.2.3.min.js"></script>
	<script src="js/mqttws31.js"></script>
	<script>
$( document ).ready(function() {
	// Erstelle einen MQTT Client
	var client = new Paho.MQTT.Client("<Organisation
ID>.messaging.internetofthings.ibmcloud.com", Number(1883), "a:<Organisation
ID>:<App ID>");

	// Setze Verbindungsoptionen
	var options = {
	    timeout: 3,   // Verbindungstimeout
	    userName: "<API Schlüssel>",
	    password: "<Authentifizierungstoken>",

	    // Alert bei erfolgreicher Verbindung
	    onSuccess: function () {
	      alert("MQTT verbunden!");
	    },
```

```
            // Alert bei fehlgeschlagener Verbindung
            onFailure: function (message) {
              alert("Verbindung fehlgeschlagen: " + message.errorMessage + "
Bitte Seite neu laden!");
            }
      };

      // Versuche eine Verbindung herzustellen
      client.connect(options);

      // Binde das Submit Event an das Formular
      $("#msg_form").submit(function(event){

            // Initialisiere lokale Variablen
            var $form = $(this);

            // Wähle alle Felder des Formulars und speichere sie
            var $inputs = $form.find("button, textarea");

            // Hole den Inhalt der Textarea
            var $msg = $("textarea[name=message]").val();

            // Erstelle die Payload
            var payload = '{"d":{"message": "'+$msg+'"}}';
            console.log(payload);

            // Deaktiviere alle Inputs für die Dauer des Ajax Requests
            $inputs.prop("disabled", true);

            // Rufe sendData() auf
            sendData(payload);

            // Reaktiviere alle Inputs
            $inputs.prop("disabled", false);

            // Prevent default posting of form
            event.preventDefault();
      });

      function sendData(payload){
            var message = new Paho.MQTT.Message(payload);  // Erstelle neue
MQTT Nachricht
            message.destinationName = "iot-2/type/<Device Type>/id/<Device
ID>/cmd/<Command ID>/fmt/json";    // API Endpunkt
            message.qos = 0;
            client.send(message);   // Sende Nachricht
            $("#status").text("Nachricht gesendet!");      // Setze Status
      }
});
      </script>

</body>
</html>
```