FH MÜNSTER
University of Applied Sciences

Abschlussbericht FEP 2018

Prof. Dr. Claus Grewe (Hrsg.)

Fachhochschule Münster - Wirtschaftsinformatik

Andres, C., Herkenhoff, T., Wallersheim, M., Woywod, T., Wörtler, F.

5. Februar 2018

Münster

Vorwort

Das Curriculum des Studiengangs Master of Science Wirtschaftsinformatik an der FH Münster beinhaltet das Wahlpflichtmodul „Forschungs- und Entwicklungsprojekt“. Das Modul ermöglicht es Studierenden, sich forschungsorientiert mit innovativen Themen auseinanderzusetzen. Die behandelten Themen werden von Jahr zu Jahr neu gewählt und umfassen sowohl aktuelle Forschungsgebiete als auch Innovationen, die den IT-Markt bzw. die Wirtschaftsinformatik gerade erreichen bzw. noch nicht durchdrungen haben.

Das Forschungs- und Entwicklungsprojekt 2017/2018 zum Thema „IoT trifft Blockchain“ startete im September 2017 und lief bis Februar 2018. Das Projektteam umfasste die fünf Mitglieder Christopher Andres, Thomas Herkenhoff, Moritz Wallersheim, Thorsten Woywod und Florian Wörtler. Ausgehend von der entwickelten Vision eines „digitalen Scheckheftes“, das die Nutzungsdaten von Fahrzeugen und industriellen Anlagen fälschungssicher erfasst und diese für cloud-basierte Analysen bereitstellt, wurden folgende Fragestellungen betrachtet:

- Wie lassen sich personenbezogene Daten in einer öffentlichen Blockchain ablegen und Zugriffsrechte hierauf individuell steuern?
- Wie lassen sich Firmware-Updates für IoT-Geräte durch den Einsatz einer Blockchain und eines dezentralen Dateisystems besser schützen?
- Welcher Ansatz eignet sich, um erfasste Maschinendaten redundant in verteilten Edge Devices zu sichern?
- Welche Vor- und Nachteile hat die Inhouse-Verarbeitung gegenüber einer externen IoT-Cloud-Lösung? Wie lassen sich Inhouse-Lösungen in die Angebote von Cloud-Anbietern migrieren?
- Welche Möglichkeiten zur Integration und Verarbeitung von IoT-Daten bieten die IoT-Plattformen Amazon Web Services und Microsoft Azure? Wie grenzen sich diese voneinander ab?

Die Ergebnisse der Untersuchungen wurden in Form von eigenständigen Beiträgen verfasst und in diesem Abschlussbericht zusammengetragen.

Münster, im Mai 2018

Prof. Dr. Claus Grewe

Inhaltsverzeichnis

ConCar – Eine Verbindung von IoT und Blockchain

Christopher Andres[1], Thomas Herkenhoff[2], Moritz Wallersheim[3], Thorsten Woywod[4], Florian Wörtler[5]

Abstract: IoT, Cloud-Computing und die Blockchain - All diese Technologien bieten Unternehmen enormes Potenzial für neue Geschäftsanwendungen und -Prozesse. Dieser Beitrag beschreibt einen prototypischen Anwendungsfall, der die Potenziale von Blockchain und IoT miteinander verbindet. Welche Schwierigkeiten ergeben sich daraus? Welche Anwendungsfälle lassen sich dadurch ableiten und welche Technologien sind für eine Umsetzung eines digitalen Scheckhefts auf Blockchain-Basis relevant?

Keywords: Internet of Things, Blockchain, Verschlüsselung, Firmware-Updates, Big Data, Evaluation, Inhouse, Edge-Computing, Distributed Data, Cloud Platform, Gerätekoordination

1 Einleitung

Dieser Tage gestalten aktuelle Themen wie das Internet of Things, Cloud-Computing oder die Blockchain den Markt der IT und befeuern diesen. Blockchain soll 2018 das Internet der Dinge sogar revolutionieren [Ke17]. Denn mit Hilfe der Blockchain lassen sich deren Daten sicher verwalten. Diese Entwicklung wird durch Cloud-Computing begünstigt, da diese Plattformen aktuelle Probleme bei wichtigen Themen wie Skalierbarkeit und Performanz bereits selbst lösen können. Zuletzt erfahren private Blockchains eine besondere Aufmerksamkeit, da sich Unternehmen hier auf einen ausgewählten Nutzerkreis bzw. begrenzte Netzwerke beschränken können. Wonach das Jahr 2017 Unternehmen dazu diente, Proof-of-Concepts in den Themen Blockchain auszuarbeiten, wird das Jahr 2018 das Jahr der Proof-of-Works, also der Umsetzung der Konzepte und deren dazugehörigen Anwendungsfällen [Ke17].

Ein weiterer Treiber ist das Thema Industrie 4.0. Hier eröffnen sich Themen wie die Früherkennung von Systemausfällen (Predictive Maintenance) oder Anomalien-Erkennung von Daten, die definitiv Potenzial für Unternehmen darstellen. Daneben wird es auch immer die klassischen Anwendungen z.B. die Anbindung von Temperatursensoren an eine Wetterstation geben. Diese Prozesse werden jedoch durch die rasche Entwicklung der IoT-

[1] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, christopher.andres@fh-muenster.de
[2] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, herkenhoff@fh-muenster.de
[3] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, mw757475@fh-muenster.de
[4] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, t.woywod@fh-muenster.de
[5] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, fw288591@fh-muenster.de

Anbieter einfacher zu realisieren und dadurch auch kostengünstiger sein.

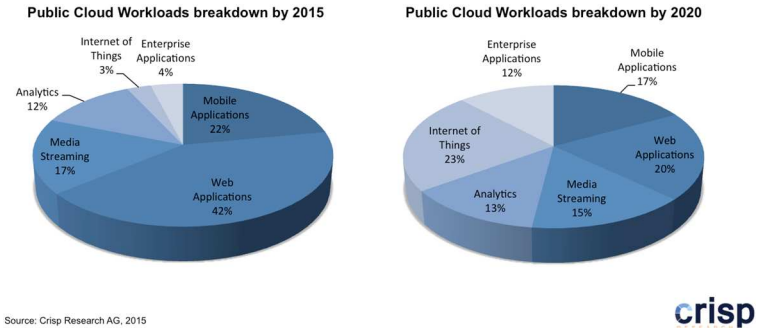

Abb. 1: Workload 2015 vs. 2020 [CR15]

Diese Entwicklung kommt Markteinsteigern durch geringe Eintrittsbarrieren (kostengünstig, bereits etablierte Systeme) bei der Realisierung von Geschäftsmodellen bzw. Anwendungsszenarien zugute. Hinzu kommt, dass der IoT-Markt zu den relevantesten Themen der zukünftigen Jahre zählen wird (siehe Abbildung 1). Ein weiterer wichtiger Punkt wird dabei die Wahl der IoT-Plattform sein [Ga17]. Manche Hersteller haben sich dabei bereits für Plattformen entschieden [Cr15].

Bei einer so hohen Relevanz der Themen IoT und Blockchain kommt eine Nutzung bzw. Verbindung beider in Betracht. Beleuchtet man diesen Bereich genauer, so kann man feststellen, dass hier noch nicht allzu viele relevante Beiträge zu finden sind. Lediglich einige verfasste Konzepte sind verfügbar [Cw17], [CCA15]. Dabei liegt es nahe, die Blockchain als Bestandteil von IoT-Anwendungen einzubinden.

Der folgende Beitrag beschreibt einen solchen Anwendungsfall konkreter wie er beispielsweise in einem Unternehmen Anwendung finden könnte. Zuerst soll jedoch in die grundlegenden Begrifflichkeiten eingeführt werden.

2 Grundlagen

2.1 IoT

Internet of Things (IoT) ist nicht nur ein gern verwendetes Schlagwort in der IT, sondern eine allgemeine Entwicklung, die in fast allen Bereichen unserer Gesellschaft eine stetige Weiterentwicklung und Optimierung durch IT-Unterstützung hervorruft. IoT gibt es in vielen verschiedenen Ausprägungen, welche viele Segmente betreffen.

Es gibt viele Definitionen bzw. Ansätze von Definitionen in der Literatur [KDB17], [Ie15], [IBM17]. Gemein haben diese, dass es sich bei IoT um einen heterogenen Verbund

vieler Objekten und Menschen handelt. Dabei fungiert IoT als Schnittstelle zwischen der virtuellen und realen Welt und ermöglicht es dadurch den Geräten, untereinander zu kommunizieren. IoT wird eingesetzt, um Probleme zu erkennen und zu analysieren sowie die Wertschöpfungskette zu überwachen und zu verwalten. Auch Qualitätskontrollen und die Validierung von Qualitätsversprechen können durch IoT-Lösungen unterstützt werden. [KDB17]

Die Analyse einer IoT-Anwendung auf der untersten Ebene, beginnend bei dem Sensor bzw. Aktuator, muss die Fragestellung, welche Art von Gerät vorliegt und welche Daten von diesem Gerät erzeugt werden, beantworten. Anschließend ist zu betrachten, in welcher Art und Weise das Gerät die Daten übermittelt (zyklisch, pull/push-Prinzip). Hieraus ergeben sich wiederum verschiedene Übertragungsmöglichkeiten und -Techniken, die je nach Art der Datenstruktur oder dem gewählten Ziel-Endpunkt gegeneinander abgewogen werden müssen. Der Endpunkt ist dabei meist ein Datenspeicher, eine selbstentwickelte Plattform oder eine Cloud-Plattform [Is15]. Diese Plattform stellt Lösungen für die Datenverarbeitung, -Persistierung und -Analyse bereit. Je nach Größe der zu entwerfenden IoT-Anwendung müssen zudem die Themen der Erweiterbarkeit und der Interoperabilität mit anderen Systemen beachtet werden [Is15].

Ein weiterer Themenbereich ist Industrial IoT (IIoT). Hierbei geht es darum, IoT in der Industrie zu verwenden, um dadurch Prozesse zu optimieren und somit Kosten zu reduzieren. Insgesamt lässt sich feststellen, dass es eine stärkere Vernetzung innerhalb der Produktion, aber auch zwischen Produkten und Kunden gibt. Produkte werden intelligenter. Nicht nur beim Kunden, sondern auch in der eigenen Produktion kennen die Produkte ihren Produktionsprozess und finden automatisch ihren Weg zur richtigen Produktionsinsel. [HPO16]

Im IIoT-Umfeld fallen oft Schlagworte wie Industrie 4.0, Machine-to-Machine-Kommunikation, Big Data, Machine Learning und Predictive Maintenance. Im Kern geht es immer darum, effizienter, besser und klüger zu werden, um am Ende Geld, Zeit und Energie zu sparen. [HPO16]

2.2 Ethereum

Ethereum ist eine Blockchain-Implementierung, die auf die Erstellung von dezentralen Anwendungen via Smart Contracts ausgelegt ist [Et18a]. Für grundlegende Informationen zur Blockchain-Technologie und der Technologie Ethereum wird auf [Br16] verwiesen. Im Folgenden sollen weiterführende Informationen zu Smart Contracts, deren Programmiersprache Solidity sowie dem Entwicklungsframework Truffle gegeben werden.

Smart Contracts können nach [Et18a] als Code beschrieben werden, der per Transaktion aktiviert und ausgeführt wird. Um die Ausführung zu ermöglichen, stellt Ethereum eine Laufzeitumgebung – die Ethereum Virtual Machine (EVM) – bereit, welche den Code auf allen Knoten im Netz ausführt. Um garantieren zu können, dass eine mögliche Zustands-

transition auf allen Knoten konsistent erfolgt, muss der Code voll deterministisch interpretiert werden. Auf der EVM wird Bytecode ausgeführt. Es existieren jedoch höhere Programmiersprachen und Entwicklungsframeworks, welche die Entwicklung erleichtern.

Eine dieser Programmiersprachen ist Solidity. Solidity bietet die Möglichkeit, mit Hilfe einer typisierten Sprache Verträge zu entwickeln, die auf der Ethereum-Blockchain bereitgestellt werden können. Zu den Besonderheiten der Programmiersprache gehören u.a. *mappings*, *structs* sowie *modifiers,* welche nachfolgend erläutert werden. Wie in [Et18b] erörtert, ermöglicht der Einsatz des *mapping*-Konstrukts die Definition einer Datenstruktur, welche mit einer HashMap zu vergleichen ist. Eine weitere Möglichkeit, Datentypen zu definieren, ist die Verwendung sogenannter *structs*. Ein *struct* bietet die Funktionalität, komplexe Datentypen, ähnlich lokaler Klassen, zu erstellen. Durch *modifier* können Funktionen beschrieben werden, die per Annotation aufgerufen werden können. Die annotierte Funktion wird vor dem Aufrufen der eigentlichen Funktion ausgeführt und ermöglicht damit z.B. eine Validierung der übergebenen Parameter. [Et18b]

Ein bekanntes Entwicklungsframework ist das Truffle-Framework[6]. Dieses bietet eine Entwicklungsumgebung für Smart Contracts, ein Testing-Framework und eine Asset-Pipeline, die über ein eigenes CLI gesteuert werden. Damit können Smart Contracts mit Konsolenbefehlen kompiliert, bereitgestellt, getestet und aktualisiert werden. Die Bereitstellung sowie die Migrationen der Contracts werden automatisiert über einen eigenen Migrations-Contract ausgeführt. Es ist also nicht notwendig, die Contract Application Binary Interfaces (ABIs)[7] und Ethereum-Adressen der einzelnen Contracts manuell zu verwalten. Stattdessen werden die ABIs automatisch in einem Build-Ordner hinterlegt und können somit auch in anderen Projekten verwendet werden. Das Schreiben von Software-Tests ist in den Programmiersprachen Solidity und JavaScript möglich, wobei JavaScript in diesem Fall einen deutlich höheren Funktionsumfang und eine Framework-Unterstützung aufweist.

3 ConCar-Konzept

Das Kapitel 2 beschrieb grundlegende Begriffe aus den Themenbereichen IoT und Blockchain. Diese Bereiche werden in dem hier erörterten ConCar-Konzept verwendet, um ein praxisbezogenes Anwendungsszenario aufzuzeigen.

3.1 Vorgehen

Nach anfänglichen Recherchen und der Vertiefung in den Themenbereichen IoT und der Blockchain-Technologie zeichnen sich verschiedene potenzielle Nutzungsszenarien ab.

[6] http://truffleframework.com/docs/
[7] https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI

Diese Nutzungsszenarien basieren dabei auf der gemeinsamen Nutzung beider Technologien. Der Grundgedanke dabei ist es, neben dem klassischen IoT-Nutzungsszenario eine sichere und verfälschungssichere Speicherung der IoT-Daten in der Blockchain zu gewährleisten. Nach einer Verdichtung der recherchierten Ergebnisse wächst der Wunsch nach der Prototypisierung eines digitalen Scheckhefts auf Basis der Blockchain.

Hierdurch soll den Nachteilen, die sich aus einem „normalen" digitalen Scheckheft ergeben, begegnet werden. Zu diesen Nachteilen eines analogen Scheckhefts zählen neben dem eigentlichen Verlust auch dessen Manipulierbarkeit. Hinzu kommt, dass die meisten Scheckhefte herstellerabhängig sind, sodass jeder Fahrzeughersteller ein digitales Scheckheft-Portal anbieten muss, um diese zu digitalisieren. Hierbei ist festzustellen, dass dies bereits einige Hersteller umgesetzt haben[89].

Durch die Technologien IoT und Blockchain ergeben sich neue Wege und Möglichkeiten, ein solches digitales Scheckheft-Portal zu realisieren. Dafür werden im Folgenden Anforderungen definiert.

3.2 Anforderungen

Beschäftigt man sich mit der Konzeption, so ergeben sich grundlegende funktionale und nicht-funktionale Anforderungen. Zu den funktionalen Anforderungen gehören:

- Geräte müssen in der Lage sein, Daten zu erzeugen und zu speichern.
- Vom Gerät erzeugte Daten müssen an die IoT-Plattform übergeben werden können.
- Bei Nichtverfügbarkeit der Plattform sollen die Daten redundant erhalten werden.
- Eine Nachverarbeitung der erfassten Daten muss ermöglicht werden.
- Die Persistierung der Daten auf kurz- sowie langfristigen Speichermedien muss sichergestellt sein.
- Die erfassten Daten müssen von einer IoT-Plattform analysiert werden können.
- Ein Zugriff auf die Daten externer Schnittstellen muss möglich sein.
- Die Plattform soll Geräteaktualisierungen ermöglichen.

Außerdem werden folgende nicht-funktionale Anforderungen aufgestellt:

- Die Anwendung soll über Drittplattformen erweitert werden können.
- Die Anwendung soll skalierbar sein.

[8] http://www.mazda.de/service-zubehoer/mazda-service/digitaler-service-nachweis/
[9] https://erwin.audi.com/erwin/showHome.do

- Die Daten sollen bis zur Übergabe an die Plattform redundant vorgehalten werden, um eine höhere Ausfallsicherheit gewährleisten zu können.
- Der Umgang mit den Daten soll transparent und sicher erfolgen.
- Die Kommunikation soll verschlüsselt erfolgen.

Im weiteren Verlauf werden die grundlegenden Fragestellungen erläutert, die sich durch die oben genannten Anforderungen ergeben.

3.3 ConCar-Prototyp

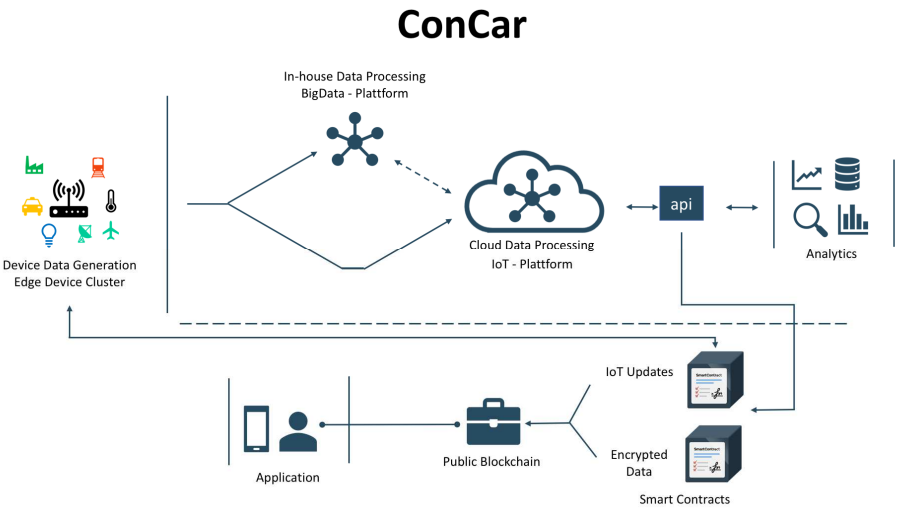

Abb. 2: ConCar-Konzeption

Abbildung 2 skizziert das Anwendungsszenario, welches sich aus den oben genannten Anforderungen ableitet.

Das Internet-der-Dinge findet dabei Anwendung, indem es die notwendigen Daten der Fahrzeuge ermittelt, aggregiert und an das IoT-Backend übermittelt. Dort werden die Daten gespeichert und anschließend ausgewertet. Neben den Fahrzeugdaten ergeben sich ggf. noch Daten aus Werkstätten für etwaige Service- oder Instandhaltungsarbeiten, welche ebenfalls an das IoT-Backend übermittelt werden müssen. Daten, die auf Edge Devices in der Werkstatt anfallen und noch nicht an eine Verarbeitungsplattform übertragen worden sind, müssen sicher gespeichert werden. Hierzu wird sich einer redundanten, verteilten Speicherung der Daten auf mehreren Edge Devices bedient.

Die IoT-Plattform ist für die Verarbeitung, Validierung und sichere Übertragung von

Scheckheft-Einträgen zuständig. Dadurch wird gewährleistet, dass nur korrekte Einträge persistiert werden. Hierbei ist es denkbar, neben einer Cloud-Variante auch eine lokale Variante zu nutzen. Die Blockchain stellt dabei sicher, dass die Einträge nachträglich nicht mehr manipuliert werden können.

Daten können z.B. die Laufleistung des Fahrzeugs oder die Betriebsstunden sein. Liegen alle Daten eines Scheckhefts vor, so hat der Anwender (Werkstattleiter, Fahrzeugbesitzer) die Möglichkeit, sich über eine Weboberfläche Scheckhefteinträge anhand einer eindeutigen Fahrzeugnummer anzeigen zu lassen. Des Weiteren können durch umfassende Analysemöglichkeiten, wie z.B. Machine Learning, Anwendungsszenarien wie die Anomalien-Erkennung oder Predictive Maintenance abgebildet werden.

Ein weiterer Aspekt, der in dem Anwendungsfall ConCar betrachtet werden soll, ist der Prozess der Firmware-Aktualisierung von IoT-Endgeräten. Soll bspw. ein Fahrzeug ein weiteres Attribut an die Plattform übermitteln oder sind Sicherheitslücken aufgetreten, so muss dies in der zugrundeliegenden Software berücksichtigt werden. Die Anpassungen der Software machen es allerdings notwendig, dass ein Update in das Fahrzeug eingespielt wird.

Aus den oben genannten Überlegungen wurden verschiedene Fragestellungen abgeleitet, welche im Folgenden genauer betrachtet werden.

4 Personenbezogene Daten auf einer öffentlichen Blockchain

Ein Teilbereich des ConCar-Beispiels betrifft die Speicherung von Daten in einer Blockchain. Die Thematik der Blockchain umfasst mehrere Bereiche. Heutzutage wird zwischen öffentlicher und privater Blockchain unterschieden. Der Unterschied zwischen den beiden Modellen lässt sich auf die Zugriffsmöglichkeit abstrahieren. Bei einer privaten Blockchain kann vom Unternehmen gesteuert werden, welche Knoten Zugriff auf die Blockchain erhalten. Der Betrieb ist jedoch aufwendig und kostenintensiv. Bei der öffentlichen Blockchain erhält jeder Teilnehmer Zugriff auf die Blockchain und kann Teil des Netzwerkes werden.

Um Transparenz hinsichtlich der Validität der in der Blockchain gespeicherten Daten sicherzustellen und die Kosten zu minimieren, ist der Einsatz einer öffentlichen Blockchain im ConCar-Beispiel angedacht worden. Die erfassten Daten werden somit öffentlich zur Verfügung gestellt. Aus Sicht der Kunden, die an der Fahrzeugabnutzung eines bestimmten Fahrzeuges interessiert sind, ist die Transparenz dabei von entscheidender Bedeutung.

Diese Eigenschaft stellt die Unternehmen vor eine große Herausforderung in Bezug auf den Datenschutz. Personenbezogene Daten in öffentlicher Hand entsprechen nicht den Datenschutzbestimmungen. Auch wenn über die Anonymität von in der Blockchain getätigten Transaktionen gesprochen wird, ist diese nur bedingt gewährleistet. Transaktionen beinhalten Daten. Wenn diesen Daten personenbezogene Informationen beinhalten, sind

diese öffentlich einsehbar. Somit stellt sich die Frage, wie Unternehmen personenbezogene Daten in einer öffentlichen Blockchain speichern können.

Die Besonderheit bei der Betrachtung dieser Fragestellung liegt darin, gespeicherte Informationen in der Blockchain im Bedarfsfall mit anderen Nutzern zu teilen. Darüber hinaus sieht der Kontext vor, dass das Unternehmen Daten in komprimierter und konsolidierter Form für den Kunden in der Blockchain speichern kann.

Somit wird ein Vorgehensmodell benötigt, welches diesen Kriterien entspricht. Unter dem Titel „Entwurf und Implementierung einer prototypischen Anwendung für die Persistierung von personenbezogenen Daten auf einer öffentlichen Blockchain.“ wird auf die Fragestellung und das Vorgehensmodell anhand einer prototypischen Anwendung eingegangen. Die Umsetzung erfolgt auf der Ethereum-Blockchain.

5 Cyber-Attacken auf IoT-Devices

Trotz oder vielleicht sogar aufgrund des aktuellen Trends zur Verwendung von immer mehr IoT-Geräten, ist die Software, die auf diesen eingebetteten Systemen betrieben wird, im Allgemeinen nicht von Grund auf sicher entworfen worden. So wurden im Jahre 2016 mehr als 300.000 IoT-Geräte mithilfe des Schadcodes Mirai unterwandert und zu einem Bot-Netz zusammengeschlossen, das ganze Internet-Hoster kurzzeitig außer Betrieb setzte [Sc16]. Während Mirai die Geräte über das Ausprobieren bekannter Standard-Passwörter unter seine Kontrolle bringen konnte, wurde im Jahr 2017 ein Fall bekannt, der auf dem Quellcode von Mirai aufbaute, um mehrere Millionen Geräte über das Ausnutzen bekannter, noch nicht geschlossener Sicherheitslücken unter Kontrolle zu bringen [La17].

Die Attacken waren nur aus dem Grund möglich, dass viele Hersteller ihre Produkte ohne Rücksicht auf Grundprinzipien der Sicherheit entworfen haben. Dieses Phänomen resultiert unter anderem aus zwei Problemen: Einerseits werden die Hersteller durch enorme Konkurrenz dazu gezwungen, die Time-to-Market möglichst kurz zu halten. Das kann zu Einsparungen im Bereich der Sicherheit führen. Des Weiteren waren eingebettete Systeme in der Vergangenheit oft von IP-Netzwerken isoliert, sodass der Aspekt der Sicherheit nicht zu beachten war. Problematisch wird es dann, wenn diese sogenannten Air Gaps aufgelöst werden, und so Geräte mit dem Internet verbunden werden, die für diesen Einsatzzweck ursprünglich nicht vorgesehen waren [Or15].

Damit die vernetzten Dinge weniger anfällig für Cyberattacken werden, ist also ein Umdenken seitens der Hersteller notwendig, welches erhöhte Investitionen in die Sicherheit der Produkte erfordert. Dass dieser Prozess bereits gestartet ist, kann anhand einer Studie von Gartner [Ga16] nachvollzogen werden, die die weltweiten Ausgaben für die Sicherheit im Segment IoT von 2014 bis 2018 untersucht hat. Steigende Investitionen in den Sicherheitsbereich können jedoch nicht garantieren, dass die zugrundeliegende Software immer frei von Sicherheitslücken entwickelt worden ist. Deshalb ist es ebenso wichtig, regelmäßig Updates auszuliefern, die den aktuellen Sicherheitsstandards entsprechen. Die

Verteilung dieser Updates erfolgt über sogenannte Software-Update-Systeme. Doch genauso wie die Geräte, können auch Software-Update-Systeme von Cyber-Attacken bedroht werden. Haben Cyber-Kriminelle eine Schwachstelle im Software-Code entdeckt, können sie diverse Techniken, beispielsweise DDoS-Attacken oder Infiltrierung des Software-Update-Systems, einsetzen, um ein Versionsupdate zu verhindern.

Um die Probleme der Zentralität und der Manipulierbarkeit der Software-Update-Systeme lösen zu können, würde sich ein Software-Update-System auf Blockchain-Basis anbieten, da gerade diese Technologie für Dezentralität und Unveränderbarkeit steht. Aus diesem Grund beschäftigt sich ein Teil dieser Sammlung mit dem Entwurf und der prototypischen Implementierung eines Software-Update-Systems auf Blockchain-Basis.

6 Datensicherung auf einem Edge Device Instant Cluster

In IoT-Umgebungen, speziell auch in IIoT-Umgebungen, gibt es oft eine große Zahl an Sensoren, die eine große Menge an heterogenen und schwer zu verarbeitende Daten produzieren. In der Regel werden diese Daten über Edge Devices oder Gateways in eine Cloud-Plattform hochgeladen, um dort weiterverarbeitet zu werden. In einigen Fällen ist es aus diversen Gründen nicht möglich oder nicht erwünscht, dass diese Daten in Echtzeit in eine Cloud-Plattform geladen werden. Zunehmend wird Edge oder Fog Computing für die Vorverarbeitung und Zwischenspeicherung von Daten verwendet. Daten werden auf Edge Devices oder Gateways gespeichert oder zwischengespeichert, wenn z.B.

- Die Daten nicht direkt in eine Cloud-Plattform geladen werden (aus technischen, kostentechnischen oder firmenphilosophisch-ethischen Gründen)
- Eine Cloud nicht eingesetzt wird und auch der Einsatz lokaler Datenverarbeitungsserver nicht erwünscht ist oder erst später erfolgt (z.B. aus Kosten oder Aufwands-Gründen)

Damit die Daten nicht verloren gehen, wenn ein Edge Device (im Folgenden Gerät genannt) ausfällt, ist es erforderlich die Daten redundant auf mehreren Geräten zu speichern. Die redundante Speicherung erfordert eine Koordination der Geräte, damit sich die Geräte gegenseitig finden und authentifizieren können. Außerdem ist die Zusicherung erforderlich, dass jede Datei mehrfach gespeichert wird und die redundante Speicherung auch nach dem Ausfall eines Gerätes erhalten bleibt.

Die Idee ist, dass sich die Geräte in Gruppen organisieren, sodass jede Gruppe eine eigene Datensicherung betreibt. Der automatische Zusammenschluss von Edge Devices zu Gruppen wird als Instant Clustering bezeichnet.

7 Inhouse- oder Cloud-IoT-Plattform

Um einen wirtschaftlichen Nutzen aus den von IoT-Geräten generierten Daten ziehen zu können, müssen diese verarbeitet werden. Aufgrund der steigenden Anzahl an IoT-Geräten müssen diese Verarbeitungssysteme in der Lage sein, auch große Mengen von Daten verarbeiten zu können. Verteilte Cluster-Ansätze sind hierbei unverzichtbar. Das System soll zudem derartig skalierbar sein, dass die Erhöhung des Datenvolumens im Bereich mehrerer Magnituden lediglich durch die Bereitstellung weiterer Ressourcen innerhalb des Clusters abgebildet werden kann. Derartige Systeme können in zwei Klassen eingeteilt werden: Inhouse- und Cloud-basierte Systeme.

Anbieter von Cloud-basierten Systemen werben mit hochverfügbaren Plattformen und einem geringen administrativen Aufwand. Einige wichtige Fragestellungen unternehmerischer und gesetzlicher Natur werden hierbei jedoch oft ignoriert. So kann die Freiheit der Wahl einer Online-IoT-Plattform bereits durch gesetzliche Bestimmungen erheblich eingeschränkt werden. Privatwirtschaftliche Faktoren, welche Einfluss auf die Plattformentscheidung nehmen, sind mitunter schwer zu identifizieren, jedoch von hoher Relevanz. Mögliche negative Auswirkungen einer Fehlentscheidung sind nicht zwingend unmittelbar ersichtlich, ihre Ausbesserung jedoch kann mit immensen Kosten verbunden sein.

Zielsetzung dieses Teils der Sammlung ist es, dem Leser ein Gefühl für die Faktoren zu vermitteln, welche einen Einfluss auf die Entscheidung zwischen einer Inhouse- oder einer Cloud-basierten Verarbeitungsplattform nehmen. Neben der Identifikation und Beleuchtung von Faktoren, welche für eine Inhouse-basierte Lösung sprechen, sollen zudem wichtige Anforderungen an diese dargestellt werden. Anschließend wird eine für den ConCar-Anwendungsfall geeignete Plattform diskutiert. Um das Szenario eines Plattformwechsels darzustellen, wird dieser konzeptuell anhand der Umstellung einer Cloud-basierten Continuous Delivery Lösung auf die vorgeschlagene Inhouse-basierte Lösung dargestellt. Abschließend erfolgt eine kritische Betrachtung der gesammelten Erkenntnisse.

8 Evaluation von IoT Plattformen: Azure vs. AWS

Immer mehr Applikationen behandeln Themen der IoT und benötigen für die Verarbeitung ihrer Daten performante und zuverlässige Endpunkte bzw. Plattformen. So auch das ConCar-Beispiel welches für die Verarbeitung der Daten ein Backend-System benötigt.

Die Plattform gestaltet sich dabei meist als Cloudplattform und bietet unterschiedliche Funktionalitäten an, die den einfachen Anwender in der Masse an Informationen ggf. überfordern können. Es ist eine Herausforderung, sich bei der Fülle der Angebote zurecht zu finden, da sich die angebotenen Dienste in ihren zugrundeliegenden Strategien unterscheiden und auf den ersten Blick nicht so einfach zu verstehen sind. Eine weitere Herausfor-

derung besteht in dem harten Konkurrenzkampf, den die Anbieter in diesem Marktsegment betreiben, wodurch hochumworbene Produkte im Zweifel den rein objektiven Blick auf die zu erfüllenden Anforderungen des Anwenders vernebeln können.

Eine erste Recherche im Internet ergibt lediglich erste Ansätze was die Evaluation bzw. den Vergleich von IoT-Plattformen angeht. Zudem fällt vermehrt auf, dass sich die Anforderungen je nach Anwendungsgebiet stark voneinander unterscheiden können, sodass das Ergebnis einer Analyse immer situationsabhängig ist.

Ziel der Analyse ist die genaue Betrachtung zweier Plattformen, um somit einen groben Überblick zu erhalten. Für die Evaluation der IoT-Plattformen wird ein konkretes Anwendungsszenario definiert, welches sich in den Grundzügen aus den Anforderungen des ConCar-Beispiel ableiten lässt. Dieses Beispiel soll prototypisch implementiert werden, um die Stärken und Schwächen beider Plattformen darstellen zu können. Die Prototypische Implementierung findet dabei mit Testzugängen in den einzelnen Plattformen statt, um somit noch spezifischere Informationen über die Plattformen geben zu können.

Durch vorangegangene Arbeiten und Recherchen werden dabei Kriterien erarbeitet, welche für einen Vergleich als Basis dienen sollen. Durch diese ist es möglich, Unterschiede in den Plattformen zu erläutern und anhand einer tabellarischen Darstellung zu veranschaulichen. Letztlich soll der Leser in der Lage sein, Angebote der Plattformanbieter vergleichen zu können, ohne in der Fülle der Informationen unter zu gehen, und sich somit ein strukturiertes Bild über die Produktportfolios machen zu können.

Literaturverzeichnis

[Br16] Brown, C. et.al.: Kryptowährungen und Smart Contracts, Abschlussbericht zum Forschungs- und Entwicklungsprojekt 2015/2016 im Studiengang Master of Science Wirtschaftsinformatik an der FH Münster, Fachhochschule Münster, Münster, 2016.

[CCA15] Cohn, J; Chang, Y; Ahluwalia, G et. al.: Empowering the edge. IBM, 2015

[Cr15] Crisp Research, https://www.crisp-research.com/iot-backend-die-evolution-der-public-cloud-anbieter-im-internet-iot/, 2015, abgerufen am 24.01.2018.

[Cw17] Computerwoche, https://www.computerwoche.de/a/die-iot-trends-in-deutschland,3330602,2, 2017, abgerufen am 24.01.2018.

[Et18a] Ethereum: Ethereum Whitepaper, https://github.com/ethereum/wiki/wiki/White-Paper, 2018, abgerufen am 24.01.2018.

[Et18b] Ethereum: Solidity Documentation, https://media.readthedocs.org/pdf/solidity/develop/solidity.pdf, 24.01.2018, S.17, 38f, 52f, 70f, abgerufen am: 24.01.2018.

[Ga16] Gartner: Internet of Things Security spending worldwide from 2014 to 2018 (in million U.S. dollars), Statista, https://www.hb.fh-muenster.de:2081/statistics/543089/iot-security-spending-worldwide/, 2016, abgerufen am 07.01.2018.

[Ga17] Gartner: Gartner Hype Cycle for Emerging Technologies,

https://blogs.gartner.com/smarterwithgartner/files/2017/08/Emerging-Technology-Hype-Cycle-for-2017_Infographic_R6A.jpg, 2017, abgerufen am 29.01.2017.

[HPO16] Hermann, M.; Pentek, T.; Otto, B.: Design Principles for Industrie 4.0 Scenarios, 2016. In: 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, S. 3928-3937, 2016.

[IBM17] IBM: Was ist das Internet der Dinge? https://www.ibm.com/internet-of-things/de-de/resources/library/what-is-iot/, abgerufen am 17.01.2018.

[Ie15] IEEE-SA Internet of Things (IoT) Ecosystem Study. New York, S. 1-5, 2015. https://ieeesastaff.imeetcentral.com/p/eAAAAAAAFQ1mAAAAAHun4h4, abgerufen am 17.01.2018.

[Is15] Isikdag, U.: „Enhanced Building Information Models: Using IoT Services and Integration Patterns“, Springer, S.17, S. 55ff, 2015.

[Ke17] Kern, E., Blockchain soll 2018 das Internet der Dinge revolutionieren, https://t3n.de/news/blockchain-2018-eco-885681/, 2017 abgerufen am 17.01.2018.

[La17] Labs, L.: Neues Botnetz über IoT-Geräte, https://www.heise.de/security/meldung/Neues-Botnetz-ueber-IoT-Geraete-3867237.html, 2017, abgerufen am 06.01.2018.

[Or15] Orebaugh, A.: Internet der Dinge: Was zu tun ist, um IoT-Security Realität werden zu lassen, http://www.searchsecurity.de/meinung/Internet-der-Dinge-Was-zu-tun-ist-um-IoT-Security-Realitaet-werden-zu-lassen, 2015, abgerufen am 06.01.2018.

[Sc16] Schirrmacher, D.: Source Code von mächtigem DDoS-Tool Mirai veröffentlicht, https://www.heise.de/security/meldung/Source-Code-von-maechtigem-DDoS-Tool-Mirai-veroeffentlicht-3345809.html, 2016, abgerufen am 06.01.2018.

Entwurf und Implementierung einer prototypischen Anwendung für die Persistierung von personenbezogenen Daten auf einer öffentlichen Blockchain

Thorsten Woywod[1]

Abstract: Um Transaktionen zu verifizieren und Transparenz zu schaffen, findet die Blockchain-Technologie immer mehr Verwendung. Diese Arbeit beschäftigt sich mit der Fragestellung: „Wie können Unternehmen eine öffentliche Blockchain nutzen, um personenbezogene Daten zu speichern?“. Die Beantwortung dieser Fragestellung wird anhand des Entwurfes und der Implementierung einer prototypischen Anwendung veranschaulicht. Beginnend beschäftigt sich die Ausarbeitung mit der Einordung des Themenbereiches, der Vermittlung von Grundlagen sowie der Beschreibung verwandter Arbeiten. Darauf aufbauend wird die Umsetzung der prototypischen Anwendung durch einen Entwurfs- und Implementierungspart geschildert. Abschließend werden die gewonnenen Ergebnisse hinsichtlich der Beantwortung der Fragestellung bewertet.

Keywords: Blockchain, Kryptographie, Smart Contract, Verschlüsselte Daten auf der Blockchain.

1 Einleitung

Durch die steigende Beliebtheit von Kryptowährungen wie z.B. Bitcoin, ist die Blockchain-Technologie, auf der Kryptowährungen basieren, bekannt geworden. Zurückzuführen ist die Blockchain-Technologie auf den Verfasser des Whitepapers „Bitcoin: A Peer-to-Peer Electronic Cash System“ Satoshi Nakamoto. Das im Jahr 2009 veröffentliche Whitepaper[2] beschreibt den Ansatz eines dezentralen Peer-To-Peer-Zahlungssystems mit vollkommener Transparenz. Das von Satoshi Nakamoto beschriebene Verfahren eignet sich nicht nur für die Durchführung von Zahlungsvorgängen, sondern ermöglicht auch, jede Art von elektronischen Transaktionen durchzuführen. Differenziert wird die Blockchain hinsichtlich ihrer Modi, dem öffentlichen und dem privaten Modus. [Ro16]

Bei der öffentlichen Blockchain werden die Daten jedem Teilnehmer im Blockchain-Netzwerk zur Verfügung gestellt. Jeder Teilnehmer des Netzwerkes hat somit die Möglichkeit, alle Daten zu lesen. Da bei einer öffentlichen Blockchain keine Restriktionen hinsichtlich der Teilnehmer existieren, kann jeder Teil des Netzwerkes werden und somit Zugriff auf die Daten erhalten. [Pl17]

[1] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, tw734552@fh-muenster.de
[2] https://bitcoin.org/bitcoin.pdf

Neben dem offensichtlichen Nutzen dieser Technologie stellt sich die Frage, wie Unternehmen die öffentliche Blockchain einsetzen können und was sie zu berücksichtigen haben. Die Ausarbeitung beschäftigt sich mit der konkreten Fragestellung, „Wie können Unternehmen personenbezogene Daten auf der Blockchain speichern?“. Die Fragestellung wird anhand des Entwurfes und der prototypischen Implementierung einer Anwendung beantwortet. Die zu entwickelnde Anwendung befähigt den Nutzer zu bestimmen, wer die eigenen personenbezogenen Daten einsehen darf oder wer in der Lage sein darf, Daten für den Nutzer zu bearbeiten.

2 Grundlagen

In diesem Kapitel werden die technischen Grundlagen beschrieben, die für das Verständnis der Arbeit notwendig sind: der Zero-Knowledge-Beweis sowie der Unterschied zwischen öffentlicher und privater Blockchain. Anschließend wird eine verwandte Arbeit beschrieben, die sich mit verschlüsselten Daten auf der Blockchain beschäftigt.

2.1 Zero-Knowledge-Beweis

Der Zero-Knowledge-Beweis bezeichnet eine Beweismethode zur Überprüfung von Geheimnissen, ohne das konkrete Geheimnis zu veröffentlichen. Diese Methode kann dafür genutzt werden, Identitäten zu überprüfen, ohne die Identität zu kennen. Dabei wird die Identität mit einer bestimmten Wahrscheinlichkeit nachgewiesen. [BSW06]

2.2 Unterschiede zwischen öffentlicher und privater Blockchain

Der Unterschied zwischen öffentlicher und privater Blockchain liegt in der Möglichkeit, Zugriff auf die in der Blockchain gespeicherten Daten zu erlangen. Die Tabelle 23-2 aus [Dr17] zeigt, dass bei einer öffentlichen Blockchain jeder die Fähigkeit besitzt, Daten aus der Blockchain zu lesen. Bei der privaten Blockchain besteht die Möglichkeit, durch Restriktionen die Lesemöglichkeit zu verweigern. Die Restriktion hinsichtlich der Lesemöglichkeit basiert darauf, dass bei einer privaten Blockchain nur autorisierte Rechner Teil des Blockchain-Netzwerkes werden können. Bei einer öffentlichen Blockchain wie z.B. der Bitcoin-Blockchain kann jeder einen Rechner zum Netzwerk hinzufügen, die Blockchain herunterladen und Zugriff auf die Daten erlangen. [Pl17]

2.3 Verwandte Arbeiten

Eine verwandte Arbeit, die sich mit der Verschlüsselung von Daten auf der Blockchain beschäftigt, ist das Hawk-Projekt[3]. In der Veröffentlichung von [Ko16] wird die Funktionsweise des Hawk-Projektes beschrieben. Das Projekt ermöglicht es, Contracts, vergleichbar mit Smart Contracts, in einer von Hawk spezifizierten Sprache zu entwickeln. Die Contracts werden mit Hilfe eines Compilers in ein Kryptographie-Protokoll zwischen der Blockchain und den Nutzern übersetzt.

Wie in der Abbildung Hawk Overview aus [Ko16] ersichtlich, besteht ein Hawk Contract aus zwei Partitionen: der öffentlichen und der privaten Partition. Die private Partition beinhaltet Daten, die vor unautorisierten Zugriffen geschützt werden. In der öffentlichen Partition sind frei zugängliche Informationen angesiedelt. Nachdem der Hawk Contract kompiliert ist, stehen drei Artefakte zur Verfügung: ein Programm, welches von allen Konsensknoten in der Blockchain ausgeführt wird, ein Programm für die Nutzereingabe sowie ein Programm für die Verwaltung und Ausführung des Contracts. Diese Artefakte sind dafür dar eine eigene Blockchain zu betreiben. [Ko16]

Das Sicherheitsversprechen von Hawk berücksichtigt die Aspekte „On-chain privacy“ sowie „Contractual security“. Mit „On-chain privacy“ wird die Integrität der Contract-Daten bezeichnet. Der Zugriff auf die privaten Daten ist lediglich involvierten Parteien gestattet. Die privaten Dateien liegen verschlüsselt in der Blockchain. Die Korrektheit der in der Blockchain gespeicherten Daten wird anhand einer Zero-Knowledge-Prüfung sichergestellt. Mit dem Aspekt „Contractual Security“ wird die Integrität innerhalb eines Contract bezeichnet und eine Autorisierung bereitgestellt. Lediglich Parteien, die an einer Transaktion beteiligt sind, haben Zugriff auf die Transaktionsdaten. [Ko16]

3 Entwurf

In diesem Kapitel wird auf den Entwurf der prototypischen Anwendung eingegangen. Der Abschnitt beleuchtet, was personenbezogene Daten sind, die rechtlichen Bestimmungen beim Einsatz einer öffentlichen Blockchain sowie die Anforderungen an den zu entwickelnden Prototypen.

3.1 Personenbezogene Daten

Für die Beantwortung der Fragestellung „Wie können Unternehmen personenbezogene Daten auf der Blockchain speichern?“ ist zu klären, was personenbezogene Daten sind und warum Unternehmen diese besonders betrachten müssen.

---

[3] http://oblivm.com/hawk/

Laut Bundesdatenschutzgesetz wird der Begriff „Personenbezogene Daten“ wie folgt definiert: „Personenbezogene Daten sind Einzelangaben über persönliche oder sachliche Verhältnisse einer bestimmten oder bestimmbaren natürlichen Person (Betroffener)“ (§3 Art. 1 BDSG). Demnach fallen nach §3 BDSG Informationen wie z.B. Name, Adresse, Geschlecht, Herkunft, Geburtsdatum und politische Meinungen, die zu einer natürlichen Person zugeordnet werden können, unter den Begriff der personenbezogenen Daten.

Nach §4 BDSG und [EU16] sind Unternehmen, die personenbezogene Daten erheben, verpflichtet die Notwendigkeit nachzuweisen und den Betroffenen auf Wunsch Auskunft über ihre erhobenen Daten zu gewähren. Die erhobenen Daten dürfen nur nach schriftlicher oder digitaler Genehmigung verarbeitet werden. Die Genehmigung kann vom Betroffenen jederzeit zurückgezogen werden. [§28 Art. 3 BDSG]

3.2 Personenbezogene Daten in der Blockchain

Wie in Abschnitt 1 beschrieben sind Daten, die in der Blockchain gespeichert werden, jedem Teilnehmer des Blockchain-Netzwerkes zugänglich. Bei einer öffentlichen Blockchain hat das Unternehmen keinen Einfluss darauf, wer Teilnehmer dieses Netzwerkes ist.

Durch die Öffentlichkeit des Blockchain-Netzwerkes ist der Schutz der personenbezogenen Daten nicht sichergestellt. [EU16] sieht jedoch vor, dass anonymisierte Daten veröffentlicht werden dürfen. Aus den anonymisierten Daten darf kein Rückschluss auf die Identität der natürlichen Person geschlossen werden können. [EU16]

3.3 Anforderungen

Der zu entwickelnde Prototyp zeigt, wie Unternehmen personenbezogene Daten auf der Blockchain speichern können. Die Anforderungen an den Prototypen leiten sich anhand der gesetzlichen Bestimmungen des Datenschutzgesetzes und des übergeordneten Anwendungsfalles ConCar aus [An18] ab. Im Folgenden werden die funktionalen sowie nichtfunktionalen Anforderungen dargestellt.

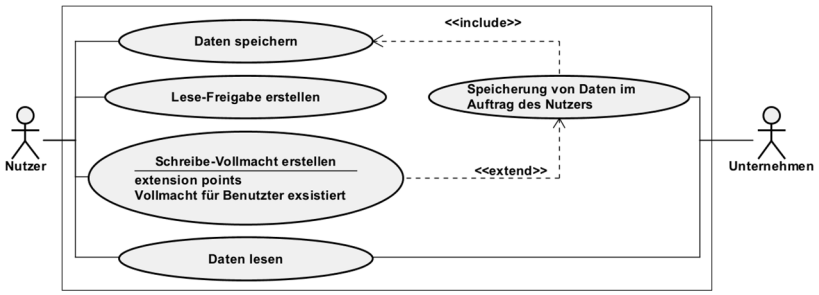

Abb. 1: Anwendungsfalldiagramm

Das Anwendungsfalldiagramm aus Abb. 1 zeigt die funktionalen Anforderungen an den entwickelten Prototypen. Das Anwendungsfalldiagramm besitzt zwei Akteure: den Nutzer und das Unternehmen. Die zu berücksichtigenden Anwendungsfälle aus Sicht des Nutzers sind: das Speichern und das Lesen von Daten, die Erstellung von Lese-Freigaben sowie die Erstellung von Schreibe-Vollmachten. Der zu betrachtende Anwendungsfall des Unternehmens besteht aus der Speicherung von Daten im Auftrag des Nutzers unter der Berücksichtigung einer erteilten Vollmacht.

Neben den funktionalen Anforderungen werden die folgenden nicht-funktionalen Anforderungen berücksichtigt:

- Verschlüsselte Persistierung der Daten in der Blockchain
- Nutzung der Ethereum-Blockchain
- Nutzung einer Angular-Webanwendung für den Zugriff auf die Blockchain

4 Implementierung

Dieser Abschnitt beschäftig sich mit der Implementierung der prototypischen Anwendung. Anhand der Systemübersicht und der Anwendungsfälle wird die Umsetzung der prototypischen Anwendung beschrieben.

4.1 Systemübersicht

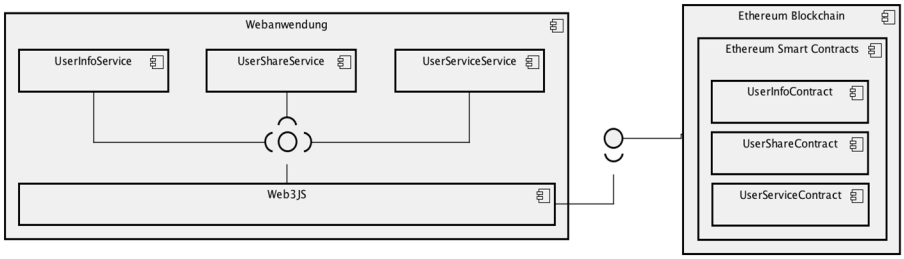

Abb. 2: Komponentendiagramm

Die Abb. 2 zeigt eine abstrakte Systemübersicht des zu entwickelten Prototypens anhand dessen Kernkomponenten. Die Komponenten *UserInfoService*, *UserShareService* und *UserServiceService* sind für die Kommunikation mit den dazugehörigen Smart Contracts zuständig. Diese greifen über die Web3JS-Komponente auf die Blockchain zu. Die *Web3JS*-Komponente stellt das Web3-Framework[4] dar.

[4] https://github.com/ethereum/web3.js/

4.2 Persistierung von personenbezogenen Daten

Um den Anforderungen an den zu entwickelnden Prototypen und den geltenden Datenschutzbestimmungen gerecht zu werden, werden alle personenbezogenen Daten verschlüsselt und somit anonymisiert in der Blockchain persistiert. Durch die Anonymisierung der personenbezogenen Daten werden die Grundsätze des Datenschutzes erfüllt. [EU16]

Eine Vielzahl an Verschlüsselungsmethoden ermöglicht, verschlüsselte Daten mit einem Empfänger zu teilen. In der Kryptographie wird zwischen symmetrischen und asymmetrischen Verschlüsselungsmethoden unterschieden. [BSW06] Das „asymmetrische Verfahren oder Public-Key-Verfahren“ [BSW06] ist für den zu entwickelnden Anwendungsfall am besten geeignet, da hierbei Sender und Empfänger verschiedene Schlüsselpaare besitzen, die ein Geheimnis entweder ver- oder entschlüsseln können. [BSW06] Ein häufig genutztes asymmetrisches Verschlüsselungsverfahren ist das RSA-Verfahren. Dieses Verfahren stellt die Grundlage für die Verschlüsselung der Daten im Prototypen dar.

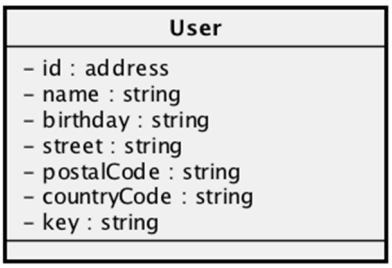

Abb. 3: User Class

Bei der Erzeugung eines Benutzers, siehe Abb. 3, werden personenbezogene Daten wie z.B. der Name und das Geburtsdatum erfasst. Diese Daten dürfen nicht im Klartext auf der Blockchain persistiert werden, da diese Rückschluss auf die natürliche Person zulassen. Alle personenbezogenen Daten werden mit Hilfe des öffentlichen *contentKeyPair*-Schlüssels verschlüsselt. Das *contentKeyPair* besteht aus einem öffentlichen und einem privaten RSA-Schlüssel, die für die Ver- und Entschlüsselung von allen Daten zuständig sind und anonymisiert in der Blockchain gespeichert werden. Das Schlüsselpaar wird bei der Erstellung des Benutzerkontos automatisch von einem Webservice der Webanwendung angelegt. Neben dem *contentKeyPair* wird ein *keyKeyPair*-Schlüsselpaar erzeugt. Dieses ist für die Ver- und Entschlüsselung der *contentKeyPair*-Schlüssel zuständig, welches in den nachfolgenden Kapiteln beschrieben ist. Der öffentliche *keyKeyPair*-Schlüssel des Nutzers ($keykeyPair_{(Bob)}$), wird im Klartext im Attribute *Key* des *Users* gespeichert.

Nachdem die zu persistierenden Daten mit Hilfe des öffentlichen *contentKeyPair*-Schlüssel ($contentKeyPair_{(Bob)}$) verschlüsselt sind, werden die verschlüsselten *User*-Daten an den Smart Contract *UserInfo* übergeben, der diese speichert und verwaltet. Für die Entschlüsselung der Daten ist immer der zugehörige private *contentKeyPair*-Schlüssel notwendig ($contentKeyPair_{(Bob)}$). Dieser muss jedem Nutzer zugänglich gemacht werden, der Zugriff auf die personenbezogenen Daten erhalten soll.

Die Ver- und Entschlüsselung wird immer frontendseitig durchgeführt und ermöglicht, das Vorgehensmodell auf verschiedene Blockchain-Technologien zu übertragen. Die frontendseitige Umsetzung setzt auf eine in TypeScript[5] geschriebene Angular[6]-Anwendung auf. Für die Verschlüsselung nach dem RSA-Verfahren wird auf das JavaScript-Module NodeRSA[7] zurückgegriffen.

4.3 Erstellen von Lese-Freigaben

Wie im Abschnitt 4.2 beschrieben, wird für die Entschlüsselung der Daten der private *contentKeyPair*-Schlüssel des Nutzers benötigt, der die Nachricht verschlüsselt hat ($contentKeyPair_{(Bob)}$). Dieser Schlüssel muss mit den beteiligten Personen ausgetauscht werden. Für den Austausch des Schlüssels können verschiedene Medien genutzt werden.

Zu den möglichen Austausch-Szenarien, die den Anforderungen des Datenschutzes entsprechen, gehören:

a) der physische Austausch über manuelle Eingabe des Schlüssels

b) der Peer-To-Peer-Austausch von Gerät zu Gerät

c) der Austausch über einen zentralen Datenspeicher oder Datenbank

d) der verschlüsselte Austausch über einen zentralen Datenspeicher oder Datenbank

e) der verschlüsselte Austausch über die Blockchain

Jedes dieser Austausch-Szenarien weist gewisse Vor- und Nachteile auf. Die Lösung a) stellt das sicherste Verfahren dar, da die Schlüssel auf keinem zentralen oder dezentralen Speicherort persistiert werden der öffentlich einsehbar ist. Die Nutzer müssen die Schlüssel manuell austauschen und eingeben. Aus Sicht des Nutzers stellt dieses einen unzumutbaren Aufwand dar, sodass dieses ausgeschlossen wird. Um Medienbrüche zu vermeiden und die Vorteile eines dezentralen Systems, wie einer öffentliche Blockchain, zu nutzen, wird der Austausch der Schlüssel mit Szenario e) umgesetzt.

Der verschlüsselte Austausch über die Blockchain erfolgt derartig, dass der private Schlüssel für jeden Nutzer (*Alice*), der eine Freigabe hat, verschlüsselt im Smart Contract *UserShares* gespeichert wird. Jeder Nutzer besitzt zwei Arten von Schlüsselpaaren: das *contentKeyPair* zum Ver- oder Entschlüsseln von personenbezogenen Daten sowie das *keyKeyPair* zum Ver- oder Entschlüsseln von geteilten Schlüsseln. Die Verschlüsselung des geteilten privaten Schlüssels ($contentKeyPair_{(Bob)}$) erfolgt somit mit dem öffentlichen *keyKeyPair*-Schlüssel von *Alice*. Der öffentliche *keyKeyPair*-Schlüssel ($keyKeyPair_{(Alice)}$)

[5] https://www.typescriptlang.org/
[6] https://angular.io/
[7] https://github.com/rzcoder/node-rsa

wird im Klartext im Contract *UserInfo* vorgehalten und kann von jedem abgerufen werden. Ein Sicherheitsrisiko durch den im Klartext gespeicherten öffentlichen Schlüssel ist nicht existent, da der *UserShare* Contract lediglich den ursprünglichen Nutzern gestattet, Freigaben zu speichern. Dritte können somit nicht im Namen des ursprünglichen Nutzers eine Freigabe erstellen.

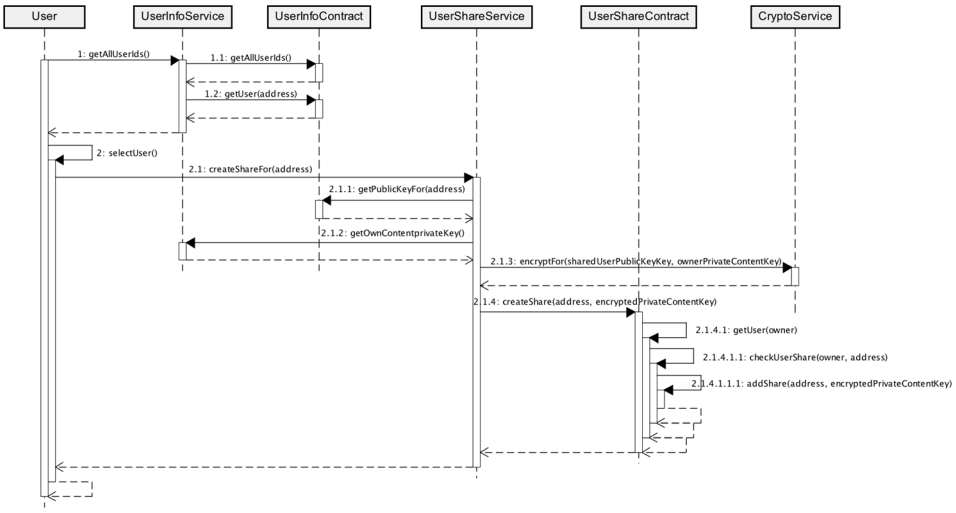

Abb. 4: Sequenzdiagramm – Lese-Freigabe erstellen

Das Sequenzdiagramm aus Abb. 4 visualisiert den Ablauf bei der Erstellung einer Lese-Freigabe von der Nutzerinteraktion bis hin zur Speicherung in der Blockchain.

Die Komponenten *UserInfoContract* und *UserShareContract* stellen die in der Blockchain bereitgestellten Smart Contracts dar. Die übrigen Komponenten repräsentieren Teile der Webanwendung.

Die Entwicklung von Smart Contracts für die Ethereum Blockchain wird mit der „Turing-kompletten“ Programmiersprache Solidity [An18] durchgeführt. Ein konkretes Problem, welches bei der Umsetzung der Contracts aufgefallen ist, ist die fehlende Funktionalität der Rückgabe von selbst definierten Datentypen sowie die Rückgabe von mehrdimensionalen Arrays. Wie in Abb. 4 zu erkennen wird für den Erhalt einer Liste von Benutzern zunächst ein Array von BenutzerIds angefragt. Im zweiten Schritt wird für jede enthaltene BenutzerId eine eigene Anfrage ausgeführt. Bei einer großen Anzahl von Datensätzen kann das Vorgehen zu Performanceproblemen führen.

Die Abb. 5 zeigt die Solidity-Funktion zur Ausgabe des *User*-Objektes. Zu beachten ist, dass das Objekt nicht als *User*-Objekt zurückgegeben wird, sondern zu einem Array konvertiert wird.

```
function getUser(address id) public returns(address, string, string, string, string,
string, string) {
  User memory user;
  user = users[id];
  return (
    user.id, user.name, user.birthday,
    user.street, user.postalCode,
    user.countryCode, user.key
  );
}
```

Abb. 5: Solidity-Funktion getUser

4.4 Zugriff auf Daten mit Lese-Freigabe

Ein Hauptargument für den Einsatz einer öffentlichen Blockchain ist die Transparenz der getätigten Transaktionen. Diese Transparenz führt dazu, dass alle Daten öffentlich eingesehen werden können. Durch die Nutzung von Verschlüsselungstechnologien kann, wie in Abschnitt 4.2 und 4.3 beschrieben, sichergestellt werden, dass die gespeicherten Daten nur von autorisierten Personen deanonymisiert werden können. Das Abrufen der verschlüsselten Daten ist jedoch von jedem möglich.

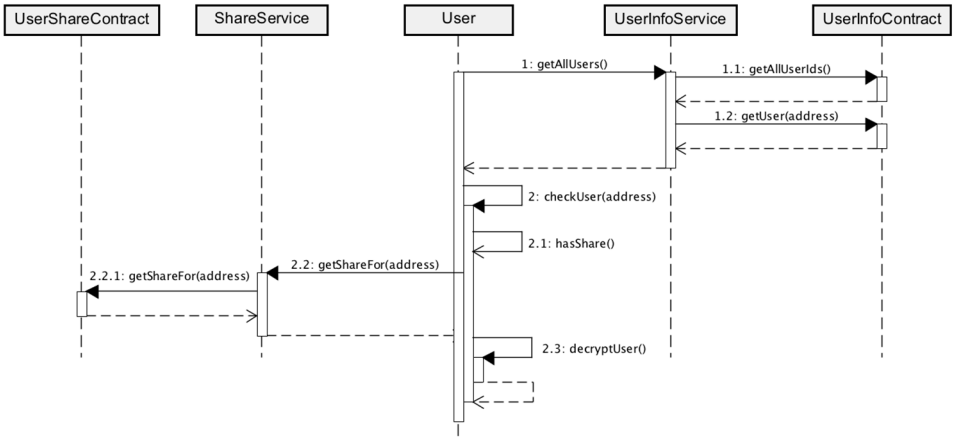

Abb. 6: Sequenzdiagramm - Benutzer mit Freigabe anzeigen

Das Sequenzdiagramm Abb. 6 zeigt den Anwendungsfall *Benutzer mit Freigabe anzeigen*, welcher in der prototypischen Anwendung behandelt wird. Hier werden alle Benutzer ausgegeben und entschlüsselt, für die dem aktuelle Benutzer eine Freigabe ausgestellt wurde.

Für den Abruf und die Entschlüsselung der Benutzerdaten werden die Smart Contracts *UserInfo* und *UserShare* benötigt. Wie in Abschnitt 4.3 geschildert, beinhaltet der

Contract *UserShare* alle Freigaben, die ein Nutzer ausgestellt hat. Über die im Contract implementierte Funktion *getShareFor* ist der Nutzer in der Lage zu überprüfen, ob der aktuelle Nutzer eine Freigabe für eine bestimmte BenutzerId hat. Der *User* repräsentiert in diesem Fall ein Content-Objekt und kann durch jede Art von Objekt ausgetauscht werden. Solange das Content-Objekt einen Rückschluss auf den Besitzer zulässt, ermöglicht das Vorgehensmodell die Persistierung jeglicher Art von Datensätzen. Durch die Freigabe können alle Daten zu einem Nutzer entschlüsselt werden. Für den Abruf des *Users* wird der Contract *UserInfo* genutzt. Wie in Abschnitt 4.3 erläutert, stellt dieser die Funktionen *getAllUser* sowie *getUser* bereit, über die eine Liste der *User* ausgegeben werden kann. Nachdem die Liste der *User*-Objekte lokal vorliegt, wird für jedes Objekt geprüft, ob der aktuelle Nutzer eine Freigabe besitzt.

Unter der Voraussetzung, dass eine Freigabe vorhanden ist, wird der private *contentKeyPair*$_{(Bob)}$-Schlüssel mithilfe des eigenen privaten *keyKeyPair*$_{(Alice)}$-Schlüssel entschlüsselt und im *ShareService* zwischengespeichert. Dieser entschlüsselte, private *contentKeyPair*$_{(Bob)}$-Schlüssel wird genutzt, um die verschlüsselten Objekt-Informationen von *Bob* zu entschlüsseln.

Die Überprüfung der Freigaben greift auf die zwischengespeicherten Schlüssel zurück, die nur für die Laufzeit der Browser Session zur Verfügung stehen. Durch den Einsatz des Zwischenspeichers wird die Anzahl der Anfragen an die Blockchain minimiert. Eine Anfrage wird nur pro Nutzer ausgeführt und nicht für jedes Content-Objekt, das auf der Blockchain persistiert ist. Bei der Übertragung des Beispiels auf ein Szenario, in dem ein Nutzer mehrere Content-Objekte abrufen kann wie z.B. das Speichern des Verlaufes von ausgewerteten IoT-Fahrzeugdaten, können die ausgeführten Anfragen minimiert werden.

4.5 Erstellung von Schreibe-Vollmachten

Die Anforderung an den zu entwickelnden Prototypen besteht darin, dass das Unternehmen im Auftrag des Kunden Daten in der Blockchain persistieren kann. Dieses wird über die Funktionalität der Vollmacht realisiert. Die Vollmacht wird einem im System registrierten Benutzer (*Bob*) ausgestellt, der das Unternehmen darstellt. Die Abwicklung der Ausstellung sowie die Abfrage von Vollmachten werden im Smart Contract *UserService* behandelt. Damit ein Nutzer (*Alice*) im Auftrag eines anderen Nutzers (*Bob*) Daten verschlüsselt in der Blockchain persistieren kann, benötigt *Bob* den öffentlichen *contentKeyPair*$_{(Alice)}$-Schlüssel. Die Speicherung des Schlüssels erfolgt wie bei der Freigabe verschlüsselt. Der öffentlichen *contentKeyPair*$_{(Alice)}$-Schlüssel wird mit dem öffentlichen *keyKeyPair*$_{(Bob)}$-Schlüssel verschlüsselt. Die Entschlüsselung des *contentKeyPair*$_{(Alice)}$-Schlüssels ist nur Besitzern des privaten *keyKeyPair*$_{(Bob)}$-Schlüssels möglich.

Damit das Unternehmen automatisiert Daten im Auftrag eines Nutzers in der Blockchain speichern kann, wird eine Schnittstelle benötigt, auf die das Unternehmen zugreifen kann. Diese Schnittstelle wird im Folgenden nicht weiter betrachtet, da diese aus zeitlichen Gründen nicht umgesetzt worden ist.

5 Ergebnisdiskussion

Der anhand der Fragestellung „Wie können Unternehmen personenbezogene Daten auf der Blockchain speichern?“ entwickelte Prototyp zeigt einen Ansatz, wie Unternehmen personenbezogene Daten auf einer öffentlichen Blockchain speichern können. Der Ansatz abstrahiert die eingesetzte Blockchain-Technologie und kann auf jede Blockchain angewandt werden, die das Speichern von Daten erlaubt. Durch den Einsatz des RSA-Verfahrens für die Verschlüsselung, sind Unternehmen in der Lage, Daten in eine öffentlichen Blockchain wie der Ethereum Blockchain zu speichern, und die aktuellen Datenschutzbestimmungen zu berücksichtigen.

Der eingeschränkte Funktionsumfang der Programmiersprache Solidity mit der Smart Contracts auf der Ethereum-Blockchain umgesetzt werden und der Einsatz der Ende-zu-Ende-RSA-Verschlüsselung erzeugen das Bild, dass die Laufzeit des verwendeten Verfahrens zur Entschlüsselung von Massendaten ungeeignet ist. Hingegen ist das Verfahren eher geeignet für wenige und komprimierte Daten, wie im betrachteten Anwendungsfall "Speicherung von personenbezogenen Daten". Die Untersuchung grenzt sich von der verwandten Arbeit hinsichtlich der Blockchain-Komponente ab. Hawk setzt auf die Bereitstellung einer neue Blockchain mit integrierten Verschlüsselungsmethoden. Der in der Ausarbeitung betrachtete Ansatz setzt jedoch auf die etablierte Ethereum-Blockchain-Technologie auf und ermöglicht die Übertragung auf andere Blockchain-Technologien.

Ein Nachteil welches das beschriebene Vorgehensmodell aufweist, ist die lokale Speicherung des privaten *keyKeyPair*-Schlüssels. Bei Verlust dieses Schlüssels, durch z.B. einen Hardware-Defekt, ist der Nutzer nicht mehr in der Lage, Zugriff auf seine Daten zu erlangen. Eine Lösung dieses Problems stellt z.B. die Verschlüsselte Speicherung des privaten *keyKeyPair*-Schlüssels in der Blockchain dar. Dieser Schlüssel könnte z.B. symmetrisch nach dem AES-Verfahren [KW11] mit einem Passwort verschlüsselt und an das *User*-Objekt angefügt werden. Die Ergänzung würde dem Nutzer ermöglichen, verschiedene Endgeräte zu verwenden.

6 Zusammenfassung und Ausblick

Abschließend lässt sich zusammenfassen, dass die Ausarbeitung den Entwurf und die Implementierung einer prototypischen Anwendung beschreibt, die personenbezogenen Daten datenschutzkonform in einer öffentlichen Blockchain speichert. Im Entwurf werden anhand der Hauptfragestellung und dem übergeordneten Anwendungsbeispiel "ConCar" Datenschutzbestimmungen dargestellt die zu berücksichtigen sind. Darüber hinaus werden relevante Anforderungen erhoben. Die Umsetzung der Anforderungen wird in dem darauffolgenden Abschnitt "Implementierung" näher erläutert. Bestandteil des Abschnittes sind die Szenarien *Daten speichern*, *Daten freigeben*, *Daten lesen* sowie *Vollmacht ausstellen*. Die Umsetzung erfolgt auf Basis der Ethereum-Blockchain. Für den Zugriff auf die gespeicherten Daten ist eine auf Angular basierende Webanwendung umgesetzt

worden. Nach der Implementierung wird das Ergebnis bewertet und das Anwendungsgebiet definiert, in dem das Vorgehensmodell einen Nutzen aufweist.

Die Ausarbeitung lässt die Betrachtung einer Geschwindigkeitsanalyse sowie das Optimierungspotential bei der Handhabung von Massendaten außer Acht. Darüber hinaus steht eine Migration des Programmcodes auf eine andere Blockchain-Technologie aus. Diese Aspekte können in zukünftigen Ausarbeitungen betrachtet werden.

Literaturverzeichnis

[An18] Andres, C.; Herkenhoff, T.; Wallersheim, M.; Woywod, T.; Wörtler, F.: Eine Verbindung von IoT und Blockchain, Abschlussbericht FEP 2018 S. 1-12, 2018.

[BSW06] Beutelspacher, A.; Schwenk, J.; Wolfenstetter, K.: Moderne Verfahren der Kryptographie: Von RSA zu Zero-Knowledge, 6. Auflage, Friedr. Vieweg & Sohn Verlag/ GWV Fachverlag GmbH, Wisbaden, S. 1f, 9, 40ff, 2006.

[Dr17] Drescher, D: Blockchain Basics: A Non-Technical Introduction in 25 Steps, Apress, Frankfurt am Main, S. 216, 2017.

[EU16] Europäischen Union: Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates - zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung), §16, §26, http://eur-lex.europa.eu/legal-content/DE/TXT/?qid=1462345886854&uri=OJ:JOL_2016_119_R_0001, Abgerufen am: 1.11.2017, 2016.

[Ko16] Kosba, A. et.al.: Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts, University of Maryland und Cornell University, 2016.

[KW11] Küsters, R.; Wilke, T.: Moderne Kryptographie: Eine Einführung, Vieweg +Teubner Verlag, 2011.

[Pl17] Ploom, T: Blockchains -wichtige Fragen aus IT-Sicht, S. 123 – 148, 2017.

[Ro16] Roßbach, P: Blockchain-Technologie und ihre Implikationen: Teil2: Anwendungsbereich der Blockchain-Technologie, http://blog.frankfurt-school.de/wp-content/uploads/2016/02/Blockchain_FSBlog_part2.pdf, Abgerufen am: 29.01.2018, 2016.

Entwurf und prototypische Implementierung eines Blockchain-basierten IoT-Software-Update-Systems

Christopher Andres[1]

Abstract: Bedingt durch das Internet der Dinge existieren in der heutigen Welt eine Vielzahl vernetzter Geräte, die alle zum Ziel von Cyber-Kriminellen werden können, welche Schwachstellen in der Software ausnutzen wollen. Deshalb ist es notwendig, die Geräte regelmäßig durch ein Software-Update-System mit Aktualisierungen zu versorgen. Diese Arbeit stellt eine Lösung für ein Software-Update-System auf Blockchain-Basis vor. Dabei wurden zwei Smart Contracts für die Ethereum-Blockchain entworfen, welche Metainformationen zu den Updates bereitstellen. Die Update-Images selbst werden im dezentralen Dateisystem IPFS, welches diese inhaltsadressiert ablegt, persistiert. Der Zugriff auf die Blockchain wird über eine Server-Schnittstelle sowie eine Weboberfläche ermöglicht.

Keywords: Software-Update-System, Blockchain, Smart Contracts, Ethereum, IoT, Firmware

1 Einleitung

Software-Updates spielen in der gegenwärtigen Zeit eine immer größere Rolle. Regelmäßig tauchen in den Nachrichten Meldungen über Sicherheitslücken jeglicher Art auf, die daraufhin von den Herstellern über ein Sicherheitsupdate geschlossen werden müssen. Um diese Aktualisierungen für die vorgesehenen Endgeräte verfügbar zu machen, werden sogenannte Software-Update-Systeme eingesetzt.

Wie in Kapitel 5 in [An18] beschrieben, sind herkömmliche Software-Update-Systeme aufgrund ihrer zentralisierten Natur und der Möglichkeit, enthaltene Software-Images zu verändern, anfällig für Cyber-Angriffe diverser Art.

Die Zielsetzung dieser Untersuchung ist es daher, die Blockchain-Technologie einzusetzen, um die genannten Schwächen herkömmlicher Software-Update-Systeme zu schließen. Die Blockchain steht nämlich für das genaue Gegenteil: Dezentralität und Unveränderlichkeit. Zu diesem Zweck sollen ein konkreter Entwurf sowie ein Prototyp entstehen, die die Machbarkeit des Ansatzes unter Beweis stellen.

Das nächste Kapitel führt in die grundlegende Thematik sowie verwendete Technologien ein und nennt verwandte Arbeiten, die die in dieser Arbeit beschriebene Lösung beeinflusst haben. Im Anschluss werden Analyse, Entwurf und prototypische Implementierung des Software-Update-Systems behandelt, welches daraufhin kritisch beurteilt wird. Das letzte Kapitel fasst die Ergebnisse zusammen und gibt einen Ausblick.

[1] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, christopher.andres@fh-muenster.de

2 Grundlagen

In diesem Kapitel werden die technischen Grundlagen behandelt, die zum Verständnis dieser Arbeit und der entwickelten Lösung notwendig sind. Bei diesen handelt es sich um Software-Update-Systeme sowie dezentrale Dateisysteme. Außerdem werden die Ergebnisse verwandter Arbeiten erläutert, die sich theoretisch und mit unterschiedlichen Schwerpunkten mit der Möglichkeit befassen, Software-Updates sicherer verteilen zu können.

2.1 Software-Update-System

Software-Updates für IoT-Geräte werden in der Regel aus Entfernung – Over-The-Air – verteilt. Herkömmliche manuelle Verfahren sind oft eher ungeeignet, da viele Endgeräte stark verteilt beziehungsweise schwer zugänglich sind.

Ein Over-The-Air-Update wird von einem zentralen Update-Server verteilt und auf den Geräten nach Empfang automatisch installiert [Si16]. Der Prozess ist in der folgenden Abbildung veranschaulicht.

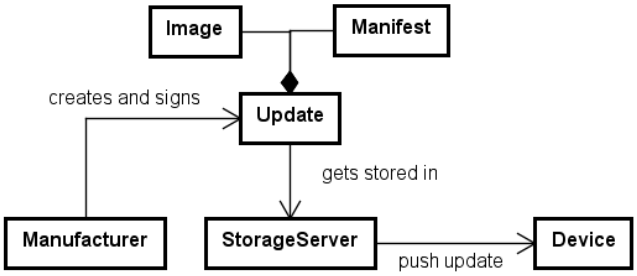

Abb. 1: OTA-Update Prozess

Da die Möglichkeit besteht, dass Update-Server kompromittiert werden können, muss eine Sicherheit zwischen Herstellern und Endgeräten gewährleistet werden. Zu diesem Zweck werden Verschlüsselungsverfahren verwendet sowie Metainformationen für jedes Update in einem Manifest bereitgestellt. Diese Metainformationen werden benutzt, um das Update-Image verifizieren zu können. Sie weisen dabei nach [MMT17] unter anderem folgende Bestandteile auf:

- Eine digitale Signatur zur Verifizierung des Herstellers
- Einen Verweis auf das Update-Image
- Die Länge und den Hashwert des Images zur Prüfung von Veränderungen
- Versionsangaben

Nachdem ein Endgerät sowohl das Manifest als auch das Image empfangen und verifiziert hat, kann das Image an einen Bootloader übergeben und installiert werden.

Ein Endgerät kann nach [CPJ16] generell auf zwei Weisen an ein Update gelangen: Entweder gelangt das Update über eine eingehende Verbindung vom Update-Server zum Gerät (Push-Prinzip) oder das Gerät fordert das Update aktiv an (Pull-Prinzip). Das Push-Prinzip gilt in dieser Hinsicht jedoch als unsicher, da Geräte auf diese Weise leichter kompromittiert werden können. Das Pull-Prinzip hingegen lässt sich etwa dadurch umsetzen, dass das Gerät ein bekanntes Repository periodisch nach neuen Updates untersucht und diese nach der Entdeckung herunterlädt.

2.2 Dezentrale Dateisysteme

Software-Update-Images sind in ihrer Dateigröße in der Regel unbegrenzt und können je nach Umfang des Updates Größenordnungen erreichen, welche gerade aus ökonomischer Betrachtung nicht mehr Teil einer Transaktion auf einer öffentlichen Blockchain wie Ethereum sein sollten. Stattdessen wird lediglich eine Referenz auf die Adresse der Datei in der Transaktion abgelegt. Um an dieser Stelle die Dezentralität und Unveränderlichkeit der Blockchain beizubehalten, sollte diese Referenz auf ein Speichermedium zeigen, welches diese beiden Eigenschaften ebenfalls aufweist.

Ein solches, dezentrales Dateisystem ist das Interplanetary File System (IPFS) [Be14]. In diesem werden Objekte inhaltsadressiert abgelegt. Das bedeutet, dass ein kryptografischer Hashwert über den zu speichernden Inhalt gebildet wird, welcher dem Dateisystem als Speicheradresse dient. Dieses Verfahren erfordert zwar, dass die Speicheradresse für das menschliche Auge unleserlich wirkt, garantiert jedoch im Gegenzug die Unveränderlichkeit des gespeicherten Objektes. Das Hochladen einer manipulierten Version der Datei würde zu einem veränderten Hashwert führen und deshalb eine andere Adressierung erhalten.

Die Dezentralität gewinnt IPFS durch die Tatsache, dass es sich um ein Peer-to-Peer-Netzwerk handelt. Die einzelnen Knoten im Netzwerk verfügen dabei über Routing-Tabellen, in welchen dokumentiert ist, welcher Knoten den angeforderten Inhalt liefern kann. Als effiziente Datenstruktur für diese Routing-Informationen dienen verteilte Hashtabellen [MM02].

Die Incentivierung einzelner Nodes im Netzwerk, Dateien dupliziert vorzuhalten, wird über das Bitswap-Protokoll[2] erreicht. Darüber hinaus setzt das Projekt *Filecoin* [PL17] auf IPFS auf und möchte Anwender mit einer eigenen Kryptowährung dazu bewegen, Speicherkapazitäten für IPFS gegen Bezahlung zur Verfügung zu stellen.

Ein weiteres dezentrales Dateisystem ist Ethereum Swarm[3]. Swarm ähnelt IPFS und setzt auf eine tiefe Integration mit der Ethereum Blockchain. Allerdings ist Swarm zum

[2] https://github.com/ipfs/go-ipfs/tree/master/exchange/bitswap
[3] https://swarm-guide.readthedocs.io/en/latest/introduction.html

Zeitpunkt dieser Ausarbeitung in der Proof-of-Concept-Version 0.2 verfügbar, wodurch keine stabile Grundlage für eine Machbarkeitsaussage gewährleistet ist.

2.3 Verwandte Arbeiten

Im Rahmen der Forschung in den Bereichen IoT und Blockchain wurden bereits einige Arbeiten angefertigt, die sich mit der Fragestellung der Bereitstellung von Software-Updates für IoT-Geräte beschäftigt haben. Im Folgenden sollen zwei Arbeiten, die für die hier erstellte Lösung relevant waren, vorgestellt werden.

Der erste Ansatz [Bo17] beschreibt, wie es theoretisch möglich ist, durch die Blockchain-Technologie bessere Verfügbarkeit für Software-Update-Systeme zu gewährleisten. Darin wird propagiert, die Update-Binaries verschlüsselt mitsamt Metainformationen in einer Blockchain-Transaktion zu veröffentlichen. Die Endgeräte prüfen periodisch, ob es eine neue Update-Transaktion gegeben hat und installieren das Update, wenn es verfügbar ist. Um die Installationsvorgänge nachvollziehen zu können, sollen die Ergebnisse von den Endgeräten ebenfalls per Transaktion in der Blockchain gespeichert werden. Eine Variation sieht vor, dass die Updates nach ihrer Veröffentlichung von einem festgelegten Schwellwert an unabhängigen Prüfstellen in der Blockchain als unbedenklich markiert werden müssen, bevor diese von Endgeräten heruntergeladen werden dürfen. Insgesamt liefert dieser Ansatz einige theoretische Ideen, die in dieser Arbeit weiterverfolgt wurden, lässt allerdings die Frage nach konkreten Blockchain-Implementierungen zur Umsetzung offen.

Der zweite Ansatz [LL16] nutzt die Blockchain-Technologie nicht zum Verteilen der Updates, sondern ausschließlich zur Verifikation. Deshalb wird lediglich der Hashwert des Update-Binaries in der Blockchain gespeichert. Das Update selbst soll aus einem Peer-to-Peer-Netzwerk heruntergeladen werden. Dieser Ansatz sieht private Blockchain- und Peer-to-Peer-Netzwerke der Endgeräte untereinander vor. Dadurch wird dieser Ansatz interessant für geschlossene Systeme. Sollten sich die einzelnen Geräte allerdings nicht im selben System befinden, ist die Gefahr vorhanden, dass die Blockchain die kritische Größe nicht erreicht, um eine nachträgliche Veränderung durch einen Angreifer ausschließen zu können.

3 Analyse

Dieses Kapitel befasst sich mit dem Requirements Engineering für das zu entwickelnde Software-Update-System.

Um einen Überblick über benötigte Funktionalitäten zu erhalten, ist ein Anwendungsfalldiagramm (Abb. 2) erstellt worden, welches die fachlichen Anforderungen an das Software-Update-System veranschaulicht. So ist der Ansatz der unabhängigen Software-Prüfung aus [Bo17] übernommen worden. Ferner soll es sowohl den Herstellern als auch

den Besitzern von Endgeräten möglich sein, den aktuellen Stand der installierten Software im Software-Update-System nachvollziehen zu können. Der Zugriff aus Sicht der Endgeräte soll über das Pull-Prinzip realisiert werden, wobei die Endgeräte je nach Ausprägung entweder autark oder gesammelt durch ein Gateway auf das Software-Update-System zugreifen können.

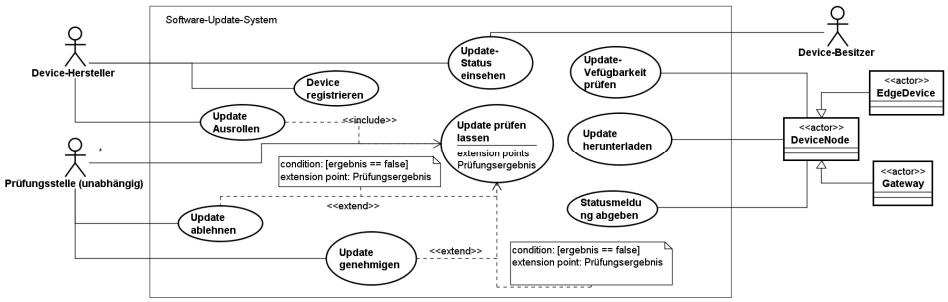

Abb. 2: Anwendungsfalldiagramm

Anhand dieser Überlegungen ergeben sich die folgenden funktionalen Anforderungen:

- Ein Hersteller soll Updates (verschlüsselt) im Software-Update-System ablegen können.
- Der Veröffentlichung soll eine Überprüfung des Updates (Integrität, Signaturprüfung, Codeanalyse) durch unabhängige Knoten vorausgehen können.
- Endgeräte sollen regelmäßig abfragen können, ob neue Softwareversionen für sie vorhanden sind (Pull-Prinzip).
- Endgeräte sollen neue Softwareversionen aus dem Software-Update-System herunterladen und deren Integrität verifizieren können. Außerdem soll gewährleistet sein, dass es sich beim Urheber der Software um eine vertrauenswürdige Quelle handelt.
- Das Software-Update-System soll eine Geräteverwaltung enthalten, in der der Hersteller Metainformationen (IDs, öffentliche Schlüssel) für jedes Endgerät pflegen und dynamisch hinzufügen kann, um Signaturen bei eingehenden Nachrichten der Geräte prüfen und ausgehende Nachrichten für die Endgeräte verschlüsseln zu können.
- Nach einer erfolgten Updateinstallation soll eine Statusmeldung des Endgeräts im Software-Update-System vermerkt werden.
- Der Software-Status der Endgeräte soll für den Hersteller sowie den Besitzer der Endgeräte per Weboberfläche einsehbar sein.

Außerdem sind die folgenden nicht-funktionalen Anforderungen zu berücksichtigen:

- Das Software-Update-System soll resistent gegen DDoS-Attacken sein.
- Alle Informationen im Software-Update-System sollen unveränderlich abgelegt werden.
- Ein modularer Aufbau ist zwecks Wiederverwendbarkeit und Wartung des Codes einzuhalten.
- Ökonomische Besonderheiten bei der Nutzung von Blockchain-Technologie sind zu betrachten und zu optimieren.

4 Entwurf und Implementierung

Dieses Kapitel behandelt den Entwurf sowie die prototypische Implementierung des Software-Update-Systems, welches die im voranstehenden Kapitel entwickelten Anforderungen umsetzen soll. Zuerst werden Technologieentscheidungen begründet. Danach wird aufgezeigt, wie eine Integritätsprüfung des Updates mit dem System ablaufen kann. Das System kann in die Smart Contracts Endgeräteverwaltung und Updateverwaltung unterteilt werden, welche ebenfalls Teil dieses Kapitels sind. Für den Zugriff auf die Contracts sind eine Schnittstelle sowie eine Weboberfläche konzipiert worden, die abschließend beschrieben werden.

4.1 Technologieauswahl

Als Blockchain-Technologie zur Umsetzung des Anwendungsfalles wurde aufgrund der Fähigkeit zur Erstellung von Smart Contracts, der Touring-kompletten Programmiersprache Solidity, der erreichten kritischen Masse zum Schutz vor DDoS-Attacken, der Größe der Entwickler-Community sowie der Auswahl an Entwicklungsframeworks wie etwa Truffle, Ethereum ausgewählt. Grundlagen zu Ethereum sind in Kapitel 2 aus [An17] zu finden. Da jede Berechnung und der benötigte Speicher eines Smart Contracts Gas kosten und solche Transaktionen mit hohen Gas-Preisen bevorzugt werden, ist es ökonomisch sinnvoll, möglichst wenig Gas zu verbrauchen [Da17]. Aus diesem Grund sollen die Update-Dateien außerhalb der Blockchain persistiert werden. Um trotzdem Dezentralität zu gewährleisten, wird der in Abschnitt 2.2 diskutierte dezentrale Speicher IPFS verwendet. Für die Schnittstelle zu den Smart Contracts wurde das serverseitige JavaScript-Framework Node.js[4] ausgewählt, da sowohl Truffle[5], Ethereum[6] als auch IPFS[7] API-Pakete für Node.js anbieten und diese im Vergleich zu APIs in anderen Programmiersprachen die, aus Sicht des Autors, verständlichsten Dokumentationen anbieten.

[4] https://nodejs.org/en/about/
[5] https://github.com/trufflesuite/truffle-contract
[6] https://github.com/ethereum/web3.js/
[7] https://github.com/ipfs/js-ipfs

4.2 Integritätsprüfung

Die Prüfung der Integrität des Updates sowie jeder Kommunikation eines Knotens mit dem Software-Update-System kann mit Hilfe von asymmetrischer Kryptographie erreicht werden. Der Vorgang der Sicherung und Prüfung der Integrität eines Updates ist nachstehendem Sequenzdiagramm zu entnehmen.

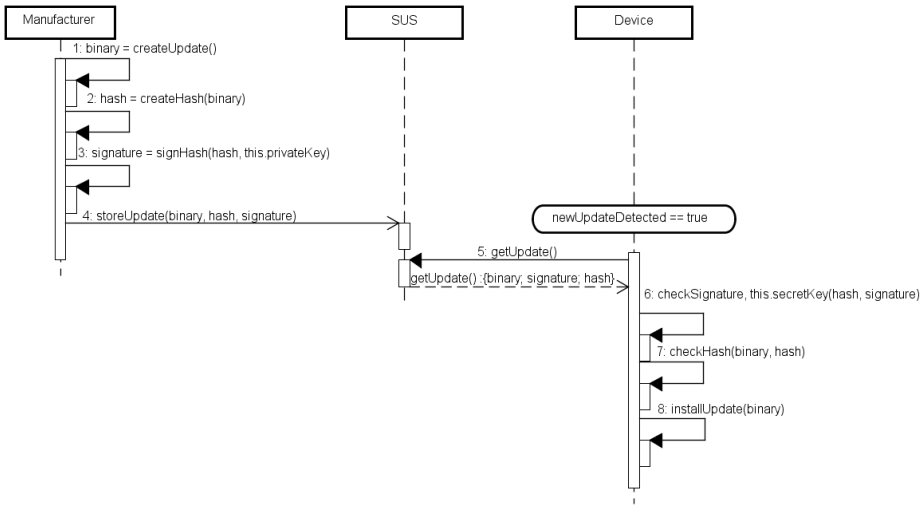

Abb. 3: Sequenzdiagramm Integritätsprüfung

Der Hersteller erstellt ein Schlüsselpaar. Auf jedem Endgerät wird vor dessen Auslieferung der öffentlichen Schlüssel des Herstellers dauerhaft persistiert. Nun fertig der Hersteller für jedes Update einen Hashwert an und signiert diesen mit seinem privaten Schlüssel. Dann werden die Artefakte entweder vom Hersteller direkt oder von einer unabhängigen Prüfstelle an das Software-Update-System übergeben. Das Endgerät kann nun die Artefakte laden, die Signatur mit dem öffentlichen Schlüssel des Herstellers prüfen und danach einen Hashwert des Updates bilden und mit dem übergebenen Wert vergleichen. Im Erfolgsfall kann das Update dann installiert werden.

Bei Bedarf kann das Update auch nach Erstellung des Hashwertes noch mit dem privaten Schlüssel des Herstellers verschlüsselt werden, um den unter Umständen öffentlich verfügbaren Quellcode des Updates der Allgemeinheit unzugänglich zu machen. Eine andere Lösung für verschlüsselte Felder in einer öffentlichen Blockchain wurde in [Wo18] entwickelt.

Um auch die Kommunikation vom Endgerät zurück zum Software-Update-System sicher durchführen zu können, kann ebenfalls mit Signaturen gearbeitet werden, die das System mit dem jeweiligen öffentlichen Schlüssel des Endgerätes prüfen kann.

4.3 Updateverwaltung

Die Updateverwaltung wird durch einen Smart Contract in der Ethereum-Blockchain abgebildet. Die Persistierung der tatsächlichen Updatedatei erfolgt aus Kostengründen wie bereits thematisiert über IPFS. Der Hersteller legt also lediglich Metadaten zu seinen Updateversionen im Smart Contract ab. Endgeräte können aus dem Smart Contract die Metadaten der jeweils aktuellsten Version abrufen und prüfen, ob ihre installierte Version der neuesten Version entspricht. Das folgende Klassendiagram visualisiert den Aufbau des Smart Contracts. Die dargestellten Variablen und Methoden werden im Anschluss erläutert.

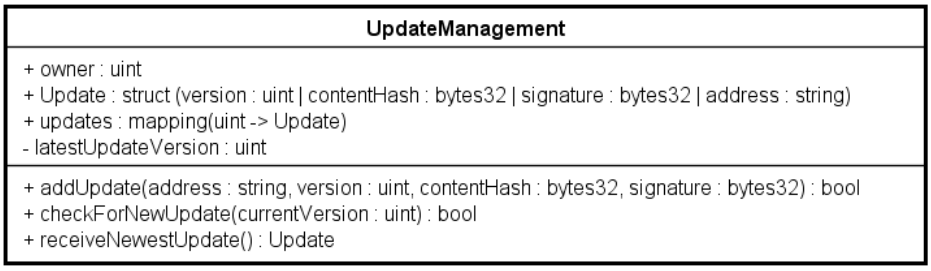

Abb. 4: Klassendiagramm Updateverwaltung

Die Variable *owner* hält den öffentlichen Schlüssel des Ethereum-Accounts, welcher den Smart Contract aktiviert hat. Darüber kann mit Hilfe sogenannter *Modifier* sichergestellt werden, dass nur der Ersteller des Contracts – in diesem Fall der Gerätehersteller – die Methode *addUpdate* ausführen darf. Die Variable *Update* definiert einen eigenen Datentyp für Metainformationen zu einem Update. Dieser enthält die Versionsnummer, den Hashwert der Updatedatei, eine Signatur über den Hashwert und die Adresse, unter welcher die Datei im IPFS zu erreichen ist. Die Variable *updates* speichert ein Mapping von Versionen zu Updates. Da es in Solidity nicht möglich ist, über solche Mappings zu iterieren, wird für den Zugriff auf das jeweils neueste Update dessen Version in der privaten Variable *latestUpdateVersion* vorgehalten. Die Methode *addUpdate* erlaubt es nun dem *owner*, Updatemetadaten in dem Contract abzulegen. Die anderen beiden Methoden dienen dem Zugriff durch die Updateempfänger. Über *checkForNewUpdate* kann überprüft werden, ob es eine neue Version der Software gibt. Dadurch wird das Pull-Prinzip der Updateverteilung realisiert. Die letzte Methode ermöglicht nun das Abrufen der Metadaten zum aktuellsten Update.

4.4 Geräteverwaltung

Die Geräteverwaltung ist ebenfalls durch einen Smart Contract in der Blockchain abgebildet. In diesem können Hersteller Endgeräte registrieren. Außerdem wird die Firmwareversion dort aktualisiert, wenn ein Endgerät ein Update erfolgreich aufgespielt hat.

Aus dem Smart Contract wird die geforderte Weboberfläche mit den Informationen zu den Versionsnummern der einzelnen Endgeräte gespeist. Ebenso werden die öffentlichen Schlüssel zu jedem Endgerät gespeichert, um Signaturprüfungen bei eingehenden Nachrichten zu ermöglichen. Die Variablen und Methoden der Geräteverwaltung, die in folgendem Klassendiagramm visualisiert wurden, werden analog zum ersten Smart Contract nachfolgend erläutert.

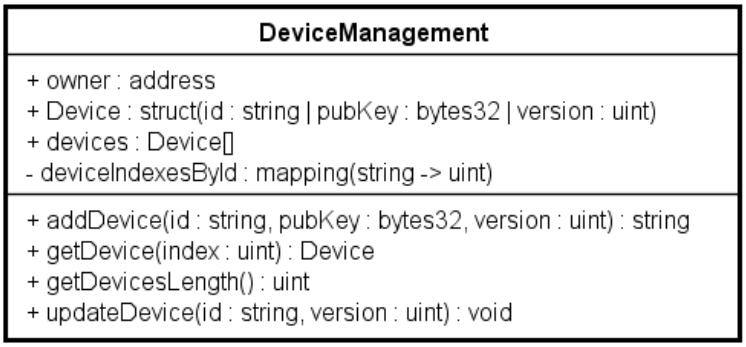

Abb. 5: Klassendiagramm Geräteverwaltung

Der Smart Contract verfügt ebenfalls durch die Variable *owner* über eine Referenz auf den Ersteller des Contracts, um sicherzustellen, dass nur der Hersteller Informationen hinzufügen kann. Die restlichen drei Variablen dienen der Speicherung der Geräteinformationen: *Device* ist ein lokaler Datentyp bestehend aus der ID, dem öffentlichen Schlüssel und der Firmwareversion. Das Array *devices* speichert alle im Contract enthaltenen Endgeräteinformationen. Das Mapping *devicesIndexedById* speichert den Index eines Structs im Array mit dessen ID als Schlüssel. Dieses Vorgehen ist notwendig, um ein *Device* anhand seiner ID aus dem Array auslesen zu können. Die Methode *addDevice* erlaubt dem *owner* das Hinzufügen eines Endgerätes in das Array. Die nächsten beiden Methoden dienen dem Auslesen aller gespeicherten Informationen. Dabei muss zunächst die Gesamtlänge des Arrays abgefragt werden, bevor über die Indexe *0* bis *Gesamtlänge – 1* die Informationen nacheinander abgefragt werden können. Diese ineffiziente Lösung ist notwendig, da es in der verwendeten Version der Programmiersprache nicht möglich ist, ein aus Structs bestehendes Array zurückzugeben. Die Methode *updateDevice* setzt die *version* des Structs mit der übergebenen ID auf die übergebene Nummer.

4.5 Server-Schnittstelle

Als Fassade [Ga15] zur Kapselung der Zugriffe auf die beiden Smart Contracts sowie IPFS als dezentrales Speichersystem ist mit Node.js eine Server-Schnittstelle entwickelt worden. Auf diese Schnittstelle können Endgeräte beziehungsweise deren Gateways sowie die Hersteller der Updates via HTTP-Requests zugreifen. Diese Art der API wur-

de gewählt, da davon ausgegangen wird, dass die Schnittstelle innerhalb eines sicheren Firmennetzwerkes eingesetzt wird. Außerdem kann programmiersprachenunabhängig von den Endgeräten beziehungsweise den Gateways auf die API zugegriffen werden, solange die verwendete Programmiersprache HTTP-Requests unterstützt. Eine Voraussetzung für den Zugriff auf die Smart Contracts ist, dass im selben System ein lokaler Ethereum-Knoten, beispielsweise über *testrpc*,[8] verfügbar ist.

Die wichtigsten verwendeten NPM-Pakete der Schnittstelle sind *web3*, *truffle-contract*, *express*[9] und *ipfs*. Das Paket *web3* implementiert eine JavaScript API für den Zugriff auf Ethereum über die JSON RPC Spezifikation[10]. Im Prototypen ermöglicht *web3* Zugriff auf den Ethereum-Account, mit dem der Knoten betrieben wird und verbindet die Smart Contracts mit den Abstraktionen, die *truffle-contract* bereitstellt. Diese Abstraktionen werden aus den JSON-Repräsentationen, die das Truffle-Framework aus den entwickelten Contracts erstellt, instanziiert und bietet unter anderem Methoden zum Aufruf einzelner Funktionen des Contracts und zur Verarbeitung der Rückgabewerte. Um die Asynchronität der Blockchain-Transaktionen zu beherrschen, ermöglicht *truffle-contract* die Arbeit mit Promise-Objekten. Das Modul *ipfs* ist eine JavaScript-Implementierung des IPFS-Protokolls. Diese Implementierung erlaubt es programmatisch, einen IPFS-Knoten zu starten sowie diesem Dateien hinzuzufügen und im Gegenzug deren Hashadressen zu erhalten. Mit diesen Hashadressen ist es daraufhin möglich, die referenzierten Dateien aus dem IPFS-Netzwerk auszulesen. Für den HTTP-Zugriff auf die in den Abschnitten 4.3 und 4.4 erläuterten Methoden der Contracts sowie den Upload und Download der Updatedateien wurde das Framework *express* verwendet, um einen Webserver zu starten und diesem einen Router für HTTP-Anfragen hinzuzufügen.

4.6 Weboberfläche

Die in Kapitel 3 als Anforderung erfasste Weboberfläche zur Ansicht der Geräteinformationen aus dem Smart Contract *Geräteverwaltung* ist mit Node.js realisiert worden. Der Zugriff auf den Smart Contract wird analog zur Server-Schnittstelle über die beiden Pakete *web3* und *truffle-contract* implementiert. Aus diesem Grund setzt auch dieses System wieder einen aktiven Ethereum-Client voraus, um den auf der Blockchain bereitgestellten Smart Contract erreichen zu können. Das Design der Oberfläche wurde im Sinne einer prototypischen Anwendung minimalistisch gehalten, sodass die Oberfläche im Wesentlichen aus einer Tabelle besteht. Diese verfügt über eine Zeile pro registriertem Endgerät. Jede Zeile besitzt Spalten für die jeweilige ID, den öffentlichen Schlüssel und die Firmwareversion. Diese Informationen werden beim Ladevorgang wie in Abschnitt 4.4 erläutert über eine Verkettung der Methoden *getDevicesLength* und *getDevice* gewonnen und dynamisch in die Tabelle gerendert. Um die asynchronen Blockchain-Transaktionen in die benötigte synchrone Reihenfolge zu bringen, wird auch an dieser Stelle mit Promise-Objekten gearbeitet. Um eine standardisierte und responsive Darstel-

[8] https://www.npmjs.com/package/ethereumjs-testrpc
[9] http://expressjs.com/de/
[10] https://github.com/ethereum/wiki/wiki/JSON-RPC

lung zu garantieren, wurde das Framework *Bootstrap*[11] eingesetzt.

5 Ergebnisdiskussion

Der Entwurf sowie die prototypische Implementierung des Software-Update-Systems sollen an dieser Stelle kritisch reflektiert werden. Es lässt sich festhalten, dass basierend auf zwei theoretischen Arbeiten zu diesem Thema ein konkreter Ansatz für ein Software-Update-System auf Ethereum-Basis entwickelt worden ist. Darüber hinaus erfolgte eine prototypische Implementierung. Damit wurde eine Basis geschaffen, die zeigt, dass es konkret möglich ist, Updates für IoT-Geräte mit Hilfe einer Blockchain zu verteilen und damit diverse Cyber-Angriffe abzuwehren. Die Voraussetzung dazu ist allerdings, dass die Bedingungen der asynchronen Verschlüsselung eingehalten werden. Da es sich jedoch um einen Prototyp handelt, sollen an dieser Stelle einige der Aspekte genannt werden, die nicht, beziehungsweise nur vereinfacht implementiert wurden: Zuerst ist der Prototyp in der Entwicklung über *testrpc* auf einer lokalen Ethereum-Implementierung bereitgestellt worden und es ist nicht auszuschließen, dass für ein erfolgreiches Deployment auf der öffentlichen Haupt-Blockchain Anpassungen im Code vorzunehmen sind. Die in Kapitel 4.2 diskutierte Möglichkeit der Signaturprüfung bei eingehenden Nachrichten von den Endgeräten ist durch das Speichern der öffentlichen Schlüssel aller registrierten Geräte zwar technisch möglich, jedoch noch nicht konkret implementiert worden. Die Möglichkeit einer fehlerhaften Update-Installation und einer negativen Statusrückmeldung ist im Rahmen dieser Machbarkeitsstudie ebenfalls nicht implementiert worden. Auch ist nicht auszuschließen, dass das Swarm-Projekt in naher Zukunft soweit entwickelt ist, dass es IPFS in seinen Möglichkeiten überholt und austauschbar macht. Eine vorherige Authentifizierung vorausgesetzt, wäre es auch möglich, die Weboberfläche komplett dezentral in IPFS oder Swarm zu betreiben, um auch an dieser Stelle von den Vorteilen der Dezentralität zu profitieren.

6 Zusammenfassung und Ausblick

Zusammenfassend lässt sich sagen, dass in dieser Arbeit die Relevanz des Themas betrachtet wurde und anschließend die theoretischen Grundlagen für die weitere Arbeit – Dezentrale Dateisysteme, Software-Updatesysteme, sowie die verwandten Arbeiten – gelegt wurden. Danach wurden Anwendungsfälle für das zu entwickelnde Software-Update-System konstruiert und Anforderungen abgeleitet. Um diese Anforderungen erfüllen zu können, wurde Ethereum als Blockchain-Technologie und IPFS als dezentrales Dateisystem ausgewählt und ein Konzept zur Integritätsprüfung vorgelegt. Die Implementierung erfolgte modular über je einen Smart Contract für die Updateverwaltung und einen für die Geräteverwaltung. Darauf aufbauend wurden eine HTTP-Schnittstelle und eine Weboberfläche für den Contract-Zugriff, die beide mit Node.js umgesetzt wur-

[11] https://getbootstrap.com/

den, entwickelt. Schließlich wurde das Ergebnis bewertend diskutiert. In Zukunft sollte versucht werden, die Contracts auf der öffentlichen Blockchain bereitzustellen und die Anwendung um weitere Fähigkeiten zu ergänzen. Das vorhergehende Kapitel fasst schon einige der ausstehenden beziehungsweise möglichen neuen Themen zusammen. Außerdem könnte zum Beispiel die Weboberfläche um das manuelle Hinzufügen von Updates erweitert werden, um die Benutzerfreundlichkeit zu erhöhen.

Literaturverzeichnis

[An18] Andres, C.; Herkenhoff, T.; Wallersheim, M.; Woywod, T.; Wörtler, F.: ConCar – Eine Verbindung von IoT und Blockchain, Abschlussbericht FEP 2018 S. 1-12, 2018.

[Be15] Benet, J.: IPFS – Content Adressed, Versioned, P2P File System (Draft 3), Whitepaper, https://arxiv.org/pdf/1407.3561.pdf, 2014.

[Bo17] Boudguiga, A. et.al.: Towards Better Availability and Accountability for IoT Updates by means of a Blockchain, IEEE Security & Privacy on the Blockchain (IEEE S&B 2017) an IEEE EuroS&P 2017 and Eurocrypt 2017 affiliated workshop, 2017.

[Da17] Dannen, C.: Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners, S. 58ff., Apress, 2017.

[CPJ16] Cox, J.; Johnston, S.; Poulter, A.: SRUP: The Secure Remote Update Protocol, https://eprints.soton.ac.uk/402143/, 2016.

[Ga15] Gamma, E. et.al.: Design Patterns – Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software, mitp, 2015.

[LL16] Lee, B.; Lee, J.-H.: Blockchain-based secure firmware update for embedded devices in an Internet of Things environment, Springer Science+Business Media, New York, 2016.

[MM02] Maymounkov, P; Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric, Whitepaper, https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf, 2002.

[MMT17] Moran, B.; Meriac, M.; Tschofenig, H.: A Firmware Update Architecture for Internet of Things Devices, https://tools.ietf.org/id/draft-moran-suit-architecture-00.html, 2017, abgerufen am 13.01.2018.

[PL17] Protocol Labs: Filecoin: A Decentralized Storage Network, Whitepaper, https://filecoin.io/filecoin.pdf, 2017.

[Si16] Simmons, C.: Software update for IoT – the current state of play, https://events.static.linuxfound.org/sites/events/files/slides/software-update-elce-2016-169.pdf, 2016, abgerufen am 13.01.2018.

[Wo18] Woywod, T.: Entwurf und Implementierung einer prototypischen Anwendung für die Persistierung von personenbezogenen Daten auf einer öffentlichen Blockchain, Abschlussbericht FEP 2018 S. 13-24, 2018.

Datensicherung auf einem Edge Device Instant Cluster: Automatische Gerätekoordination und Clusterbildung von Edge Devices in einer Industrie 4.0-Umgebung zur Sicherung von IoT-Sensordaten

Florian Wörtler[1]

Abstract: Diese Arbeit beschäftigt sich mit dem automatischen Zusammenschluss von Edge Devices. Das Ziel ist es, dass sich Edge Devices in Cluster zusammenschließen, um gegenseitig Datensicherung zu betreiben. Dazu werden zunächst die dezentralen Speichersysteme Tahoe-LAFS, Infinit und GlusterFS hinsichtlich ihrer Eignung analysiert und GlusterFS ausgewählt. Mit Hilfe von DHCP-Triggern wird ein zentraler Konfigurationsdienst entworfen, der GlusterFS konfiguriert und die Edge Devices zu einem dezentralen Speichersystem ohne einen Single Point of Failure konfiguriert. In dem Speichersystem können die Daten mehrfach vorgehalten werden, sodass einige Edge Devices ausfallen können, ohne dass es zu Datenverlust kommt.

Keywords: Dezentrales Speichersystem, dezentrales Dateisystem, P2P, Edge Device, IoT, Industrie 4.0, Device Coordination, Sensordaten zwischenspeichern

1 Einleitung

In einer IoT-Umgebung fallen viele Daten auf vielen Edge Devices an, die in einigen Fällen erst zwischengespeichert werden müssen, bevor sie verwendet werden [An18]. Diese Arbeit beschäftigt sich mit dem Zwischenspeichern von Daten auf Edge Devices im Rahmen des Edge Computings. Das Ziel ist es, eine Lösung zu entwickeln, die die Edge Devices koordiniert und die Daten sicher und redundant speichert.

Die Entwicklung der Lösung kann in zwei Teilbereiche aufgeteilt werden. Zum einen sollen die Daten redundant gespeichert werden, damit der Ausfall eines Edge Devices nicht zu Datenverlust führt. Zum anderen sollen sich die Edge Devices im (W-)LAN automatisch koordinieren, sodass eine manuelle Konfiguration der großen und veränderlichen Zahl an Edge Devices nicht erforderlich ist. Anforderung ist es, dass die Edge Devices Gruppen der Größe *n* bilden, oder sich anderweitig organisieren, um die Daten *n*-fach vorzuhalten. Dies wird als Instant Clustering bezeichnet.

Für die erste Teillösung, der Speicherung der Daten, werden verschiedene verteilte Dateisysteme bezüglich ihrer Eignung für die Anforderungen analysiert (siehe Kapitel 4). Die Lösung des zweiten Teils, dem Instant Clustering, wird in Kapitel 5 entworfen und teilweise implementiert.

[1] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, fw288591@fh-muenster.de

2 Grundlagen

Edge Computing nutzt im Gegensatz zu Cloud Computing die vorhandenen Verarbeitungsressourcen in Edge Devices. Das Ziel ist es, die Cloud-Infrastruktur zu erweitern, um neben der erweiterten Rechenkapazität Kosten einzusparen und zu verhindern, dass sensible Daten das lokale Netzwerk verlassen. [TM17]

Edge Devices (in dieser Arbeit auch Geräte genannt) im Kontext von IoT sind Geräte, die Sensoren mit dem Netzwerk verbinden. Sie befinden sich also am Rand des Netzwerkes und bilden die „Edges“. Die Aufgabe von Edge Devices ist es, Daten von direkt angeschlossenen Sensoren entgegenzunehmen. Anschließend kann eine Vorverarbeitung durchgeführt werden, bevor die Daten in eine Cloud oder an einen anderen Ort versendet werden. Dabei übersetzen sie die Protokolle der Sensoren in andere Netzwerkprotokolle. Auch der entgegengesetzte Weg, dem Senden von Daten zu Aktoren, ist möglich. [BS15], [TM17]

3 Verwandte Arbeiten

Es gibt verwandte Arbeiten, die sich mit automatischer Gerätekonfiguration beschäftigen. Diese setzen jedoch andere Technologien ein und gehen anders vor. In [KTK10] wird ein „secured grid overlay network“ erstellt, welches mit Multicasts kommuniziert. Die Authentifizierung erfolgt über Hubs. Wie genau wird leider nicht spezifiziert. Der Netzwerk-Verkehr wird mit RSA-Schlüsseln verschlüsselt. Es gibt Gruppenschlüssel, um eine Art Autorisierung umzusetzen.

IBM stellt in einem Paper [CCA15] ein sehr interessantes Produkt vor, welches sich mit den Kernthemen diese Arbeit beschäftigt (P2P-Messaging, verteilter Datentransfer und automatische Gerätekoordination). Für die Datenverteilung wird BitTorrent verwendet. Die automatische Gerätekoordination wurde mit Smart Contracts auf Ethereum-Basis implementiert. Es handelt sich bei dem Paper eher um Marketingfolien, weshalb technische Details zur Umsetzung leider nicht genannt werden. Beispielsweise bleibt offen, wie ein neues Gerät so konfiguriert wird, dass es auf die Konfigurationsdaten in der Blockchain zugreifen kann.

4 Analyse und Auswahl eines Speichersystems

In diesem Kapitel werden zunächst die Anforderungen erhoben, anschließend werden die relevantesten Speichersysteme analysiert und das geeignetste ausgewählt.

Die Edge Devices, für die diese Lösung entwickelt wird, sind sehr leistungsschwache Geräte. Es handelt sich um Mini-Rechner mit sehr sparsamen, passiv gekühlten Prozessoren, 2 GB Arbeitsspeicher und einer Festplatte in Form einer SD-Karte. Als Betriebs-

system wird CentOS ohne GUI verwendet. Bei der Auswahl des geeigneten Speichersystems ist es also von Bedeutung, dass die leistungsschwache Hardware berücksichtigt und eine möglichst leichtgewichtige Lösung ausgewählt wird.

Verteilte Speicher- oder Dateisysteme eignen sich gut für diesen Anwendungsfall, da sie im Gegensatz zu lokalen Dateisystemen die Aufgabe übernehmen, Daten über mehrere Speichergeräte zu verteilen.

Dezentrale Speicher- oder Dateisysteme, auch Peer-to-Peer-Dateisystem (P2P-Dateisystem) genannt, sind eine Unterkategorie von verteilten Speichersystemen und eignen sich noch besser für diese Lösung, weil jeder Teilnehmer des verteilten Speichersystems Server und Client gleichzeitig sein kann und einzelne Server nicht zum zentralen, verwaltenden Server ernannt werden müssen. Dies vermeidet die Entstehung eines Single Point of Failures (SPF) und es entfallen keine Ressourcen darauf, dass einige der Edge Devices verwaltende Aufgaben übernehmen müssen. Dies ist zum Beispiel beim Hadoop Distributed File System Name Node der Fall, der ein SPF darstellt und selbst nur verwaltende Aufgaben übernimmt [Ve13].

Wichtig für das Speichersystem ist auch, dass es kontrolliert ist. Einige Speichersysteme bieten diese Fähigkeit nicht. Zum Beispiel bietet das P2P-Netzwerke BitTorrent von sich aus keine Verwaltung über die eigenen Daten an. Es sind manuelle Eingriffe notwendig, um sicherzustellen, dass eine Datei von einer gewissen Anzahl an Teilnehmern gespeichert wird [KCC11]. Auch werden die Daten innerhalb des Netzwerkes nicht aktiv verwaltet. Einmal geschriebene Dateien verweilen dort und werden nur durch Vergessen gelöscht[2] . Eine Datei wird vergessen, wenn sie eine längere Zeit nicht mehr nachgefragt worden ist. Ein weiterer Aspekt ist die Authentifizierung. Die Geräte sollen die Möglichkeit haben ihre Authentizität gegenseitig festzustellen (z.B. mit Signaturen) oder die Daten mit einer Verschlüsselung vor fremden Zugriffen zu schützen. Außerdem soll es einen konfigurierbaren Replikationsfaktor für die redundante Speicherung der Daten geben und das Speichersystem soll sich nach Ausfall eines Speicherknotens selbst heilen, um den konfigurierten Replikationsfaktor wieder zu erreichen.

Konsistenz ist für viele Speichersysteme ein wichtiger Faktor. Aufgrund der Tatsache, dass die Sensordaten in eine Datei gespeichert werden und diese abschließend geschrieben wird, gibt es keine Änderung an den Dateien und somit ist die Konsistenz nicht so wichtig wie die Verfügbarkeit des Dateisystems.

Aus diesem Grund werden verteilte Speichersysteme, die nicht dezentral und nicht kontrolliert sind, aus der Betrachtung ausgeschlossen. Übrig bleiben einige kontrollierte und dezentrale Speichersysteme. Eine Vielzahl an Speichersystemen konnten nach einer kürzeren Analyse ausgeschlossen werden. Am Ende bleiben drei Speichersysteme (Tahoe-LAFS, Infinit und GlusterFS) übrig, die die für diese Umsetzung relevantesten Speichersysteme sind. In den folgenden Kapiteln werden diese genauer analysiert.

[2] https://blog.docker.com/2017/01/docker-storage-infinit-faq/

4.1 Tahoe-LAFS

Tahoe Least-Authority File System (Tahoe-LAFS) ist ein verteiltes, dezentrales Speicher-System auf Open-Source-Basis. Bei der Entwicklung wurde das Kriterium Hochverfügbarkeit stärker gewichtet als Konsistenz[3]. Tahoe-LAFS stellt sich nach dem Ausfall eines Nodes selbst wieder her, was jedoch ein langwieriger Prozess sein kann[4]. [SF14]

Die Daten werden für die einzelnen Benutzer verschlüsselt, sodass andere (z.B. Besitzer eines Nodes) die Daten nicht einsehen können. Außerdem gibt es ein Autorisierungssystem sowie Dateisignaturen, die unerlaubte Dateiveränderungen verhindern. [SF14]

Tahoe-LAFS beherrscht Erasure Coding, einen Algorithmus, der die Dateien in Fragmente aufteilt [CR05]. Das Erasure Coding spart gegenüber einer einfachen Duplizierung der Daten Speicherplatz. [SF14]

Die Nodes von Tahoe-LAFS können verschiedene Rollen einnehmen. Ein Node übernimmt auch die Rolle des Coordinator Nodes. Ihn gibt es nur einmal und er stellt somit einen SPF dar. Der Coordinator Node ist der sogenannte Introducer. Er ist der Einstiegspunkt für neue Server oder neue Clients, die sich beim Introducer melden und eine Liste der verfügbaren Speicherknoten erhalten. Ein Ausfall des Coordinator Nodes bedeutet, dass keine neuen Clients oder Server dem Netzwerk beitreten können. Bereits vorhandene Nodes sind davon nicht betroffen. Dieser SPF wird daher als unkritisch eingestuft, denn ein bestehendes Speichersystem würde bei einem Ausfall des Coordinator Nodes ohne Einschränkungen weiterarbeiten. [SF14]

Aus Geschwindigkeitsmessungen in [SF14] lässt sich schließen, dass Tahoe-LAFS grundsätzlich langsam ist. Insbesondere für große Dateien ist es schlecht optimiert. Viel Overhead entsteht auch, weil für jede einzelne Datei ein Schlüsselpaar generiert wird, bevor diese gespeichert wird. Zudem bedeutet das Erasue Coding beim Schreiben sowie beim Lesen viel Overhead. Das Nachrichten-Passing wird ebenfalls als ineffizient eingestuft. Es gibt häufige und teure Hash-Suchen beim Lesen und Schreiben von Daten.

Abschließend lässt sich schlussfolgern, dass Tahoe-LAFS ein für diese Lösung interessantes Speichersystem ist, welches über alle nötigen Funktionalitäten verfügt und mit dem Erasure Code-Algorithmus eine ausgeklügelte und platzsparende redundante Speicherung ermöglicht. Eben dieser Algorithmus bringt leider auch signifikante Nachteile mit sich. Die allgemeine Performanz ist langsam und Schreib- sowie Leseoperationen sind mit viel Overhead verbunden. Da nach einem leichtgewichtigen Speichersystem für hardware-schwache Geräte gesucht wird, ist Tahoe-LAFS nicht optimal.

[3] https://github.com/tahoe-lafs/tahoe-lafs/blob/master/docs/specifications/mutable.rst
[4] https://tahoe-lafs.org/trac/tahoe-lafs/wiki/FAQ#Q9_use_raid_with_tahoe_lafs

4.2 Infinit

Infinit[5][6] ist ein dezentrales, fehlertolerantes und flexibles P2P-Speichersystem. Neue Server können unterbrechungsfrei hinzugefügt und entfernt werden. Entwickelt wurde Infinit von der gleichnamigen Firma. Im Dezember 2016 wurde Infinit von Docker gekauft. Obwohl Infinit von Docker so gedacht ist, für Docker-Container benutzt zu werden, ist es weiterhin als eigenständige Lösung verfügbar.

Die Entwicklung von Infinit ist noch nicht abgeschlossen und steht noch am Anfang. Dazu zählen unter anderem noch ausstehende Stresstests, um Resilienz zu garantieren, sowie einige Schnittstellen, die noch nicht fertiggestellt worden sind.

Die Authentifizierung von Benutzern geschieht über RSA-Schlüssel-Paare. Die Nutzdaten werden für jeden Benutzer einzeln verschlüsselt. Berechtigte Benutzer können sogenannte Passports erstellen. Die Passports sind als Einladungen zu verstehen, verbinden andere Benutzer mit dem Netzwerk und haben verschiedene Autorisierungs-Stufen. Durch entsprechende Passports können auch andere Benutzer neue Nodes hinzufügen und das Netzwerk nutzen.

Die Datenübertragung geschieht zur Reduzierung des Traffic-Overheads per UDP. Beim Speichern werden Dateien in Chunks aufgeteilt, verschlüsselt, signiert und auf Speicherknoten verteilt. Dabei lässt sich der Replikationsfaktor einstellen. Bei Ausfall eines Servers wird eine automatische Fehlerbehebung angestoßen und das Speichersystem heilt sich selbst. Durch das Signieren der Daten kann der Client beim Lesen eine ungewollte Änderung erkennen. [Qu10]

Das Infinit-System ist skalierbar und deshalb dafür ausgelegt, sehr groß wachsen zu können. Dies stellt für die Bewahrung der Konsistenz eine Herausforderung dar. Um das Erreichen der Konsistenz zu vereinfachen, tauschen sich jeweils nur relevante Nodes aus. Die Konsistenz muss nur innerhalb eines sogenannten Quorums vorhanden sein. Das Quorum ist eine variable, und damit veränderliche Komposition aus Nodes, deren Anzahl vom Replikationsfaktor abhängt. Ein Block-Konsens wird erreicht, indem sich das Quorum, welches über den konkreten Block verfügt (ihn speichert), einigt. Das Quorum einigt sich in einem transaktionalen Schreibprozess, bestehend aus zwei Commit-Phasen, wenn eine definierte Mehrheit einen Commit aussendet. Dadurch ist Infinit mehrheitlich konsistent und verfügbar, solange genügend Nodes vorhanden sind und kann mit dem Ausfall von Nodes umgehen. [Qu10]

Beim Testen von Infinit hat sich herausgestellt, dass der Infinit Hub mit Infinit sehr eng verzahnt ist. Infinit Hub ist ein zentraler Cloud-Dienst, welcher das Austauschen von Informationen zwischen fremden Benutzern oder Netzwerken vereinfacht. Das Veröffentlichen solcher teils sensibler Informationen auf dem Infinit Hub ist jedoch nicht im

[5] https://infinit.sh/documentation/reference
[6] https://infinit.sh/product

Interesse dieser Lösung. Es ist daher Vorsicht geboten, damit bei der Benutzung nicht versehentlich Informationen auf dem Infinit Hub veröffentlicht werden.

4.3 GlusterFS

Gluster File System (GlusterFS) ist ein skalierbares, verteiltes Open-Source-Dateisystem, welches von Red Hat gepflegt wird. Das Grundgerüst von GlusterFS basiert auf dem FUSE-Projekt. Ursprünglich wurde GlusterFS von der Gluster Company entwickelt, 2011 jedoch von Red Hat aufgekauft. Es ist lizensiert unter der GPL-Lizenz. [XZL15], [Ve13], [JDB12]

Die Skalierung kann sowohl vertikal als auch horizontal bis in den Petabyte-Bereich erfolgen. GlusterFS eignet sich auch für günstige Hardware. Der Fokus von GlusterFS liegt auf Skalierbarkeit und Konsistenz. Im Falle von einzelnen inkonsistenten Nodes gibt es Einschränkungen bei der Verfügbarkeit, bis die Konsistenz wiederhegestellt ist. Eine schwächere Form von Konsistenz, die spätere Konsistenz (engl. eventual consistency), ist konfigurierbar und für diese Lösung ausreichend. [XZL15], [Sa15]

GlusterFS aggregiert diverse Speicherkapazitäten zu einem virtuellen Dateisystem. Auf den einzelnen Nodes wird jeweils ein Dateisystem auf dem lokalen Dateisystem abstrahiert und somit wird ein virtuelles Laufwerk auf dem Echten erstellt. Dies ermöglicht den Zusammenschluss von heterogenen Nodes. Auch Arbeitsspeicher kann als Speicher verwendet werden. Wenn Nodes ausfallen, werden Selbstheilungs-Prozesse gestartet, die im Hintergrund laufen. [XZL15], [Ve13]

Die einzelnen virtuellen Laufwerke werden Bricks genannt. Ein Node stellt mindestens einen Brick bereit. Die Bricks werden zu Volumes zusammengefasst. Dabei können verschiedene Konfigurationen erstellt werden, die auch kombinierbar sind. Es gibt die Möglichkeit Daten zu duplizieren und zufällig über Bricks zu verteilen, Daten zu teilen und die Teile über Bricks zu verteilen oder Bricks zusammenzuschließen um mehr Speicherplatz zu erhalten. [Ve13]

Der Zusammenschluss von Nodes geschieht mit einem Trusted Storage Pool (TSP). In einem TSP befinden sich nur Nodes, denen vertraut wird. Der Beitritt zu einem TSP funktioniert nur, indem ein TSP-zugehöriger Node einen neuen Node hinzufügt. Neue Nodes werden mit Hilfe der IP-Adresse oder, wenn ein DNS-Server verfügbar ist, mit dem Hostnamen gefunden und hinzugefügt. Durch die Benutzung des Hostnamens ergibt sich die Möglichkeit, die Authentifizierung mit Zertifikaten sicherer zu machen. Dazu lassen sich die Nodes über eine Certificate Authority (CA) zertifizieren und andere Nodes können die Echtheit des Zertifikats überprüfen. Auch die Authentifizierung von Clients ist so möglich.

Zur Geschwindigkeitsoptimierung wird Hashing verwendet, um Datei-Anfragen an die passenden Peers zu verteilen [JDB12]. Durchgeführte Geschwindigkeitsmessungen zeigen, dass durch die horizontale Skalierung auch die Geschwindigkeit des Systems

linear steigt. Beim Lesen steigt die Geschwindigkeit deutlich mehr an, als beim Schreiben. Die Größe der Datei ist dabei uninteressant. Insgesamt wird die Geschwindigkeit als gut bewertet. [XZL15]

4.4 Entscheidung für ein Speichersystem

Die drei analysierten Speichersysteme eignen sich grundsätzlich alle für die zu entwickelnde Lösung. Tahoe-LAFS erzeugt durch das Erasure Coding einen großen Overhead, ist dadurch langsamer und deutlich rechenaufwändiger. Infinit und GlusterFS sind schnell und explizit für schwache Hardware entwickelt worden. GlusterFS existiert schon seit langer Zeit, wurde genauestens erprobt, stetig weiterentwickelt und gilt als ausgereift. Diese Tatsache, zusammen mit der zu engen Verzahnung von Infinit mit dem Infinit Hub, lassen die Entscheidung zu, GlusterFS zu benutzen.

5 Automatischer Zusammenschluss von Edge Devices

Das Ziel ist es, dass sich die Edge Devices, die in einem Netzwerk verbunden sind, gegenseitig automatisch finden, verbinden und ein gemeinsames System bilden, um Daten redundant zu speichern. Wenn ein Gerät ausfällt, soll es eine Mindestzahl *n-1* an weiteren Geräten geben, die die Daten gesichert haben. Daher sollten die Geräte Gruppen bilden oder sich anderweitig organisieren, sodass jede Datei auf *n* Geräten gespeichert wird. Dies wird Instant Cluster genannt. Zu einem undefinierten späteren Zeitpunkt werden dann die Daten ausgelesen und weiterverarbeitet. Letzteres ist jedoch nicht Teil dieser Lösung.

5.1 Analyse

GlusterFS ist das zugrundeliegende Speichersystem. Es wird im Folgenden analysiert, wie ein Instant Cluster mit GlusterFS gebildet werden kann. Im nächsten Kapitel wird dann eine konkrete Lösungsmöglichkeit entworfen und implementiert.

Die Analyse, welche Schritte notwendig sind, um ein weiteres Gerät zum Speicher-Netzwerk hinzuzufügen, wurde praktisch in einer Testumgebung durchgeführt. Es lässt sich feststellen, dass ein neues Gerät nur auf Einladung von einem Teilnehmer des Gluster-Netzwerkes beitreten kann.

Die Schwierigkeit besteht darin, dass sich die Geräte automatisch gegenseitig finden müssen. Das Bilden von Gruppen (Cluster-Bildung) ist nicht nötig, denn durch den im GlusterFS konfigurierbaren Replikationsfaktor werden die Daten automatisch entsprechend redundant gespeichert. Wenn ein neues Gerät hinzugefügt wird, muss es von einem bestehenden Teilnehmer bemerkt und die Prozedur zum Hinzufügen des neuen Gerätes zum Gluster-Netzwerk gestartet werden.

5.2 Entwurf und Implementierung

In diesem Kapitel werden Lösungsansätze diskutiert und ein zu entwickelnder Ansatz entworfen und in Teilen implementiert.

Die Herausforderung beim Entwurf besteht darin, dass der Zusammenschluss der Edge Devices zu einem Speichernetzwerk automatisch geschehen soll. Es muss daher ein Weg gefunden werden, wie ein neues Gerät das bestehende Netzwerk erkennt oder von diesem erkannt wird und das Netzwerk das neue Gerät einbindet.

Es wird ein Ansatz entworfen, der eine Eigenentwicklung erfordert, jedoch auch vorhandene Infrastrukturen nutzt. Ein Teil der Netzwerk-Infrastruktur ist ein DNS-DHCP-Server. Der DHCP-Server ist die erste Instanz, die ein neues Gerät in Empfang nimmt und bietet einen idealen Ansatzpunkt, neue Geräte zu erkennen und die nötigen Schritte einzuleiten, um ein neues Gerät hinzuzufügen. Der ISC-DHCP-Server bietet Schnittstellen, die für diesen Anwendungsfall auf Eignung geprüft werden.

Die gesuchte DHCP-Schnittstelle bietet eine Möglichkeit, um sämtliche gültigen DHCP-Einträge (auch Leases genannt) auslesen zu können. Die ausgelesenen Werte können dann aktiv ausgewertet werden, um nach neuen Geräten im Netzwerk zu suchen. Anschließend kann eine Prozedur gestartet werden, die das neue Gerät hinzufügt.

Eine Schnittstelle zum Auslesen der Leases-Datenbank ist das Object Management Application Programming Interface (OMAPI). Dieses kann die Anforderungen nicht erfüllen, denn über das OMAPI kann nur anhand einer bekannten IP-Adresse ein spezifisches Lease ausgelesen werden. Eine Übersicht über alle Leases kann nicht ausgegeben werden[7][8][9].

Die verbleibende Option ist das Verwenden von Triggern in der Konfigurations-Datei *dhcpd.conf* des DHCP-Dienstes. In der Datei können beliebige Befehle ausgeführt werden, wenn ein neues Lease erstellt wird, ein Lease ausläuft oder ein Lease auf Wunsch eines Clients beendet wird. Für diese Lösung wird ein Python-Skript ausgeführt, welches gespeicherte Prozeduren (Stored Procedures) in einer MySQL-Datenbank ausführt und dadurch alle Leases des DHCP-Servers in der Datenbank pflegt.

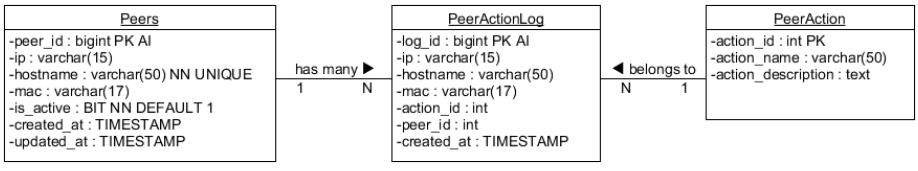

Abb. 1: Domänenmodell der Datenbank

[7] https://www.isc.org/wp-content/uploads/2017/08/dhcp43omapi.html
[8] https://www.systutorials.com/docs/linux/man/1-omshell/
[9] https://linux.die.net/man/1/omcmd

In Abbildung 1 ist das Domänenmodell der Datenbank zu sehen. Die Tabelle *Peers* speichert die Leases des DHCP-Servers unter andrem mit den Attributen *IP* und *Hostname*. In der *PeerActionLog*-Tabelle werden Veränderungen gespeichert, um nachvollziehen zu können, wann welcher Client eine neue IP erhalten hat oder wann der Client sich am DHCP-Server an- oder abgemeldet hat. Die Tabelle *PeerAction* löst die *ActionIds* der *PeerActionLog* in für Menschen verständliche Beschreibungen auf.

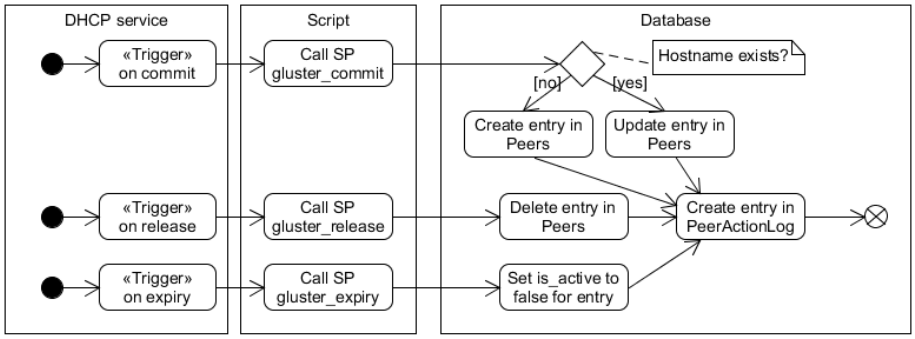

Abb. 2: Aktivitätsdiagramm der Datenbank-Aktualisierungen

Die Datenbank-Aktualisierungen, die durch die DHCP-Trigger[10] ausgelöst werden, sind in Abbildung 2 dargestellt. Die Trigger rufen jeweils das Python-Skript auf, welches eine gespeicherte Prozedur in der Datenbank aufruft. Die gespeicherten Prozeduren aktualisieren die Datenbank und schreiben das *PeerActionLog* fort.

Bei der Implementierung ist aufgefallen, dass das unter anderem dem CentOS zugrundeliegende Sicherheits-Framework AppArmor dem DHCP-Server die Ausführung von ausführbaren Dateien verbieten kann. AppArmor kontrolliert einzelne Anwendungen und kann mit Profilen angepasst werden[11]. Auf Basis einer Dokumentation[12] wurde das Profil angepasst, damit der DHCP-Server das Python-Skript ausführen kann.

Nachdem nun die Datenbank über die immer aktuellen Informationen über das Netzwerk verfügt, wird ein Modul benötigt, welches die Informationen nutzt und das Gluster-Netzwerk entsprechend aktualisiert. Eine Veränderung am Gluster-Netzwerk ist, wie bereits analysiert wurde, nur von einem Peer, der Teil des Netzwerkes ist, möglich. Es bietet sich an, dass der DHCP-Server Teil des Gluster-Netzwerkes ist. Es ist nicht nötig Speicherkapazitäten bereitzustellen, um Teil des Gluster-Netzwerkes sein zu können. Daher reicht für den DHCP-Server eine einfache Basis-Installation von GlusterFS aus. Die Einschränkung, die diese Lösung hat, ist, dass der DNS-DHCP-Server einen Single Point of Failure darstellt. Wenn der Server ausfällt, funktioniert das Netzwerk solange weiter, bis die DNS-Einträge auf den Geräten ungültig werden. Der Ausfall würde auch

[10] https://www.isc.org/wp-content/uploads/2017/08/dhcp41conf.html#REFERENCE:%20EVENTS
[11] https://wiki.ubuntuusers.de/AppArmor/
[12] http://manpages.ubuntu.com/manpages/xenial/en/man5/apparmor.d.5.html

bedeuten, dass automatisch keine neuen Geräte mehr sowohl in das LAN als auch in das Gluster-Netzwerk eingebunden werden können. Es ist also ratsam, diesen Server redundant auszulegen. Die redundante Auslegung von DNS-DHCP-Servern ist aber kein Gegenstand dieser Arbeit. Für diese Lösung ist nur wichtig, dass bei einer redundanten Auslegung des DNS-DCHP-Servers beide Datenbanken konsistent auf dem gleichen Stand sind. Auch empfiehlt es sich die DHCP-Leases mit einer langen Lebenszeit zu versehen.

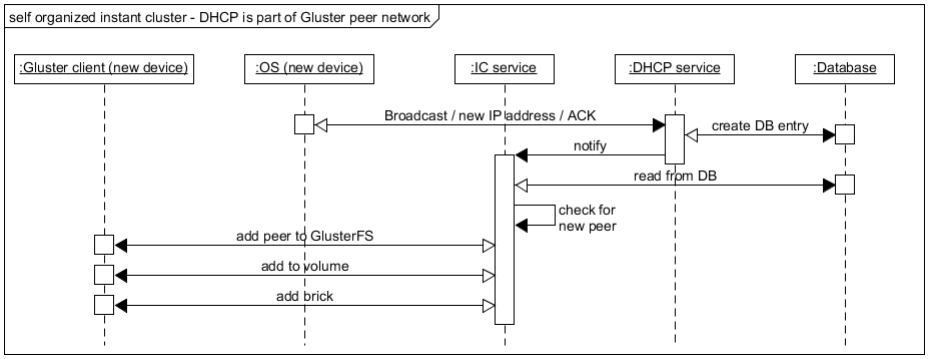

Abb. 3: Entwurf des Instant Cluster-Dienstes

In Abbildung 3 ist der Entwurf des Instant Cluster-Dienstes (ICS) zu sehen. Das vereinfachte Sequenzdiagramm besteht aus fünf Kommunikationspartnern. Der Gluster-Client sowie das Betriebssystem befinden sich auf dem neuen Gerät. Der ICS, der DHCP-Dienst und die Datenbank befinden sich auf dem DHCP-Server. Nachdem das Betriebssystem des neuen Gerätes erfolgreich eine vom DHCP-Dienst zugewiesene IP-Adresse erhalten hat, wird ein neuer Datenbank-Eintrag erstellt. Anschließend ruft der DHCP-Dienst den ICS auf oder benachrichtigt ihn darüber, dass es Veränderungen gibt. In dem Sequenzdiagramm ist nur der Fall berücksichtigt, bei dem es sich um ein neues Gerät handelt. Der ICS erhält auch in den anderen Fällen eine Benachrichtigung vom DHCP-Dienst, und zwar immer, wenn der DHCP-Dienst einen Trigger ausführt. Der ICS liest in jedem Fall erst die Daten aus der Datenbank und vergleicht diese dann mit dem Status des Gluster-Netzwerkes. Bei Unterschieden führt der ICS entsprechende Aktionen aus. In dem dargestellten Fall wird das neue Gerät zum Gluster-Peer-Netzwerk hinzugefügt, für das Volume eingetragen und der Brick des neuen Gerätes wird zum Volume hinzugefügt.

6 Ergebnisdiskussion

Mit dem Edge Device Instant Cluster können Edge Devices für IoT-Sensoren zuverlässige und dezentrale Datensicherung betreiben. In ein einfaches Netzwerk mit DHCP- und lokalem DNS-Server ist der Betrieb einer Vielzahl an Edge Devices automatisch möglich. GlusterFS wurde als Speichersystem ausgesucht und erfüllt von sich aus viele Aufgaben, die für das Bilden der Instant Cluster nötig sind. Das Bilden von Gruppen in denen die Geräte Datenaustauch betreiben sollen entfällt, da GlusterFS durch einen definierbaren Replikationsfaktor die Daten *n*-Fach speichert. Mit Hilfe des zentralen Instant Cluster-Dienstes wird das den Geräten zugrundeliegende Speichersystem GlusterFS automatisch konfiguriert. Nach vollendeter Konfiguration handelt es sich um ein dezentrales Speichersystem, welches ohne zentrale Einheit auskommt und keinen Single Point of Failure besitzt, solange die DHCP-Leases gültig sind. Der Ausfall einiger Geräte wird durch GlusterFS mit den Selbstheilungs-Funktionen abgefangen. Für das Auslesen der Daten kann ein beliebiger GlusterFS-Client benutzt werden, der direkte Zugriff auf ein Edge Device ist dazu nicht mehr notwendig.

Die Implementierung des Instant Cluster-Dienstes wurde nicht durchgeführt. Der Dienst konnte allerdings entworfen werden. Mit Hilfe der DHCP-Trigger ist es möglich in einer Datenbank stets den aktuellen Netzwerk-Status zu speichern und den Instant Cluster-Dienst bei Veränderungen im Netzwerk zu benachrichtigen und GlusterFS-Konfigurationen durchzuführen. Auch die dauerhafte Abwesenheit von Geräten kann erkannt und das System bereinigt werden.

7 Zusammenfassung und Ausblick

Für die Entwicklung des Edge Device Instant Clusters wurden in dieser Arbeit zunächst einige Anforderungen an dezentrale Speichersysteme analysiert. Drei der relevantesten Speichersysteme wurden genauer analysiert. Tahoe-LAFS bietet durch das Erasure Coding von Daten den Vorteil, platzsparende Redundanz zu ermöglichen. Nachteilig ist der erhöhte Overhead. Infinit ist ein neues dezentrales Speichersystem mit vielen Funktionen und einfacher Bedienung. Nachteilig ist, dass es noch nicht fertig entwickelt ist und die Verzahnung mit dem Infinit Hub zu eng ist. GlusterFS ist ein ausgereiftes Dateisystem, welches über viele Funktionen verfügt und alle Anforderungen erfüllt. Schließlich wurde GlusterFS zum Speichersystem für den Edge Device Instant Cluster ausgewählt. GlusterFS kann mit CA-Zertifikaten versehen werden, sodass sich die Gluster-Peers gegenseitig authentifizieren können.

Für das Bilden der Instant Cluster, also der automatischen Konfiguration der Geräte in einem Netzwerk, werden DHCP-Server-Trigger verwendet. Die Trigger lösen bei Veränderungen im Netzwerk aus und speichern Daten über das Netzwerk in einer Datenbank. Ein von den Triggern angesprochener Dienst kann auf die Trigger reagieren, das

Gluster-Netzwerk mit den Informationen aus der Datenbank vergleichen und entsprechend eine GlusterFS-Konfiguration vornehmen.

In Zukunft wird noch der Instant Cluster-Dienst entwickelt. Der Entwurf für den Dienst liegt bereits vor. Eine weitere Überlegung betrifft die Nutzung von NoSQL-Datenbanken anstelle der Nutzung eines Dateisystems. Mit NoSQL-Datenbanken können die Sensordaten strukturiert gespeichert werden und Datenanalysen müssten nicht erst die Daten vorverarbeiten und strukturieren. Dazu muss zunächst analysiert werden, wie viel Overhead der Einsatz eines NoSQL-Systems bedeutet und ob dieser den leistungsschwachen Edge Devices zuzumuten ist.

8 Literaturverzeichnis

[An18] Andres, C.; Herkenhoff, T.; Wallersheim, M.; Woywod, T.; Wörtler, F.: ConCar: Eine Verbindung von IoT und Blockchain, Abschlussbericht FEP 2018, pp. 1-12, 2018.

[BS15] Bial, D.; Scheuch, R.: Architecting for the Internet of Things (IoT). objektspektrum.de, 2015

[CCA15] Cohn, J; Chang, Y; Ahluwalia, G et. al.: Empowering the edge. IBM, 2015

[CR05] Castro, M.; Renesse, R.: Peer-to-Peer Systems IV. Springer, Ithaca, NY, USA, pp. 227-228, 2005.

[JDB12] Juve, G; Deelman, E.; Berman, B. et al: An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2. In (S Toor et al): J. Phys.: Conf. Ser. 513 062047, IOP Publishing, 2012

[KCC11] M. Kryczka, R. Cuevas, C. Guerrero, et al: Measuring the bittorrent ecosystem: Techniques, tips, and tricks. In: IEEE Communications Magazine, vol. 49, no. 9, pp. 144-152, 2011.

[KTK10] Y. J. Kim, M. Thottan, V. Kolesnikov et al: A secure decentralized data-centric information infrastructure for smart grid. In: IEEE Communications Magazine, vol. 48, no. 11, pp. 58-65, 2010.

[Qu10] Quintard, J.: Towards a worldwide storage infrastructure. University of Cambridge, pp. 47-49, 2010

[Sa15] Sarjaz, B.: Behrooz File System (BFS). University of Waterloo, pp. 22, 2015.

[SF14] M. Selimi and F. Freitag: Tahoe-LAFS Distributed Storage Service in Community Network Clouds. In: IEEE Fourth International Conference on Big Data and Cloud Computing, Sydney, NSW, pp. 17-24, 2014.

[TM17] A. Taivalsaari and T. Mikkonen: A Roadmap to the Programmable World: Software Challenges in the IoT Era. In: IEEE Software, vol. 34, no. 1, pp. 72-80, 2017.

[Ve13] Verkuil, S.: A Comparison of Fault-Tolerant Cloud Storage File Systems. University of Twente, Enschede, 2013

[XZL15] Xiao, D.; Zhang, C.; Li, X.: The Performance Analysis of GlusterFS in Virtual Storage. In: International Conference on Advances in Mechanical Engineering and Industrial Informatics (AMEII 2015), pp. 199-203, 2015.

Evaluation von IoT-Plattformen: Azure vs. AWS

Moritz Wallersheim[1]

Abstract: Welche Funktionalitäten bieten die großen IoT-Plattformen AWS und Azure um IoT-Geräte anzubinden und deren Daten zu analysieren? Welche Services sind zwingend notwendig um Prognosen bzw. Analysen von übergebenen Daten geben zu können? Und worin ergeben sich wesentliche Unterschiede bei den Plattformen? Der Beitrag diskutiert diese Fragen unter Betrachtung eines bestimmten Anwendungsfalls. Zur Umsetzung dieser Methodik wird eine Evaluation anhand von fest definierten Kriterien durchgeführt und präsentiert.

Keywords: Internet of Things, Cloud Platform, Evaluation of IoT Platforms, Azure vs. AWS

1 Einleitung

Das Bestreben der großen Gerätehersteller, Cloudanbieter und die zahlreichen Berichte von Analysten [De15] zeigen, dass die Zahl der technischen Geräte in den nächsten Jahren stetig ansteigen wird [Ga17]. Produkte, Maschinen und Anlagen aller Art sind immer stärker miteinander vernetzt und generieren tagtäglich Millionen von Daten. Ein großer Anteil sind IoT-Geräte und deren Plattformen.

Dabei ergeben sich für Anwender und Entwickler von IoT-Anwendungen immer mehr lukrative Möglichkeiten, die in den Anfangsjahren individuell geschaffenen IoT-Lösungen, durch standardisierte Produkte von Cloudanbietern nun kostengünstig zu erweitern, abzulösen oder neue Anwendungen schneller am Markt platzieren zu können (Time-to-Market). Große Cloud-Anbieter haben den Trend erkannt und erweitern ihre Angebote an standardisierten IoT-Produkte stetig[2], zum Teil auch durch den Zukauf von Unternehmen [Te15]. Grund genug also, um sich mit dem Thema der IoT-Plattformwahl auseinander zu setzen und diese nach eigenen Anforderungen und Bedürfnissen zu analysieren, zugleich die richtige Wahl der Plattform für den Erfolg eines Projektes entscheidend sein kann.

Dieser Beitrag befasst sich mit den Unterschieden zwischen den Plattformen „Azure IoT Suite“[3] und „AWS IoT“[4] anhand eines konkreten Anwendungsfalls, welcher prototypisch in beiden Plattformen umgesetzt wird. Zielsetzung ist es, einen ersten Überblick beider Anbieter zu erhalten und dem Leser Nutzungsszenarien der einzelnen Plattformen, zu geben.

[1] Fachhochschule Münster, Fachbereich Wirtschaft, Wirtschaftsinformatik, mw757475@fh-muenster.de
[2] Crisp Research, https://www.crisp-research.com/aws-summit-berlin-2016-indikator-fur-das-steigende-interesse-der-cloud-deutschland/
[3] Azure IoT Hub, https://azure.microsoft.com/de-de/services/iot-hub
[4] Amazon Web Services IoT, https://aws.amazon.com/de/iot-core/

2 Grundlagen und verwandte Arbeiten

Wie bereits in [An18] dargelegt, geht es bei IoT darum, Objekte über das Internet miteinander zu vernetzen, deren Daten zu sammeln und je nach Anwendungsfall auszuwerten. Ergebnisse von umfassenden Analysen können dabei als Auslöser oder Datengrundlage für andere Services dienen. Unternehmen haben dann beispielsweise die Möglichkeit, verschiedene Nutzungsszenarien ihrer Kunden zu verstehen, wenn deren verkaufte Produkte Informationen zurückliefern. Im Folgenden werden die verschiedenen Teile einer IoT-Plattform genauer erläutert und vorangegangene Arbeiten eingeordnet.

Mazhelis & Tyrvainen [MT14] beschreiben in ihrer Arbeit eine erste gute Basis für den Vergleich von Plattformen. Mit der Entwicklung eines Evaluation-Frameworks haben diese eine erste Unterscheidung von Plattformen vorgenommen. Sie betrachten in ihrer Arbeit jedoch ausschließlich die Sichtweise eines Applikation Anbieters und gehen dabei auf Aspekte wie die Umsetzung der zu benutzende Weboberfläche oder Funktionen wie die Abrechnung von Applikationen und Diensten ein. Ganugly [Ga16] ergänzt diese Ergebnisse mit seiner Arbeit, welche die Erstellung eines Basis-Sets für den Vergleich von Plattformen beinhaltet, um die Betrachtung der funktionalen und nicht-funktionalen Anforderungen. Dabei nennt und vergleicht er diverse Plattformen, ignoriert jedoch wichtige Lösungen, da diese entweder die geforderten Mindestanforderungen nicht erfüllen oder aber Stand der Arbeit noch nicht evaluationsfähig sind.

Diese Analyse knüpft an die beiden genannten Ergebnisse an und befasst sich ausschließlich mit den beiden großen Plattformanbietern Microsoft und Amazon, da diese bereits zum heutigen Zeitpunkt 60 % des Markts[5] beherrschen. Zudem ist in der Zukunft eine rasche und positive Entwicklung beider Anbieter im Markt[6] zu erwarten. Das nächste Kapitel beschreibt grundlegende Eigenschaften einer solchen Plattform.

2.1 Kommunikationsmuster und -protokolle

Die verschiedenen Plattformen nutzen unterschiedliche Kommunikationsmuster, die sich darin unterscheiden, wie die Nachrichten zwischen den Parteien fließen und welche Ziele sie haben. Folgende Muster finden in IoT-Lösungen Anwendung [Dz17]:

- **Telemetrie:** Daten fließen in eine Richtung vom Gerät zu anderen Systemen, um Statusänderungen im Gerät selbst zu übermitteln (z.B. Sensoren auslesen).
- **Anfrage:** Anfragen von dem Gerät, um die erforderlichen Informationen zu sammeln oder Aktivitäten zu initiieren.
- **Befehl:** Das Senden eines Befehls von einem System an ein Gerät oder eine Gruppe von Geräten, um bestimmte Aktivitäten auszuführen. Um daraufhin ein Ergebnis aus

[5] https://www.crisp-research.com/reinvent-2017-aws-muss-weiter-tempo-machen-der-kampf-um-die-marktanteile-wird-haerter
[6] https://www.crisp-research.com/iot-backend-die-evolution-der-public-cloud-anbieter-im-internet-iot/

der Befehlsausführung oder zumindest einen Status dafür zu erwarten.

- **Benachrichtigung:** Informationen fließen in einer Richtung von Systemen zum Gerät oder einer Gruppe von Geräten um Statusänderungen zu übermitteln.

Jedes Muster hat eigene Anforderungen. So nutzen manche einen Speicher- und Weiterleitungsmechanismus (Warteschlangen, Themenkanäle mit einem Nachrichten-broker) oder den Weg der Direktnachricht, wobei der Empfänger online sein muss, damit die Nachricht übermittelt werden kann. Ersteres wendet hauptsächlich Befehls-Muster an, wobei das Gerät zum Zeitpunkt des Nachrichtenversands möglicherweise offline bzw. nicht verfügbar sein kann. Dabei wird oft mit dem Attribut Time-to-Live (TTL) gearbeitet, das dem automatisierten Löschen der Nachricht nach einer gewissen Zeit dient. Dies ist nützlich, damit ein Gerät zu dem Zeitpunkt indem es wieder online ist bspw. eine veraltete Nachricht nicht ausführt.

Die Nutzung des Mechanismus der Direktnachricht wird häufig bei Telemetrie-Mustern verwendet [Dz17], auch wenn in einigen Anwendungsfällen das Speichern von Telemetrie-Daten ebenfalls nützlich sein könnte, um z.B. Datenverlust vorzubeugen. In mancher Hinsicht kann sich das Telemetrie-Muster zum Ereignis-Muster entwickeln, um bspw. eine bessere Quality-of-Service (QoS) in Bezug auf die Lieferung der Daten zu erreichen. Der Vorteil des Telemetrie-Musters ist die schnelle Auslieferung der Daten, da selbst, wenn ein Datenwert verloren geht, ein neuer, aktualisierter Wert in sehr kurzer Zeit gesendet wird.

Somit lässt sich feststellen, dass die Hauptprotokolle der IoT und die Art und Weise wie diese eingesetzt und implementiert werden, sich an den oben genannten Mustern orientieren. Bekannte Protokolle, die diese Muster implementieren, sind MQTT, AMQP bzw. HTTP, zugleich sich auch Abwandlungen von Herstellern etabliert haben[7].

2.2 IoT Referenzarchitektur

In der Welt der IoT dreht sich alles um die Kommunikation zwischen Geräten, Gateways und der Cloud. Eine IoT-Lösung ermöglicht es, ein Problem aus der realen Welt zu adaptieren und anhand von technischen Komponenten in der Welt der IT abzubilden. Das zumeist auftretende Problem, das es dabei zu lösen gilt, ist die Handhabung von Millionen von Geräten und Nachrichten pro Sekunde, diese zu verwalten, zu überwachen und aus der Ferne zu steuern.

Ein wichtiger Bestandteil bei der Lösung dieses Problems ist dabei das Backend-System der IoT-Lösung, das auch als IoT-Plattform bezeichnet wird [Pe16]. Eine IoT-Plattform besteht aus mehreren Microservices, die zumeist cloudbasiert, verteilt und über standardisierte Internet-Protokolle [Mü15, Is15, Pe16] miteinander kommunizieren. Dies dient der Skalierbarkeit, da die Plattform eingehende Daten sehr schnell skalieren und verarbeiten muss, ohne den gesamten Prozess zu verlangsamen. Hier spricht man auch von

[7] AWS, https://docs.aws.amazon.com/de_de/iot/latest/developerguide/protocols.html

dem oben genannten Telemetrie-Muster.

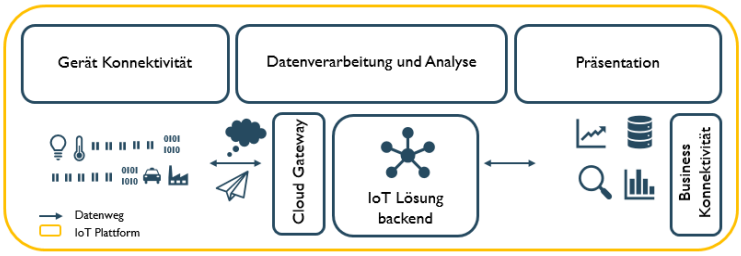

Abb. 1: IoT-Plattform - Referenzarchitektur

Zunächst kann man die Architektur einer IoT-Lösung in drei große Teile unterscheiden: Die IoT-Geräte, die verarbeitende Plattform (beinhaltet Datenverarbeitung und Analyseschicht) und die Präsentationsschicht. Letztere interagiert auch mit anderen Geschäftsanwendungen wie z.B. SAP (siehe Abbildung 1).

IoT-Geräte lassen sich klassifizieren in IP-fähige und nicht IP-fähige Geräte. IP-fähige Geräte sind in der Lage auf Basis des TCP/IP-Stacks direkt mit einem Gateway über eine Wifi- oder Ethernet-Verbindung zu kommunizieren. Darüber hinaus existieren Geräte, welche Protokolle wie z.B. Bluetooth für die Sensordatenerfassung nutzen und daher für eine direkte Kommunikation mit dem Gateway eher ungeeignet sind. Diese nicht IP-fähigen Geräte werden aus diesem Grund nicht weiterführend betrachtet.

Das Cloud-Gateway bietet, durch die Nutzung von typischen Cloud-Computing Merkmalen, wie dem Befehls- oder Benachrichtigungsmuster [Is15, Fa15], beide Szenarien zur Verfügung. Also die Datenübergabe bzw. -Annahme auf der einen und die Geräteverwaltung von dutzenden Geräten auf der anderen Seite, an. Gateways kommunizieren in der Regel bidirektional. Durch einen sicheren Anmeldungsmechanismus bzw. Authentifizierungsprozess ist eine Kommunikation zwischen Gerät und Cloud-Gateway möglich, sodass Daten über die bereits oben genannten Muster und Protokolle ermittelt und übermittelt werden können.

Daraufhin wird das Importsystem aktiv, welches die Daten entgegennimmt und verarbeitet und anschließend an die Business Engine, also das IoT-Backend oder andere Drittanbieterplattformen, übergibt. Um die Qualität der Daten [Ra15] zu gewährleisten, sollten gewisse Bedingungen definiert werden, sodass bspw. fehlerhafte Daten im frühen Stadium erkannt und im anschließenden Prozess nicht weiter betrachtet werden (Data-Cleaning [Ra15]). Darüber hinaus kann es erforderlich sein, Daten der verschiedenen Geräte anzureichern, zu aggregieren oder nach festen Regeln zu verändern. So ist etwa eine Umrechnung von Fahrenheit in Celsius, oder die Datenerfassung nach einer festen Zeitzone eine potenzielle Anforderung. Im Anschluss können Daten an andere Plattformen, Services oder Anwendungen zur Verfügung gestellt werden.

3 Analyse

In diesem Kapitel wird die Herangehensweise der Analyse beider Plattformen beschrieben. Ziel der Analyse ist es, die Plattformen „AWS IoT“ und „Azure IoT Hub“ einzusetzen und diese anhand von Kriterien zu vergleichen.

Dafür werden zunächst Kriterien erarbeitet und erklärt, die dann im weiteren Verlauf als eine Art Grundlage bzw. Vergleichsbasis dienen. Eine weitere Grundlage ist das Anwendungsszenario, das neben der Darstellung der Gemeinsamkeiten und Unterschiede dazu dienen soll, die Stärken und Schwächen der einzelnen Plattformdienste zu beleuchten. Das Ergebnis der Analyse soll schließlich, neben einem Überblick der Dienste auch eine Gegenüberstellung dieser sein. In Form einer Tabelle sollen die Ergebnisse zusammengefasst dargestellt werden.

3.1 Vergleichsbasis

Für einen Vergleich der beiden Plattformen bedarf es sowohl funktionaler als auch nichtfunktionaler Anforderungen, die zugleich als Grundlage einer Bewertung dienen können. Zur Verbesserung der Übersichtlichkeit folgen dabei nur einige wichtige Punkte.

A. Unterstützte Lizenz oder Bezahlmodelle
Aktuell existieren sowohl kostenpflichtige als auch kostenlose (Open Source) Plattformen. Kostenpflichtige Plattformen zeichnen sich dadurch aus, dass diese sich in der Bezahlart und den angebotenen Preisstaffellungen unterscheiden.

B. Unterstützte Anwendungsprotokolle
Jede Plattform unterstützt zumeist mehrere Anwendungsprotokolle. Dabei bildet REST die meist verbreitete und genutzte Variante. Websockets bieten eine gute Lösung für die Echtzeitkommunikation. Darüber hinaus existieren IoT-spezialisierte Protokolle wie MQTT und AMQP. Einige Plattformen bieten dem Nutzer darüber hinaus eigene veränderte Protokolle an oder die Möglichkeit der Veränderung.

C. Unterstützte Gerätehardware und SDKs
Einige Anbieter stellen individuell angepasste Betriebssysteme für IoT-Geräte zur Verfügung. Darüber hinaus gibt es SDKs in verschiedenen Programmiersprachen wie z.B. Node.JS, C#, Java, Python. Eine Limitierung der eingesetzten Geräte ist somit nahezu nicht gegeben, da heutzutage alle gängigen Hardwareplattformen wie z.B. Raspberry Pi, Arduino, Intel, ARM und weitere problemlos unterstützt werden.

D. Unterstützte Serialisierungsformate
Beachtet man, dass IoT-Geräte zumeist über wenig Rechenleistung bzw. Speicher verfügen, so ist ein schlankes Serialisierungsformat umso mehr von Bedeutung. Auf der einen Seite wird so die ohnehin schon geringe Leistung des Geräts geschont und zum anderen können durch ein schlankes Datenformat Kosten bei der Übermittlung eingespart werden. Gängige und genutzte Formate für Daten sind JSON und XML.

E. Benutzer- und Rollenverwaltung
Mandantenfähigkeit in einer Plattform ermöglicht dem Inhaber des Master Accounts, Benutzer zu erstellen oder diese zu bearbeiten. Anwender sollen darüber hinaus die Möglichkeit haben, sich selbstständig in wenigen Schritten an der Plattform registrieren zu können. Um heutige Sicherheitsanforderungen zu erfüllen, besteht die Möglichkeit, Benutzerinformationen in ein Gerät auszulagern, sodass auch nicht autorisierte Nutzer in der Lage sind, die Plattform zu nutzen. Darüber hinaus ist die Erstellung von Benutzergruppen und die feingranulare Verwaltung von Geräten ein Grundbedürfnis von Anwendern, sodass z.B. einem Gerät nur bestimmte Funktionen gewährt werden.

F. Geräteverwaltung
Hier gilt es zwischen zwei Gerätetypen zu unterscheiden. Typ 1 ist ein Gerät, das vorkonfiguriert ausgeliefert und der Plattform in der Regel bekannt ist. Im Laufe der Zeit wird dieses verbunden. Typ 2 ist ein Gerät, das bei Bedarf verbunden wird. Dies bedeutet, dass das Gerät in der Regel der Plattform noch unbekannt ist und somit bspw. seine eindeutige ID erst von der Plattform über eine entsprechende Schnittstelle erfragen muss. Dies erfordert entsprechende Mechanismen und Anpassungen an der Plattform.

G. Geräte- und Eventmanagement
Die meisten Plattformen bieten vorgefertigte Ereignismodelle aus dem Alltag wie z.B. das Ein- und Ausschalten einer Glühbirne [Is15]. Die Modellierung solcher Modelle ist jedoch schwierig und wird nur von wenigen Plattformen unterstützt. Es wird erwartet, dass der Anwender eine Möglichkeit erhält, vor und nachgelagerte Funktionen bzw. Event-Handler zu implementieren um bspw. das Senden von Benachrichtigungen oder Alarmen an Drittsysteme zu ermöglichen. Häufig findet sich hier ein Regelwerk.

H. Standortwahl
Unternehmen, die in mehreren Geolokationen aktiv sind, müssen sich mit der Frage der Standortwahl eines Cloud-Rechenzentrums auseinandersetzen.

I. Monitoring
Zur Übersicht und der Auswertung von Leistungsdaten ist eine Visualisierung der verbundenen Geräte und Dienste zwingend notwendig. Darüber hinaus sollten Warnungen über Missbrauchsfälle dem Anwender gezeigt werden.

J. Sicherheit
Da im Zweifel sensible Daten ausgetauscht werden, muss eine Sicherheit zwischen Gerät und Plattform gewährleistet werden, welche durch Verschlüsselungsmechanismen Verbindungs- bzw. Datenseitig erfolgen kann.

K. Analytics
Um Daten auszuwerten kann ebenfalls in zwei Typen unterschieden werden. Typ 1 stellt die Echtzeitanalyse dar, in der die Daten sofort verarbeitet und für eine Analyse zur Verfügung stehen. Typ 2 beschreibt die Offlineanalyse also die Analyse bei der bspw. Vergangenheitsdaten ausgewertet werden.

L. Dokumentation
Eine detaillierte Dokumentation von Schnittstellen oder bereitgestellten Diensten kann hilfreich sein und teure Supportkosten ersparen. So können z.B. Tutorials und Einführungsvideos hilfreich sein und ergänzend herangezogen werden.

M. Skalierbarkeit
Die wohl meistbenötigte nicht-funktionale Anforderung ist der Aspekt der Skalierbarkeit. Von der Plattform wird erwartet, dass sie problemlos mehrere tausende Geräte unterstützt. Bei der raschen Entwicklung von IoT-Anwendungen muss die Plattform in der Lage sein, hier Schritt zu halten und den Anwender dahingehend zu unterstützen.

N. *Sonstiges*
Die oben genannte Liste bietet einen ersten umfassenden Blick auf wichtige Kriterien. Je nach Anwendungsfall sollten diese hinsichtlich ihres Schwerpunkts und individuellen Anforderungen um eigene Kriterien ergänzt werden. Denkbar wäre es z.B. die Forenaktivität der Community oder die Unterstützung von Drittanbietern genauer zu beleuchten.

3.2 IoT-Anwendungsszenario

Das nun folgende Anwendungsszenario soll möglichst viele Technologien der zugrundeliegenden Plattformen verwenden, um die Stärken und Schwächen hervorzuheben. So soll der Prototyp bestehend aus einem simulierten IoT-Gerät und einer IoT-Plattform in der Lage sein, Daten auszutauschen, zu verarbeiten und dem Anwender anzuzeigen. Für die Umsetzung des Szenarios wurden folgende Anforderungen definiert:

- Eine Testanwendung (z.B. Java) soll als Simulation eines IoT-Geräts dienen.
- Über die Testanwendung sollen Daten simuliert werden.
- Die Testanwendung soll für die Kommunikation das SDK der Plattform nutzen.
- Die Testanwendung soll über IoT-Protokolle Daten an die Plattform übermitteln
- Die Plattform soll auf Daten reagieren und diese exemplarisch verarbeiten.
- Die Daten sollen in einem nicht flüchtigen Speicher persistiert werden.
- Die Verarbeitung der Daten soll als Echtzeit-Variante und zudem als Offline-Variante erfolgen.
- Die Plattform soll die verarbeitenden Daten anzeigen können.
- Die Plattform soll Möglichkeiten bieten, Drittanbieter-Anwendungen an die Plattform anbinden zu können.

Im nächsten Kapitel werden die Plattformen und ihre Funktionen näher erläutert.

3.3 Vergleich der Plattformen

Die im Folgenden genannten Informationen der Plattformen wurden über deren Webseiten ermittelt. Für die Realisierung des zugrundeliegenden Prototyps wurde zudem je Plattform ein Testzugang erstellt, wodurch wesentlich detailliertere Informationen ermittelt werden konnten.

Kriterium	Funktion	Azure IoT Hub	AWS IoT
A	Preisgestaltung, Lizenzmodelle	Je nach Editionswahl des IoT Hub Pakets und ggf. IoT Hub Provisioning-Dienst	Je nach Nutzung der Dienste Anbindung, Messaging, Thing Shadows, Thing Registry und Rules-Engine
B	Unterstützte Protokolle	HTTP, AMQP, MQTT und benutzerdefinierte Protokolle	HTTP, MQTT und Abwandlungen davon
C	Unterstützte Plattformen	>= 10+ z.B. Intel, Qualcomm, Broadcom, Nvidia, Raspberry Pi 2,	>= 15 z.B. Broadcom, Qualcomm, Texas Industries
C.1	Geräte SDK	Android, iOS, Swift	Android, iOS, Arduino Yun,
C.2	Anwendung SDK	C, Python, Node.js, Java, .NET, Ruby	C++, Python, Node.JS, Java, PHP, .NET
D	Unterstützte Dateiformate	JSON, XML	JSON, XML
E	Authentifizierung	pro Gerät mit einem SAS-Token	X.509 Zertifikat, IAM service, Cognito service
F	Geräte Verwaltung	Device Explorer über SAS-Token	Thing Registry über Secret Key
G	Geräte Event Management	Event Hubs, Functions	Rules Engine, Lambda
H	Lokation der Rechenzentren	über 36 Regionen weltweit, in Deutschland Frankfurt, Magdeburg	über 18 Regionen weltweit, in Deutschland nur Frankfurt
I	Monitoring	Log Analytics, Monitor	CloudWatch, CloudTrail
J	Sicherheitsmechanismen	TLS (Serverseitig)	TLS (beidseitig)
K	Analytics	Stream Analytics, Power BI	Kinesis Analytics, Quicksight
L	Dokumentation	Azure-Codebeispiele	AWS Developer Guide

Tab. 1: Ergebnistabelle – Azure vs. AWS

Microsoft nennt seinen IoT-Dienst „IoT Hub“, durch den eine bidirektionale Kommunikation zwischen Gerät und Geschäftsanwendung (bzw. Cloud-Backend) ermöglicht wird. Hierbei wird das oben erläuterte Telemetrie-Muster wie auch das Befehlsmuster verwendet, um Nachrichten vom Gerät zur Cloud (D2C) bzw. von der Cloud zum Gerät (C2D) zu übermitteln.

Die Verbindung ist TLS basiert, wodurch die Kommunikation zuverlässig und verschlüsselt ist. Die Authentifizierung wird vom IoT Hub (Server) bereitgestellt, der den vom Gerät gesendeten SAS-Token anhand der Richtlinien überprüft. Der Server führt bei der Authentifizierung sein eigenes X.509 Zertifikat beim Senden der Daten an das Gerät beim TLS Handshake mit. Für die Übertragung von Nachrichten nutzt der IoT Hub das AMQP-Protokoll, das zugleich eine native Unterstützung für Anforderungs- und Antwortpfade mitbringt. Des Weiteren ist auch eine Kommunikation über Protokollstandards wie HTTP oder MQTT möglich. Alle Nachrichten nutzen dabei die Dateiformaten JSON oder XML.

Neben der Möglichkeit, den IoT Hub über standardisierte Protokolle anzusprechen, bietet Microsoft darüber hinaus verschiedene SDKs in unterschiedlichen Programmier-

Sprachen und Plattformen an. Neben dem .NET Framework, das vor allem in UWP-Anwendungen zum Einsatz kommen dürfte, werden auch Java oder Javascript (NodeJS) basierte SDKs unterstützt. Für Kleinstsysteme wird zudem eine Embedded C Variante angeboten. Darüber hinaus existieren verschiedene Partnerschaften mit Hardwareherstellern und ein Zertifizierungsprogramm[8]. Dies führt zu einer Vielzahl von nutzbaren Plattformen wie z.B. Raspberry Pi, die mittels SDK fähig sind, den IoT Hub zu benutzen.

Die Nutzung des Dienstes „Event Hub“ ermöglicht es Nachrichten an andere Endpunkte weiterzuleiten. Dadurch ist es bspw. möglich, den Dienst „Stream Analytics“ anzureihen und Daten in Echtzeit zu analysieren. Darüber hinaus besteht natürlich die Möglichkeit, alle Speicherdienste (SQL, NoSQL, blob, cache) für die kurzfristige oder langfristige Speicherung der Daten zu verwenden.

Des Weiteren sollte man bei der Betrachtung des IoT Hub nicht nur die Rolle der Dateneinnahme betrachten, sondern auch die Entwicklung von vollumfassenden Backend-Anwendungen in Betracht ziehen. Der Dienst „Power BI“ ermöglicht eine ansehnliche Art und Weise der Visualisierung und Anzeige von Daten gegenüber dem Endanwender. Darüber hinaus stellt der Dienst „Machine Learning“ Möglichkeiten der vorrausschauenden Analyse mittels vordefinierten Modellen zur Verfügung.

All diese Dienste sind ab einer gewissen Basis kostenpflichtig (meist abhängig von Anzahl der Nachrichten). Dabei gestalten sich die Preise in drei verschiedenen Editionen.

AWS IoT bietet im Grunde die gleichen Objekte nur findet die Kommunikation innerhalb der Dienste in einer anderen Art und Weise statt. Über eine Geräteregistrierung ist es möglich, gerätebezogene Informationen bzw. Attribute (Hersteller, Seriennummer) im Gerät zu hinterlegen. Zudem können Geräte über eine Weboberfläche bzw. Kommandozeile erstellt, verwaltet, aktualisiert oder gelöscht werden.

Jedes Gerät, das über eine gültige Anmeldung verfügt, kann Nachrichten übermitteln oder sich für die Zustellung dieser anmelden. Die Nachrichten sind themenbezogen und werden von einem sog. Message Broker verteilt, welcher nach dem Publish / Subscribe – Prinzip vorgeht. Anders als bei Microsoft ermöglicht „AWS IoT“ die Interaktion mit Geräte-Schatten. Ein Geräte-Schatten ist aus Sicht eines Entwicklers im Wesentlichen nichts anderes als ein JSON-Dokument, das zum Speichern oder Abrufen aktueller oder vorhergesehener Statusinformationen eines Geräts dient. Für Schatten-Operationen (Update, Get und Delete) die ebenfalls im JSON-Format erfolgen, bietet der Message Broker einige reservierte Themen an. Empfängt also bspw. ein Gerät einen Status auf einem reservierten Thema, so wird dieser automatisch im Schatten repliziert.

Die Kommunikation mit dem Message Broker erfolgt über das TLS Protokoll und der Client authentifiziert sich mit einem X.509-Client-Zertifikat. Die Zertifikate werden mittels Weboberfläche oder Kommandozeile erstellt, aktiviert oder gelöscht. Darüber hinaus können auch eigene Zertifikate verwendet werden. Weitere Möglichkeiten der

[8] Azure IoT Certified Program, https://azure.microsoft.com/de-de/marketplace/iot-partners/

Anmeldung ergeben sich durch die Dienste „IAM“ und „Cognito“. Recherchen ergeben, dass Zertifikate bei MQTT-Anwendungen, HTTP bei IAM[9] und Cognito bei Endanwender-Szenarien verwendet werden.

Die Kommunikation mit der Plattform und dahinterliegenden Diensten kann wie bei Azure auch über verschiedene SDKs[10] erfolgen. Hier werden mitsamt alle gängigen Programmiersprachen und Plattformen unterstützt, zuletzt sogar das Microsoft Framework .NET und C#. Für die einfache Interaktion mit Geräte-Schatten bietet das SDK eine darüber gelagerte Anwendungsschicht an. Neben bekannten Hardwareplattformen ist es darüber hinaus möglich, über eine C Variante des SDK eigene Hardware zu entwickeln.

Für die eventbasierte Verarbeitung von Daten bietet AWS eine „Rules-Engine“ an. Über diese ist es möglich eine oder mehrere Regeln für ein bestimmtes Thema zu definieren, die dann Aktionen ausführen. Dabei kann für jede Regel eine Zugriffs- und Ausführungsberechtigung definiert werden. Die Engine unterstützt diverse Typen von Aktionen wie z.B. die Integration von Speicherdiensten (wie z.B. „DynamoDB“, „S3“, „Glacier“) oder Systeme für die weiterführende Verarbeitung (wie z.B. Lambda, zur funktionalen Programmierung oder Kinesis für die Echtzeit-Verarbeitung der Daten).

Der Dienst „Quicksight**“** bietet dem Anwender die einfache und schnelle Möglichkeit, Daten zu visualisieren. Zudem bietet AWS seit diesem Jahr den Dienst IoT Analytics an, der die Ausführung von anspruchsvollen Analysen an IoT-Massendaten erleichtern soll. Viele der oben genannten Funktionen bietet AWS für bestimmte Volumina im ersten Jahr kostenlos an. Der Gesamtpreis ergibt sich feingranular aus den Teilen Anbindungszeit, Messaging (Anzahl der Nachrichten), Device Shadow Registry (Anzahl Operationen) und der Rules-Engine (Anzahl ausgeübte Regeln, Aktionen). Zusammenfassend finden sich die Ergebnisse in der obigen Tabelle wieder (siehe Tabelle 1).

4 Ergebnisdiskussion

Microsoft und Amazon haben ihre Plattformen mit verschiedenen Optionen entwickelt und bieten durchweg interessante Produkte an. Beginnt man zunächst bei den grundlegenden Protokollen bedeutet dies AMQP vs. MQTT. Amazon verwendet hier das MQTT-Protokoll was durch die Akquisition des Startups „2elemtry“ [Te15] und deren Entwicklung bedingt sein dürfte. Darüber hinaus verwenden beide das HTTP-Protokoll, wodurch keine wesentlichen Unterschiede ausgemacht werden können. In der hardwareseitigen Geräteunterstützung finden sich Unterschiede (z.B. unterstützte Bausätze[11]) die jedoch als verschwindend gering zu bewerten sind. Ein ähnliches Bild zeichnet sich auch bei der Plattform- und Sprachenunterstützung der SDKs ab.

[9] https://aws.amazon.com/de/iot/sdk/
[10] https://docs.aws.amazon.com/IAM/latest/APIReference/Welcome.html/
[11] https://catalog.azureiotsuite.com/

Erste wesentliche Unterschiede finden sich in der Interaktion mit den Geräten. IoT Hub benutzt Nachrichten, die Daten oder Befehle für Aktionen tragen, AWS hingegen bildet Geräte-Stati anhand eines JSON-Dokuments ab, was eine weitere Datenabstraktionsschicht mit sich bringt. Bei der zugrundeliegenden Sicherheitsimplementierung nutzen beide eine TLS-basierte Kommunikation, bei der Authentifizierung nutzt AWS ihren eigenen IAM-Dienst, Microsoft hingegen nutzt einen SAS-Token zur Identitätsverwaltung.

Die Gestaltung der Preise ist hingegen völlig unterschiedlich und könnte je nach Anwendungsfall ausschlaggebender Grund für das Produkt sein. Sieht man von den kostenlosen Nutzungskontingenten einmal ab und kalkuliert ein beispielhaftes Nutzungsszenario[12], überzeugt Azure. Stellschrauben sind hier die Preisstruktur sowie Art und Weise der Dienstnutzung. So verrechnet Azure bspw. Nachrichten in Blöcken von je 4 KB. Wo hingegen AWS 5 KB als Einheit zugrunde legt. Letztlich dürfte auch die Standortwahl des Dienstes einen Einfluss auf die Preisgestaltung haben. Da z.B. Heimatregionen der Cloudanbieter (AWS, Azure in Nordamerika) i. d. R. günstiger sind. Bei einem Vergleich der Visualisierungswerkzeuge merkt man, dass Microsoft mit „Power BI“ langfristige Erfahrung in puncto Berichtserstellung und Analysen mit sich bringt. Entwickler, die mit Reporting Services oder dergleichen bereits vertraut sind, werden sich hier eventuell leichter tun als mit dem Produkt „Quicksight“ von Amazon.

Besonders hervorzuheben ist der Microsoft Dienst „Machine Learning“, welcher durch vordefinierte Modelle überzeugt und auch fachfremden Entwicklern durch die einfache Modellierung einen leichten Einstieg ermöglicht. Generell bietet die „Azure IoT Suite“ einige out-of-the-Box Lösungen und gute Dokumentationen für Entwickler, die eine Vielzahl an Services abdecken und bei AWS vermisst werden. Auf Seiten Amazons ist besonders die „Rules-Engine“ und deren benutzerfreundliche Weboberfläche zu nennen, über die es nahezu mühelos erscheint, Daten an andere Dienste weiterzuleiten. Hier könnte Microsoft an ihrem „Event Hub“ noch an Übersichtlichkeit nachbessern.

5 Zusammenfassung und Ausblick

Anhand der vorliegenden Arbeit wird dem Leser ein umfassender Einblick in die beiden Plattformen Azure und AWS gegeben. Dabei werden verschiedene Kriterien für eine Unterscheidung von Plattformen erläutert und ein erster kurzer Vergleich dieser durchgeführt. So bieten Microsoft und Amazon beide ein Set aus IoT-Microservices an, mit denen vollwertige Lösungen innerhalb kürzester Zeit und vertretbarem Aufwand zu realisieren sind. Durch die Kombination mit eigenen Microservices kann dies zu einer hochskalierbaren, fehlertoleranten und zuverlässigen Gesamt IoT-Lösung führen. Für eine Entscheidung für oder gegen eine Plattform sollten neben den genannten Kriterien dieser Arbeit potenzielle Plattformen nach individuell angepassten Bedürfnissen betrachtet werden. Zudem ist zu beachten, dass je nach Sichtweise des Anwenders, sich die

[12] 10.000 Geräte welche 40 Nachrichten a 5 KB versenden (Region Frankfurt), Azure 50,6€, AWS 61$

Relevanz der Anforderungen grundlegend ändern kann. So kann bspw. ein geringerer Preis andere Vorzüge oder Besonderheiten einer Plattform in den Hintergrund rücken lassen.

Auch in Zukunft werden neue Plattformen hinzukommen, die die Entwicklung einfacher machen, ohne dabei das Rad neu zu erfinden. Dabei ist zu erwähnen, dass sich durch die Masse an neuen Geräten in den nächsten Jahren auch der Markt der IoT-Plattformen rasant entwickeln und der Konkurrenzkampf sich unter den Anbietern verstärken wird. Dies wiederum wird die Vielfalt an neuen Funktionalitäten beflügeln und die Preise der IoT-Plattformen und -Dienste mittelfristig sinken lassen.

Literaturverzeichnis

[An18] Andres, C.; Herkenhoff, T.; Wallersheim, M.; Woywod, T.; Wörtler, F.: Eine Verbindung von IoT und Blockchain, Abschlussbericht FEP 2018 S. 1-12, 2018.

[De15] Deloitte: Inside IoT, https://www2.deloitte.com/content/dam/insights/us/articles/iot-primer-iot-technologies-applications/DUP_1102_InsideTheInternetOfThings.pdf, 2015, abgerufen am: 17.01.2018.

[Dz17] Patierno, P.: Strength and Weaknesses of IoT Communication Patterns, https://static.dzone.com/2017_IoT_Guide2/index.html#p=25, 2017, abgerufen am: 17.01.2018.

[Fa15] Familiar Bob, „Microservices, IoT, and Azure: Leveraging DevOps and Microservice Architecture to Deliver SaaS Solutions“, Springer, S. 132ff, 2015.

[Ga16] Ganguly Pankaj.: „Selecting the right IoT Cloud Platform“, International Conference on Internet of Things and Applications (IOTA), 2016.

[Ga17] Gartner, Press Release, https://www.gartner.com/newsroom/id/3598917, 2017, abgerufen am: 17.01.2018.

[Is15] Isikdag Umit, „Enhanced Building Information Models: Using IoT Services and Integration Patterns“, Springer, S. 71ff, S. 95, S. 96, 2015.

[MT14] Mazhelis O. & Tyrvainen P., „A Framework for Evaluating Internet-of-Things Platforms: Application Provider Viewpoint“, World Forum on Internet of Things (WF-IoT), 2014 IEEE.

[Mü15] Münzl, G.; Pauly, M.; Reti, M.: Cloud Computing als neue Herausforderung für Management und IT, Springer, S. 8, S. 49, 2015.

[Pe16] Perry J. Matthew.: „Evaluating and Choosing an IoT Platform“, O'Reilly Media, Inc., S. 14ff, 2016.

[Ra15] Rahm, E.; Saake, G.; Sattler, K.-U: Verteiltes und paralleles Datenmanagement, Springer, S. 15, 2015

[Te15] Lunden, I.: https://techcrunch.com/2015/03/12/amazon-has-quietly-acquired-2lemetry-to-build-out-its-internet-of-things-strategy, 2015, abgerufen am: 17.01.2018.