HAW HAMBURG

BACHELOR THESIS
Herberto Werner

Datenpipeline-Vergleichsstudie mit Echtzeitanbindung am Beispiel von ausgewählten closed- und open-source Ansätzen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Herberto Werner

Datenpipeline-Vergleichsstudie mit Echtzeitanbindung am Beispiel von ausgewählten closed- und open-source Ansätzen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke
Zweitgutachter: Prof. Dr. Jan Sudeikat

Eingereicht am: 19.12.2022

Herberto Werner

Thema der Arbeit

Datenpipeline-Vergleichsstudie mit Echtzeitanbindung am Beispiel von ausgewählten closed- und open-source Ansätzen

Stichworte

Industrie 4.0, Datenpipeline, IIoT

Kurzzusammenfassung

Eine Überwachung von Industrieanlagen in einer Produktionsumgebung bringt Herausforderungen mit sich, da es verschiedene Anforderungen, Use-Cases, Randbedingungen sowie Möglicheiten für den Aufbau und Implementierung gibt. Industrie 4.0 und „Industrial Internet of Things (IIoT)“ stellen neue Paradigmen, Architekturen und einen großen Pool von Technologien vor, die genutzt werden können. Das Ziel von Industrie 4.0 und IIoT ist es, Geschäfts- und Produktionsprozesse zu verbessern. Eine sichere Produktion, eine verbesserte Qualität der Produkte und Prozesse, maximaler Durchsatz etc. sind Ziele von Industrie 4.0 und IIoT. Durch eine Überwachung können einige Ziele von Industrie 4.0 und (IIoT) erreicht werden. Hierfür müssen die Daten aus den Industrieanlagen extrahiert, verarbeitet und visualisiert werden. Die vorliegende Arbeit vergleicht zwei skalierbare Datenpipelines anhand zwei Performance Metriken: Latenz und Durchsatz. Die Datenpipelines werden zwischen Open-Source und Closed Ansatz unterschieden und enthalten die Schritte: Extrahierung, Verarbeitung und Visualisierung. Das Ziel dieser Arbeit ist es, beide Datenpipeline anhand den Metriken zu evaluieren und eine Datenpipeline mit gegebenen Anforderungen, Use-Cases und Randbedingungen für die Überwachung auszuwählen. Im Laufe der Untersuchung zeigte sich, dass der Open-Source-Ansatz aufgrund seiner höheren Parametrisierbarkeit und aufgrund der Ergebnisse der Latenz- und Durchsatzmessungen besser zur Überwachung geeignet ist.

Herberto Werner

Title of Thesis

Data pipeline comparison study with real-time connectivity using selected closed- and open-source approaches as an example

Keywords

Industry 4.0, Datapipeline, IIoT

Abstract

Monitoring industrial plants in a production environment brings challenges as there are different requirements, use cases, constraints and options for setup and implementation. Industry 4.0 and Industrial Internet of Things (IIoT) introduce new paradigms, architectures and a large pool of technologies that can be used. The goal of Industry 4.0 and IIoT is to improve business and production processes. Safe production, improved quality of products and processes, maximum throughput etc. are goals of Industry 4.0 and IIoT. Some of the goals of Industry 4.0 and (IIoT) can be achieved through monitoring. For this, the data from the industrial plants must be extracted, processed and visualised. This paper compares two scalable data pipelines based on two performance metrics: Latency and Throughput. The data pipelines are distinguished between open source and closed approach and contain the steps: extraction, processing and visualisation. The aim of this work is to evaluate both data pipelines against the metrics and to select a data pipeline with given requirements, use cases and constraints for monitoring. In the course of the investigation, it became apparent that the open-source approach is better suited for monitoring due to its higher parameterisability and based on the results of the latency and throughput measurements.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Industrie 4.0 und „Internet of Things“ schaffen neue Möglichkeiten in Produktionsprozessen. Die digitale Transformation ermöglicht die dezentrale, autonome, digitale Steuerung und Überwachung von Produktionssystemen und Industrieanlagen [30]. Für Internet of Things in der Produktion wird laut [10] in den kommenden Jahren bis 2030 200 Millionen vernetzte Geräte und Dinge erwartet. Die Kommunikation und Vernetzung von Maschinen, intelligenten Sensoren, Aktoren, Systemen und Produkten in einer Produktionsumgebung oder sogenannten „Smart Factories“ , wird „Industrial Internet of Things (IIoT)“ genannt [21]. Der Marktwert von „Industrial Internet of Things (IIoT)“ wird laut [13] in wichtigen Marktregionen auf 110.6 Billionen U.S Dollar erwartet. Zusammenfassend kann man sagen, dass mehr Daten durch die Kommunikation und Vernetzung entstehen.

Geschäfts- und Produktionsprozesse zu verbessern durch Autonomie, höhere Flexibilität und sichere, überwachte Produktion und darausfolgend eine verbesserte Qualität der Produkte und Prozesse sind Ziele von Industrie 4.0 und IIoT. [53] [61] [50]

Aus diesem Grund müssen die Daten aus den Industrieanlagen und Produktionsprozessen gesammelt, ausgewertet und visualiert werden. Mithilfe einer Überwachung können Ziele Industrie 4.0 und IIoT erreicht werden.

Die Automatisierungpyramide ist eine etablierte, hierarchische Struktur in einer Produktion [23]. Durch die Transformation auf Industrie 4.0 wird die Struktur aufgrund neuen Verbindungen und der Kommunikation zu anderen Systemen aufgebrochen [53] [60]. Eine speicherprogrammierbare Steuerung (SPS) ist Bestandteil dieser Automatisierungpyramide, da sie Daten aus Sensoren und Aktoren sammelt, verarbeitet und die Informationen an andere Ebenen der Pyramide schickt [60]. Hierbei bietet die SPS die Möglichkeit Daten direkt aus der SPS zu extrahieren, verarbeiten und zu visualisieren [60].

Eine Überwachung der Industrieanlagen und Produktionen mithilfe einer SPS mit den Stationen: Datenextrahierung, - verarbeitung und visualisierung bringt ebenfalls Herausforderungen mit sich. Die Herausforderung ist es, aus dem Pool von Cloud-Plattformen, Open-Source Big Data Frameworks, Protokollen und Bibliotheken eine skalierbare Datenpipeline mit den genannten Stationen zu erstellen. Beim Aufbau und Implementierung einer Datenpipeline unterscheidet man ebenfalls zwischen einem Open-Source und Closed-Ansatz. [30] [58] [41]

Folgende Fragen lassen sich durch die Problemstellung stellen:

1. Welche Plattformen und Big Data Frameworks können für eine Überwachung einer Industrieanlage durch eine skalierbare Datenpipeline gewählt und verglichen werden?

2. Welche Datenpipeline (Open-Source oder Closed-Ansatz) ist anhand den Performance Metriken: Latenz und Durchsatz für eine Überwachung eher geeignet?

3. Welche Latenzen und welcher Durchsatz ergeben sich bei der Überwachung mit einer SPS?

Die vorliegende Arbeit konzentriert sich anhand von Performance Metriken auf den Vergleich zweier Datenpipelines, die jeweils einen Closed- und Open-Source Ansatz haben. Der Closed Ansatz wird mit „Amazon Web Services (AWS)“ realisiert. Der Open-Source Ansatz wird mit ausgewählten Open-Source Frameworks und Bibliotheken realisiert. Das Ziel der Arbeit ist es, beide Datenpipelines zu vergleichen und eine Datenpipeline für die Überwachung von Industrieanlagen auszuwählen. Hierfür werden beide Datenpipelines aufgebaut, implementiert und anhand den Performance Metriken: Latenz und Durchsatz verglichen. Es wird anschließend anhand den Metriken evaluiert, welche Lösung für die Überwachung geeignet ist.

Als Erstes werden Definitionen und Begriffe in Kapitel 2 erklärt, um die Datenpipelines in die Domäne und den Kontext einzuordnen. Es soll ebenfalls zeigen, welche Herausforderungen und Probleme für den Aufbau und Einsatz der Datenpipelines entstehen.

Im Kapitel 3 werden die Anforderungen und Use-Cases, die sich aus der Fragestellung bilden, vorgestellt. Hinzu kommt, dass Randbedingungen beschrieben werden, die die Experimente in einem Rahmen festhalten und bestimmen. Die Anforderungen, Use-Cases und Randbedingungen bilden die Grundlage für die Funktionen und Umfang der Datenpipelines.

Anschließend werden in Kapitel 4 Architekturstile erläutert, die jeweils beide Datenpipelines verwenden. Beide Datenpipelines haben unterschiedliche Aufbauten. Es dient als Grundlage für die Erklärung des Aufbaues im Experimenten Kapitel.

Beide Datenpipelines verwenden andere Technologien. In Kapitel 5 werden die Technologien beider Datenpipelines erklärt und dargestellt. Die Parameterkonfigurationen werden ebenfalls erklärt, um die Auswirkungen dieser Parameter auf die Messungen zu verstehen.

Nach der Vorstellung der Anforderungen, Randbedingungen, Use-Cases, Einführung in den Kontext sowie die Erklärung der Architektur und Technologien werden in Kapitel 7 die Latenz und der Durchsatz in beiden Datenpipelines untersucht und verglichen. Der Aufbau beider Datenpipelines wird anhand Verteilungssichten dargestellt. Anschließend wird durch Sequenzdiagramme der Datenfluss für die Latenz- und Durchsatzmessung erklärt. Die Parameterkonfiguration für die jeweilige Messung wird ebenfalls beschrieben. Die Ergebnisse der Latenz- und Durchsatzmessungen werden in Tabellen festgehalten.

Die Arbeit wird mit der Diskussion, welche Datenpipeline anhand den gemessenen Werten die geeignetere Wahl ist, beendet. Hierbei wird zwischen Erwartung und Ergebnissen verglichen und die Unterschiede und Gemeinsamkeiten diskutiert.

2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Definitionen eingeführt. In Abschnitt 2.1 und Abschnitt 2.2 die Begriffe Industrie 4.0 und „Internet of Things“ sowie in Abschnitt 2.3 die Automatisierungpyramide in der Industrie und Cyber-physische Systeme in Abschnitt 2.4. Weitere Begriffe wie „Edge“ „Fog“ und „Cloud“ werden in Abschnitt 2.5 eingeführt. Das Kapitel wird mit der Einführung des Begriffes Datenpipeline in Abschnitt 2.6 abgeschlossen.

2.1 Industrie 4.0

Der folgende Abschnitt stellt die Entwicklung und Ziele der Industrie in der Produktion und IT dar. Die Ziele der Industrie müssen bei der Anforderungsanalyse und -spezifikation betrachtet werden. Daraus resultiert die Entscheidung über den Architektur- und Grobentwurf und die Auswahl von Technologien, wie Cloud-Diensten, Plattformen und Bibliotheken.

Die erste industrielle Revolution führte mechanische Maschinen in Fabriken zur Steigerung der Produktivität ein. Die zweite industrielle Revolution steigerte die Produktivität durch die Einführung des Fließbandes und einer modernen Arbeitsteilung. Die dritte industrielle Revolution war die Automatisierung der Produktion durch speicherprogrammierbare Steuerungen (SPS). Die Marktnachfrage nach Flexibilität in der Produktion von Systemen, Lösungen für komplexe Lieferketten und Fortschritt der Technologien führt zur vierten industriellen Revolution (Industrie 4.0). [53]

Die vierte industrielle Revolution führt neue Paradigmen ein, die eine digitale, autonome und dezentrale Steuerung von Produktionssystemen ermöglichen. Durch Industrie 4.0 ist es möglich immer größere Mengen an Daten von „Smart Factories“ zu extrahieren [30]. So beinhaltet Industrie 4.0 ebenfalls neue Technologien, Methodiken und Prozesse, die für die Datenextrahierung und Datenverarbeitung verwendet werden [30]. Das Ziel ist es

ein maximaler Durchsatz der Produktionswertschöpfungskette zu erreichen, eine flexible Produktion und eine erhöhte Produktivität zu gewährleisten. [61] [50] Außerdem gibt es durch Industrie 4.0 neue Referenzarchitekturen. Diese sollen für verschiedene Domänen die Interoperabilität zwischen Systemen ermöglichen, Kosten bei der Entwicklung und Risiken in Software Projekten reduzieren [53].

Für die Automatisierung von Industrie 4.0-Produktionsprozessen sind spezifische Softwaretechnologien, Plattformen und Werkzeuge nötig. In Kapitel 5 werden diese für die Use-Cases in Kapitel 3 eingeführt. Die Einführung von Industrie 4.0 ist für diese Arbeit wichtig, um die Datenpipelines in die Domäne und Kontext einordnen zu können. Hinzu kommt, dass die Ziele von Industrie 4.0 bei der Wahl der Technolgien beachtet werden müssen.

2.2 Internet der Dinge

Internet der Dinge, bzw. „Internet of Things (IoT)“ ist Teil der industriellen, vierten Revolution. Nach den Definitionen in [50] [63] beschreibt das Internet der Dinge ein System, dass mittels eindeutiger Adressen eine Kooperation und Integration von physischen Objekten oder Dingen über ein Netzwerk stattfindet. Physische Objekte oder Dinge sind Geräte wie Sensoren, Aktuatoren, die verschiedene Kommunikationsmethoden und Kapazitäten haben. Diese können mit anderen Objekten, der Umwelt oder beiden interagieren. Durch die Interaktionen aller Objekte und Dinge ist es möglich kooperativ Probleme zu lösen und eine erhöhte Interkonnektivität zwischen den Objekten und Menschen zu schaffen. Die Objekte sind mit der Umgebung, mit anderen Objekten sowie mit dem Internet verbunden. Informationen, wie zum Beispiel der Zustand eines Aktuators, werden zwischen den Objekten über das Internet ausgetauscht. Durch den Informationsaustausch über das Internet ist eine Steuerung und Überwachung dieser Objekte, wie zum Beispiel Aktoren möglich.

In [35] werden verschiedene Definitionen des IoT-Konzeptes aufgezeigt. Nach [35] gibt es überblickend vier charakteristische Objekt-Kategorien:

- Verfolgbare Objekte
  Verfolgbare Objekte sind für die Identifizierung und Bestimmung der Position zuständig.

- Datenobjekte
  Datenobjekte halten eigene Zustände und die Zustände der Umgebung mittels Sensoren fest.
- Interaktive Objekte
  Interaktive Objekte interagieren und beeinflussen die Umgebung.
- Intelligente Objekte
  Intelligente Objekte bearbeiten Daten und steuern das Verhalten.

Industrielles Internet der Dinge, auch „Industrial Internet of things (IIoT)“ genannt und als Industrie 4.0 bekannt, beschreibt den Einsatz des IoT-Konzeptes im industriellen Kontext [23]. Im Fokus hierbei steht die Errichtung einer industriellen Echtzeitumgebung. Es beinhaltet Künstliche Intelligenz, Maschine-zu-Maschine-Kommunikation „M2M“ , „Big Data“ und Automatisierungstechnologien. Im IIoT spielen Berechnungen über die „Cloud“ , Nutzung von Cloud-Infrastrukturen und Big Data-Technologien für die Umsetzung eines IIoT-Systems eine große Rolle [63]. Vernetzte Geräte im Haushalt, der Medizin oder in der Industrie können neue Möglichkeiten und Anwendungsfelder in der Kommunikation und Interaktionen für die Steuerung und Überwachung schaffen [23].

Die Einführung von IoT und IIoT ist wichtig, da Sensoren und Aktoren an der SPS angeschlossen sind. Diese sind Teil der Industrieanlage und Produktionsumgebung. Sie können den Zustand der Umgebung über eine Messung festhalten. Die Informationen über den Zustand können über das Internet verschickt werden. Sensor- und Aktorwerte sind Daten, die in beide Datenpipelines eingespeist werden.

2.3 Automatisierungspyramide

2.3.1 Referenzmodell

Der folgende Abschnitt befasst sich mit einem Referenzmodell, welches die Automatisierungspyramide genannt wird. Die Automatisierungpyramide zeigt in welcher Ebene sich die SPS befindet. Die SPS ist für den Vergleich die Datenquelle. In beiden Architekturansätzen in Kapitel 4 werden Daten, wie z.B.: Sensordaten aus der SPS extrahiert.

Aktuell ist die Automatisierungspyramide ein etabliertes Referenzmodell [23] [67], welches im fertigungsnahen Umfeld verwendet wird. Abbildung 2.1 zeigt, dass das Modell in

einer hierarchische Struktur, welches eine Aufteilung von Verantwortlichkeiten in der Produktionsstätte darstellt. Die Pyramide besteht aus fünf Ebenen [23] [67] [63].

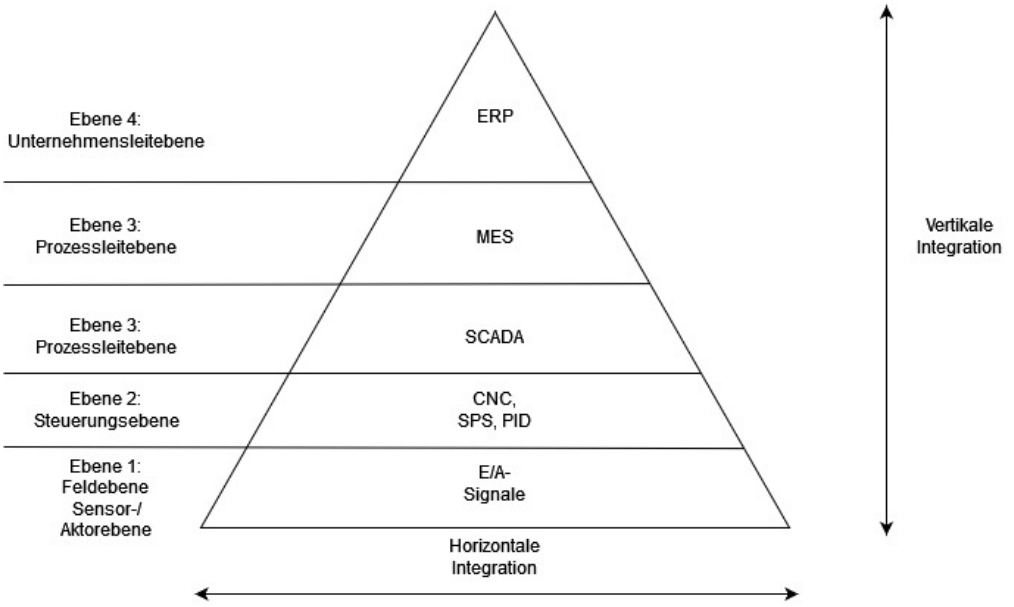

Abbildung 2.1: Automatisierungpyramide mit allen Ebenen

- 1.Ebene: Feldebene
  - Die unterste und erste Ebene ist die Feldebene mit physikalischen Entitäten wie Sensoren, Aktoren.
- 2.Ebene: Steuerungsebene
  - Die zweite Ebene ist die Steuerungsebene und enthält speicherprogrammierbare Steuerungen (SPS), „Computerized Numerical Control (CNC)“ , Maschinen oder proportionale, integrale und derivative Regler (PID).
- 3.Ebene: Prozessleitebene
  - Die dritte Ebene ist die Prozessleitebene, welche „Supervisory Control und Data Aquisition (SCADA)“ Systeme enthält. Diese Ebene kontrolliert und überwacht Prozesse in der Feld- und Steuerungsebene. Außerdem kann es eine Mensch-Maschine-Schnittstelle für einen Bediener enthalten.
- 4.Ebene: Betriebsleitebene

  - Die vierte Ebene ist die Betriebsleitebene und enthält ein Fertigungsmanagementsystem „Manufacturing-Execution-System (MES)“ , die den Produktionsprozess von Auftrags-, Material-, Betriebsmittel-, Personal-, Energie- und Qualitätsmanagement bis zu Datenerfassung und Feinplanung und -steuerung als Aufgaben übernimmt.

- 5.Ebene: Unternehmensebene
  - Die oberste Ebene ist die Unternehmensebene, die Geschäftsprozesse in der Logistik, Produktion, Finanz- und Personalwesen mit betriebswirtschaftlichen „Enterprise-Ressource-Planning-System (ERP)“ Systemen leitet.

Betrachtet man die Reaktionszeiten und Echtzeitanforderungen, müssen Entscheidungen in den unteren Ebene in Millisekunden getroffen werden. Andererseits werden auf den höheren Ebenen der Pyramide mittelfristige und langfristige Entscheidungen und Planungen auf Unternehmensebene getroffen. Das bedeutet, dass die Reaktionszeiten zwischen Tagen, Wochen und Monaten liegen können [23]. Die Zahl der Informationen nimmt in den Ebenen der Pyramide von oben nach unten zu. Gleichzeitig unterscheidet sich in den Ebenen die Informationsqualität für Entscheidungen [23]. Eine weitere Eigenschaft der Struktur ist, dass der Informationsfluss ebenorientiert ist. Das bedeutet, dass der Informationsaustausch an den angrenzenden Ebenen geschieht. Nach Abbildung 2.1 sieht man ebenfalls eine Unterscheidung zwischen vertikaler und horizontaler Integration. Die horizontale Integration ist die Kommunikation zwischen Akteuren innerhalb einer Ebene oder weiteren Automatisierungspyramiden auf gleicher Ebene. Hierfür werden unteranderem Feldbussysteme verwendet. Bei vertikaler Integration betrachtet man die Kommunikation zwischen den Ebenen. Ein „Gateway“ kann zwischen den verschiedenen Protokollen als Vermittler dienen, um Daten zwischen den Ebenen austauschen zu können [23].

Anhand der Flexibilität und Komplexität der Produktion, die durch Industrie 4.0 erforderlich werden, verschwinden die Grenzen der Ebenen. Das Ergebnis ist eine Vernetzung aus Systemen und Ebenen. Die Automatisierungpyramide lässt sich auf die Schichten, Ebenen, Teile und Prozesse der Industrie 4.0 Referenzarchitekturen abbilden und für neue Anwendungsfelder erweitern. [53] zeigt neueste relevante Referenzarchitekturen für Industrie 4.0 sowie deren Umfang und bildet sie auf die Automatisierungspyramide ab. [53]

Die Vorstellung der Automatisierungpyramide ist für die Arbeit relevant, da die SPS die Datenquelle für beide Datenpipelines ist. Für die Überwachung einer Industrieanlage

muss der aktuelle Einsatz und Standard gezeigt werden. Der Aufbau der Datenpipelines zeigt, dass die Strukturen aufgebrochen werden, da der Datenfluss nicht mehr in höhere Ebenen verläuft.

2.4 Cyber-physische Systeme

Die Konzepte aus Industrie 4.0 und die kabellose, kabelverbundene Netzwerkverbindungen sind für den Einsatz der cyber-physischen Systeme die Elemente für eine digitalisierte Produktion [46]. Cyber-physische Systeme in der Industrie und im Kontext der Automatisierung werden auch Cyber-physische Produktionssysteme (CPPS) genannt [43]. Die Auflösung der hierarchischen Strukturen (siehe Abbildung 2.1) in den Produktionsstätten führen laut [40] zu intelligenten Fabriken, die eine Kombination aus intelligenten Geräten, IoT und Software beinhalten. Die Wandel von einer Automatisierungpyramide zu intelligenten Fabriken beschreibt die Entwicklung von Industrie 3.0 auf Industrie 4.0. Die Kombination bildet cyber-physische Systeme ab. Die Definition von cyber-physischen Systemen ist weltweit in der Literatur nicht einheitlich und eindeutig definiert [66] [46]. Der Begriff wird oft in der Literatur mit den Begriffen „Industrie 4.0“ und „Internet of Things (IoT)“ verbunden. In der Literatur gibt es verschiedene Zusammenhänge zwischen cyber-physischen Systeme (CPS) und IoT. Beide Begriffe haben teilweise Schnittmengen und Äquivalenzen. Ebenfalls wird CPS als ein Teil von IoT sowohl auch IoT als Teil von CPS betrachtet [35]. Laut [48] [46] ist „Cyber-Physical System (CPS)“ ein Zusammenspiel aus eingebetteten, vernetzten Rechnersystemen und physischen Prozessen. Die Prozesse beeinflussen Berechnungen durch die Steuerung und Überwachung in einer Rückkopplungsschleife [48] [46]. Hierbei sind die Systeme in einer heterogenen, vernetzten Zusammensetzung und Struktur. Für die dynamische Anpassung eines cyber-physischen Systems zur Betriebszeit gibt es Mensch-Maschine-Schnittstellen. Über diese Schnittstellen können Prozesse durch Aktoren und Sensoren überwacht oder gesteuert werden [40] [46].

CPS in der Industrie 4.0 soll Automatisierungsziele, wie Autonomie, höhere Flexibilität, verbesserte Qualität, sichere Produktion, Reduktion von Arbeit und Kosten sowie eine höhere Zuverlässigkeit erreichen. Daraus folgt, dass neue, vielfältige Anwendungen und Möglichkeiten von Geschäftsmodellen entwickelt und umgesetzt werden können [40] [64]. Für die neuen Anwendungen und Möglichkeiten ergeben sich Nutzenpotenziale

wie zum Beispiel für Management und Steuerung industrieller Maschinen und Anlagen, zeitliche Vorhersagen und die Optimierung von technischen Kundendiensten sowie Ferndiagnose[39]. Der Einsatz und die Potentiale für Anwendungsgebiete von CPS in der Industrie sind vielfältig und erstrecken sich von Transportwesen und Mobilität über Maschinenbau, medizinische Geräte sowie Robotik bis zur Energieversorgung und das intelligente Netz [48] [50].

Die vorliegende Arbeit bietet eine Möglichkeit, eine sichere Produktion durch die Vernetzung und Überwachung zu gewährleisten. Es bietet die Grundlage die extrahierten Daten für den weiteren Ausbau eines Cyber-physischen Systems.

2.5 Edge, Fog und Cloud

Der Abschnitt dient zur Definition und Abgrenzung der Begriffe. Für den Aufbau einer Datenpipeline als Closed-Ansatz werden Cloud-Dienste verwendet. Die Begriffe werden ebenfalls in verwandte Arbeiten (Kapitel 6) erwähnt.

2.5.1 Edge

Die Ebene „Edge“ befindet sich zwischen der „Fog“ Ebene und IoT-Geräten in „Gateways“ und Servern. Das bedeutet, dass sie eine Grenze zwischen dem Firmennetzwerk und dem Internet formt und mit der „Fog“ Ebene kommuniziert. Als „Gateway“ kann es zum Beispiel für die Kommunikation zwischen dem Internet und IoT-Geräten in „Low Power Wide Area Network(LoRaWAN)“ Netzwerken genutzt werden. LoRaWAN ist eine Netzwerkarchitektur, in der zwischen Servern und Endgeräten eine Verbindung durch „Gateways“ hergestellt wird [55]. [65] [49]

Ein „Edge Gateway“ hat zum Beispiel die Aufgabe Daten zu sammeln und diese an eine IoT Plattform zu senden. Das „Edge Gateway “ verwendet HTTP, HTTPS, „Advanced Message Queuing Protocol (AMQP)“ , „Message Queue Telemetry Transport (MQTT)“ als Internetprotokolle für den Transport [65] [49] [67]. Mithilfe von „Edge Computing“ können die gesammelten Daten gespeichert oder für Berechnungen vor Ort, direkt an einem vernetzten Endpunkt eines Gerätes, verwendet werden[67].

2.5.2 Fog

Die Ebene „Fog“ befindet sich zwischen den Ebenen „Edge“ und „Cloud“ „Fog Computing“ ist laut [68] eine Erweiterung vom Cloud Konzept, welches am Rand der Cloud dezentralisiert Dienste anbietet. Die Aufgaben der Dienste beinhalten die Verarbeitung der Daten, Bestimmung welche Daten an die zentrale Cloud gesendet werden, Netzwerkdienste und die Speicherung von Daten. Der Unterschied zwischen „Edge Computing“ und „Fog Computing“ ist der Standort. In der Fog Ebene, welche sich zwischen IoT Gerät und Cloud befindet, gibt es dezentralisierte, separate Netzwerkknoten für die intelligente Verarbeitung und Speicherung von Daten während beim „Edge Computing“ die Verarbeitung und Speicherung in der Nähe der Geräte erfolgt. Durch „Edge Computing“ und „Fog Computing“ können durch die Zwischenverarbeitung - und speicherung Latenzzeiten und das Datenübertragungsvolumen verringert werden.[68]

2.5.3 Cloud

In der Cloud-Ebene werden Dienste und Ressourcen jederzeit und unabhängig vom Ort über das Internet für Nutzer und Kunden angeboten. Die Nutzung der Dienste und Ressourcen nennt man „Cloud Computing“ [57]. Es gibt die Möglichkeit eine Infrastruktur dynamisch aufzubauen und die benötigten Dienste aus den verfügbaren Diensten zusammenzustellen [65]. Eine „Cloud“ ist in vier Ebenen aufgeteilt:

1. Hardware
2. Infrastruktur
3. Plattform
4. Applikation

Die Ebenen können über die Kommandozeile, „GUIs“ und Programmierschnittstellen, die vom Cloud-Anbieter zur Verfügung gestellt werden, erreicht werden.

In der untersten Ebene „Hardware“ befinden sich in Daten- und Rechenzentren, die Prozessoren und Router anbieten. Der Nutzer kann auf die Hardware nicht zugreifen.

In der nächsten Ebene „Infrastruktur“ befinden sich virtuelle Server, Speichergeräte als Speicher- und Berechnungsressourcen.

Die nächste, höhere Ebene „Plattform“ beinhaltet komplexe Soft- und Hardwareschichten, wie z.B. Betriebssysteme, Netzwerk, Server und Entwicklungstools, die der Anbieter auf einer Plattform anbietet. Auf dieser Ebene soll es den Kunden und Programmierern möglich sein, Applikationen zu entwickeln und bereitzustellen. Auf der obersten Ebene „Applikation“ werden den Kunden und Nutzern fertige Applikationen und Anwendungen bereitgestellt. Der Nutzer hat ebenfalls die Möglichkeit die Applikationen zu konfigurieren und anzupassen [65]. Die Anwendungen und Applikationen laufen auf der Infrastruktur des Anbieters. Die Komplexität und Infrastruktur des Anbieters sind für Kunden und Nutzer nicht bekannt.

Ein Cloud Model eines Cloud-Anbieters hat verschiedene charakteristische Eigenschaften, Deployment- und Dienstmöglichkeiten. Die Dienste lassen sich laut [65] in die genannten Ebenen einordnen.

- „Software as a Service (SaaS)“ lässt sich in die Applikationsebene einteilen.
- „Plattform as a Service (PaaS)“ ordnet man in die Plattformebene zu.
- „Infrastructure as a Service (IaaS)“ lässt sich in die Hardware- und Infrastrukturebene einteilen.

Neben den Dienstmöglichkeiten unterscheidet man bei Cloud Hosting Deployment Modellen zwischen öffentlichen, privaten, hybriden und der Gemeinschaft. Bei einer öffentlichen Cloud legt der Cloud-Anbieter vordefinierte Regeln, Richtlinien und Preismodelle fest, während eine private Cloud für den Eigenbedarf intern in öffentlichen Einrichtungen oder Unternehmen genutzt wird. Eine hybride Cloud ist eine Kombination aus beidem. Bei einer Gemeinschafts-Cloud teilen sich verschiedene Organisationen die Cloud. Ein Drittanbieter stellt die Infrastruktur zur Verfügung. [57] [22]

Die Datenpipeline im Closed Ansatz verwendet die Dienste in der Applikationsebene (SaaS). Der Cloud-Anbieter hat bei der Nutzung bestimmte Regeln und Richtlinien aufgestellt. Diese müssen bei den Experimenten und für die Diskussion ebenfalls betrachtet werden.

2.6 Datenpipeline

Eine Datenpipeline ist ein Prozess, der mehrere Stufen durchläuft. Der Prozess einer Datenpipeline beinhaltet Datenextrahierung und -verarbeitung, Weiterleitung an weitere

Systeme für eine Erkenntnisgewinnung sowie als Unterstützung für weitere Entscheidungen. Je nach Anwendungsfall gibt es unterschiedliche Anforderungen und Arbeitsflüsse für eine Datenpipeline. Es gibt ein Konzept, welches Merkmale für den Aufbau aller Datenpipelines beschreibt. Daten werden für einen Anwendungsfall, wie z.B.: die Überwachung von Industrieanlagen extrahiert und vorbereitet. Im nächsten Schritt werden die Daten verarbeitet. Eine Datenverarbeitung kann zum Beispiel eine Datenfilterung sein. In einer Datenbank werden abschließend die Ergebnisse gespeichert. Für die technische Umsetzung einer Datenpipeline gibt es sechs Stufen: Datenerhebung, Datenbereinigung, Visualisierung, Modellierung, Interpretation und Auslieferung in die Produktionsumgebung. [56]

In Kapitel 4 werden unterschiedliche Architekturstile der Datenpipelines für den Vergleich gezeigt.

3 Anforderungen

Im folgenden Kapitel werden die Anforderungen, Use-Cases und Randbedingungen beschrieben. Die Anforderungen, Use-Cases, Randbedingungen leiten sich von der Fragestellung in Kapitel 1 ab. Die Anforderungen, Use-Cases und Randbedingungen werden nach ARC42 erstellt.

Als Erstes werden Anforderungen für die Überwachung einer SPS in einer Industrieanlage durch eine Datenpipeline (siehe 3.2) dargestellt. Anschließend werden in diesem Abschnitt die Use Cases der Datenpipelines, welche sich aus den Anforderungen ableiten, aufgezeigt. Abschließend werden die Randbedingungen für die Überwachung aufgezählt. Der Aufbau, Implementierung und Einsatz der Datenpipeline wird anhand der Anforderungen, Use-Case und Randbedingungen bestimmt.
In der Tabelle 3.2 sind die Anforderungen für eine Überwachung einer Industrieanlage zu sehen. Diese leiten sich von der Fragestellung ab.

ID	**Thema**	**Anforderung**	**Spezifikation**
A1	Überwachung	Daten aus einer SPS können ausgelesen werden	Daten, wie Sensorwerte, Alarme, Statusinformationen können aus einer Industrieanlage ausgelesen und abgefragt werden.
A2	Standort	Die SPS befindet sich in einem anderen Standort	Die Sensorwerte einer SPS werden aus einem anderen Standort ausgelesen/abgefragt.
A3	Daten	Die Daten können gefiltert werden	Daten können nach Eigenschaften, wie z.B. Temperatur, Strom, Spannung etc. gefiltert werden.

A4	Daten	Es sind Datenmanipulationen möglich	Manipulationen, wie z.B.: Aggregationen, Gruppierungen, Join-Operationen sind möglich.
A5	Daten	Daten werden in eine Datenbank gespeichert	Daten mit einem oder ohne Zeitstempel sollen in eine Datenbank gespeichert werden. Es können auch zwei Datenbanken verwendet werden.
A6	Überwachung	Daten werden in einer Visualisierung gezeigt	Die Daten, wie Sensorwerte, Alarme, Statusinformationen müssen in einer Visualisierung abgefragt und dargestellt werden.
A7	Überwachung	Die Überwachung erfolgt kontinuierlich	Es gibt keine festegelegten Zeitintervalle oder Echtzeitanforderungen für die Überwachung. Der Datenfluss erfolgt kontinuierlich.

Tabelle 3.2: Anforderungen für die Applikation (Datenpipeline)

3.1 Use-Case

Die Use-Cases werden von den Anforderungen (siehe Tabelle 3.2) abgeleitet. Die Referenzierung zu den Anforderungen sieht man anhand der Nummer bzw. ID. Die Priorität zeigt an, wie wichtig das Use-Case ist. Die Use-Cases für die Überwachung einer Industrieanlage sind in den Tabellen 3.4 3.6 3.8 3.10 zu sehen.

Nr. / ID	A1, A2, A5, A6, A7	**Name**	Sensorwerte aus einer SPS auslesen	**Priorität**	hoch
Beschreibung	Eine SPS ist in einer Industrieanlage in Deutschland im Einsatz. Sensorwerte werden von der SPS gesammelt. Der Datenfluss erfolgt kontinuierlich. Ein Nutzer der Visualisierung fragt einen Sensorwert ab und möchte diese beobachten.				
Vorbedingung	1. Die SPS eingeschaltet und mit einem entfernten Server für die Datenextrahierung (Datenpipeline Station) verbunden 2. Sensorwert ist vorhanden				
Ablaufbeschreibung	Die SPS schickt kontinuierlich die Sensordaten in die Datenpipeline.				
Nachbedingung	Die Sensordaten erreichen die Stationen der Datenpipeline. Die Sensorwerte sind in der Datenbank gespeichert. Die abgefragten Sensorwerte sind in der Visualisierung zu sehen.				

Tabelle 3.4: Use-Case 1: Sensorwerte aus einer SPS auslesen

Use-Case 1 aus Tabelle 3.4 beschreibt das Auslesen der SPS. Für die Experimente werden Daten aus der SPS ausgelesen. Use-Case 1 zeigt dies mit Sensorwerten.

Nr. / ID	A1, A3, A5, A6, A7	**Name**	Datenfilterung der Sensorwerte	**Priorität**	hoch
Beschreibung	Der Nutzer der Visualisierung möchte nur bestimmte Sensordaten, wie z.B.: Strom- oder Spannungswerte aus der Industrieanlage betrachten.				
Vorbedingung	1. Die SPS eingeschaltet und mit einem entfernten Server für die Datenextrahierung (Datenpipeline Station) verbunden 2. Sensorwert ist vorhanden 3. Filterung nach Strom, Spannungswerten möglich				

Ablaufbeschreibung	Eine Abfrage (SQL etc.) filtert den kontinuierlichen Datenstrom nach den gesuchten Sensorwerten. Die Datenbank wird abgefragt.
Nachbedingung	Der gesuchte Sensorwert wird zurückgegeben und angezeigt

Tabelle 3.6: Use-Case 2: Datenfilterung der Sensorwerte

Use-Case 2 in Tabelle 3.6 zeigt die Filterung der Sensorwerte. Für eine Überwachung muss eine Filterung vorgenommen werden, um Daten sortieren und punktuell auswählen zu können.

Nr. / ID	A1, A4, A5, A6, A7	**Name**	Datenmanipulation der Sensorwerte	**Priorität**	hoch
Beschreibung	Der Nutzer der Visualisierung möchte bestimmte Sensorwerte zusammenfassen, um diese für die prädiktive, präventive Überwachung analysieren zu können.				
Vorbedingung	1. Die SPS eingeschaltet und mit einem entfernten Server für die Datenextrahierung (Datenpipeline Station) verbunden 2. Sensorwert ist vorhanden 3. Manipulation von mehreren Datenströmen möglich				
Ablaufbeschreibung	Eine Abfrage (SQL etc.) manipuliert mehrere kontinuierliche Sensordatenströme und speichert diese in eine Datenbank Die Datenbank wird abgefragt.				
Nachbedingung	Sensorwerte wurden manipuliert bzw. zusammengefasst etc. und sind in der Visualisierung zu sehen				

Tabelle 3.8: Use-Case 3: Datenmanipulation der Sensorwerte

Das Use-Case 3 in Tabelle 3.8 soll zeigen, dass ebenfalls Datenmanipulationen bei einer Überwachung möglich sein sollen.

Nr. / ID	A1, A2, A6, A7	**Name**	Sensorwerte in der Visualisierung anzeigen	**Priorität**	mittel
Beschreibung	Der Nutzer der Visualisierung möchte Sensorwerte von der entfernten SPS auslesen können				
Vorbedingung	1. Die SPS eingeschaltet und mit einem entfernten Server für die Datenextrahierung (Datenpipeline Station) verbunden 2. Sensorwert ist vorhanden 3. Visualisierung vorhanden 4. Alte Werte vom Sensorwerte				
Ablaufbeschreibung	Die SPS schickt aktualisierte Sensordaten an die Datenpipeline.				
Nachbedingung	Aktualisierte Sensorwerte, die der Nutzer sehen möchte, sind in der Visualisierung zu sehen				

Tabelle 3.10: Use-Case 4: Sensorwerte in der Visualisierung anzeigen

Eine Überwachung erfordert auch die Möglichkeit der Visualisierung. Das Use-Case 4 in Tabelle 3.10 beschreibt die Visualisierung der Sensorwerte.

3.2 Randbedingungen

Die Randbedingungen bestimmen die Experimente und halten diese in einem Rahmen fest. Diese leiten sich aus der Fragestellung, Anforderungen und Use-Cases ab. In diesem Abschnitt werden die Randbedingungen für die Datenpipelines definiert. Folgende Randbedingungen gibt es für die Datenpipeline:

- **Skalierbarkeit**
  - Geographisch
    - Die verwendetete Hardware für den Aufbau und Implementierung befinden sich in Deutschland. Die Datenübertragung erfolgt über das Internet.
    - Datenpipeline AWS: Die verwendeteten Dienste befinden sich in der Region „EU-CENTRAL-1 (Frankfurt)“ .

        * Datenpipeline Open-Source: Der Ubuntu Server von NetCup GmbH befindet sich in Deutschland.

    - Administrativ

        * Ein Nutzer

- Eine SPS ist an einem Router in Hamburg angeschlossen und simuliert eine SPS in einer Anlage.
- **Zykluszeit**
    - Die Zykluszeit wird bei der Latenzmessung nicht betrachtet, da es sich um eine Simulation einer Anlage handelt. Das verwendete SPS Programm wird nur zu experimentellen Zwecken genutzt.
- **Sicherheit**
    - Die Sicherheit bei der Datenübertragung sowie Zertfizierungen werden vernachlässigt.
- Die Uhrensynchronisation der AWS Datenpipeline erfolgt über den NTP Server „time.aws.com“ .
- Es ist keine Echtzeitüberwachung. Es gibt keine Echtzeitgarantien.
- Übertragungswege zwischen Diensten in Rechenzentren einer Region können nicht beeinflusst werden.

4 Architekturtypen

Es werden zwei Architekturstile erläutert. Im Abschnitt 4.1 wird die Service-orientierte Architektur vorgestellt. Ein weiterer Architekturstil ist die ergeinisbasierte Architektur im Abschnitt 4.2. Beide Architekturstile sind die Grundlagen für die Open-Source und AWS Datenpipeline (Closed Ansatz) in Kapitel 5. Die Anwendung der Architekturstile auf die Arbeit erfolgt in Abschnitt 4.1.3 und Abschnitt 4.2.3.

4.1 Service-orientierte Architektur (SOA)

Laut [65] ist eine serviceorientierte Architektur (SOA) eine verteilte Anwendung oder ein System, welche ein Zusammenschluss aus vielen verschiedenen Diensten ist. Applikationen und Dienste sind in einer heterogenen Landschaft von verschiedensten Technologien. Die Motivation einer service-orientierte Architektur ist es, Dienste und Applikation in einer heterogenen Landschaft von Technologien zu integrieren [33].

4.1.1 Eigenschaften einer service-orientierten Architektur

Eine SOA ist durch bestimmte Eigenschaften definiert. Die erste Eigenschaft Service Kapselung bedeutet, dass Dienste gekapselt werden und die Schnittstelle des Dienstes das Verhalten beschreibt. Die Implementierung des Dienstes ist für den Dienstnutzer nicht sichtbar. Die zweite Eigenschaft ist die lose Kopplung der Dienste. Dienste werden bei Bedarf zur Laufzeit von Applikationen oder anderen Diensten gesucht, gefunden und dynamisch eingebunden. Bei einer dynamische Bindung, welche das Suchen, Finden und die Einbindung eines Dienstes beinhaltet, muss es ein Verzeichnisdienst für Dienste geben. Dies ist die dritte Eigenschaft einer SOA. In einem Verzeichnisdienst sind verfügbare, zugreiffbare Dienste gegliedert und registriert. Sobald eine Anwendung einen Dienst suchen und auf diesen zugreifen muss, wird das Verzeichnis benötigt. Eine weitere Eigenschaft, ist die Verwendung von Standards. Sobald ein Aufrufer einen Dienst

im Dienstverzeichnis erfolgreich findet, folgt die Kommunikation durch die Schnittstelle als nächsten Schritt. Hierbei müssen sich Aufrufer und Anbieter des Dienstes über einen offenen Standard austauschen können. [51]

In [33] werden Service Autonomie und Wiederverwendbarkeit als weitere Eigenschaften ergänzt. Service Autonomie ist die Eigenverantwortung eines Teams für Implementierung, Wartung etc. und als Dienstanbieter die Beachtung von Verpflichtungen gegenüber den Kunden. Gleichzeitig bedeutet das, dass Dienste für Kunden wiederverwendbar sind.

4.1.2 Rollen und Aktionen in einer service-orientierten Architektur

In einer service-orientierten Architektur gibt es grundlegend drei Akteure, die bestimmte Aufgaben erfüllen und Rollen haben. Die Akteure sind Dienstverzeichnis, -anbieter und -nutzer.[51] Die Abbildung 4.1 zeigt in Form eines Dreiecks einen Ablauf für das Zusammenspiel zwischen Dienstverzeichnis, -anbieter und -nutzer. Die Pfeile beschreiben jeweils eine Aktion und zeigen das Verhältnis zwischen den Rollen.

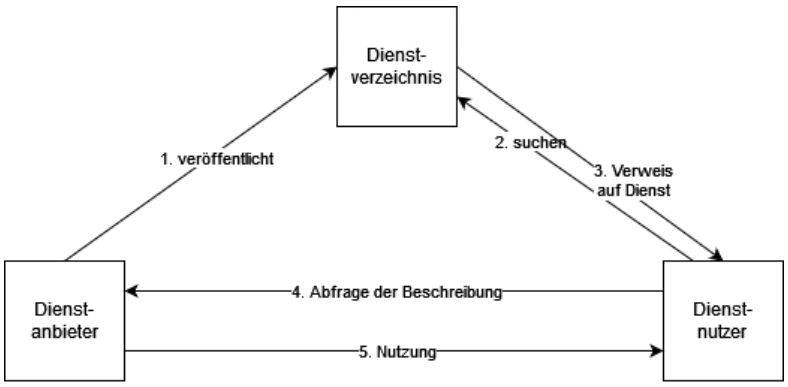

Abbildung 4.1: Ablauf zwischen Dienstverzeichnis, -anbieter und -nutzer [51]

Im Kontext der Abbildung 4.1 ist ein Dienst ein Programm oder eine Softwarekomponente. Dienste können über öffentliche Schnittstellen, die als Dienstbeschreibung beschrieben sind, vom Dienstnutzer zugegriffen werden. Eine Dienstbeschreibung ist eine Beschreibung der öffentlichen Schnittstelle mithilfe Beschreibungssprachen, die Programmiersprachen und Plattform unabhängig sind. Eine Beschreibungssprache gibt die Signatur eines Dienstes wieder. Des Weiteren gibt es auch nichtfunktionale Anforderungen, die in sogennanten „Service Level Agreements(SLA)“ Bedingungen, wie zum Beispiel maximale Antwortzeiten, Verfügbarkeit und Nutzungskosten, festgelegt sind. [51]

Auf der linken Seite ist der Dienstanbieter zu sehen. Der Dienstanbieter ermöglicht einen Zugriff auf einen Dienst. Der Zugriff über die öffentliche Schnittstelle kann lokal oder von außen bzw. Internet erfolgen. Gleichzeitig ist die Implementierung der Dienste für den Dienstnutzer nicht sichtbar. Für die Zugriffsmöglichkeit muss der Dienstanbieter seine Dienste in einem Verzeichnisdienst registrieren. Weitere Aufgaben des Dienstanbieters sind die Entwicklung und Wartung der Infrastruktur der Plattform sowie eine Verfügbarkeit des Betriebs zu garantieren. Zusätzlich muss der Dienstanbieter eine Sicherheit durch Authentifizierung und Authentisierung für die Nutzer gewährleisten. Neben der selbstentwickelten Diensten gibt es auch die Möglichkeit andere Dienste vom Netz zu kapseln und einen vereinfachten Zugriff zu ermöglichen. Auf der rechten Seite ist der Dienstnutzer zu sehen. Der Dienstnutzer ruft einen gewünschten Dienst auf und nutzt die öffentliche Schnittstelle. Beim Aufruf des Dienstes ist der Dienstnutzer auf offene Standards zur Kommunikation mit dem Anbieter angewiesen. Für die Kommunikation muss der Standard beiden Teilnehmern bekannt sein, denn dieser kann über ein Protokool, wie zum Beispiel „Simple Object Access Protocol (SOAP)“ erfolgen. In der Mitte des Dreiecks gibt es das Dienstverzeichnis. Der Anbieter registriert seine Dienste im Verzeichnis. Diese Dienste kann vom Dienstnutzer zugegriffen werden. Bei einem Dienstverzeichnis ist es das Ziel, dass Nutzer ihre benötigten Dienste im Verzeichnis finden. [51]

Betrachtet man die Aktionen an den Pfeilen, muss der Dienstanbieter als Erstes seine Dienste im Dienstverzeichnis veröffentlichen. Im zweiten Schritt sucht der Dienstnutzer nach dem gesuchten Dienst im Dienstverzeichnis. Wenn dieser gefunden wird, wird dieser im dritten Schritt vom Dienstverzeichnis verwiesen. In vierten Schritt wird die Dienstbeschreibung bzw. Signatur der Schnittstellen des gefundenen Dienstes abgefragt. Weitere Vorrausetzungen, wie die Authentifizierung und sonstige Richtlinien können ebenfalls abgefragt werden. Bei einer erfolgreichen Aushandlung findet im letzten Schritt die Nutzung des Dienstes durch den Dienstnutzer statt.[51]

4.1.3 Anwendung Closed-Ansatz

Der Architekturstil „SOA“ wird für den Aufbau der AWS Datenpipeline (Closed Ansatz) angewendet. Die folgende Abbildung 4.2 zeigt den technischen Kontext sowie die Anwendung.

Auf der Abbildung 4.2 sieht man auf der linken Seite eine Industrieanlage mit einer SPS. Auf der rechten Seite sind die Dienste von „Amazon Web Services (AWS)“ zu sehen. Die

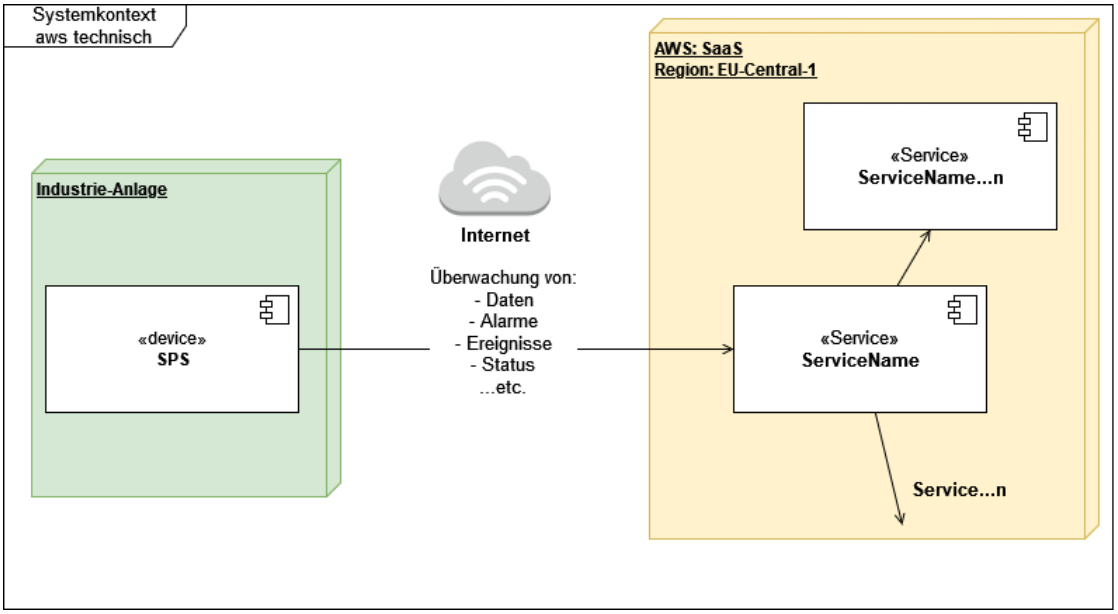

Abbildung 4.2: Technischer Systemkontext für den Closed-Ansatz

SPS verschickt über das Internet Sensordaten, Alarme, Ereignisse an den Dienstanbieter AWS. Je nach Konfiguration der AWS Dienste werden die Ereignisse, Alarme und Sensordaten an weitere AWS Dienste verschickt. AWS bietet zahlreiche Dienste an. Die genutzten Dienste sind in Kapitel 5 Abschnitt 5.2 zu finden. Der Aufbau orientert sich an Referenzarchitekturen [5] [12] von AWS, die Dienste für eine Überwachung vorstellen. Die genaueren Details des Aufbaues werden in Kapitel 7 in Abschnitt 7.1.1 beschrieben.

4.2 Ereignisbasierte Architektur

Grundlagen und Begriffe

Die ereignisbasierte Architektur ist ein Architekturstil für verteilte Softwarearchitekturen, bei dem Ereignisse im Fokus stehen [25] [33]. Das Ziel der ereignisbasierten Architektur ist die Effizienz und Agilität in Anwendungssystemen eines Unternehmen zu integrieren [25].

Ein Ereignis ist eine Veränderung eines Zustands, welches erwartet oder unerwartet sein kann. Das bedeutet, dass der Wert einer Eigenschaft eines realen oder virtuellen Objekts

sich verändern kann. Es wird zwischen technischen System- und Geschäftsereignissen unterschieden. Ein Beispiel für ein technisches Ereignis ist die Änderung eines Sensorwertes, wie Temperatur, Luftfeuchtigkeit etc. Mithilfe der Ereignisse lassen sich Abläufe in Geschäftsprozesse oder Systeme steuern.Bei Eintreffen eines Ereignisses von einer internen oder externen Quelle kann das Verhalten eines Systems, Geräts, Softwarekomponente etc. gesteuert werden. Das bedeutet, dass der Verarbeitungsprozess des Systems, Geräts oder Softwarekomponente etc. ereignisgesteuert ist. [25]

Ein ereignisgesteuertes System hat laut [25] prinzipiell drei Grundschritte, die sich in einem Zyklus befinden:

- 1.Schritt: Erkennen
  Wichtige Informationen und Sachverhalte, wie zum Beispiel durch Sensoren werden als Ereignisse unmittelbar ohne zeitlicher Verzögerung erkannt und erfasst. Es folgt die Generierung eines Ereignisobjektes, welches einen Zustand hält.
- 2.Schritt: Verarbeiten
  Im zweiten Schritt erfolgt die Verarbeitung. Ereignisse aus den unterschiedlichen Quellen können zusammengefasst, kategorisiert, vereinfacht und verworfen werden. Außerdem werden in den Ereignisströmen nach Muster, wie zum Beispiel Beziehungen und Abhängigkeiten gesucht.
- 3.Schritt: Reagieren von bzw. auf Ereignisse
  Im letzten Schritt erfolgt als Ergebnis auf die Analyse des Ereignisstromes eine Reaktion. Eine Reaktion kann weitere Prozesse auslösen, wie zum Beispiel eine Meldung als Warnung und/oder weitere Dienstaufrufe aussenden. Des Weiteren können auch neue Ereignisse als Folge generiert werden. Nach dem letzten Schritt fängt der Zyklus wieder mit Schritt 1 an.

4.2.1 Event-Driven Architecture (EDA)

Ein weiterer Grundbegriff, der nach [25] eingeführt und definiert wird, ist der Architekturstil „Event-Driven Architecture (EDA)“ . In einem EDA sind Komponenten ereignisgesteuert und kommunizieren, in dem sie untereinander Ereignisse austauschen. Bei der Modellierung, dem Entwurf und Kommunikation ist das zentrale Konzept die Ereignisverarbeitung [25]. Eine EDA hat laut [25] zwei wichtige Eigenschaften:

- Das Verarbeitungsmodell von EDA
  Ein ereignisgesteuertes System besitzt laut [25] ein Verarbeitungsmodell, welche folgende Eigenschaften und Elemente besitzt:

  - Ereignisquelle/Produzent:
    Ein Produzent hat die Aufgaben wichtigen Informationen zu erkennen und Ereignisse bzw. Ereignisobjekte zu erzeugen. Die Ereignisquelle bzw. Produzent versendet eine Nachricht zu einem bestimmten Zeitpunkt an einen Mediator, sobald ein neues Ereignis auftritt. Der Mediator ist für die richtige Verteilung der Nachrichten verantwortlich. Aufgrund der direkten Übermittlung des Ereignisses zeichnet sich die EDA vorallem durch seine hohe Aktualität aus.

  - Ereignissenke/Konsument:
    Als Ereignissenke bzw. Konsument bezeichnet man eine Softwarekomponente, der Ereignisse empfängt. Bei Eintritt einer Nachricht als Ereigniss folgt eine Reaktion durch die Ereignissenke.

  - Ereignisobjekt:
    Der Konsument empfängt Ereignisse als Nachrichten und führt als Reaktion Operationen und Verarbeitungsschritte durch. Ein Ereignisobjekt bzw. Ereignis enthält keine Informationen über die nächsten Aktionen oder Verarbeitungsschritte.

- Das Kommunikationsmuster von EDA
  In einem ereignisgesteuerten System gibt es einen Austausch von Ereignissen. Folgende Kommunikationsmuster mit drei Eigenschaften gibt es laut [25]:
  1. Asynchronität
  Bei der Asynchronität werden zwei Interaktionsarten zwischen den Komponenten unterschieden.

  - Synchrone Interaktion: Die synchrone Interaktion ist eine Anfrage/Antwort Interaktion. Es verhält sich wie ein „Callback“ . Dabei ist die aufrufene Komponente und die Verarbeitung blockiert. Bei Erhalt der Antwort steht die aufrufene Komponente wieder zur Verfügung.

  - Asynchrone Interaktion: Bei einer asynchronen Interaktion muss die aufrufende Komponente auf die Antwort der aufgerufenen Komponente nicht warten.

Während eines Kommunikationsvorganges muss die aufgerufene Komponente nicht verfügbar sein.

2. Publish/Subscribe-Interaktion
In einer Publish/Subscribe Interaktion unterscheidet man zwischen Sender bzw. Produzent und Empfänger bzw. Konsument. Ein Produzent sendet Daten in Form von Nachrichten an eine Middleware, die allen Interessenten unter einer bestimmten Nachrichtentyp veröffentlicht. Ein oder mehrere Konsumenten können diese Nachrichten abonnieren, sodass jeder Konsument nach dem Versenden der Nachrichten diese zu einem bestimmten Zeitpunkt empfangen kann. Die Reihenfolge der Verarbeitungsschritte ist bei diesem Kommunikationsmuster nicht geordnet.
3. Push-Modus
Im Push-Modus bestimmt nur die Ereignisquelle zu welchem Zeitpunkt ein Nachrichtenaustausch stattfinden soll.

Eine Entkopplung der beteiligten Komponenten erreicht man mit dem Kommunikationsmuster Publish/Subscribe im Push-Modus. Eine Kombination aus Asynchronität und Entkopplung schafft die Basis für Skalierbarkeit und Interoperabilität in einem EDA. Entkopplung, Skalierbarkeit und Asynchronität gehören zu den Vorteilen eines EDA. Verzögerungen aufgrund zu vielen Ereignissen und hoher Aufwand für die Verarbeitung (Filtern, Sortieren etc.) der Ereignisse gehören zu den Nachteilen. [25]

4.2.2 Arten der Ereignisverarbeitung

In der Ereignisverarbeitung gibt es laut [36] [52] [25] generell drei Arten: einfach, flüssig und komplex.

1. „Simple Event Processing“: Ein einfaches Ereignis tritt auf. Eine Aktion wird daraufhin aktiviert. Ein einfaches Ereignis kann zum Beispiel eine Zustandsänderung eines Sensors sein. Als Aktion kann eine Warnmeldung ausgegeben werden.

2. „Stream Event Processing“: Ereignisse treffen als Strom ein und werden verarbeitet. „Stream Event Processing“ wird für Echtzeit-Informationslflüsse zum Beispiel für die Steuerung von zeitnahen Entscheidungen im Unternehmen verwendet.

3. „Complex Event Processing“: Komplexe Ereignisse aus mehreren Ereignisquellen treffen ein und werden analysiert und bewertet, um Muster und Korrelationen zu

finden und Daten herauszufiltern. Aufgrund der Bewertung folgt eine Reaktion. Der Eintritt und Bewertung der Ereignisse können in einem bestimmten Zeitraum stattfinden.

4.2.3 Anwendung Open-Source Ansatz

Der Architekturstil einer ereignisbasierten Architektur wird für den Aufbau der Open-Source Datenpipeline angewendet. Die folgende Abbildung 4.3 zeigt den technischen Kontext sowie die Anwendung.

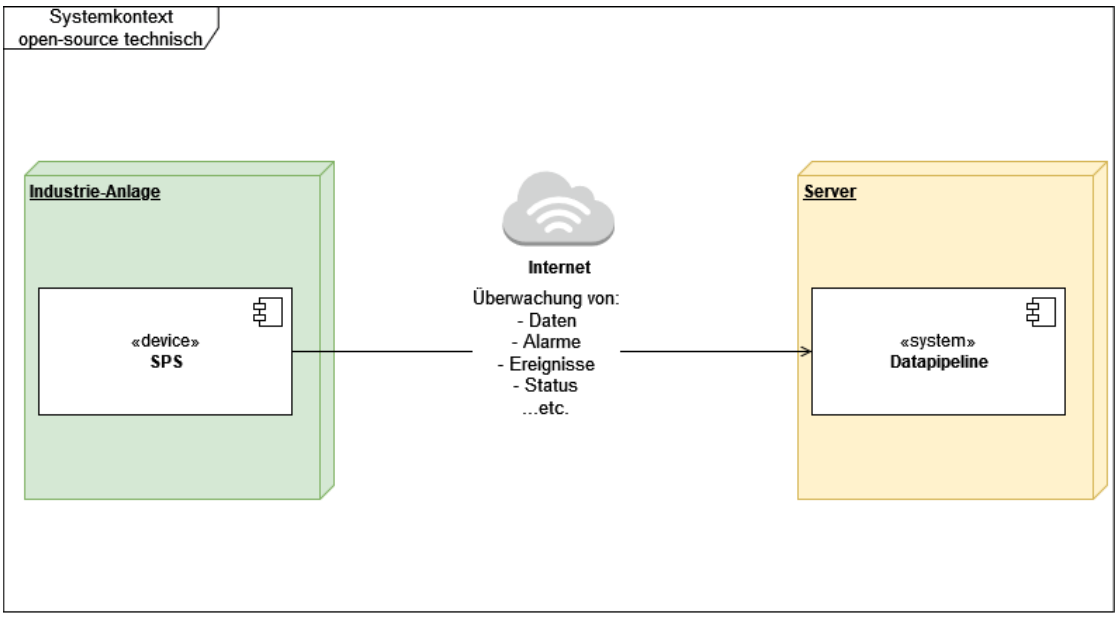

Abbildung 4.3: Technischer Systemkontext für den Open-Source Ansatz

Auf der linken Seite versendet die SPS Daten an das System, welches auf der rechten Seite zu sehen ist. Die SPS ist in diesem Fall die Ereignisquelle. Die Datenpipeline ist die Ereignissenke und enthält Ereignisobjekte. Die Verarbeitung des Systems ist „Simple Event Processing“. Die Daten durchlaufen alle drei Grundschritte eines ereignisbasierten Systems. Das Versenden der Nachrichten von der SPS zur Open-Source Datenpipeline wird mit dem Publish/Subscribe Kommunikationsmuster realisiert. Es wird das Protokoll „MQTT“ ,welches ebenfalls nach dem Publish/Subscribe Prinzip funtkioniert, verwendet. In der Datenpipeline findet der Austausch asynchron (Kommunikationsmuster: Asynchronität) statt. Die genaueren Details zum Aufbau sind im Kapitel 7 in Abschnitt 7.1.2 beschrieben.

4.3 Ergänzung der Service-orienterte Architektur

Es ist ebenfalls möglich, dass EDA eine SOA ergänzen kann [33]. Die Kombination aus EDA und SOA wird auch als „SOA 2.0“ oder „Event-Driven SOA“ bezeichnet [33] [26]. Das bedeutet, dass es Interaktionen zwischen SOA und EDA gibt. Es können laut [52] zwei Arten von Interaktionen stattfinden.

- In der ersten Interaktion tritt ein internes oder externes Ereignis im System auf. Das Ereignis kann Dienste aufrufen. Die Dienste können einfache Aktionen sein oder Geschäftsprozesse auslösen. Die erste Interaktion nennt man ereignisbasierte SOA.
- In der zweiten Interaktion kann ein Dienst ein Ereignis generieren. Das Ereignis kann ein Problem, ein Hinweis auf ein bevorstehendes Problem, Schwellenwert oder Abweichung sein. Das Ereignis wird an alle interessierten Komponenten gesendet. Die Komponenten bewerten das Ereignis. Nach der Bewertung kann eine ereignisgesteurte Aktion als Antwort gesendet werden. Die Aktion kann ein Dienstaufruf, einen Geschäftsprozess auslösen oder weitere Informationen an andere Komponenten veröffentlichen. Der Dienst ist in dieser Interaktion eine der Ereignisquellen in einer ereignisgesteurten Architektur.

Komponenten mit einer starken Abhängigkeit kommunizieren über Serviceaufrufe. Externe, lose gekoppelte Komponenten, die ein Unternehmen verwendet, kommunizieren über Ereignisse. [33]

In Abschnitt 4.1.3 wird die Anwendung von SOA beschrieben. Der Aufbau des Closed-Ansatzes ist eine „Event-Driven SOA“.Die Daten durchlaufen ebenfalls bei AWS alle drei Grundschritte eines ergeinisgesteuerten Systems. Daten werden erfasst, verarbeitet bzw. gefiltert und in einer Visualisierung angezeigt. Die Reaktion auf Ereignisse, wie z.B.: in Form von Alarmen und Steuerungen sind aufgrund der definierten Use-Cases und Fokus in dieser Arbeit nicht realisiert worden. AWS Dienste bieten jedoch die Möglichkeit auf bestimmte Ereignisse und Werte zu reagieren.

5 Verwendete Technologien

Für den Aufbau und die Vergleichstudie zwischen Open-Source und Closed werden verschiedene Dienste, Frameworks, Bibliotheken verwendet. Das Kapitel beginnt mit der Erklärung von speicherprogrammierbare Steuerungen (SPS), welches als Echtzeitanbindung und Datenquelle verwendet wird. In Absatz 5.2 wird der Closed-Ansatz vorgestellt. Absatz 5.3 schließt das Kapitel 5 mit dem Open-Source Ansatz ab.

5.1 Hardware

5.1.1 Speicherprogrammierbare Steuerungen (SPS)

Als Datenquelle für beide Datenpipelines wird eine SPS verwendet. Eine speicherprogrammierbare Steuerung (SPS) ist für Überwachungs-, Regelungs- oder auch Steuerungsaufgaben zuständig. Diese kontrolliert komplexe Automatisierungsprozesse, wie z.B.: Abläufe, Maschinenfunktionen etc. Speicherprogrammierbare Steuerungen unterscheiden sich von Leistungsmerkmalen, wie z.B.: Verarbeitungsgeschwindigkeit, Vernetzungsfähigkeit oder Peripherie-Baugruppen. Für die Programmierung und Entwicklung von Steuerungen, einer Überwachung oder Regelungen gibt es mehrere Programmiersprachen, die man nutzen kann. Es gibt Richtilinien für die Programmierung, wie zum Beispiel die internationale Norm IEC 1131. [42] [37]

Aufbau

Funktionale Grundstruktur
Eine SPS hat nach [37] eine funktionale Grundstruktur. Die Funktionen sind für die Stromversorgung, Speicherung, Schnittstellen und für das Betriebssystem. Es gibt hierfür drei Funktionen:

- Stromversorgungsfunktionen
- Signalverarbeitungsfunktionen
  - Verarbeitung der Signale von Sensoren und internen Datenspeichern.
  - Erzeugung von Signale, um diese an Aktoren und weitere Speicher weiterzuleiten.
  - Funktionen für die Speicherung des Anwenderprogramms, Daten und das Betriebsystem.
- Schnittstellenfunktionen
  - Datenaustausch mit anderen Prozessen, Bedienern, Programmierern und Systemen.
  - Umwandlung von Signalen

Hardwareaufbau
Eine SPS besteht im einfachsten Aufbau aus einer Stromversorgungseinheit, einer Zentraleinheit(CPU), die mit Ein- und Ausgabeeinheiten verbunden sind. Ein- und Ausgabeinheiten sind modular nach Domäne, Aufgabenbereich, Anforderungen und Use-Cases austauschbar. [37]

- Zentraleinheit (CPU)
  - Besteht aus einem Steuerwerk, Lade-, System- und Arbeitspeicher. In diesen Komponenten werden Steuerungsanweisungen und -aufgaben bearbeitet.
- Ein- und Ausgabeeinheiten
  - Bearbeitung von Ein- und Ausgangssignalen

Arbeitsweise

In einer SPS ist die Bearbeitung der Anweisungen und Programme zyklisch. Jeder Zyklus beginnt mit einer Abfrage der Eingänge. Anschließend werden Anweisungen nacheinander abgearbeitet. Sprünge zwischen den Anweisungen sind ebenfalls möglich. Der Zyklus endet mit dem Setzen der Ausgänge. Die Zykluszeit, welches die Zeit für einen Durchlauf bestimmt, variiert mit der Größe des Programms. Bei der Abarbeitung der Programme

gibt es eine Struktur unter den Bausteinen, die die zyklische Bearbeitung bestimmt. In einem Programm bzw. einer Anweisung werden Bausteine aufgerufen. Die Bausteine unterscheiden sich durch ihre Funktion und ihrem Zweck. Man unterscheidet zwischen Programm- und Datenbausteine. Programmbausteine dienen zur Implementierung von Programmlogiken, Funktionen etc. Auf der anderen Seite dienen Datenbausteine für die Speicherung von Variablen, wo der Zugriff durch Programmbausteine entweder lokal oder global erfolgen kann. [37] [59]

5.2 Closed Ansatz

In diesem Abschnitt werden die verwendeten Technologien des Closed Ansatzes beschrieben. Für den Closed Ansatz wird eine Datenpipeline mithilfe „Amazon Web Services (AWS)“ aufgebaut. Die Eigenschaften und das Konzept von Rollen und Aktionen in einer SOA ist anhand von „Amazon Web Services (AWS)“ mit „Amazon“ als Anbieter wiederzufinden. Verschiedenste Dienste werden angeboten, wie z.B.: Virtualisierung, Applikationen, Infrastrukturen, Plattformen etc. Die Nutzung der Dienste kann über die AWS Management Konsole, Kommandozeile, über API Schnittstellen erfolgen. Die nachfolgenden Dienste sind Teil der Datenpipeline des Closed Ansatzes. Die Architekur der Closed Ansatzes ist nach dem Stil 4.3 aufgebaut, da einige Dienste ereignisgesteuert sind. [1]

5.2.1 AWS IoT

Um die SPS mit der Cloud-Plattform zu verbinden und die Daten weiterzuverarbeiten, wird der Dienst „AWS IoT“ genutzt. „AWS IoT“ ermöglicht die einfache und schnelle Verbindung von Geräten mit AWS Diensten und anderen Geräten. Mögliche Kommunikationsprotokolle für den Datenaustausch sind MQTT, HTTP oder WebSockets. Des Weiteren können die Nachrichten an weitere Dienste, wie z.B.: siehe Abschnitt 5.2.2 weitergeleitet werden. Die Voraussetzung hierfür, ist die Erstellung von Regeln, die die Konfigurationen, wie z.B.: Auswahl der Datenquelle, Datenbank, Puffergröße der Nachrichten festlegen. [20] [8]
Zu den wichtigsten Funktionen für die Datenpipeline gehören nach [7] [6] [20]:

- Device Gateway

  - Der Zugangspunkt für IoT-Geräte, die sich mit AWS verbinden.
  - Verwaltung aller aktiven Geräteverbindungen für die effiziente Kommunikation mit dem „AWS IoT Core“ .

- Message Broker
  - „Pub/Sub-Message Broker“ mit hohem Durchsatz und geringer Latenz bei der Übertragung von Nachrichten.
  - Automatisch skalierbar gemäß Nachrichtenvolumen

- Authentifizierung
  - Authentifizierung auf Basis des X.509-Zertifikats
  - Bietet die Authentifizierung und Verschlüsselung an allen Verbindungspunkten.

- Rules Engine
  - Sammlung von Daten von den verbundenen Geräten, Verarbeitung, Analyse.
  - Auswertung von ankommenden Nachrichten, Transformation und Weiterleitung an ein anderes Gerät oder „Cloud-Service“ .
  - Regeln werden mit einer SQL-Syntax festgelegt. Zum Beispiel kann eine Schwellenwert Überschreitung eine Regel auslösen.

5.2.2 AWS Kinesis Data Firehose

Für die Bereitstellung der Daten in eine Datenbank gibt es „AWS Kinesis Data Firehose“ , der ein vollständig verwalteter Dienst ist. Die Aufgabe von „AWS Kinesis Firehose“ ist es, Echtzeit-Datenströme an Datenbanken, wie z.B.: „Amazon Simple Storage Service (Amazon S3)“ , „Amazon Redshift“ zu übermitteln. Dienstrichtlinien bestimmen den genauen Ressourcenverbrauch eines Nutzers. Datenproduzenten, Ziel der Daten, Puffergröße und Pufferungszeit etc. können konfiguriert werden. Außerdem ist es möglich während einer Übertragung eine Datentransformation in ein kompakten Dateiformat, wie z.B.: Apache Parquet durchzuführen. [15]

5.2.3 AWS Timestream

Für die Speicherung von Daten in der Datenpipeline gibt es „AWS Timestream“ . Es ist eine schnelle, skalierbare, vollständig verwaltete Zeitreihendatenbank. Es ermöglicht die Speicherung und Analyse von aktuellen und historischen Daten. „AWS Timestream“ ist serverlos und skaliert automatisch. Die Kapazität und Leistung werden je nach Nutzung angepasst. Jedoch gibt es hierfür auch Dienstrichtlinien. Weitere wichtige Kernfunktionen von AWS Timestream sind die Hochverfügbarkeit und Dauerhaftigkeit. Hochverfügbarkeit wird durch die automatische Replikation von Daten bei Schreib- und Leseanforderung für mindestens drei „Availability Zones (AZs)“ gewährleistet. Verfügbarkeit wird durch eine weitere Speicherung der Daten und automatische Replikation dieser Daten über sogenannte „Availability Zones (AZs)“ gewährleistet. AZs sind weitere Datenzentren, die ebenfalls Dienste, Infrastrukturen etc. anbieten. Verknüpfungen mit Visualisierungen wie „Amazon Managed Service for Grafana“ sind ebenfalls möglich. [18]

Architektur

Die Architektur von „AWS Timestream“ besteht aus voneinander, unabhängigen, entkoppelten Systemen. Die Architektur besteht laut [3] aus:

- Dateneingabe
  - Verarbeitung von Datenschreibvorgängen
  - Durchführung von Aufgaben: Indizierung, Erstellung von Partitionen, Replikation.
- Speicherung
  - Organisation der Aufbewahrung von Daten und deren zeitlichen Reihenfolge (Zeitstempel).
  - Aufteilung der Partitionen bei wachsenden Datensätzen
- Abfrageverarbeitung
  - Verarbeitung der Abfragen mithilfe einer Abfrage-Engine.
  - Bietet Abfragen in SQL-Syntax.

5.2.4 Amazon S3

„Amazon S3“ ist eine weitere Datenbank von AWS. „AWS IoT“ kann durch deine Erstellung einer Regel Daten in „Amazon S3“ weiterleiten. Es ist ein Objektspeicherdienst. Es zeichnet sich durch Skalierbarkeit, Datenverfügbarkeit, Sicherheit und Leistung aus. Die Speicherung von beliebigen Datenmengen als Objekte erfolgt in „Buckets“ . Ein Objekt ist eine Datei mit Metadaten. Ein Bucket, welches ein Container für ein Objekt ist, kann beliebig viele Objekte enthalten. Ein Objekt ist über einen Schlüssel eindeutig identifizierbar. PUT und GET-Befehle können auf die Buckets zugreifen. Die Daten können innerhalb „Amazon S3“ organisiert, optimiert und konfiguriert werden. „Amazon S3“ bietet noch weitere Funktionen, wie z.B.: Zugriffs- und Speicherverwaltung, Versionskontrolle, hohe Verfügbarkeit durch Speicherung der Daten in andere Rechenzentren an oder hohe Konsistenz an. [17]

5.2.5 Amazon Glue

Daten in der Datenpipeline müssen laut Anforderungen manipulierbar und für Visualisierungen integrierbar sein. Eine Möglichkeit dafür bietet „AWS Glue“ . Es ist ein serverloser Datenintegrationsservice für Daten aus mehreren Datenquellen, wie z.B.: „Amazon S3“ , „Amazon Athena“ . Funktionen von „AWS Glue“ lassen sich laut [19] in drei Kategorien einordnen:

- Erkennen und Organisieren von Daten
- Transformieren, Vorbereiten und Bereinigen von Daten für die Analyse
- Erstellen und Überwachen von Datenpipelines

Konzept

Laut [19] müssen bei „AWS Glue“ Aufträge definiert werden. Diese Aufträge bestimmen wie Daten extrahiert, transformiert und aus einer Datenquelle geladen werden.

Um die Daten erkennen, extrahieren und verarbeiten zu können, muss als Erstes ein „Crawler“ definiert und ein Datenkatalog mit Datenstromeigenschaften erstellt werden. Ein „Crawler“ durchsucht einen Datenspeicher, um einen Datenkatalog füllen zu können. Der „Crawler“ untersucht das Schema des Datenspeichers. Ein Datenkatalog definiert

Tabellen, Aufträge und Steuerungsinformationen. Der Datenkatalog enthält ebenfalls Metadaten, die für eine Transformation der Daten in einem Auftrag verwendet werden können.

Alle benötigten Daten für die Extrahierung, Transformation und Verarbeitung werden für den Auftrag gesammelt und als Skript ausgeführt.

5.2.6 Amazon Athena

Eine Datenquelle für eine Visualisierung in AWS ist „Amazon Athena“ . Es ist ein serverloser Dienst, welches interaktive Abfragen aus anderen Datenspeicher und -quellen, wie z.B.: „Amazon S3“ , „AWS Glue Datacatalog“ etc. durchführt. Der Zugriff der Daten erfolgt über einen „Abfrage-Editor“ . Die Abfragen verwenden den Standard-SQL Syntax. [4]

5.2.7 Amazon Managed Service for Grafana

Für die Visualisierung bietet AWS „Amazon Managed Service for Grafana“ als Datenvisualisierungstool an. Es ist vollständig verwaltetet und sicher. Die Funktionen und Möglichkeiten werden in Abschnitt 5.3.4 beschrieben. Ein Unterschied zum Open-Source Grafana ist, dass mehr Datenquellen verfügbar sind. [16]

5.3 Open-Source Ansatz

In den folgenden Abschnitten werden verwendete Frameworks, Bibliotheken und Werkzeuge für den Aufbau der Open-Source Datenpipeline eingeführt. Im Abschnitt 5.3.1 wird die „Message Queue“ Apache Kafka, in Abschnitt 5.3.2 Apache Spark für die Datenverarbeitung und in Abschnitt 5.3.3 die Datenbank Apache Cassandra eingeführt. Das Kapitel wird mit dem Visualisierungswerkzeug Grafana in Abschnitt 5.3.4 abgeschlossen.

5.3.1 Apache Kafka

Apache Kafka ist eine Open-Source Event-Streaming-Plattform [45] [2]. Nach [2] kombiniert Apache Kafka drei Kernfunktionen für Anwendungsfälle mit Ereignisströme:

- Das Veröffentlichen und Abonnieren von Ereignisströmen („Publish-Subscribe"-Prinzip). Veröffentlichen bedeutet das Schreiben von Datenströmen. Abonnieren bedeutet das Lesen aus den Datenströmen. Kontiniuierliche Daten können aus anderen Systemen importiert oder in andere Systeme exportiert werden.
- Speichern von Ereignisströmen
- Verarbeitung von Ereignisströmen. Die Verarbeitung kann bei Eintritt oder im Nachhinein erfolgen.

Hauptkonzept und Funktionsweise

Apache Kafka hat die Eigenschaften eines hoch skalierenden, fehlertoleranten, elastischen, sicheren, verteilten System. Das verteilte System von Kafka besteht aus Servern und Clients. Server und Clients kommunizieren über TCP. Apache Kafka kann in einem Cluster bestehend aus einem Server oder mehreren Server betrieben werden. [2]

Ein Server kann zum Beispiel die Speicherebene als „Broker" abbilden. Ein weiteres Beispiel für die Anwendung des Servers, ist der Export und Import von kontiniuierlichen Daten mithilfe Kafka Connect. Bei Kafka Connect können Daten aus bestehenden Systemen, wie relationale Datenbanken, MQTT in Kafka Cluster integriert werden. Ein Kafka Cluster ist hoch skalierbar und fehlertolerant, da bei einem Server-Verlust andere Server die Daten und Operationen übernehmen. [2]

Mit Kafka „Clients" ist es möglich verteilte Anwendungen und „Microservices" zu schreiben. Die verteilten Anwendungen und „Microservices" können Ereignisströme lesen, schreiben und verarbeiten. Hierbei ist es durch das fehlertolerante System von Kafka möglich, die Lese-, Scrheib-, und Verarbeitungsoperationen bei Networkproblemen oder Rechnerausfällen auszuführen. [2]

Produzent und Konsumenten

Ein Kafka Client als Kafka Produzent kann Ereignisse produzieren und diese veröffentlichen. Ein Kafka Client als Kafka Konsument kann die Ereignisse abonnieren, lesen und

verarbeiten. Durch das „Publish-Subscribe“-Prinzip mit entkoppelten Produzenten und Konsumenten ist eine hohe Skalierbarkeit möglich. Des Weiteren müssen Produzent auf die Ereignisse der Konsumenten nicht warten. Für die Nachrichtenübermittlung bietet Kafka drei verschiedene Garantien an. Ein Beispiel hierfür ist die „Exactly once“ Garantie. Bei „Exactly once“ darf eine Nachricht nur einmal übermittelt werden. [2] [45]

Kafka Broker

Ein Kafka Broker ist größtenteils für E/A-Operationen und für die dauerhafte Persistenz innerhalb eines Clusters zuständig. Es hat die Aufgabe eines Vermittlers zwischen Produzent und Konsumenten. Es erleichtert die Interaktionen zwischen zwei Parteien. Der Kafka Broker ist zustandsorientiert und bewahrt die veröffentlichten Daten vom Produzenten auf, sodass ein Konsument dauerhaft über einen Zeitraum auf die Daten zugreifen kann. Mithilfe der Kafka Broker ist Kafka skalierbar. Erhöht man die Anzahl der Broker, desto mehr E/A-Operationen können ausgeführt werden. Außerdem wird die Verfügbarkeit und Beständigkeitseigenschaft verbessert. [2] [45]

Jede Partition hat genau einen Broker als Partitionsführer. Eine Gruppe von null oder mehr Anhänger-Brokern kann die replizierten Daten einer Partition enthalten. Partitionsführer und Anhänger bezeichnet man als Kollektiv Replikate. Die Last wird unter Partitionsführern und Anhängern aufgeteilt. Es ist möglich, dass ein Broker Knoten für bestimmte Replikate der Partitionsführer ist, während dieser für andere als Anhänger dient. Die Rollen können sich auch im Falle eines Ausfalls ändern. Die Wahrscheinlichkeit eines Datenverlusts ist geringer, je mehr Replikate es in einem Cluster gibt. [2] [45]

Cluster Controller

Für die Rollenzuweisung der Broker, wie zum Beispiel die Zuweisung eines Partitionsführers in einem Cluster, gibt es einen Cluster „Controller“ . Ein Cluster „Controller“ hat die Aufgabe die Zustände der Partitionen und Replikaten zu verwalten und die Ausführung von administrativen Aufgaben. [2] [45]

Zookeeper

Ein Zookeper Knoten ist kein interner Bestandteil von Kafka sondern ein eigenständiges Open-Source-Projekt. Es ist ein Cluster, welches aus kooperierenden Prozessen besteht. Zookeeper ist verantwortlich für die Wahl eines „Controllers“ und stellt sicher, dass ein Broker Knoten die Aufgabe eines „Controllers“ übernimmt. Sobald ein „Controller“ Knoten fehlschlägt oder einen Cluster verlässt, übernimmt ein anderer Broker Knoten die

Aufgabe. Es ist auch ein konsistenter und hochverfügbarer Speicherort für Metadaten, Nutzerinformationen. [2] [45]

Ereignisse, Datensatz und Themen

In Kafka ist ein Datensatz, auch „Record“ genannt, die elementarste Einheit der Persistenz [45]. Abgrenzend zum Begriff „Datensatz“ kann ein Ereignis mehrere Kafka Datensätzen hervorbringen. Ein Ereignis kann ebenfalls nur aus einem Datensatz bestehen. Datensätze sind in sogenannten Themen gespeichert und organisiert. Die Datensätze existieren in den Themen so lange, bis man diese nicht mehr benötigt. Das bedeutet, dass man Datensätze wiederholt lesen kann. Über eine Konfigurationseinstellung lässt sich pro Thema festlegen, wie lange die Datensätze aufbewahrt und welche alten Datensätze verworfen werden sollen. Daten können in ein Thema von keinem, einem oder mehreren Produzenten geschrieben werden. Auf der anderen Seite können Themen von keinem, einem oder mehreren Konsumenten abonniert werden. [2] [45]

Partitionen und Themen

Partitionen sind elementare Einheiten des Datenstroms und eine vollständig geordnete, unbeschränkte Menge von Datensätzen. Ein logische Zusammenstellung von Partitionen nennt man Thema. Es kann eine oder mehrere Partitionen umfassen. Eine Partition muss ein Teil von genau einem Thema sein. Durch die Verteilung der Daten in Partitionen ist es Klienten-Anwendungen möglich gleichzeitig Lese- und Schreiboperationen von/zu vielen Brokern durchzuführen. Die verteilte Platzierung von Daten und die Möglichkeit des Lesens und Schreibens von/zu vielen Brokern ist eine Eigenschaft der Skalierbarkeit, Parallelität und Lastausgleich. Sobald ein neuer Datensatz von einem Produzenten veröffentlich wird, wird diese an das Kopfende einer Partition hinzugefügt. Bei der Anordnung von neuen Datensätzen können Beziehungen zwischen Produzenten und Partitionen entstehen. Kafka garantiert den Konsumenten beim Auslesen einer Themen Partition, dass die Reihenfolge in denen die Daten in die Partitionen geschrieben worden sind, eingehalten wird. Die Anordnung und Identifikation erfolgt über die Offsets der Datensätze. Der Offset agiert wie ein Primärschlüssel. Mit jedem neu hinzugefügten Datensatz wird der Offset als monoton ansteigende Ganzzahl in einem Adressraum erhöht Es ist ebenfalls möglich Daten durch Kopien über mehrere Broker aus anderen Regionen oder Datenzentren zu replizieren. Auf der anderen Seite werden Partitionen neu zugewiesen, sobald ein Konsument innerhalb einer Frist die Datensätze nicht ausliest oder fehlerhaft sind. Kafka ist anhand der Datenreplikation fehlertolerant und hat eine hohe Datenverfügbarkeit. [2] [45]

Konsumentengruppen und Lastverteilung

In Absatz 5.3.1 wird Kafka Konsument eingeführt. Es ist ein Prozess oder Thread, welches sich über eine Bibliothek an einem Kafka-Cluster anschließt. Eine Gruppe von Konsumenten wird als Lastausgleichsmechanismus für die Zuweisung der Partitionen verwendet. Durch die Partitionszuweisungen wird die Last auf die einzelnen Konsument annähernd gleich verteilt. Ein Konsument hat jedoch keinen Einfluss auf das Thema und Unterteilungen der Partitionen. Außerdem entnimmt ein Konsument keinen Datensatz aus dem Thema, sondern verwaltet intern einen Offset. Der Offset verweist bei jedem Lesevorgang auf den nächsten Datensatz einer Partition. Das bedeutet, dass der Konsument intern bei Verbrauch von Datensätzen den Offset verschiebt. Beim Auslesen der Partitionen gibt es Regeln, wenn zwei oder mehr Konsumentengruppen mit den jeweiligen Konsumenten Daten auslesen wollen. Pro Partition darf nur ein Konsument der jeweiligen Konsumentengruppe auslesen. Anhand den genannten Eigenschaften stellen Konsumentengruppen eine Verfügbarkeit sicher. [2] [45]

Architektur

Die folgende Abbildung 5.1 zeigt die Beziehungen zwischen den Kernkonzepten. Es werden alle eingeführten Grundbegriffe der Kafka Architektur in einem Zusammenhang gezeigt.

In Abbildung 5.1 sieht man einen Produzenten, zwei Konsumentengruppen, einen Broker Knoten als Gruppenkoordinator und Controller, die Abbildung eines Kafka-Cluster und ein Thema, welches Partitionen mit Datensätzen enthält.

Ein Kafka-Cluster hat ein oder mehrere Themen und besteht aus einem Broker Knoten. Der Kafka-Cluster zeigt auf einen Cluster Controller. Ein Broker Knoten ist genau einer Partition als Partitionsführer zugeordnet. Dabei kann es keinen oder mehrere Anhänger-Broker geben. Diese nennt man Replikate. Gleichzeitig kann der Broker Knoten der Cluster Controller und Gruppenkoordinator sein. Ein Gruppenkoordinator beaufsichtigt eine Konsumentengruppe. Konsumentengruppen sind unabhängig voneinander. Eine Konsumentengruppe besteht aus ein oder mehreren Konsumenten. Konsumenten teilen sich eine Partition und sind jeweils exklusiv einer Partition zugeordnet. Eine Konsumentengruppe abonniert ein Thema. Ein Produzent veröffentlicht Daten zu einem Thema. Ein Thema enthält mehrere Partitionen. Ein Partition enthält einen oder mehrere Datensätze.

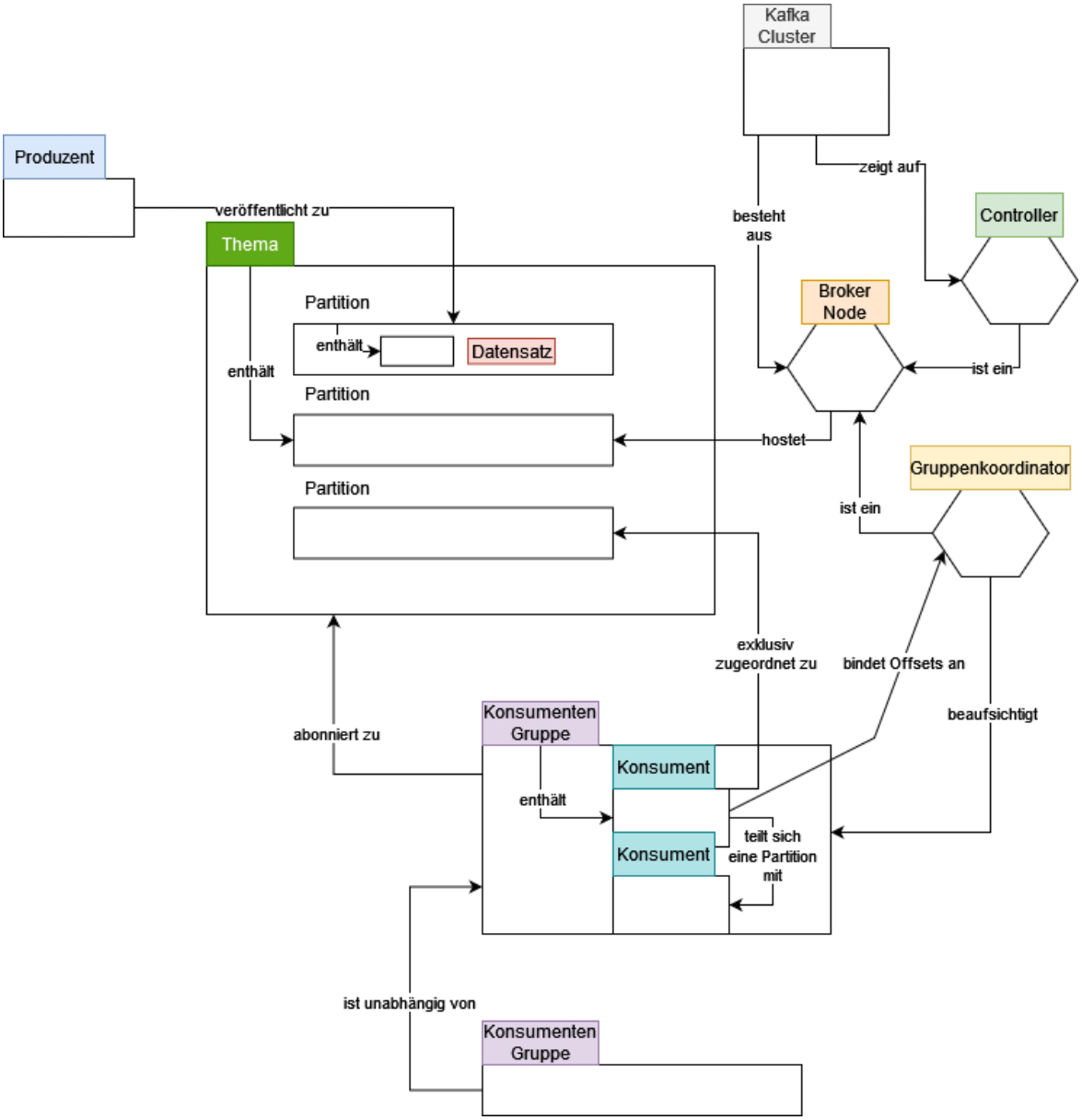

Abbildung 5.1: Beziehungen zwischen den Kernkonzepten

Veränderbarkeit von Apache Kafka

Im Rahmen der Durchsatz- und Latenzmessungen gibt es veränderbare Parameter für Kafka Proudzenten und Konsumenten. Die Parameter konzentrieren sich auf jeweilige Designziele, sodass die veränderbaren Parameter für Latenz und Durchsatz konkurrierend sind. In der folgenden Aufzählung werden die Parameter erklärt [2] [28] [27]:

- Produzent: *linger.ms*
  - Es baut eine Verzögerung ein, um Anfragen zu bündeln und die Anzahl an Anfragen zu minimieren.
- Produzent: *compression.type*
  - Kompremiert die Daten nach einem Komprimierungscodec. Die Nutzung der Netzwerkbandbreite wird verringert.
- Produzent: *batch.size*
  - Es gibt an, wie viele Bytes an Daten gebündelt werden sollen, bevor diese geschickt werden. Es verringert es Anzahl an Anfragen. Sind *linger.ms* und *batch.size* eingestellt, wird auf die Größe des *batch.size* nicht gewartet. Das Senden richtet sich dann nach der Einstellung von *linger.ms*.
- Konsument: *fetch.min.bytes*
  - Es gibt die Mindestmenge an Daten vor, die der Server geschickt hat. Erreicht die Abfrage nicht die Mindestmenge, wird gewartet bis die Mindestmenge erreicht wird.

5.3.2 Apache Spark

Apache Spark ist eine Open-Source einheitliche „Verarbeitungsengine“ und beinhaltet eine Reihe von Bibliotheken für die Parallelverarbeitung von Daten auf Clustern. Spark hat eine große, variable Ansammlung an Bibliotheken für verschiedene Aufgaben. Diese reicht von SQL-Abfragen bis hin zu Streaming und maschinellem Lernen. Der Fokus in dieser Arbeit liegt auf die Datenverarbeitung mit SQL-Abfragen und die Weiterleitung der Daten zu einer „Big Data“ Datenbank. Mit Spark ist es möglich schnelle Berechnungen im Speicher auszuführen und gleichzeitig effizient komplexe Berechnungen auf der Festplatte auszuführen. Ein weiterer Vorteil für die Nutzung von Spark, ist die Kombination von verschiedenen Verarbeitungstypen in einer Engine. Es sind keine weiteren verteilte Systeme für Batch-Anwendungen, iterative Algorithmen oder iteraktive Abfragen und Streaming erforderlich. Die Verwaltung und Wartung von seperaten Werkzeugen/Bibliotheken wird ebenfalls verringert. [31] [44]

Nach [11] gibt es folgende Operationen bei Apache Spark:

- **Extrahieren:** Das Erfassen von Daten aus einer oder mehreren Datenquellen.
- **Transformation:** Daten transformieren und mit Abfragen bearbeiten. Für die Transformation von Daten gibt es Möglichkeiten eine Filterung, Sortierung, Aggregation, Verknüpfung, Bereinigung, Reduplizierung, Überprüfung der Daten durchzuführen.
- **Laden:** Verschieben der Daten in einen neuen Datenspeicher.

Architektur

Die Konzepte der Architektur von Spark sind Cluster, Spark Applikationen und APIs mit „DataFrames“ und SQL [31].

Cluster

Eine Gruppe von Computern bzw. eine Zusammenschluss von Ressourcen vieler Rechner/Maschinen nennt man Cluster. Mit den Clustern ist es möglich alle hinzugefügten Ressourcen zu nutzen, als ob ein Einzelrechner wäre. Die Koordination für die Ausführung von Aufgaben zwischen den Clustern und Verwaltung der Cluster übernimmt Spark. Für die Ausführung von Aufgaben verwendet Spark einen Cluster Manager. Dieser teilt und genehmigt Ressourcen für eingereichte Spark Applikationen zu. [11] [31]

Spark Applikationen

Spark Applikationen bestehen aus einem Treiberprozess und ein oder mehreren „Executor“ . Der Treiberprozess führt die „main()“ Funktion aus und ist Teil eines Knoten im Cluster. Das bedeutet, dass Code vom Anwender in einer „SparkSession“ ausgeführt wird. Mithilfe einer „SparkSession“ können anwenderspezifische Manipulationen zwischen den Cluster ausgeführt werden. [31]

Nach [31] ist der Treiberprozess für drei Punkte verantwortlich:

- Informationen von der Spark-Applikation verwalten.
- Auf Eingaben eines Benutzers und Programmänderungen reagieren.
- Aufgaben der „Executor“ analysieren, zu verteilen und zu planen.

Ein „Executor“ ist für die Ausführung, der ihm vom Treiberprozess zugeteilten Aufgabe zuständig. Außerdem gibt der „Executor“ dem Treiberprozess eine Rückmeldung zum Stand der Berechnung. Die parallele Bearbeitung der Aufgaben geschieht durch die Aufteilung der Daten in Partitionen. [31]

Verarbeitung Spark unterstützt die Verarbeitung der Daten in zwei Modi: „Batch“ und „Streaming“ . Hierbei unterscheidet man, ob die Verarbeitung kontinuierlich oder ob die großen Datenmengen in kleinere Stücke (Batches) aufgeteilt werden soll. Die Aufteilung in mehrere Batches minimiert Anomalien und Fehler. [31]

5.3.3 Apache Cassandra

Als Datenbank der Open-Source Datenpipeline wird Apache Cassandra verwendet. Apache Cassandra ist eine verteilte, dezentralisierte, skalierbare, hochverfügbare, fehlertolerante, nicht-relationale Datenbank [24]. Es bietet eine bessere Skalierbarkeit und Fehlertoleranz gegenüber traditionellen Single-Master-Datenbanken [24]. Die Struktur mit Tabellen, Zeilen und Spalten wie in relationalen Datenbanken existiert ebenfalls bei Cassandra. Das bietet strukturierte Zugriffe der Datensätze und Modellierungen an. Mithilfe „Cassandra Query Language (CQL)“ können Datenbankschemata erstellt, aktualisiert und Daten zugegriffen und Cassandra-Knoten innerhalb eines Clusters organisiert werden [14].

Verteilt und Dezentral

Cassandra ist ein verteiltes System. Das bedeutet, dass es auf mehreren Rechnern laufen kann, während es für den Benutzer wie ein einheitliches System erscheint Es gibt keinen „Single Point of Failure“ , da kein „Master-Slave“-Setup bei Cassandra vorhanden ist. Cassandra ist dezentral. Jeder Cassandra-Knoten ist identisch und unterscheiden sich nicht bei der Ausführung von organisatorischen Operationen. Der dezentrale Ansatz ist ebenfalls ein Schlüsselprinzip für die Hohe Verfügbarkeit. Für die Synchronisation und Abfrage der Knoten, ob sie tot oder lebendig sind, hat Cassandra ein Peer-to-Peer Protokoll, welches das Gossip-Prinzip verwendet. [24] [29]

Horizontale Skalierbarkeit

Apache Cassandra ist horizontal skalierbar. Horizontale Skalierbarkeit bedeutet, dass es eine Möglichkeit gibt weitere Speicher- und Verarbeitungskapazitäten zu erweitern. Das

Hinzufügen von weiteren Server zu einem Cluster ist eine Möglichkeit. Bei traditionellen „Single-Master“ Datenbanken ist die Kapizität an der Serverkapazität gebunden. Auf der anderen Seite hat Cassandra eine uendliche Kapazität für das Speichern und Verarbeiten von Daten. Grund hierfür ist, dass bei weiterem Kapazitätsbedarf mehr Maschinen/Computer zum Cluster hinzugefügt werden können. Die existierenden Daten werden durch die neu hinzugefügten Daten für die Balance im Cluster neu organisiert. [24] [29]

Der Aufbau von Cassandra besteht aus einem Cluster von Instanzen, die sich alle gegenseitig kennen. Daten können von der Client-Anwendung in jeder Instanz des Clusters geschrieben oder gelesen werden. Eine Instanz des Cluster bezeichnet man als Knoten. Je nach Zugehörigkeit der Daten leiten Knoten Daten an die jeweilige Instanz des Clusters weiter. [24] [29]

Hohe Verfügbarkeit

Die hohe Verfügbarkeit zeichnet sich durch das dezentralisierte Design von Cassandra aus. Bei einer Schreib- oder Leseoperation aus der Datenbank gibt es keinen „Single Point of Failure“ . Grund hiefür, ist dass Daten auf mehreren Knoten repliziert werden. Es gibt im Vergleich zu relationalen Datenbanken, wie zum Beispiel NoSQL keinen „Master-Slave“ oder „Master Copy“ . Sobald eine Maschine oder Knoten ausfällt, werden die Daten weiterhin auf andere Knoten, die Daten mit der ausgefallenen Maschine geteilt haben, geschrieben. Die Operationen werden in eine Warteschlange reingestellt. Der ausgefallene Knoten wird aktualisiert, sobald dieser am Cluster teilnimmt. [24] [29]

Konsistenz

Im Vergleich zu anderen verteilten Systemen oder „Single-Master“ Datenbanken wie MySQL und PostgreSQL, die eine „strict“ oder „immediate“ Konsistenz haben, hat Cassandra eine „eventual“ Konsistenz. Das heißt, dass die Aktualisierungen der Daten in allen Replikaten einige Zeit andauern, bis diese im verteilten System an alle Replikate weitergegeben werden. [24] [29]

Veränderbarkeit von Apache Cassandra

Im Rahmen der Durchsatz- und Latenzmessungen gibt es veränderbare Parameter für Apache Cassandra. Folgende Parameter werden anhand der Dokumentation [14] und [29] erklärt:

- *replicationfactor*
  - Der Replikationsfaktor gibt die Gesamtzahl der Replikate in einem Cassandra-Cluster an. Je höher der Replikationsfaktor, desto mehr Kopien gibt es in anderen Knoten. Alle Replikate sind gleich. Dadurch erreicht man eine höhere Datenverfügbarkeit und hohe Fehlertoleranz.
- *consistency level*
  - Der Konsistenzlevel gibt an, wie viele Knoten antworten müssen, um um auf einen erfolgreichen Lese- oder Schreibvorgang abschließen zu können.
  - Schreiben: Je höher der Konsistenz Level, desto mehr Replikate müssen antworten, sodass erst danach der Schreibbefehl abgeschlossen ist.
  - Lesen: Eine höhere Konsistenzstufe bedeutet, dass mehr Knoten auf die Abfrage antworten müssen, was mehr Sicherheit bedeutet. Es wird sichergestellt, dass die Werte auf jeder Replikat dieselben.

5.3.4 Grafana

Für die Analyse und Visualisierung der Daten wird Grafana verwendet. Grafana ist eine Open-Source-Analyseplattform für die Überwachung und Analyse von Daten. Die vielfältigen, wählbaren Datenquellen für die Visualisierung gehören zu den Stärken von Grafana. Ein weiteres Merkmal von Grafana, dass Alarme auf Basis der erfassten Daten und Ereignisse gesetzt werden können. Aufgrund der Visualisierungsmöglichkeiten und Alarme, die Statusänderungen, Überschreitungen von Werten usw. anzeigen können, ist Grafana für Überwachung geeignet. Ein grundlegender Visualisierungsbaustein von Grafana ist das sogenannte „Panel“ . Es beinhaltet einen Abfrage-Editor. Der Abfrage-Editor ist für den Panel gewählte Datenquelle zuständig. Die Panels können auf dem Dashboard per Drag und Drop verschoben und neu angeordnet werden. Die Größe ist ebenfalls veränderbar. Mithilfe Abfragen kommunizieren Grafana-Panels mit Datenquellen, um Daten visualisieren zu können. Eine Abfrage ist eine Frage, die in einer Abfragesprache geschrieben wird. Die Abfragesprache wird von der Datenquelle verwendet. In den Datenquellenoptionen kann eingestellt werden, wie viele Datenpunkte gesammelt werden. Außerdem kann die Anzahl der Abfragen, die an die Datenquelle gesendet wird, eingestellt werden. [9] [62]

6 Verwandte Arbeiten

Im folgenden Kapitel werden verwandte und aktuelle Arbeiten vorgestellt, um den Vergleich der Datenpipelines in der Forschung und Diskussion einordnen zu können. Da nicht alle wissenschaftliche Artikel, Ausarbeitungen betrachtet werden können, werden bestimmte, ausgewählte Arbeiten gesichtet. Als Erstes wird bei Abschnitt 6.1 die Grundlage und Stand der Forschung für den Aufbau und Implementierung der Datenpipeline erläutert. Abschließend wird in Abschnitt 6.2 verschiedene Arbeiten, die für die Diskussion wichtig sind, vorgestellt.

6.1 Grundlage

Als Grundlage für den Aufbau und Implementierung der Datenpipelines werden folgende Ausarbeitungen,wissenschaftliche Artikel etc. betrachtet. Außerdem ordnen diese Arbeiten in die Domäne und Kontext ein. Folgende Ausarbeitungen sind wichtig für diese Arbeit:

Der Artikel *Industry 4.0 reference architectures: State of the art and future trends* verschafft einen Einblick über den aktuellen Stand und Trends von Referenzarchitekturen für die Industrie 4.0. Es zeigt die Unterschiede und Anwendungsmöglichkeiten der Referenzarchitekturen. Es verdeutlicht ebenfalls den Wandel von alten, hierarchischen Strukturen, wie die Automatisierungspyramide zu modularen, heterogenen Systemen. [53]

Eine weitere Übersicht und Hilfestellung für den Aufbau einer Datenpipeline in einer Produktionsumgebung, ist die Ausarbeitung *Practical Guide to Smart Factory Transition Using IoT, Big Data and Edge Analytics.* Technologien, Fachbegriffe der Geschäftslogik lassen sich in verschiedenen Ebenen (Produktion, Geschäfte, IoT Plattform, Visualisierung) einteilen und einordnen. Diese Grundlage ist wichtig für den Aufbau und Strukturierung der Arbeit. [41]

Die Möglichkeiten eine SPS in Industrie 4.0 Szenarien einzusetzen, zeigt das Paper *Concepts for Retrofitting Industrial Programmable Logic Controllers for Industrie 4.0 Scenarios*. Hierbei wird ebenfalls der Einsatz einer SPS in Altsystemen als Nachrüstung für Industrie 4.0 Szenarien beachtet. Es zeigt bekannte Kommunikationsprotokolle und die Unterstützung dieser Protokolle mit den jeweiligen verschiedenen SPSen von verschiedenen Herstellern. Fünf Anwendungsfälle werden betrachtet, aus denen sich verschiedene Nachrüstungskonzepte für Altsysteme ableiten. Anhand den Erkenntnissen aus dieser Arbeit kann ein Kommunikationsprotokoll für die Datenextrahierung für beide Datenpipelines gewählt werden. Aus diesem Grund ist diese Arbeit relevant. [60]

Es gibt viele Cloud-Anbieter, um eine Datenpipeline in einer IoT-Architektur zu realisieren. Es ist ebenfalls eine Herausforderung diese in „Cloud Computing“ mit zu integrieren. Eine Performance Übersicht bietet die Ausarbeitung *Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison*. In der Ausarbeitung werden die drei bekanntesten Cloud-Plattformen (Google Cloud Platform, Amazon Web Services und Mircosoft Azure) hinsichtlich der verfügbaren Dienste für IoT verglichen. Ziel der Studie ist es, die Leistung des Zugangspunkt in allen Plattformen zu vergleichen und eine Hilfestellung für die Auswahl einer Plattform zu geben. In allen Plattformen ist das MQTT Protokoll bzw. ein MQTT Broker als Zugangspunkt verfügbar. In der Ausarbeitung wird die Ende-zu-Ende Zeit vom Veröffentlichen bis zur Annahme der Nachricht durch einen Subscriber oder Applikation gemessen. Hinzu kommt, dass die Anzahl der Publisher-Clients, Subscriber-Clients, Anzahl der ausgetauschten Nachrichten, Größe der Nachrichten, Durchsatz der Nachrichten verändert werden. Die Messungen sind jedoch allgemein und nicht auf eine Anwendung spezifiziert. Außerdem hat die höhere Last die Leistung der Plattformen nicht beeinträchtigt. Diese Ausarbeitung ist wichtig für den Aufbau des Closed Ansatzes. [58]

6.2 Stand der Diskussion

In diesem Abschnitt werden andere Arbeiten vorgestellt, die ebenfalls eine ähnlichen Open-Source, Closed oder hybriden Ansatz für eine Datenpipeline erarbeitet haben. Arbeiten mit dem gleichen Aufbau, Anforderungen sowie Bedingungen sind nicht zu finden. Stattdessen werden ähnliche IoT Architekturen, Datenpipelines vorgestellt, die für die Diskussion in Betracht gezogen werden können. Diese haben eine entscheidene Rolle für den Aufbau des Open-Source Ansatzes.

Ein ähnlicher Open Source Ansatz für eine Datenpipeline schlägt die Ausarbeitung *Scalable data pipeline architecture to support the industrial internet of things* vor. Es wird eine Architektur für eine skalierbare Pipeline zur Verarbeitung und Verteilung von Daten aus mehreren Datenquellen und Ebenen (ERP, MES etc.) vorgestellt. Für die Datenextrahierung wird MQTT und eine REST API vorgeschlagen. Als Middleware wird Apache Kafka und ein MQTT Broker verwendet. Durch Kafka Connect werden Datenkonsumenten und -produzenten mit dem Kafka Broker verbunden. Die SPS als Datenproduzent schickt Daten an den MQTT Broker. Die Daten werden in der Datenbank MongoDB gespeichert oder an weitere Applikationen weitergeleitet. In dieser Ausarbeitung wird nur die Implementation der Architektur betrachtet. [38]

Das Paper *Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework* zeigt einen Open Source Datenpipeline Aufbau für IoT Ereignisse mit historischen und aktuellen Daten. Ein Aufbau mit Apache Kafka, Apache Spark und Dashing-Dashboard wird vorgestellt. Es wird nur die Implementation beschrieben und auf Funktionalität geprüft. [32]

In vielen IoT Architekturen und Datenpipelines wird Apache Kafka verwendet. In der Ausarbeitung *Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Enviroment* wird nach einer „Big Data Plattform“ gesucht, die im Industrie-Umfeld eingesetzt werden kann. In der Fragestellung wird der Einfluss von mehreren SPSen auf die Plattform untersucht. Die Systemarchitektur besteht aus drei Schichten: Datenaufnahme, Datenverarbeitung und -speicherung. Hierfür wird Kafka, Spark und Hadoop mit HBase verwendet. Außerdem gibt es ein Public-Key-Verfahren für die Verschlüsselung der Daten. Für die Messung werden Kafka Konfigurationen verändert und Multithreading für die Partitionen angewendet. Die Messungen haben die Veränderbarkeit der Parameter für die „Big Data“ Technologien aufgezeigt. Die Erkenntnis, dass Apache Kafka und Apache Spark die Leistung und Genauigkeit der Datenspeicherung, -verarbeitung und -sicherung in der Fertigungsumgebung verbessert, ist wichtig. Diese Arbeit ist besonders relevant. [47]

In *Scalable Analytics Platform for Machine Learning in Smart Production Systems* wird eine „Big Data“ und Machine Learning Referenzarchitektur für die industrielle Automatisierung vorgeschlagen, die dem Referenzarchitekturmodell für Industrie 4.0 entspricht. Für die Evuluation wird die konzeptionelle Architektur in der Demonstrationsplattform SmartFactoryOWL implementiert. Dabei werden Performance und Skalierbarkeit anhand von industriellen Daten untersucht. Eine SPS schickt mit OPC UA Nachrichten an Kafka.

MongoDB wird als Datenbank für Meta- und Prozessdaten und InfluxDB für die Verwaltung von Zeitreihen verwendet. Die Skalierbarkeit wird anhand einer Durchsatzmessung gemessen. Die Ergebnisse besagen, dass die Architektur mit den eingesetzten „Big Data“ Technologien linear skalierbar und an „Machine Learning“ Use-Cases adaptierbar ist. Die Betrachtung der Skalierbarkeit durch die Messung ist relevant für die Arbeit. [21]

Eine hybride Lösung aus Open-Source und Closed bietet *Development An Open-Source Industrial IoT Gateway*. Der Fokus liegt auf die Datenextrahierung und Weiterleitung der Nachrichten an Cloud-Plattformen. Es wird ein Aufbau und die Implementierung eines industriellem IoT Gateway mithilfe Docker, Node-RED und weiterer Linux-basierten Software vorgestellt. Die Lösung der Arbeit verwendet eine Peer-to-Peer Kommunikation, um über andere TCP basierende industrielle Protokolle, wie z.B.: Modbus, S7, MQTT Pakete ein- und entkapseln zu können. Der Prototyp wird anhand der Latenz untersucht und der „Proof of Concept“ evaluiert. Die SPS sendet Daten an eine lokale SQL Datenbank. Über HTTP werden die Daten an einen RESTFUL Server von Amazon EC2 geschickt. Neben der HTTP Lösung werden Daten von der lokalen Datenbank mit MQTT an die Cloud-Plattform von IBM gesendet. Die vorgestellte Lösung setzt eine lokale Docker-Instanz voraus, die die Daten über das S7-Protokoll Daten aus der SPS extrahiert. In den Latenzmessungen werden geringe Latenzen erreicht, jedoch handelt es sich um eine Messung vom Gateway zur Cloud-Plattform. Die weitere Bearbeitungsdauer für Datenoperationen und -manipulationen in den Cloud-Plattformen wird nicht beachtet. [54]

Das Paper *On the Performance of Cloud Services and Databases for Industrial IoT Scalable Applications* betrachtet die Auswirkung von „Database as a Service(DBaaS)“ hinsichtlich der Latenz in einem skalierbaren IoT Kontext bzw. Architektur. Als Use-Case wird die prädiktive Überwachung gewählt. Die Cloud-Plattform und Dienste von IBM werden genutzt. Es wird eine Bewertungsstrategie vorgeschlagen, um die Latenz von der Datenquelle bis zum Zielpunkt in einer Cloud-Plattform bzw. in einem IoT Kontext so einfach wie möglich messen zu können. In den Messungen wird gleichzeitig der Einfluss von Cloud-Datenbankdiensten auf die Anwendung untersucht. Die Wege der Daten werden mithilfe Latenzmessungen miteinander verglichen und evaluiert. Die Erkenntnisse sind wichtig und relevant für die Arbeit, da es Möglichkeiten aufzeigt einfache Latenzmessungen in einer Cloud-Plattform durchzuführen. Es werden die einzelnen Stationen aufgezeigt, wo Zeitstempel entommen werden. Hinzu kommt, dass die Auswirkung der beteiligten Geräte hinsichtlich der Skalierbarkeit betrachtet wird. [34]

Zusammenfassend kann man sagen, dass viele Arbeiten davon handeln bestehende Altsysteme in IoT-Architekturen zu überführen bzw. einzugliedern. Des Weiteren werden neue Open-Source, Closed oder hybride Prototypen für den industriellen Einsatz vorgeschlagen. Der Aufbau und die Implementierung der Prototypen unterscheiden sich durch die Use-Cases, Anforderungen und Randbedingungen. Oftmals werden Geräte, Industrieanlagen und Produktionstätte simuliert und es fehlt der eigentliche Einsatz und Messung in einer realen Umgebung. Messungen hinsichtlich der Latenz, Anzahl an teilnehmenden Komponenten und Nachrichten sowie die Designziele der Architektur spielen für den Einsatz eine wichtige Rolle.

7 Experimente

Im folgenden Kapaitel wird in Abschnitt 7.1 der Aufbau der Datenpipelines für die Experimente erläutert. Anschließend wird in Abschnitt 7.2 die Durchführung der Experimente für die Metriken Latenz und Durchsatz beschrieben.

7.1 Aufbau

In beiden Aufbauten wird die Vernetzung und Abhängigkeiten unter den Komponenten mithilfe Komponentendiagrammen dargestellt. Des Weiteren werden dadurch einzelne Komponenten für das Experiment und ihre Funktion vorgestellt. Die Verteilungssicht im AWS Aufbau zeigt die Kommunikation der Dienste. Die Verteilungssicht im Open-Source Aufbau zeigt die Virtualisierung und Kommunikation zwischen den Docker-Containern. Für die Übersicht und Lesbarkeit sind die Verteilungssichten in mehreren Sichten aufgeteilt.

Beide Aufbauten verwenden eine S7-1200 mit einer Kompakt-CPU 1214C DC/DC/Relais mit der Spezifizierung 6ES7214-1HG40-0XB0. Der Programm/-Datenspeicher ist 100Kb groß.

Der AWS Aufbau verwendet die Dienste in der AWS Region EU-Central-1-Frankfurt. Der Open-Source Aufbau verwendet einen gemieteten Server mit 12 Cores, 24 GB RAM, 960 GB SSD und 80TB Traffic.

7.1.1 AWS Datenpipeline

Die folgende Abbildung 7.1 zeigt als Erstes die Ebene 0. Für die Durchführung der Experimente im Closed Ansatz wird zwischen der SPS Komponente auf der linken Seite und dem AWS Datenpipeline auf der rechten Seite unterschieden. Im AWS Aufbau ist der

Datenaustausch für die Experimente bidirektional. Der Datenaustausch findet mithilfe MQTT statt.

In der SPS ist ein Programm für die Messung geladen. Es nutzt eine Bibliothek für MQTT. Aus der Bibliothek lassen sich MQTT-Bausteine beliebig instanziieren. Eine MQTT-Instanz veröffentlicht unter einem Topic Daten an den MQTT Broker der AWS Datenpipeline. Eine zweite MQTT-Instanz abonniert und empfängt Nachrichten, die vom MQTT Broker gesendet werden.

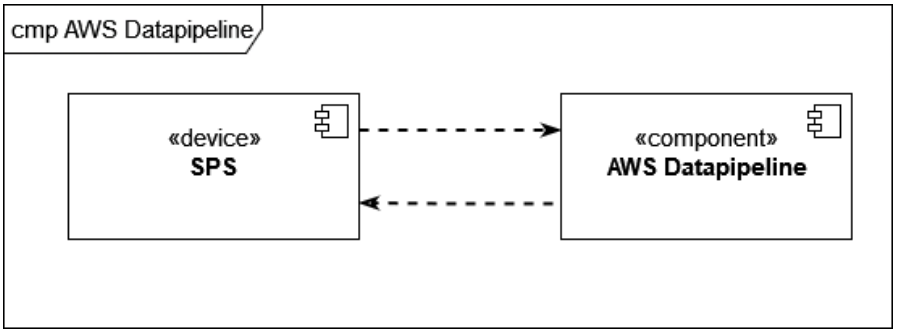

Abbildung 7.1: Ebene 0: Aufbau als Komponentendiagramm

Die folgende Abbildung 7.2 zeigt eine tiefere Ebene der AWS Datenpipeline Komponente:

Es sind die Dienste AWS CloudWatch, AWS IoT Core, AWS Kinesis Data Firehose, AWS Timestream, AWS Glue, Amazon S3, AWS Athena, AWS Grafana zu sehen. Die Dienste werden als „Software-as-a-Service (SaaS)“ genutzt.

AWS IoT Core: Der MQTT-Client (SPS) veröffentlicht Nachrichten und schickt diese an den AWS IoT Core. Die Datenformattierung der Nachrichten ist JSON. Es gibt für die Weiterleitung von Nachrichten drei Regeln. Für die Weiterleitung der Nachrichten wird eine Datenfilterung vorgenommen. Eine Regel leitet die Nachrichten bzw. Daten über den MQTT-Broker zurück an die SPS. Diese Regel wird für die „Round-Trip-Time“ Messung der Latenz-Metrik verwendet. Eine zweite Regel sendet die Daten weiter an die Zeitreihendatenbank AWS Timestream. Eine dritte Regel leitet die Daten in einen Bereitstellungsstream von AWS Kinesis Data Firehose.

AWS Timestream: Die weitergeleiteten Daten kommen in der Zeitreihendatenbank AWS Timestream an. Alle Daten, die die Datenbank erreicht, werden mit einem Zeitstempel versehen. AWS Timestream wird als Datenquelle für Grafana festgelegt. Mit einer Abfrage können die Daten in Grafana angezeigt werden.

AWS Kinesis Data Firehose: Die Komponente hat die Aufgabe die Daten als Stream an andere Komponente zuzustellen. Die weitergeleiteten Nachrichten werden von AWS Glue nach dem Schema geprüft und mit der AWS Glue Tabellendefinition abgeglichen. Die Nachrichten werden als PUT-Befehl an die Datenbank Amazon S3 gesendet.

AWS Glue: In AWS Glue gibt es einen Datenkatalog. Dieser besteht aus einer Datenbank. Die Datenbank hat eine Tabelle. Die Tabelle hat ein bestimmtes Schema. Das Schema muss mit den Daten in den gespeicherten S3-Objekte gleich sein. Mit Hilfe eines sogenannten Crawlers von AWS Glue, werden die Objekte eines Buckets in Amazon S3 durchsucht. Hierbei wird das Schema der Tabelle mit dem Schema der Daten in den S3 synchronisiert und festgelegt. Zusätzlich werden die Daten aus dem Bereitstellungsstream von AWS Kinesis Data Firehose validiert bzw. anhand des Datenformats geprüft. Die Daten der Objekte sowie die Daten des Bereitstellungsstreams sind dieselben, da Amazon S3 als Datenquelle für die AWS Glue Datenbank festgelegt ist.

Amazon S3: In einem Bucket werden die Daten aus dem Bereitstellungsstream als Objekte gespeichert. Die Daten sind ebenfalls in der AWS Glue Datenbank bzw. in einer Tabelle zu finden.

AWS Athena: Die Datenquelle für die Abfragen ist die Datenbank in AWS Glue. Dadurch sind die Daten aus Amazon S3 mit einem GET-Befehl zugreifbar. AWS Athena wird als Datenquelle für Grafana festgelegt. Aus diesem Grund können Daten in Grafana mit einer Abfrage angezeigt werden.

AWS Cloudwatch: Dieser Dienst sammelt von allen Diensten Metriken für eine Auswertung. Über ein Dashboard können ausgewählte Metriken von den jeweiligen Diensten gebündelt angezeigt werden.

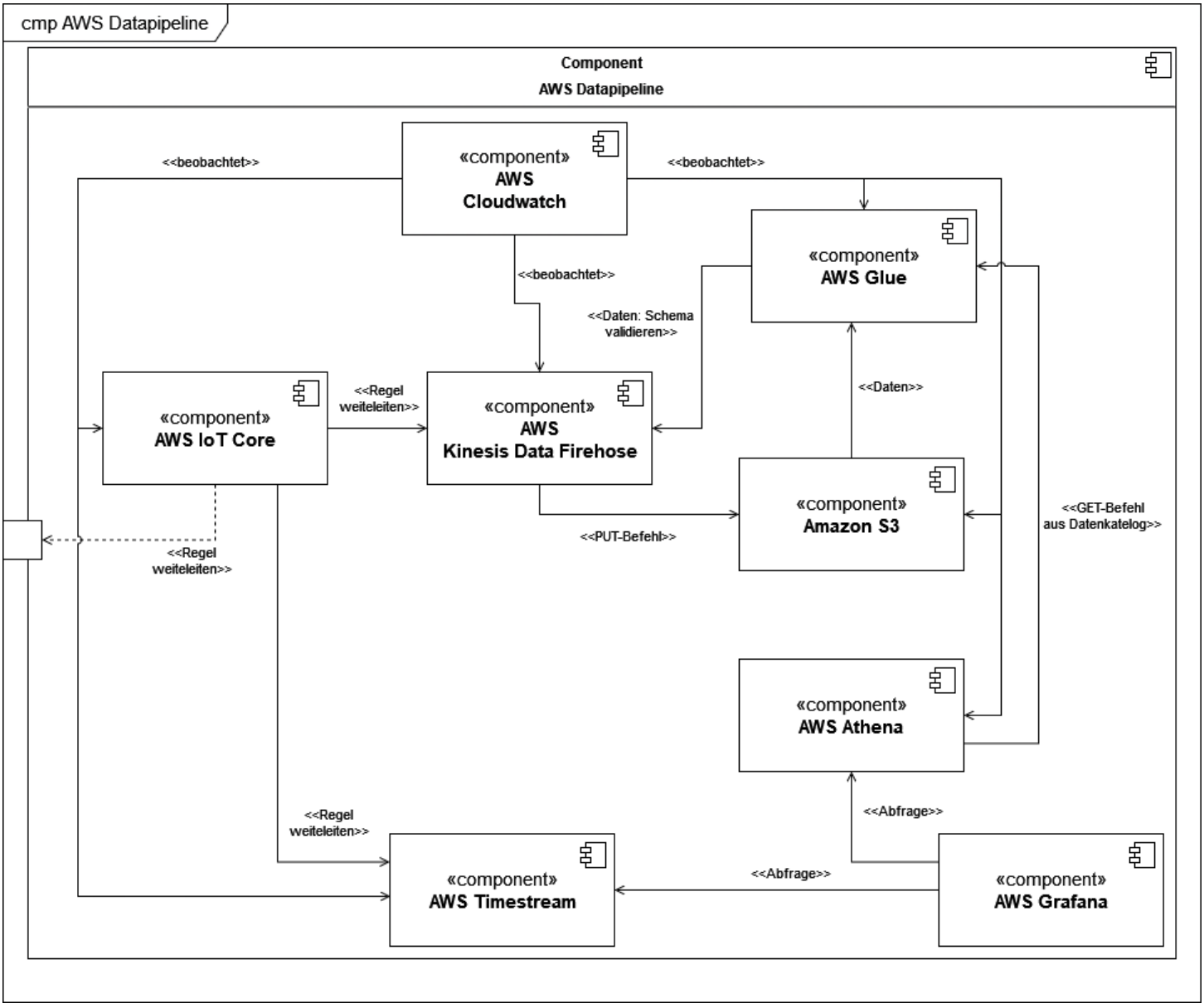

Abbildung 7.2: Ebene 1: AWS Datenpipeline Komponentendiagramm

Verteilungssicht

Die Verteilungsschicht ist in drei Teile: Datenextrahierung, Datenverarbeitung und Datenvisualisierung aufgeteilt.

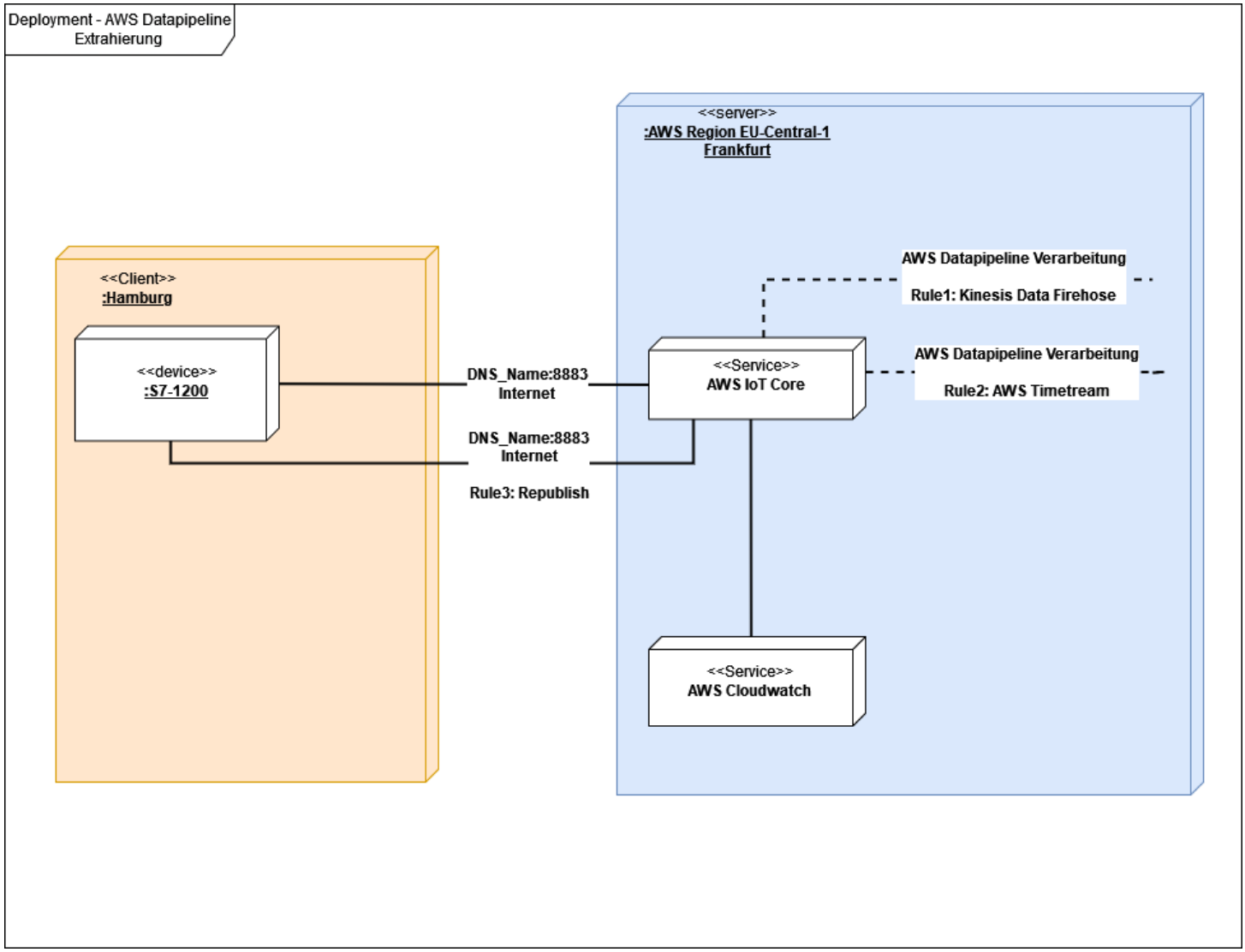

Abbildung 7.3: Verteilungssicht: Datenextrahierung

Die Abbildung 7.3 zeigt die Datenextrahierung. Auf der linken Seite ist zu sehen, dass die SPS als Client in Hamburg sich befindet. Auf der rechten Seite ist der AWS Server zu sehen, welcher sich in der AWS Region EU-Central-1-Frankfurt befindet. Der AWS IoT Core extrahiert mithilfe MQTT Daten aus der SPS. Der Datenaustausch zwischen

SPS und AWS IoT Core ist durch Zertifikate verschlüsselt (Port: 8883). Die Möglichkeit für eine Verbindung wird durch die Angabe eines DNS Namen bzw. Geräte-Endpunktes festgelegt. Die MQTT-Bausteine zum Senden und Empfangen von Nachrichten enthalten als Broker-Adresse den Geräte-Endpunkt. Der Geräte-Endpunkt wird von AWS IoT Core ausgegeben. Die festgelegten Regeln zeigen die Weitergabe der Nachrichten an die Verteilungssicht: Datenverarbeitung sowie die Veröffentlichung der Nachrichten an den SPS Client. AWS Cloudwatch sammelt Nutzungsdaten von AWS IoT und zeigt sie in einem Metrik-Dashboard an.

Die folgende Abbildung 7.4 zeigt die Verteilungssicht: Datenverarbeitung. Alle Dienste der Datenverarbeitung sind in der selben AWS Region EU-Central-1-Frankfurt. Die Verbindungen bzw. die Service-Aufrufe sind die selben wie in Abbildung 7.2. Es gibt durch die Regeln Verbindungen zur Datenextrahierung. Es gibt ebenfalls Verbindungen zur Datenvisualisierung. AWS Cloudwatch sammelt ebenfalls alle Nutzungsdaten der Dienste und zeigt sie in einem Metrik-Dashboard an.

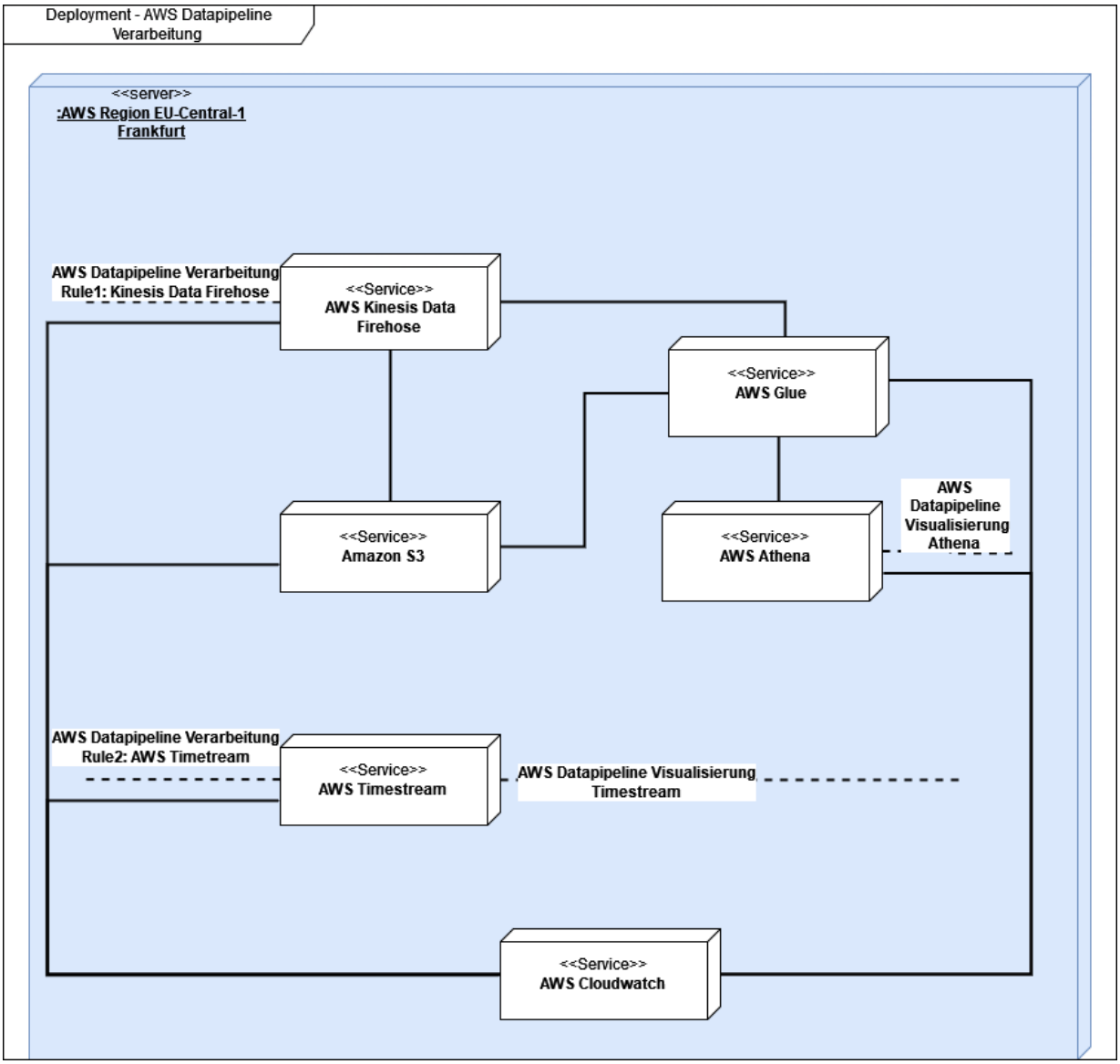

Abbildung 7.4: Verteilungssicht: Datenverarbeitung

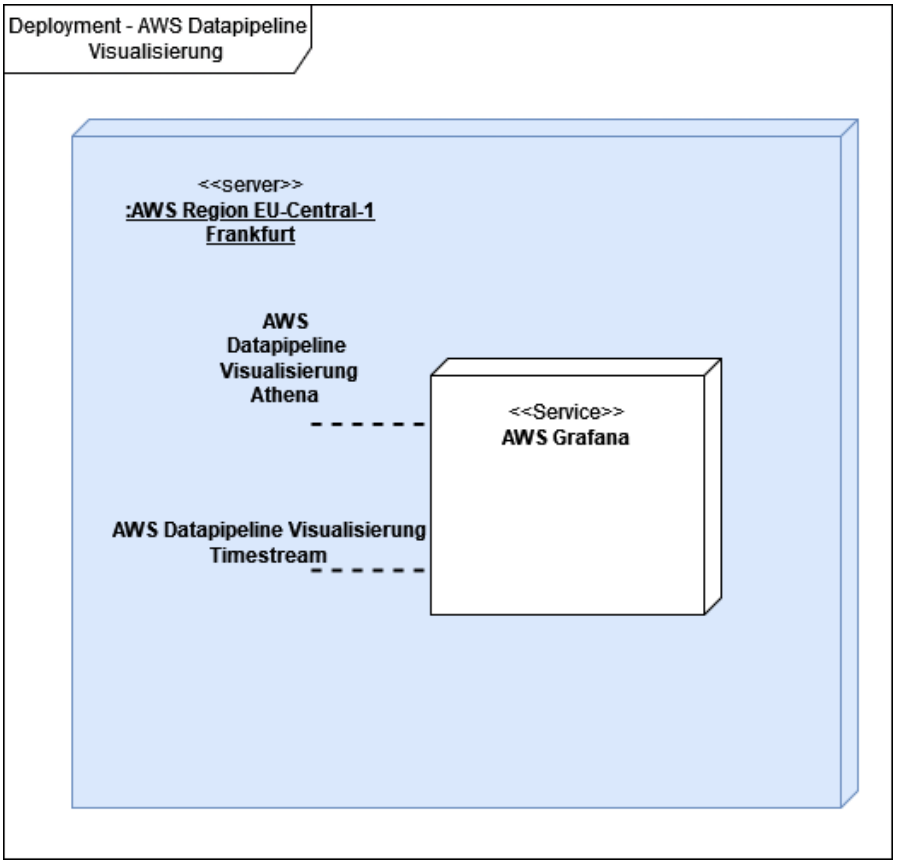

Abbildung 7.5: Verteilungssicht: Datenvisualisierung

Die Abbildung 7.5 zeigt die Verteilungssicht: Datenvisualisierung. AWS Grafana befindet sich ebenfalls in der selben AWS Region EU-Central-1-Frankfurt und ist mit den Diensten aus der Datenverarbeitung verbunden.

7.1.2 Open-Source Datenpipeline

Der Aufbau der Open-Source Datenpipeline wird ebenfalls anhand des Komponentendiagramms und Verteilungssichten erläutert. Die Komponentendiagramme zeigen die Abhängigkeiten und Schnittstellen sowie die Verbindungen zwischen den Komponenten. Die Verteilungssichten zeigen die Kommunikation zwischen den Containerns sowie die Umgebung, in denen die Artefakte, Container laufen.

Komponentendiagramm

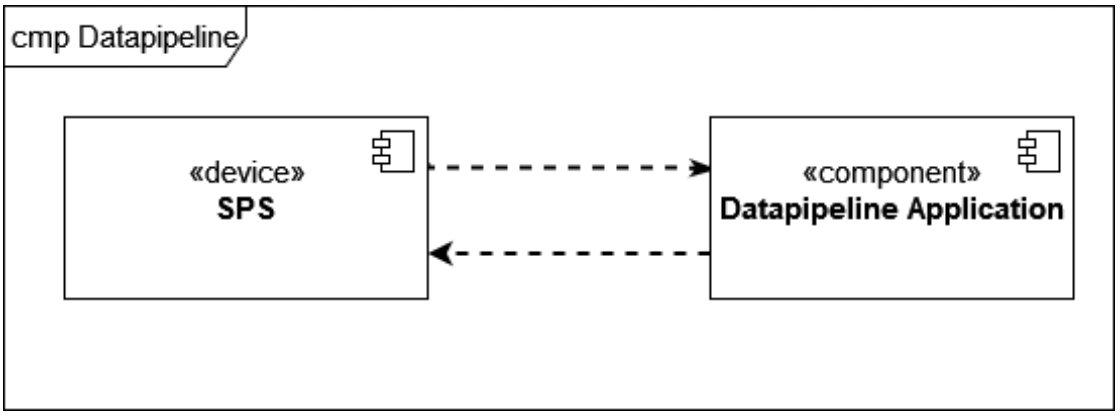

Abbildung 7.6: Ebene 0: Aufbau als Komponentendiagramm

Die Abbildung 7.6 zeigt die Ebene 0 des Aufbaues. Auf der linken Seite ist die SPS. Auf der rechten Seite ist die Datenpipeline Applikation zu sehen. Es gibt ebenfalls einen bidirektionalen Datenaustausch für die Durchführung der Experimente.

In der SPS läuft dasselbe Programm wie im AWS Aufbau. Das bedeutet, dass zwei MQTT-Instanzen aus der Bibliothek genutzt werden, um Nachrichten Senden und Empfangen zu können.

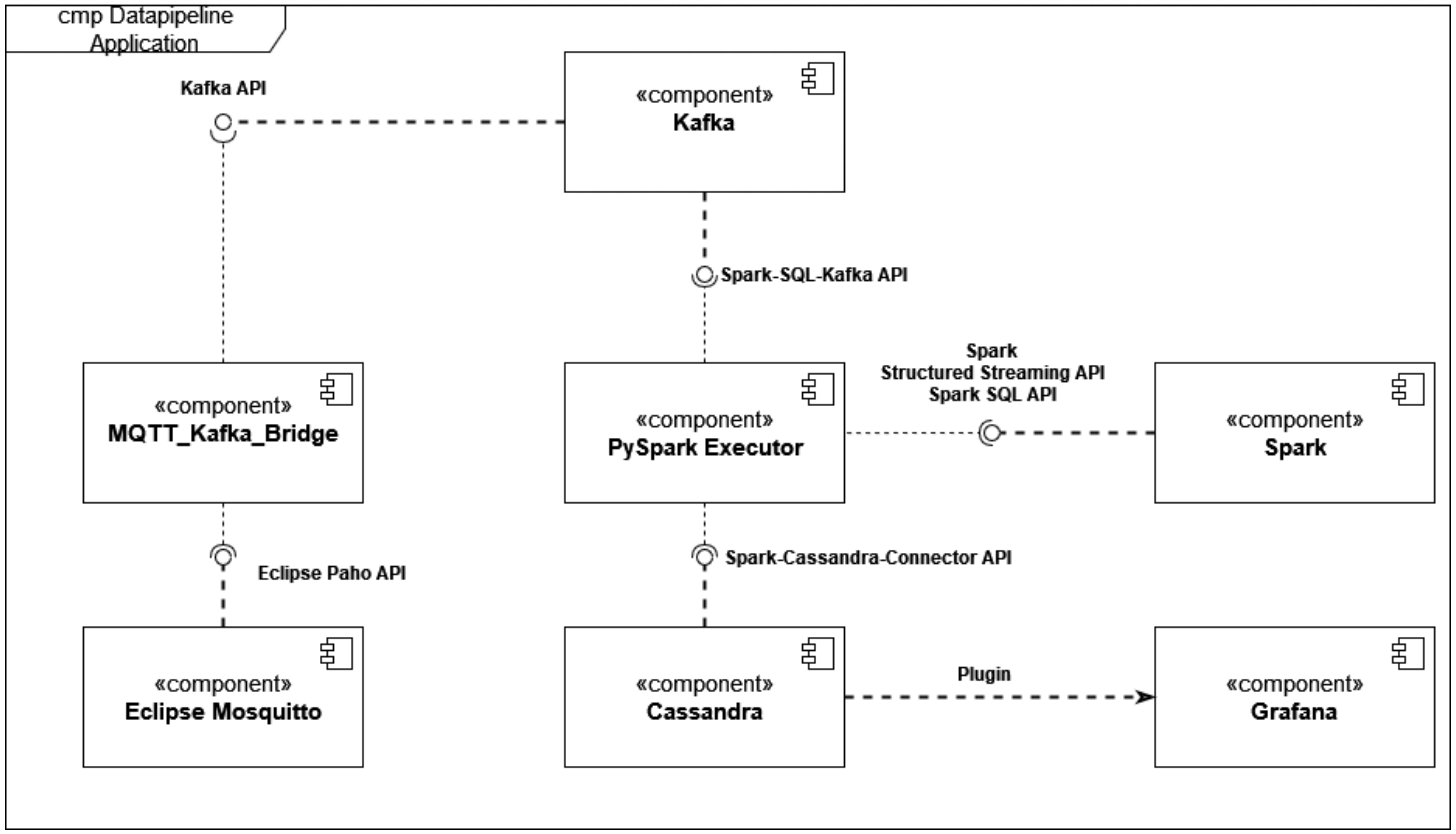

Abbildung 7.7: Ebene 1: Datenpipeline Applikation

Die Abbildung 7.7 zeigt eine tiefere Ebene der Open-Source Datenpipeline Applikation. Ebene 1 zeigt die Nutzung der APIs sowie die Abhängigkeiten zwischen den Komponenten. Es sind die Komponenten MQTT_Kafka_Bridge, Eclipse Mosquitto, Kafka, Spark, PySpark Executor, Cassandra, Grafana zu sehen.

MQTT_Kafka_Bridge: Die Komponente ist für die Kommunikation zwischen der SPS und Kafka zuständig. Es ist eine Brücke zwischen MQTT und Kafka. Es verwendet die Eclipse Paho API. Mithilfe der Schnittstelle wird ein MQTT-Client erstellt und konfiguriert. Methoden für die Veröffentlichung und das Abonnieren von MQTT-Nachrichten werden ebenfalls verwendet. Für das Experiment werden Nachrichten von der SPS über die Brücke an Kafka weitergeleitet. Des Weiteren nimmt es auch Nachrichten von Kafka entgegen und schickt diese zurück an die SPS. Weitere Details sind in Abbildung 7.8 zu sehen.

Die Komponente nutzt auch die Kafka API. Mithilfe der Schnittstelle werden Verbindungen zu Kafka hergestellt, Kafka Konsumenten und Kafka Produzenten erstellt. Über die API werden die Konsumenten und Produzenten für die Messung der Latenz verändert.

Eclipse Mosquitto: Die Komponente stellt einen MQTT Broker für den Datenaustausch zwischen SPS und Open-Source Datenpipeline zur Verfügung. Dadurch ermöglicht diese Komponente, dass eine SPS als Datenquelle für die Experimente genutzt werden kann.

Kafka: Die Komponente stellt eine Verbindung zwischen den Komponenten MQTT_-Kafka_Bridge, Kafka und PySpark Executor her. Einerseits wird über Kafka Nachrichten an Spark weitergeleitet. Anderseits verschickt der PySpark Executor durch die Nutzung der Spark-SQL-Kafka API Daten zurück an die MQTT_Kafka_Bridge. Dadurch ist eine Messung der „Round-Trip-Time“ möglich.

PySpark Executor: Die Komponente nutzt die Spark-SQL-Kafka API, Spark-Cassandra-Connector-API und Spark Structured Streaming API von anderen Komponenten. Für die Experimente erstellt es eine Spark-Session mit Jobs für Spark. Der Job enthält Datenfilterungsaufgaben. Durch die Nutzung von Spark Structured Streaming API gibt es einen kontinuierlich Datenstrom, der pro Batch verarbeitet wird. Für die Latenz-Messung empfängt es Daten von Kafka und versendet diese über Kafka und die MQTT_Kafka_Bridge an die SPS zurück. Es schickt auch Daten an Cassandra Datenbank.

Spark: Die Komponente ist mit dem PySpark Executor verbunden. Diese erhält Aufgaben vom PySpark Executor. Der Spark-Master teilt die Aufgaben den Spark-Worker zu, um diese pro Batch abzuarbeiten.

Cassandra: Die Komponente erhält die Daten vom PySpark Executor. Die Daten können gespeichert und durch ein Plugin von Grafana abgefragt werden. Die Datenbank verfügt über eine Tabelle für die Speicherung der Daten.

Grafana: Die Komponent kann über ein Plugin Daten von Cassandra abfragen. Cassandra wird als Datenquelle eingetragen.

Die Abbildung 7.8 zeigt eine tiefere Ebene der MQTT_Kafka_Bridge Komponente. Es sind die Klassen MQTT_Kafka_Bridge.java, Kafka_StringConsumer.java, Application_pipeline.java, Kafka_StringProducer.java sowie das Interface MqttCallback zu sehen.

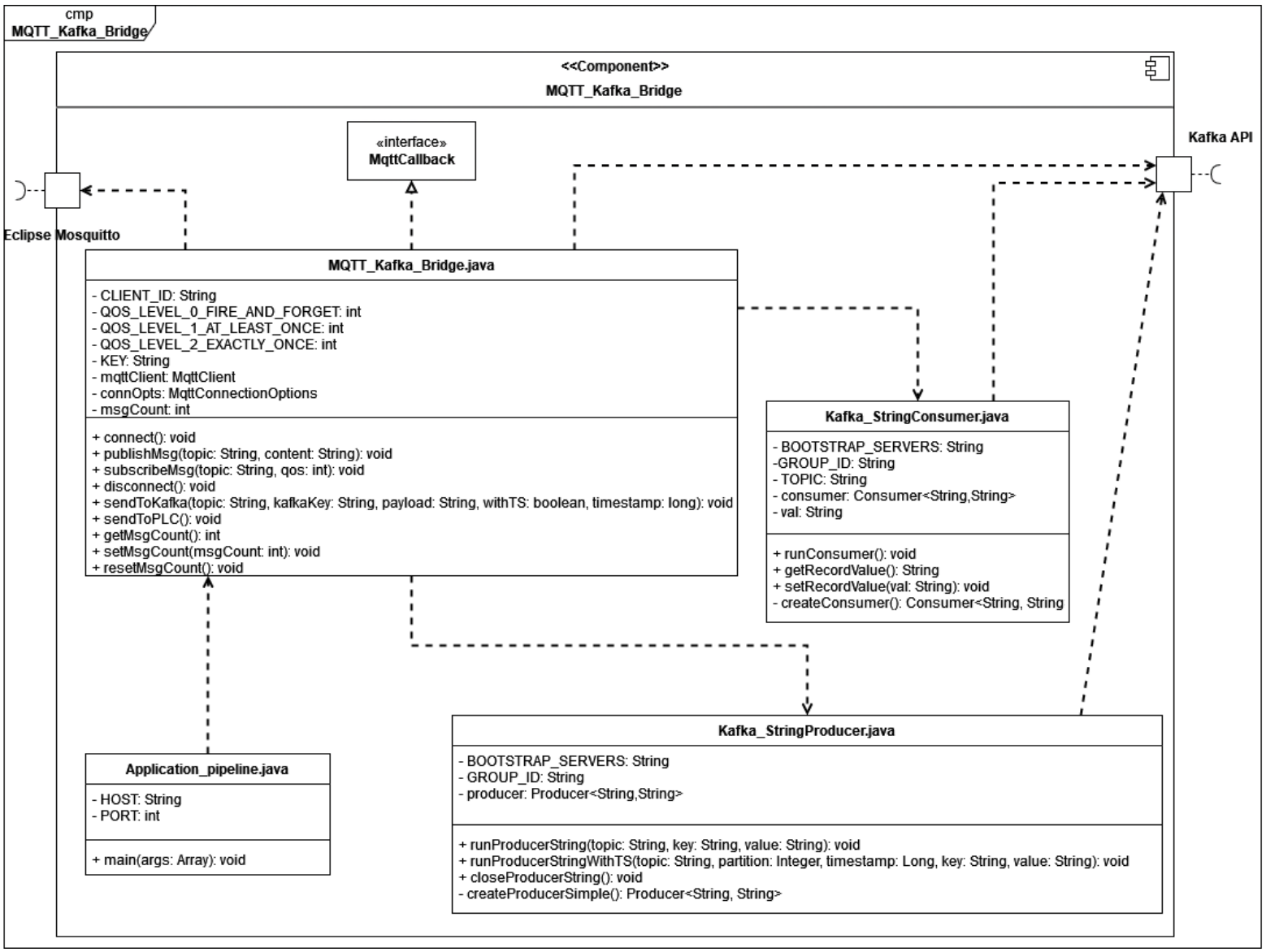

Abbildung 7.8: Klassendiagramm MQTT_Kafka_Bridge

MQTT_Kafka_Bridge.java Die Klasse MQTT_Kafka_Bridge.java implementiert die Methoden des MqttCallback Interfaces und nutzt die Eclipse Mosquitto Paho API sowie die Kafka API. Es ist der MQTT-Client, der die Nachrichten von der SPS erhält und sie an Kafka weiterleitet. Mit jeder ankommenden Nachricht wird ein Callback aufgerufen, der die Nachricht direkt an Kafka weitergeleitet. Hierfür wird die Kafka_StringProducer.java verwendet. Bei der Durchführung der Experimente werden hier die Konfigurationen vorgenommen. Die Nachrichten vom PySpark Executor werden von der Kafka_StringConsumer.java gepollt und an die SPS zurückgesendet.

Application_pipeline.java Die Klasse ist der Einstiegspunkt. Der Hostname und Port sind einstellbar. Es ist der Verbindungspunkt zum MQTT-Broker des Open-Source Servers. Nähere Informationen zur Kommunikation ist in Abbildung 7.9 zu sehen.

Kafka_StringConsumer.java Die Klasse erstellt einen Kafka Konsumenten und verbindet sich mit dem Kafka Broker. Die Parameter für die Latenzmessung werden in dieser Klasse verändert.

Kafka_StringProducer.java Die Klasse erstellt einen Kafka Produzenten und verbindet sich mit dem Kafka Broker. Die Parameter für die Latenzmessung werden in dieser Klasse verändert.

Verteilungssicht

Die Verteilungssicht ist in drei Teilen aufgeteilt: Datenextrahierung, -verarbeitung und -visualisierung. Die Sichten zeigen das Container-Netzwerk und die Kommunikation zwischen den Containern sowie die Kommunikation zur SPS.

Alle Container in allen Verteilungssichten sind durch die „volumes“ mit der Uhr des Servers synchronisiert.

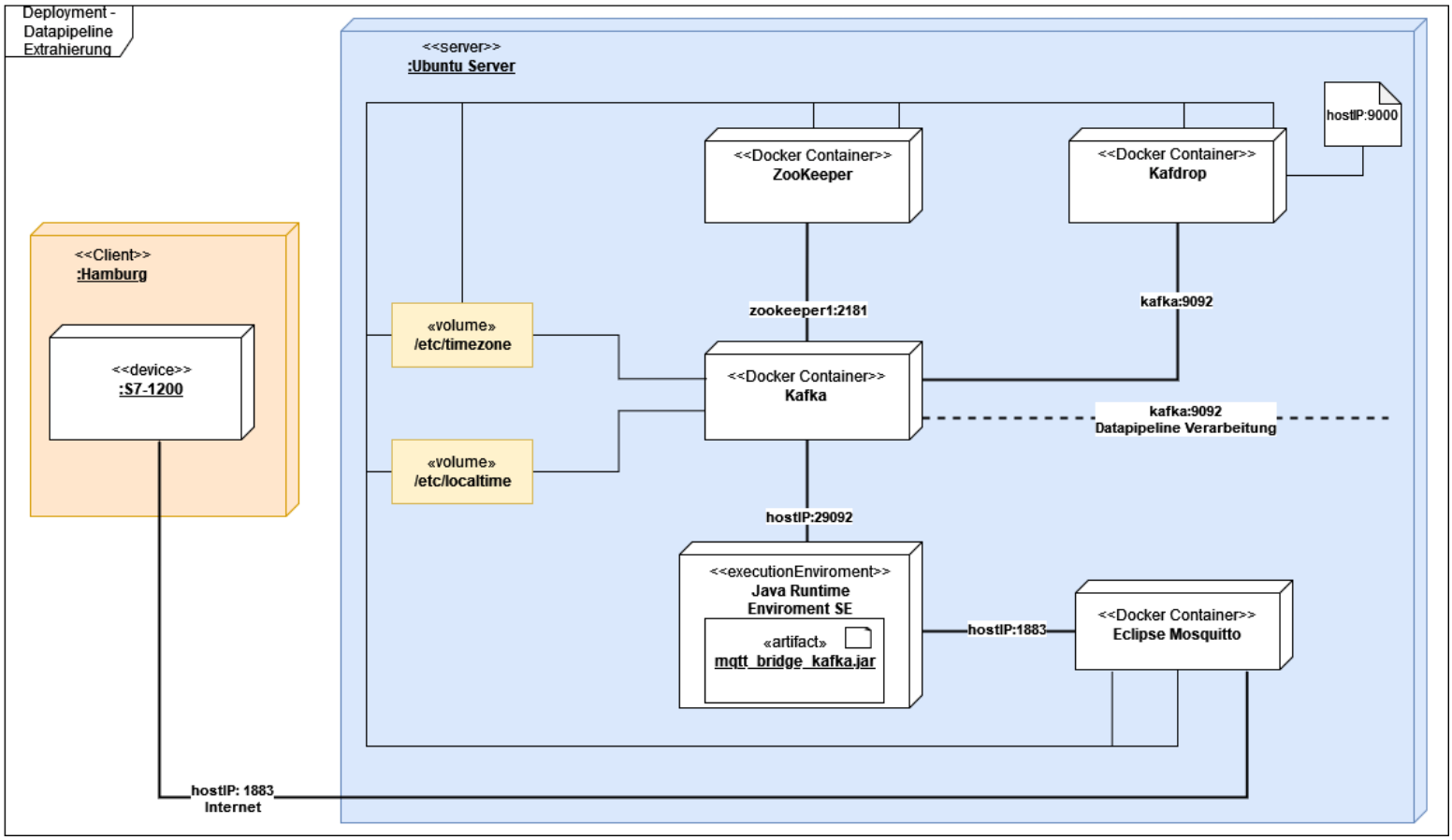

Abbildung 7.9: Verteilungssicht: Datenextrahierung

Die Datenextrahierung besteht aus den Containern: Kafka, Eclipse Mosquitto, Kafdrop und Zookeeper. Auf der linken Seite ist die SPS zu sehen, die über das Internet mit dem Server verbunden ist. Für die Verbindung wird eine Host-IP gebraucht.

Die SPS ist als MQTT-Client mit dem MQTT-Broker über die Host-IP und Port 1883 verbunden. Die Verbindung ist unverschlüsselt. Das selbe SPS-Programm, wie im AWS Aufbau wird verwendet.

Die Java-Applikation, welches auf dem Server läuft, ist als MQTT-Client ebenfalls mit dem MQTT-Broker über den Port 1883 verbunden. Diese verschickt die Daten über die Host-IP und Port 29092 an Kafka.

Kafdrop ist ein Monitoring-Werkzeug für Kafka. Es ist über den Port 9092 und Hostnamen kafka mit Kafka verbunden. Alle Nachrichten von Kafka können durch Angabe der Host-IP sowie Port 9000 per Fernzugriff beobachtet werden.

Zookeeper ist über den Hostnamen zookeeper1 und Port 2181 mit Kafka verbunden.

Kafka ist über die Ports 29092 und 9092 erreichbar. Kafka leitet die Nachrichten über den Hostnamen kafka und Port 9092 Nachrichten an den PySpark Executor weiter.

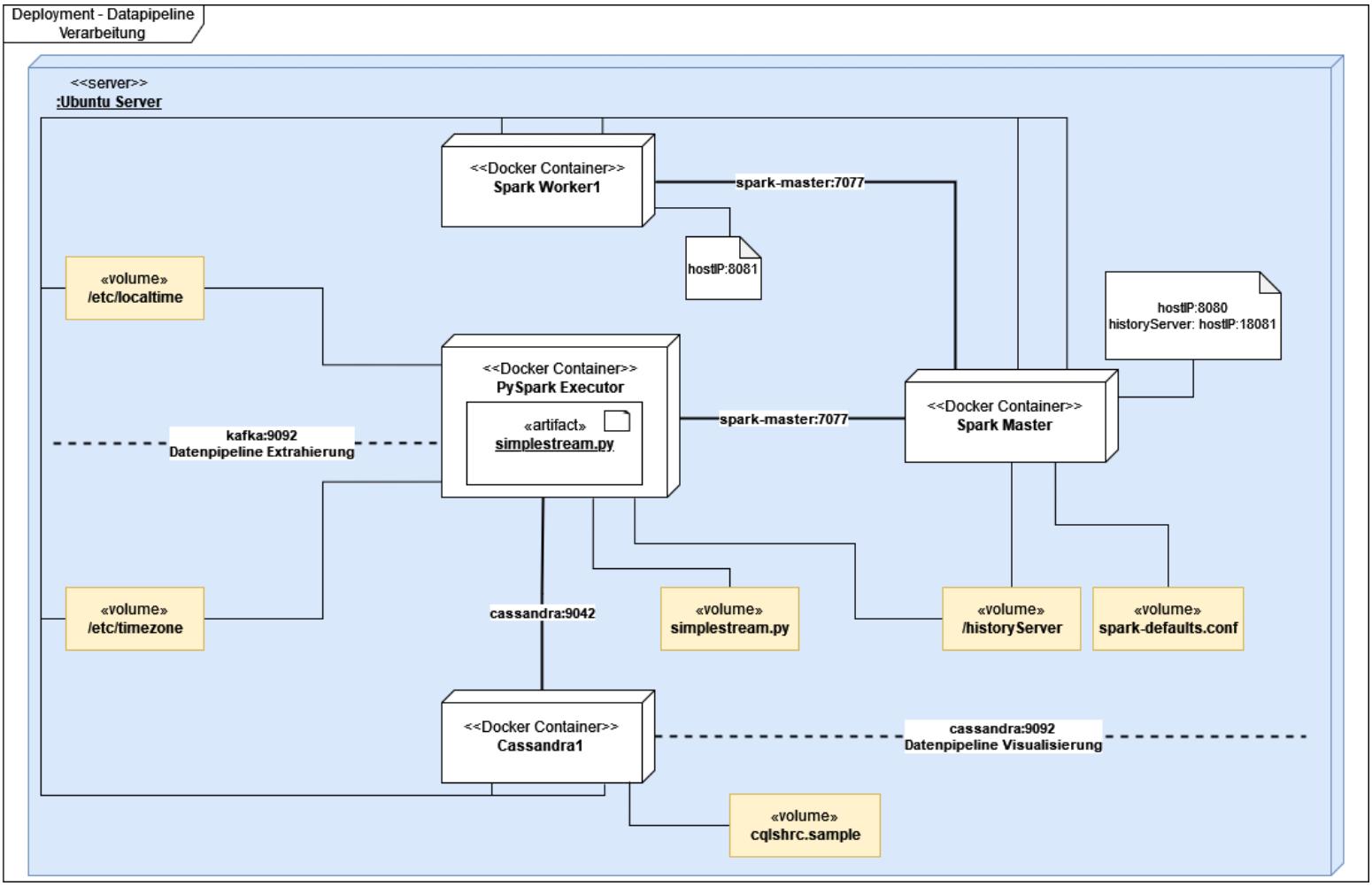

Abbildung 7.10: Verteilungssicht: Datenverarbeitung

Die Datenverarbeitung besteht aus den Containern: Spark Worker1, Cassandra1 und Spark Master.

Der PySpark Executor ist über den Hostnamen kafka und Port 9092 mit dem Kafka-Container verbunden. Außerdem ist der PySpark Executor mit der Datenbank über den Hostnamen cassandra und Port 9042 verbunden. Für die Ausführung der Spark-Jobs ist der PySpark Executor mit dem Spark-Master über den Host spark-master und Port 7077

verbunden. Der PySpark Executor enthält ein Python-Skript, welches über ein Docker Volume mit dem Server synchronisiert ist. Es lädt Jar Pakete runter. Diese enthalten Abhängigkeiten und Bibliotheken zu Cassandra, Spark und Kafka. Die Pakete sind für die Erstellung der Spark Session und zur Verarbeitung der Daten notwendig. Bei der Ausführung des Python-Skriptes werden Streams gestartet. Die Streams werden vom Spark Worker1 parallel als Batch verarbeitet. Es werden drei Streams ausgeführt. Ein Stream liest Daten von Kafka und fügt Zeitstempel hinzu. Die Daten werden in einen sogenannten „Dataframe“ gespeichert. Auf dem „Dataframe“ wird eine Datenfilterungsoperation ausgeführt. Der bearbeitete „Dataframe“ schickt die Daten mit einem Stream an Cassandra1. Ein weiterer Stream schickt Daten an Kafka zurück.

Der SparkWorker1 Container ist im Internet über die Host-IP und Port 8081 erreichbar. Des Weiteren ist der SparkWorker1 mit dem Spark Master über den Hostnamen spark-master und Port 7077 verbunden. Es können beliebig viele SparkWorker Container hinzugefügt werden. Die Einschränkung liegt bei der Hardware des Servers.

Der Spark Master ist im Internet über die Host-IP und Port 8080 erreichbar. Der Spark Master befindet sich im Cluster-Modus, weil Spark Aufgaben von einem Python Skript bekommt. Der History-Server von Spark ist im Internet über die Host-IP und Port 18080 erreichbar. Es ist ein Monitoring-Werkzeug, um Details zu Ressourcenverbrauch, Spark Jobs, Phasen und Aufgaben etc. zu beobachten. Der History Server sammelt über den Docker Volume Daten zu Spark Jobs vom PySpark Executor. Über den Hostnamen spark-master und Port 7077 ist der Spark-Master mit dem PySpark Executor verbunden. Spark lässt sich über den Docker Volume *spark-default.conf* konfigurieren. Die Konfiguration gibt in diesem Aufbau den Speicherort der History Server Daten an.

Der Cassandra Docker-Container ist mit dem PySpark Executor über den Hostnamen cassandra und Port 9042 verbunden. Es wird ebenfalls als Datenquelle für die Datenvisualisierung über den Hostnamen cassandra und Port 9042 angegeben. Es können beliebig viele Cassandra Container hinzugefügt werden. Die Einschränkung liegt bei der Hardware des Servers. Cassandra1 lässt sich über den Docker Volume *cqlshrc.sample* konfigurieren.

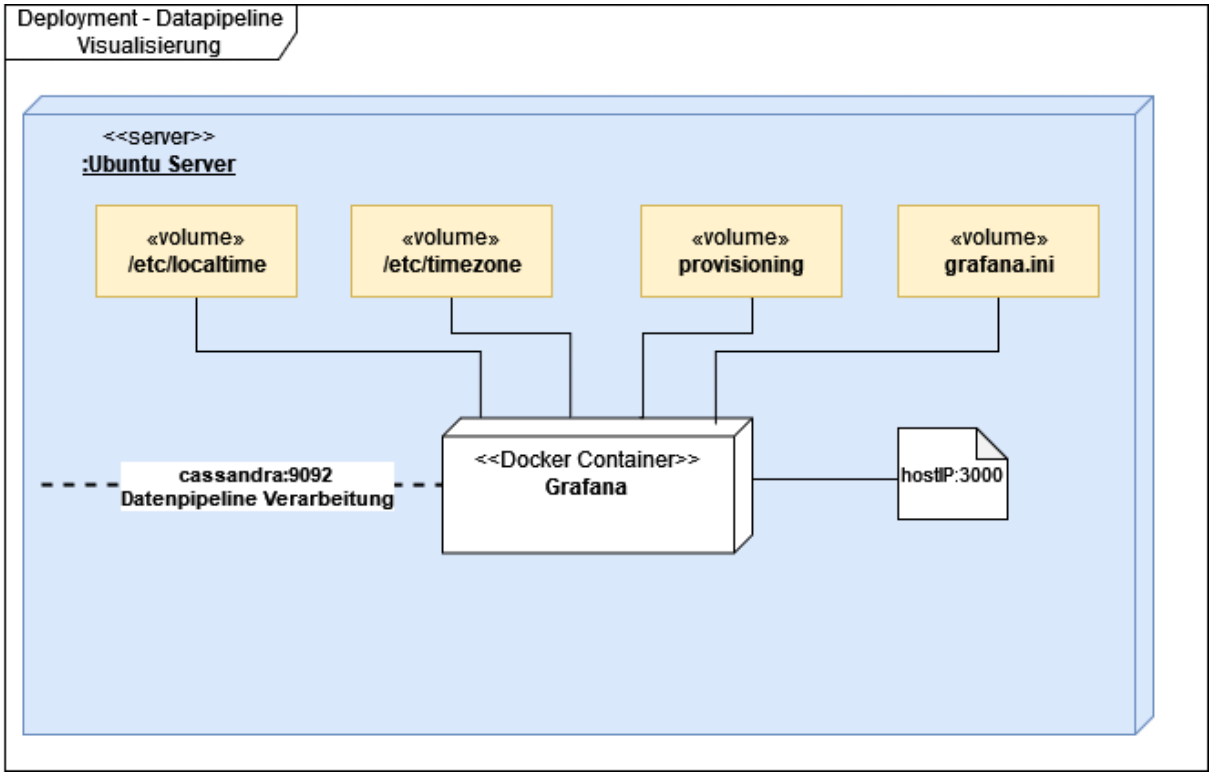

Abbildung 7.11: Verteilungssicht: Datenvisualisierung

Die Datenvisualisierung besteht aus dem Grafana Container, wo das Dashboard im Internet über die Host-IP und Port 3000 erreichbar ist. Der Cassandra1 Container ist eine Datenquelle. Beide kommunizieren über den Hostnamen cassandra und Port 9042. Mit Hilfe von Abfragen kann Grafana in bestimmten Zeitintervallen nach Daten bzw. SPS Daten abfragen. Das Dashboard wird durch das Docker Volume *provisioning* dauerhaft gespeichert. Grafana wird über das Docker Volume *grafana.ini* konfiguriert.

7.2 Durchführung

Der folgende Abschnitt beschreibt die Durchführung der Experimente. In den Messungen wird zwischen zwei Metriken unterschieden:

- Latenz
  - Es wird die Ende-zu-Ende Zeit für die Datenübertragung von der SPS bis zur Datenbank gemessen
- Durchsatz
  - Die Anzahl an gesendeten Daten, die in einem Zeitintervall geschickt werden

Als Erstes werden die Voraussetzungen für die Messungen aufgezählt. Beide Datenpipelines sind durch Parameter konfigurierbar. Es gibt für jedes Experiment andere Parameter, die einen Einfluss auf die Messungen nehmen. Die Parameter werden hierfür dargestellt und erläutert. Der Datenfluss wird anhand von Sequenzdiagrammen beschrieben. Die dazugehörigen Zeitstempel für die Messung werden ebenfalls dargestellt. AWS Dienste werden über die AWS Management Konsole konfiguriert.

Randbedingungen für die Latenz- und Durchsatzmessung:

- AWS Datenpipeline
  - Die SPS befindet sich im RUN-Modus
  - Datenbank und Tabelle in AWS Glue erstellt
  - Datenbank und Tabelle in AWS Timestream erstellt
  - S3 Bucket in Amazon S3 erstellt
  - Alle drei Regeln definiert
- Open-Source Datenpipeline
  - Die SPS befindet sich im RUN-Modus
  - MQTT Java Applikation läuft
  - Cassandra Datenbank und Tabelle erstellt
  - Python Skript wird ausgeführt

7.2.1 Messung: Latenz

Bei der Latenzmessung wird die Strecke bis zur Datenbank gemessen, da eine Abfragedauer von der Datenvisualisierung von der Abfrage abhängt. Diese Zeit wird nicht betrachtet. Die Nutzlast in den Nachrichten bleibt gleich. MQTT QoS ist in allen MQTT Clients auf 1 eingestellt. Die Nachrichten müssen mindestens einmal ankommen.

Der Host-PC ist mit den AWS Diensten mit dem NTP-Server „time.aws.com“ synchronisiert. Der Host-PC enthält Java-Applikationen, die Zeiten aus AWS Timestream und Amazon S3 entnehmen. Außerdem wird für die Differenzberechnung der Zeiten ein Python-Skript ausgeführt.

SPS-Programm

Die SPS verwendet in allen Messungen den MQTT-Baustein im SPS-Programm. Es gibt in allen Sequenzdiagrammen den MQTT-Baustein für das Senden und Empfangen von Nachrichten. Die Messungen laufen so lange, bis der Zähler *100/1000/10000* Durchläufe erreicht hat.

MQTT-Baustein1-Senden

Mit der Variable *MQTTDb.publish* und Wechsel auf die positive Flanke vom MQTT-Baustein wird die Nachricht versendet. Sobald die Nachricht verschickt wird, wechselt die Variable *MQTTDb.output* des MQTT-Bausteins auf *true*. Diese Bestätigung setzt die Variable *MQTTDb.publish* zurück. Danach wird *MQTTDb.output* wieder zurückgesetzt. Der Zähler wird anhand des Wechsels zur positiven Flanke von *MQTTDb.publish* erhöht. Das Setzen und Zurücksetzen von *MQTTDb.publish* wird so lange durchlaufen, bis der Zähler seinen eingestellten Endstand erreicht hat.

Bei jeder positiven Flanke von *MQTTDb.publish* wird ein Zeitstempel *Tsps_start* gespeichert.

AWS Parameter

Viele Dienste vom AWS Aufbau werden autoskaliert und haben durch die Dienstrichtlinien bestimmte Grenzen. Die Anzahl der MQTT-Clients, Datenbanken, Tabellen und S3 Bucket bleiben in den jeweiligen Diensten, die in Abschnitt 7.1.1 beschrieben sind, gleich.

Folgende Parameter werden für die Latenzmessung verändert:

- AWS Kinesis Data Firehose
  - Pufferintervall: Die Zeit, um Nachrichten zu sammeln. Nach Ablauf der Zeit werden die Nachrichten an S3 gesendet.
  - Puffergröße der Nachrichten: Wird die Puffergröße erreicht, werden die Nachrichten an S3 gesendet

Ablauf AWS

Der folgende Abschnitt zeigt den Datenverlauf und den Ablauf der Messungen. Die Sequenzdiagramme sollen zeigen, an welchen Stellen in der AWS Datenpipeline ein Zeitstempel entnommen wird. Die Latenzmessung wird jeweils 100,1000 und 10000 mal ausgeführt.

Messung: SPS zu AWS Timestream

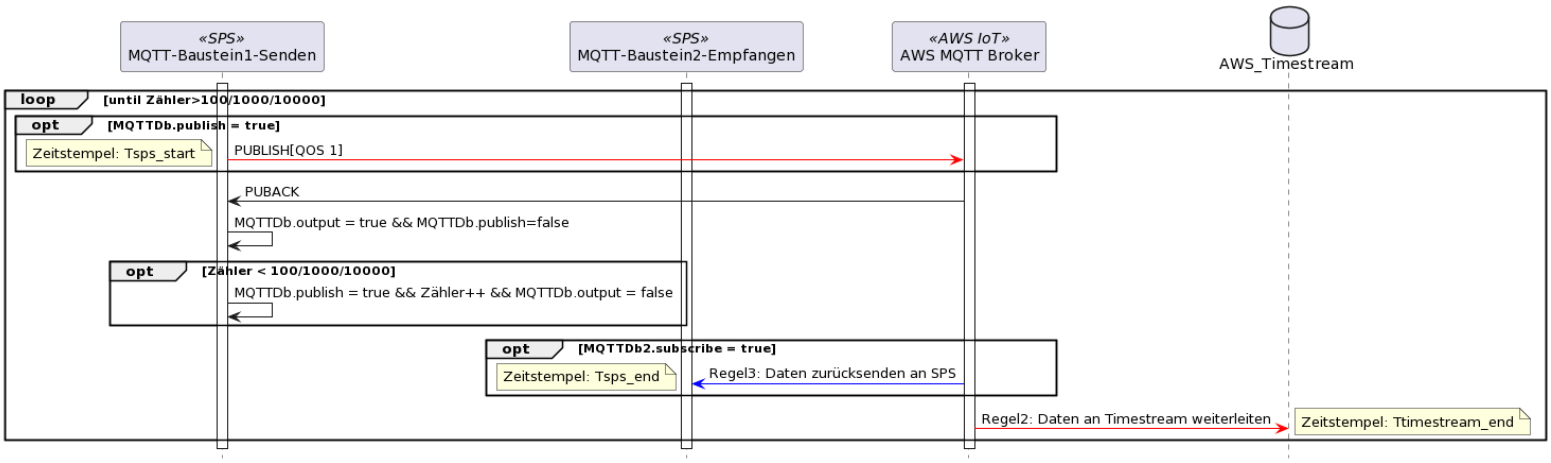

Abbildung 7.12: Datenfluss von der SPS zu AWS Timestream

Die Abbildung 7.12 zeigt die Messung von der SPS bis AWS Timestream. Es ist eine AWS MQTT Broker Instanz und AWS Timestream als Datenbank zu sehen. Sobald die Nachricht den AWS MQTT Broker erreicht, wird *Regel3* ausgeführt. *Regel3* schickt die Daten mit QoS 1 zurück. Die Daten erreichen den MQTT-Baustein2-Empfangen. Es erstellt einen Zeitstempel, sobald die Nachricht angekommen ist. Die Variable *MQTTDb2.subscribe*

muss dafür auf *true* gesetzt werden. *Regel2* leitet die Daten an Timestream weiter. Sobald die Daten die Tabelle erreichen, wird ein automatisch von Timestream eine Spalte mit den Zeitstempel erstellt. Die Zeitstempel von AWS Timestream werden für die Differenzrechnung erstellt.

Folgende Differenzen werden subtrahiert, um die Gesamtlatenz zu messen:

Zeitdifferenz: SPS und AWS MQTT Broker

Übertragungsverzögerung: $\frac{Tsps\_end - Tsps\_start}{2}$

Zeitdifferenz: SPS und AWS Timestream

Verzögerung_SPS_Timestream: $Ttimestream\_end - Tsps\_start$

Messung: SPS zu Amazon S3

Bei der Ausführung wird zwischen zwei Parameterkonfigurationen für den *PUT-S3-Stream* unterschieden:

- Ausführung der Messung 1 (schlechtester Fall):
  - Puffergröße: 5 MiB
  - Pufferintervall: 300 Sekunden
- Ausführung der Messung 2 (bester Fall):
  - Puffergröße: 1 MiB
  - Pufferintervall: 60 Sekunden

Das Sequenzdiagramm 7.13 zeigt die Durchführung der Messung von der SPS zu AmazonS3. Es ist ebenfalls ein AWS MQTT Broker sowie eine AWS Kinesis Data Firehose Instanz und AmazonS3 Datenbank zu sehen. Im Vergleich zum Sequenzdiagramm 7.12 sendet die *Regel1* die Daten über den *PUT-S3-Stream* weiter. Der *PUT-S3-Stream* wird jeweils für zwei Parameterkonfigurationen verändert. Sobald die Daten erreichen AmazonS3 erreichen, werden diese in Objekte gespeichert. Die Objekte von AWS jeweils einen Zeitstempel. Ein Objekt kann eine Bündelung von Datensätze enthalten. Hierbei kommt es auf die Konfiguration der Puffergröße an.

Folgende Differenzen werden subtrahiert, um die Gesamtlatenz zu messen:

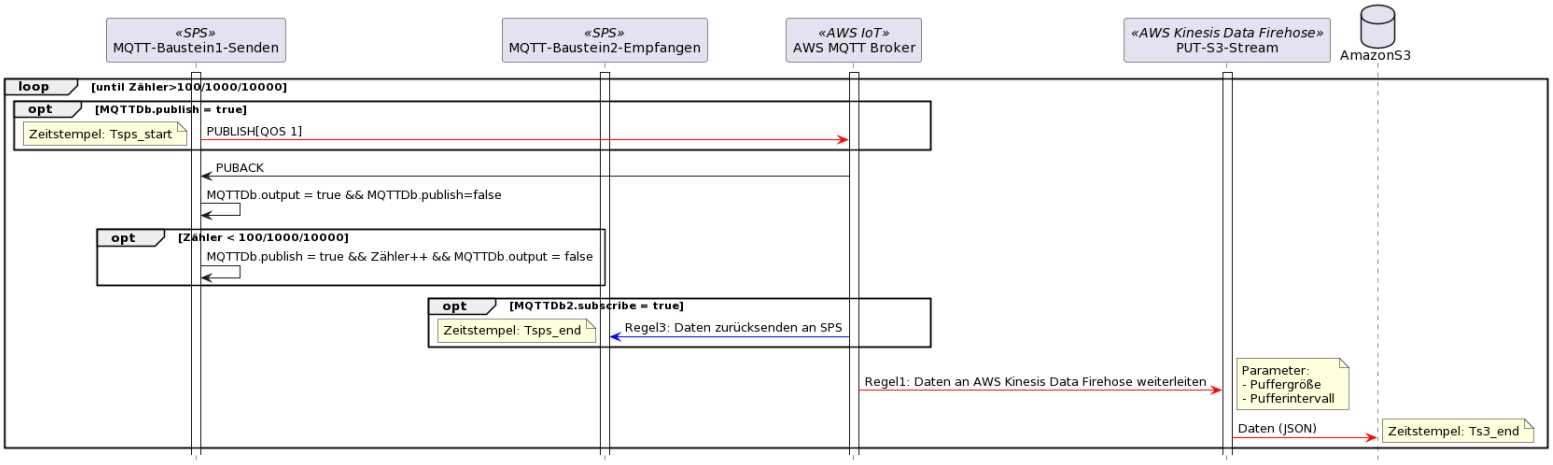

Abbildung 7.13: Datenfluss von der SPS zu Amazon S3

Zeitdifferenz: SPS und AWS MQTT Broker

Übertragungsverzögerung: $\frac{Tsps\_end - Tsps\_start}{2}$

Zeitdifferenz: SPS und Amazon S3

Verzögerung_SPS_S3: $Ts3\_end - Tsps\_start$

Open-Source Parameter

Es gibt viele Parameter in den einzelnen Komponenten, die die Gesamtlatenz erhöht oder verringert. Daher werden für die Durchführung der Messungen nur bestimmte, ausgewählte Parameter betrachtet. Die Parameter für Kafka werden in Kapitel 5 Abschnitt 5.3.1, für Cassandra in Kapitel 5 Abschnitt 5.3.3 erklärt.

Folgende Parameter werden für die Latenzmessung verändert:

- Kafka
  - Produzent: *linger.ms*
  - Produzent: *compression.type*
  - Produzent: *batch.size*

  - Konsument: *fetch.min.bytes*

- Spark
  - Anzahl der Spark Worker
  - Container Umgebungsvariable *SPARK_ WORKER_ MEMORY*: Größe des Speichers
  - Container Umgebungsvariable *SPARK_ WORKER_ CORES*: Anzahl der CPU Kerne
- Cassandra
  - Anzahl der Cassandra Knoten
  - *replicationfactor*
  - *consistency level*

Ablauf

Für die Übersicht wird der Datenfluss in zwei Teilen aufgeteilt. Das Sequenzdiagramm 7.14 zeigt den Datenfluss der Datenextrahierung. Die Sequnezdiagramme 7.16 und 7.15 zeigen zwei unterschiedliche Parameterkonfigurationen. Aufgrund des Uhrensynchronisationsproblem wird die Strecke SPS zu Spark als „Round-Trip-Time“ gemessen. Da alle Container durch die Docker Volumes */etc/timezone* und */etc/localtime* und im gleichen Server laufen, wird die Strecke von Spark zu Cassandra ohne NTP-Server oder anderen Synchronisation, gemessen.

Die folgende Abbildung 7.14 zeigt den Datenfluss bis zum Kafka Broker. Es sind wieder dieselben MQTT-Bausteine zu sehen. Es sind ebenfalls die Klassen Application_pipeline.java, MQTT_Bridge_Kafka.java, StringProducer.java und StringProducer.java zu sehen. Die rot, markierte Farbe an den Aufrufen zeigt den Hinweg der Daten. Auf der anderen Seite zeigt die blaue, markierte Farbe an den Aufrufen den Rückweg der Daten. Die Application_pipeline.java erstellt ein Objekt der Klasse MQTT_Bridge_Kafka.java. Es wird ein MQTT-Client erstellt, der eine Verbindung zum MQTT-Broker herstellt und ein Topic abonniert. MQTT_Bridge_Kafka.java erstellt ein Objekt der Klasse StringProducer.java. Beim Aufruf wird ein Objekt mit der Möglichkeit die Parameter zu ändern, erstellt. MQTT_Bridge_Kafka.java erstellt ebenfalls ein Objekt der Klasse StringConsumer.java. Beim Aufruf wird ein Objekt mit der Möglichkeit die Parameter zu ändern, erstellt.

Datenfluss und Zeitstempel

Es handelt sich um das gleiche SPS-Programm, welches für die AWS Messung verwendet wird. Die Stellen für die Zeitstempel Messung bleiben gleich. MQTT-Baustein1-PUB sendet die Nachrichten an den MQTT-Broker. Die MQTT_Bridge_Kafka.java erhält diese Nachrichten. Hierbei wird durch den MqttCallback die Methode *messageArrived* aufgerufen. Es wird intern die *sendToKafka()* Methode aufgerufen. Diese greift die Methode *runProducerString()* vom StringProducer zu. Der StringProducer schickt die Nachrichten an den Kafka Broker weiter.

Um die „Round-Trip-Time“ zu messen, müssen die Daten von Spark über den Kafka Broker zurückgesendet werden. Der StringConsumer ruft hierbei die Methode *runConsumer()* auf und pollt nach Kafka Nachrichten. Die Methode *runConsumer()* wird wiederrum durch die Methode *sendToPLC()* aufgerufen. Die Methode *sendToPLC()* von der Klasse MQTT_Bridge_Kafka wird von Application_pipeline.java aufgerufen. Diese Methode befindet sich in einer Dauerschleife, die die Kafka Nachrichten vom StringConsumer an den MQTT Broker weiterleitet bzw. veröffentlicht. Der MQTT Broker leitet diese Nachricht an den MQTT-Baustein2-SUB weiter.

Bei der Ausführung wird zwischen zwei Parameterkonfigurationen für den Kafka Produzent und Kafka Konsumenten unterschieden:

- Ausführung der Messung 1 (schlechtester Fall):
  - Kafka Produzent

        * LINGER_MS_CONFIG = 100
        * BATCH_SIZE_CONFIG = 200000
        * COMPRESSION_TYPE_CONFIG = lz4
    - Kafka Konsument
        * FETCH_MIN_BYTES_CONFIG = 1000000
- Ausführung der Messung 2 (bester Fall):
    - Kafka Produzent
        * LINGER_MS_CONFIG = 0
        * BATCH_SIZE_CONFIG = 10000
        * COMPRESSION_TYPE_CONFIG = none
    - Kafka Konsument
        * FETCH_MIN_BYTES_CONFIG = 1

Folgende Differenz werden gebildet, um die Latenz zwischen der SPS und Spark zu messen:

Zeitdifferenz: SPS und Spark

Strecke zu Spark: $\frac{Tsps\_end - Tsps\_start}{2}$

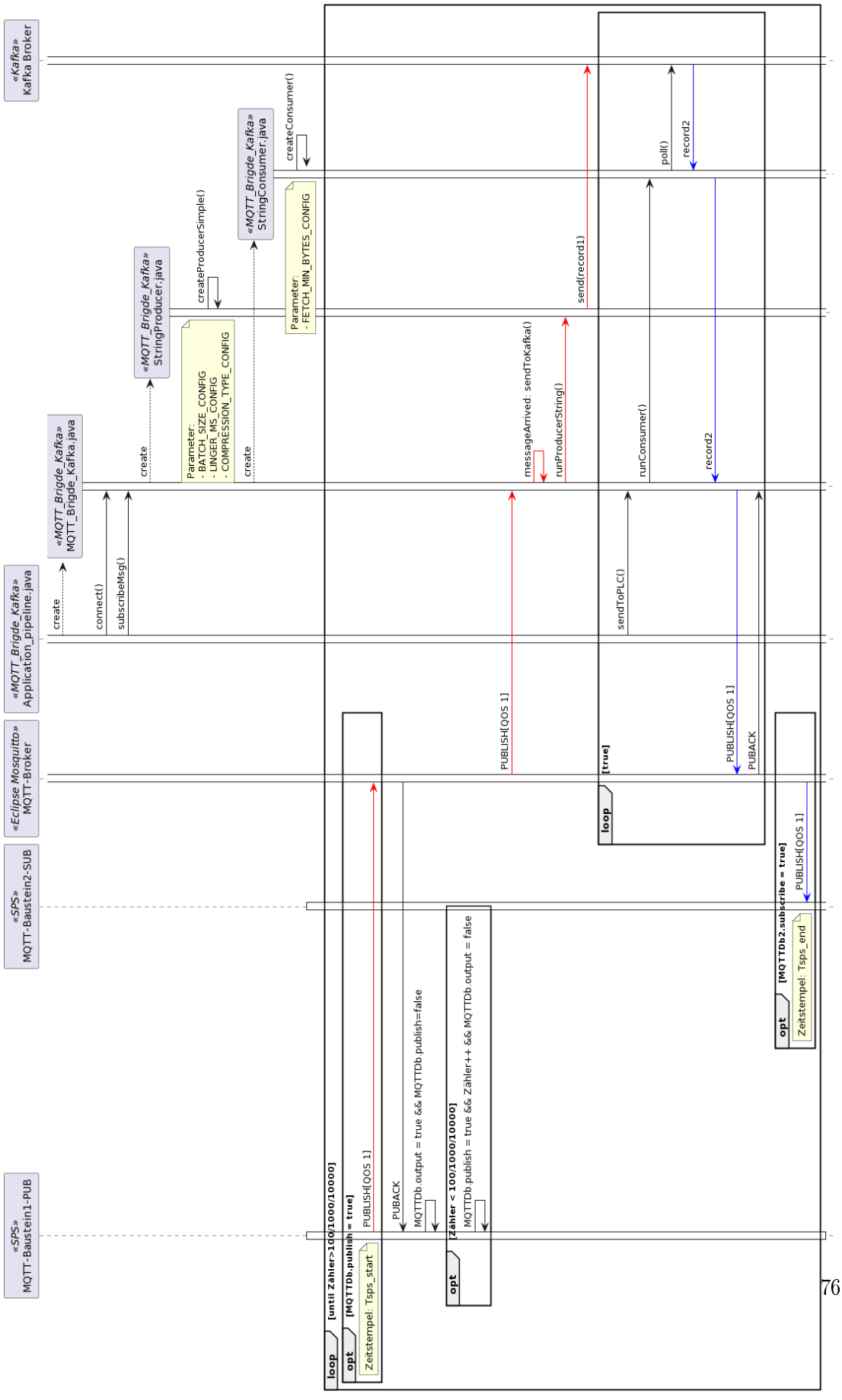

Abbildung 7.14: Datenfluss von der SPS zum Kafka Broker

Die Abbildung 7.15 zeigt die Parameterkonfiguration und das Sequenzdiagramm für die Verarbeitung der Daten im schlechtesten Fall:

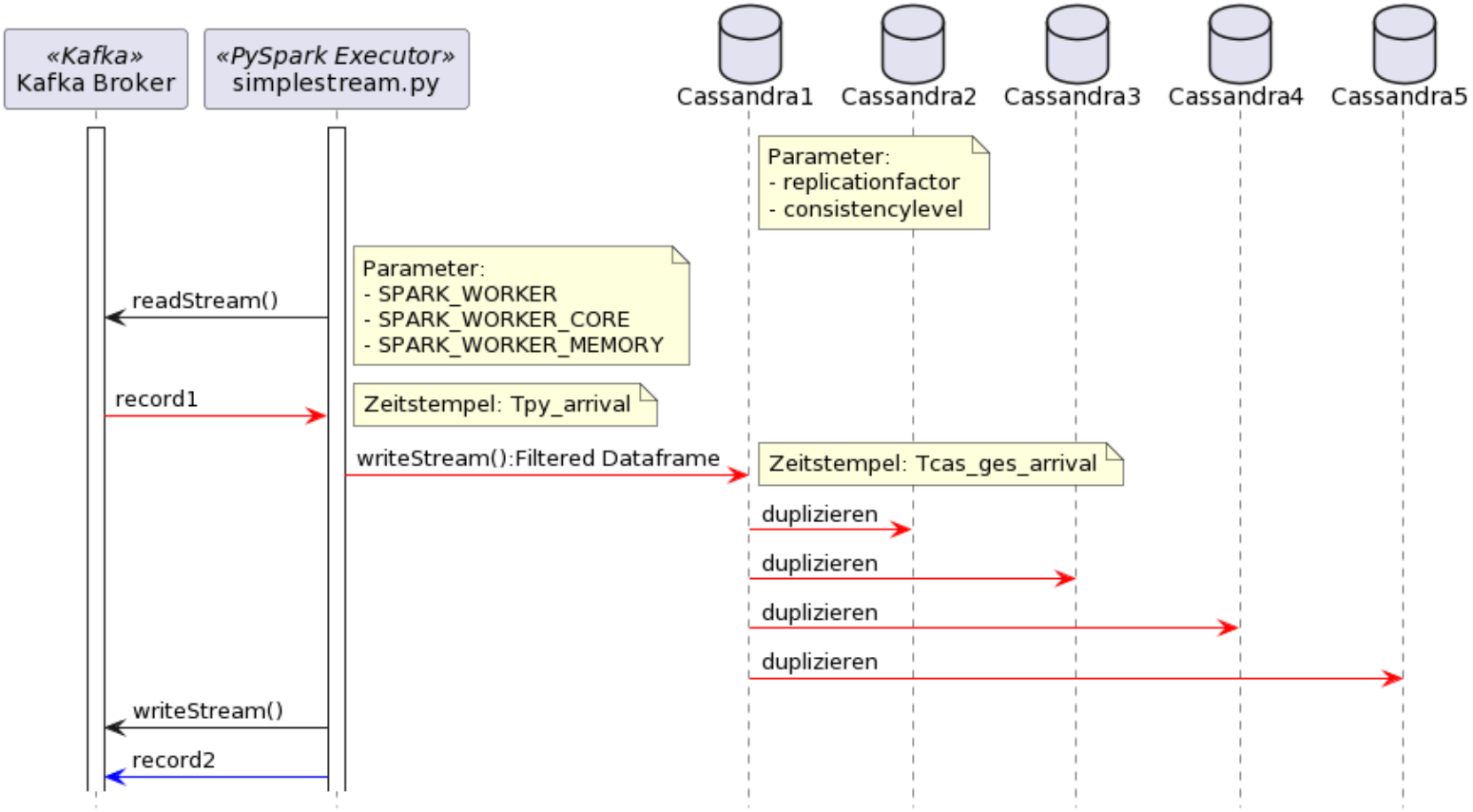

Abbildung 7.15: Datenfluss vom Kafka Broker zu Cassandra im schlechtesten Fall

Auf der Abbildung 7.15 ist ein Kafka Broker, der PySpark Executor und fünf Cassandra Datenbanken zu sehen. Wird der *simplestream.py* ausgeführt, werden drei Streams erstellt. Ein Stream liest von Kafka, während ein anderer Stream die Nachrichten für die „Round-Trip-Time“ an Kafka zurücksendet. Ein weiterer Stream leitet die Kafka Nachrichten weiter an Cassandra. Sobald die Daten im PySpark Executor ankommen, wird ein Zeitstempel angefügt. Im schlechtesten Fall werden die Daten in fünf weitere Cassandra Tabellen repliziert. Im schlechtesten Fall werden ebenfalls die Spark Ressourcen verändert.

Bei der Ausführung sind folgende Parameter konfiguriert:

- Ausführung der Messung 1 (schlechtester Fall):
  - Spark
    - SPARK_WORKER = 2
    - SPARK_WORKER_MEMORY = 1G
    - SPARK_WORKER_CORE = 1
  - Cassandra
    - replicationfactor = 5
    - consistencylevel = all

Folgende Differenz wird gebildet, um die Latenz zwischen dem PySpark Executor und Cassandra zu messen:

Zeitdifferenz: SPS und Cassandra

Strecke zu Cassandra: $Tcas\_ges\_arrival - Tpy\_arrival$

Die Abbildung 7.16 zeigt die Parameterkonfiguration und das Sequenzdiagramm für die Verarbeitung der Daten im besten Fall:

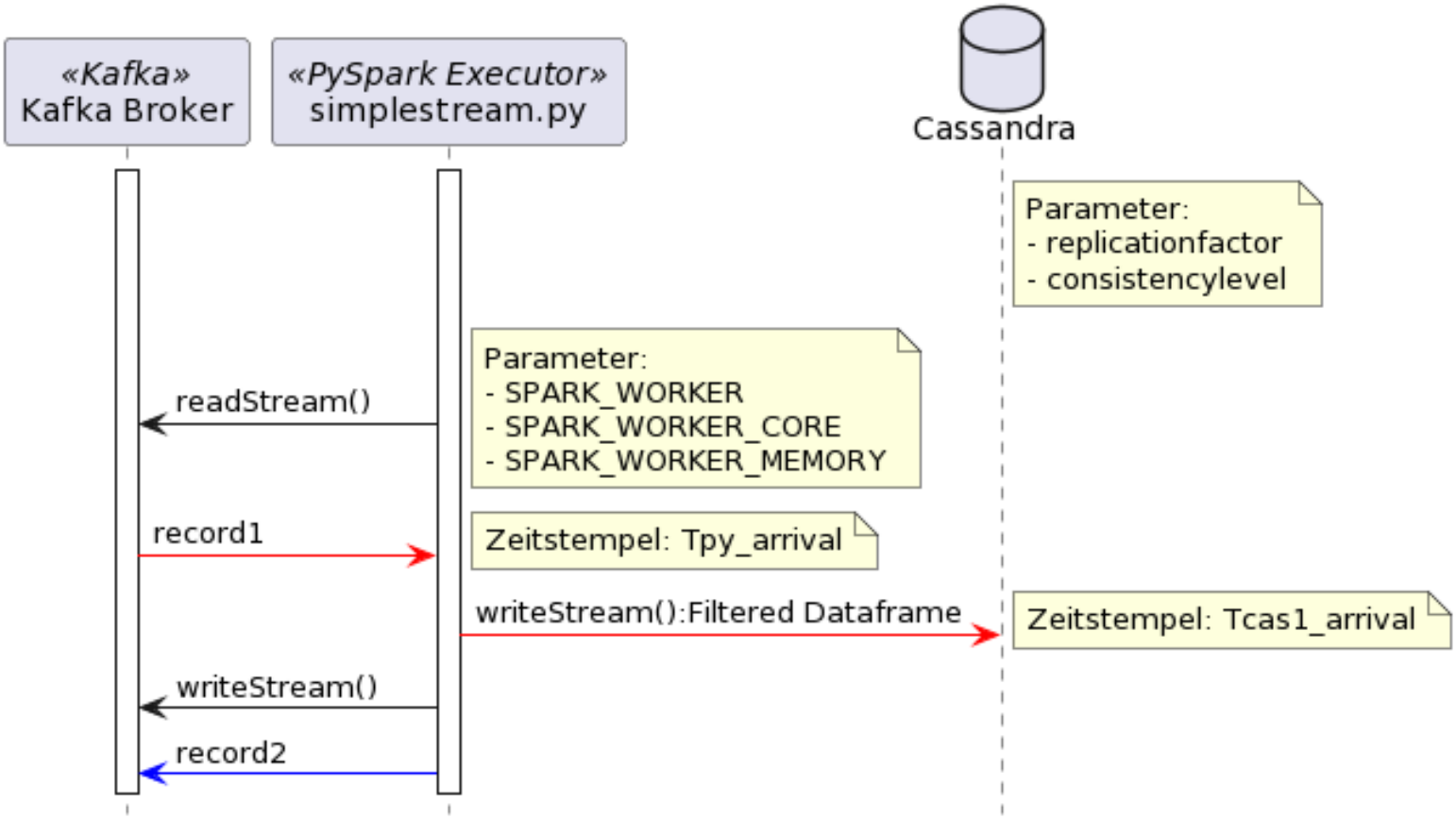

Abbildung 7.16: Datenfluss vom Kafka Broker zu Cassandra im besten Fall

Der Unterschied zur Abbildung 7.15 ist die Parameterkonfiguration. Der Datenaustausch und die Differenzrechnung bleibt gleich.

Bei der Ausführung sind folgende Parameter konfiguriert:

- Ausführung der Messung 2 (bester Fall):
  - Spark
    - SPARK_WORKER = 2
    - SPARK_WORKER_MEMORY = 6G
    - SPARK_WORKER_CORE = 4

- Cassandra
  - replicationfactor = 1
  - consistencylevel = 1

7.2.2 Messung Durchsatz

Es wird die Anzahl an Nachrichten, die in der Datenbank angekommen sind, gemessen. In beiden Aufbauten werden keine weiteren Konfigurationen getätigt.

Das SPS-Programm sieht folgendermaßen aus:

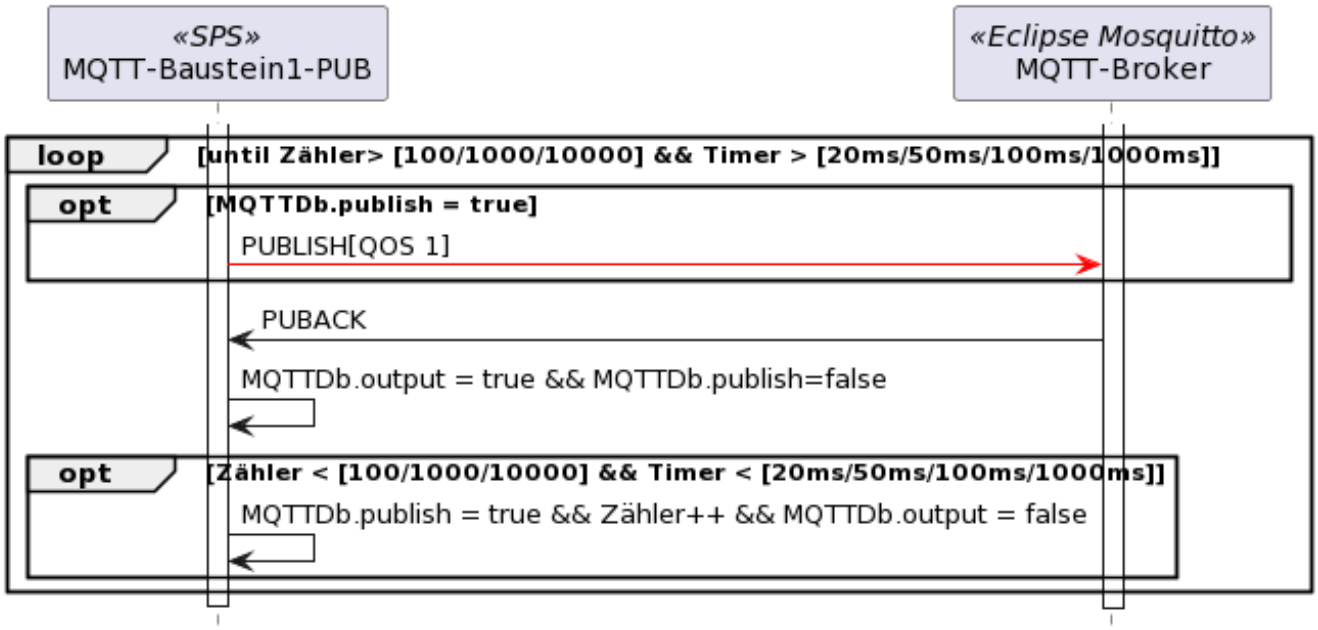

Abbildung 7.17: SPS Programm für die Durchsatzmessung

Die Abbildung 7.17 zeigt den MQTT-Baustein1 und beispielsweise einen MQTT-Broker. Das Programm gilt ebenfalls für die AWS Datenpipeline. Im Vergleich zum SPS-Programm für die Latenzmessung, kommt noch ein Timer hinzu. Dieser läuft in einem bestimmten Zeitintervall ab. In diesem Intervall wird die Variable *MQTTDb.publish* gesetzt und zurückgesetzt, sodass so viele Nachrichten, wie möglich geschickt werden. Die Messung wird für die Intervalle *[20, 50, 100, 1000]ms* wird 100, 1000 und 10000 mal wiederholt.

7.3 Ergebnisse

Der folgende Abschnitt stellt die Ergebnisse der Messungen dar.

7.3.1 Latenz

AWS

Stationen	Anzahl	Worst Case avg in ms	Best Case avg in ms
SPS >AWS IoT	100	40,36	36,64
SPS >S3		59997,91	299900,85
SPS >Timestream		1434,6	1425,32

Tabelle 7.2: Latenz: Messung bei 100 Wiederholungen(AWS)

Stationen	Anzahl	Worst Case avg in ms	Best Case avg in ms
SPS >AWS IoT	1000	164,72	87,839
SPS >S3		34652,80	263171,32
SPS >Timestream		1350,193	1403,75

Tabelle 7.4: Latenz: Messung bei 1000 Wiederholungen (AWS)

Stationen	Anzahl	Worst Case avg in ms	Best Case avg in ms
SPS >AWS IoT	10000	91,77	88,75
SPS >S3		30902,65	165659,18
SPS >Timestream		1177,84	1250,23

Tabelle 7.6: Latenz: Messung bei 10000 Wiederholungen(AWS)

Open-Source

Stationen	**Anzahl**	**Worst Case avg in ms**	**Best Case avg in ms**
SPS >Spark	100	118,7	53,8
Spark >Cassandra		483,76	577,18
Gesamt in ms		**602,46**	**630,98**

Tabelle 7.8: Latenz: Messung bei 100 Wiederholungen(OP)

Stationen	**Anzahl**	**Worst Case avg in ms**	**Best Case avg in ms**
SPS >Spark	1000	125,374	68,144
Spark >Cassandra		175,147	196,19
Gesamt in ms		**300,521**	**264,334**

Tabelle 7.10: Latenz: Messung bei 1000 Wiederholungen(OP)

Stationen	**Anzahl**	**Worst Case avg in ms**	**Best Case avg in ms**
SPS >Spark	10000	128,19	51,48
Spark >Cassandra		130,07	132,688
Gesamt in ms		**258,26**	**184,168**

Tabelle 7.12: Latenz: Messung bei 10000 Wiederholungen(OP)

7.3.2 Durchsatz

AWS

Zeitintervall in ms	**Anzahl Wiederholungen**	**Datenbank erreicht**	**Avg Nachrichten/Zeitintervall**
20	100	90	**0,9**
50		133	**1,33**
100		52	**0,52**
1000		450	**4,5**

Tabelle 7.14: Durchsatz: Messung bei 100 Wiederholungen(AWS)

Zeitintervall in ms	Anzahl Wiederholungen	Datenbank erreicht	Avg Nachrichten/Zeitintervall
20	1000	486	**0,486**
50		337	**0,337**
100		381	**0,381**
1000		3851	**3,85**

Tabelle 7.16: Durchsatz: Messung bei 1000 Wiederholungen(AWS)

Zeitintervall in ms	Anzahl Wiederholungen	Datenbank erreicht	Avg Nachrichten/Zeitintervall
20	10000	3215	**0,3215**
50		3516	**0,3516**
100		5373	**0,5373**
1000		33681	**3,37**

Tabelle 7.18: Durchsatz: Messung bei 10000 Wiederholungen(AWS)

Open-Source

Zeitintervall in ms	Anzahl Wiederholungen	Datenbank erreicht	Avg Nachrichten/Zeitintervall
20	100	110	**1.1**
50		170	**1,7**
100		285	**2,85**
1000		2197	**20,92**

Tabelle 7.20: Durchsatz: Messung bei 100 Wiederholungen(OP)

Zeitintervall in ms	Anzahl Wiederholungen	Datenbank erreicht	Avg Nachrichten/Zeitintervall
20	1000	747	**0,747**
50		1544	**1,54**
100		2628	**2,6**
1000		15990	**15,99**

Tabelle 7.22: Durchsatz: Messung bei 1000 Wiederholungen(OP)

Zeitintervall in ms	Anzahl Wiederholungen	Datenbank erreicht	Avg Nachrichten/Zeitintervall
20	10000	5566	**0,56**
50		12592	**1,26**
100		11701	**1,17**
1000		28986	**2,8986**

Tabelle 7.24: Durchsatz: Messung bei 10000 Wiederholungen(OP)

8 Diskussion

Im folgenden Abschnitt werden die Ergebnisse aus den Messungen diskutiert. Es werden die Erwartungen und Ergebnisse verglichen. Hinzu kommt, dass Unterschiede und Gemeinsamkeiten ebenfalls diskutiert werden.

Erwartung an Latenz

Ein Erwartung ist, dass es keine Echtzeitgarantien gibt, da die Kommunikation aufgrund der verwendeten Technologien mit „Best Effort“ durchgeführt wird. Die Open-Source Datenpipeline bietet mehr Möglichkeiten die Parameter zu ändern, um geringere Verzögerungen zu erreichen. Auf der anderen Seite gibt es für die Nutzung der Dienste spezifizierte Dienstrichtlinien. Die Dienste skalieren automatisch nach Anzahl der Daten, Last etc. Daher ist die Erwartung, dass die Open-Source Datenpipeline geringere Latenzen hat.

Latenz: Ergebnisse

Betrachtet man die AWS Ergebnisse (siehe Abschnitt 7.3.1) für die Latenzmessung, sieht man geringere Latenzen über AWS Timestream. Die Latenzen in der Timestream Strecke bleiben ungefähr gleich. Die Parameter von AWS Kinesis Data Firehose ändern signifikant die Latenzen. Im Vergleich zur Timestream Strecke, sind diese sehr groß. Da der Weg von der SPS zum AWS Server jedes Mal unterschiedlich ist, kann die Übertragungsverzögerung vernachlässigt werden bzw. es ist für die Gesamtmessung nicht relevant.

Betrachtet man die Open-Source Ergebnisse (siehe Abschnitt 7.3.1), kann man erkennen, dass in beiden Fällen die Latenzen geringer als die AWS Ergebnisse sind. Es ist ebenfalls erkennbar, dass mit der Erhöhung der Wiederholungen die Latenzen geringer werden. Die Einstellungen der Parameter verringern die Latenzen von der Strecke SPS zu Cassandra. Es ist erkennbar, dass mehr Ressourcen für Spark (hier: Anzahl an Worker, CPU Cores und Speicher) keine schnellere Verarbeitung mit sich bringen. Das sieht man anhand in

der Strecke von Spark zu Cassandra. Die Übertragungsverzögerung ist hier ebenfalls zu vernachlässigen.

Latenz: Vergleich mit Ergebnisse und Erwartung

Vergleicht man die Erwartung und Ergebnisse miteinander, sind die Erwartungen erfüllt worden. Die Open-Source Datenpipeline hat geringere Latenzen und die Ergebnisse reichen für keine Echtzeit-Überwachung. Der Unterschied der Latenzen im besten Fall für beide Datenpipelines ist jedoch unerwartet groß.

Erwartung an Durchsatz

Es ist zu erwarten, dass die gleiche Anzahl an ankommenden Daten in beiden Datenpipelines zu finden ist. Es ist ein hoher Durchsatz in beiden Datenpipelines zu erwarten. Das bedeutet, dass viele Nachrichten pro Zeitintervall ankommen. Durch die Einstellung MQTT QoS 1 ist kein Datenverlust zu erwarten. Der Durchsatz hängt ebenfalls von der Datenquelle ab.

Durchsatz: Ergebnisse

Vergleicht man beide Tabellen, sieht man dass der Durchsatz in der Open-Source Datenpipeline pro Zeitintervall größtenteils höher ist. Es ist ein höherer Durchsatz bei beiden Aufbauten erkennbar je größer der Zeitintervall ist. In der Tabelle 7.24 und Tabelle 7.14 ist zu sehen, dass im Zeitintervall 50ms ein höherer Durchsatz vorhanden ist. Außerdem ist in Tabelle 7.16 zu sehen, dass im Zeitintervall 20ms ein höherer Durchsatz zu sehen.

Durchsatz: Vergleich mit Ergebnisse und Erwartung

Die Erwartung, dass die Anzahl der ankommenden Nachrichten gleich ist, wird nicht erfüllt. Es ist ein höherer Durchsatz in der Open-Source Datenpipeline zu sehen. Der Grund, dass die Anzahl der ankommenden Nachrichten unterschiedlich sind, liegt am MQTT-Baustein. Der Befehl im MQTT-Baustein, um die Nachrichten zu senden, benötigt mehrere Zyklen, um die Nachrichten zu senden. Das SPS Programm bei beiden Messungen ist gleich, jedoch ist die Zykluszeit jedes Mal verschieden, obwohl keine Last oder keine große Verarbeitungszeit der Bausteine vorhanden ist. Die Werte in der Spalte „Durchschnitt pro Zeitintervall“ zeigen, dass der Durchsatz sich nicht stetig erhöht. Man kann daraus schließen, dass der MQTT-Baustein nicht für kontinuierlichen Datenversand geeignet ist. Es ist eher für einen Nachrichtenversand mit einer Pause geeignet.

Gesamtbetrachtung

Betrachtet man die Metriken, kann man sagen dass die Open-Source Datenpipeline unter den Anforderungen und Randbedingungen für die Überwachung eher geeignet ist. Beide Aufbauten haben Vor- und Nachteile. Die Open-Source Datenpipeline hat mehr Möglichkeiten zur Veränderung der Parameter. Dadurch lässt sich die Gesamtlatenz verringern. Jedoch sind die Wartung, der Aufbau und der Aufwand für die Konfiguration Nachteile, wenn man die Datenpipeline verwenden möchte. Es ist ebenfalls erkennbar, dass mehr Ressourcen für Spark keine signifikanten Vorteile mit sich bringen. Apache Spark ist für eine einfache Überwachung ohne Echtzeitanforderungen eine gute Wahl. Für Echtzeit ist die Technologie nicht geeignet, da die Verarbeitung in Batches erfolgt. Das bedeutet, dass Spark eher bei einer Verarbeitung mit großen Datenmengen effizienter ist. Viel mehr sollte man Apache Flink oder Apache Storm für eine Echtzeitverarbeitung nutzen.

Die AWS Datenpipeline zeigt, dass Amazon S3 ebenfalls eher für eine Verarbeitung von großen Datenmengen, die in Batches verpackt sind, eher geeignet ist. AWS Timestream ist eine Möglichkeit, jedoch hat es größere Latenzen als die Open-Source Datenpipeline. Die Nutzung durch die AWS Management Konsole sowie die Dienste als SaaS bietet einige Vorteile und Nachteile. Der Aufwand für den Aufbau ist gering, da alles verwaltet ist. Die Nutzung ist einfach. Des Weiteren kann dadurch schnell eine Lösung angeboten werden, jedoch gibt es geringe Parameteränderungsmöglichkeiten. Der Entwickler hat nicht die vollständige Kontrolle, da eine Autoskalierung stattfindet und Dienstrichtlinien vorhanden sind. Ein weiterer Nachteil ist, dass die Dienste kosten.

In den Durchsatzmessungen ist deutlich geworden, dass der MQTT-Baustein für einen kontinuierlichen Datenstrom nicht geeignet. Dies ist ein Schwachpunkt für beide Aufbauten. Die Implementierung von MQTT im Baustein sollte hierfür bearbeitet werden.

9 Zusammenfassung und Ausblick

Alles in allem zeigt die vorliegende Arbeit, dass für eine Überwachung einer Industrieanlage mit den gegebenen Anforderungen, Randbedingungen und verwendeten Technologien die Open-Spurce Datenpipeline eine bessere Wahl ist. Hierfür sind die Metriken Latenz und Durchsatz für beide Aufbauten verglichen worden. Die Realisierung der AWS Datenpipeline mithilfe „Software-as-a-Service(SaaS)“ und der AWS Management Konsole ist aufgrund den Messungen die schlechtere Wahl. Man kann ebenfalls sagen, dass anhand den gewählten Parameter untersucht wurde, welche Latenzen sich ergeben und welcher Durchsatz sich ergibt. Dabei wurde auch überprüft, ob eine Annährung der Echtzeit-Überwachung überhaupt möglich ist. Es ist auch deutlich geworden, dass beide Aufbauten jeweils Vor- und Nachteile aufzeigen. Für den Einsatz in einer Produktionsumgebung und Aufbau müssen diese Vor- und Nachteile von allen Stakeholdern abgewogen werden.

Der Vergleich hat auch gezeigt, dass vor allem die richtige Wahl der Technolgien wichtig ist. Hierbei kann es durch die falsche Wahl der Technolgien zu großen Latenzen kommen. Vorallem sind Amazon S3 in der AWS Datenpipeline und Apache Spark in der Open Source Datenpipeline nicht geeignet. Zum Beispiel kann Apache Flink oder Apache Storm für die Datenverarbeitung in der Open-Source Datenpipeline für eine schnellere Verarbeitung sorgen. In beiden untersuchten Fällen findet eine Batch Verarbeitung statt, die nicht geeignet ist. Der Austausch der Technologien und Art der Verarbeitung kann als Ausblick für weitere Messungen untersucht werden.

Als weiteren Ausblick und zukünftige Arbeit kann man beide Aufbauten unter Last, mit verschiedenen Nutzlasten untersuchen. Die Messungen sollten auch zukünftig in einer realen Produktionsumgebung getestet werden, um neue Erkenntnisse hinsichtlich der Latenz und Durchsatz gewinnen zu können. Simulierte SPSen oder sogenannte „Mockups“ könnten hierbei helfen.

Eine weitere Bedingung für eine bessere Performance ist, dass auch die interne Implementierung des MQTT-Bausteins umprogrammiert werden muss, um einen kontinuierlichen

Datenstrom zu erzeugen. Aktuell ist die MQTT-Implementierung in der SPS nur für eine Aktivierung in bestimmten Zeitintervallen programmiert worden. Ein weiterer wichtiger Aspekt ist, dass die Sicherheit in dieser Arbeit nicht beachtet wurde. Für den zukünftigen Einsatz der Datenpipeline in einer Produktionsumgebung sollte der Datenaustausch verschlüsselt sein. Grund hierfür, ist dass SPSen auch in einer kritischer Infrastruktur verwendet werden. In einer kritischen Infrastruktur ist die Kommunikation nach außen (hier: Internet) zum Beispiel nicht erwünscht.

Eine Erweiterung für beide Aufbauten wäre die Verwendung der Daten aus der Datenpipeline, um eine prädiktive, präventive Überwachung durchzuführen. Dabei können „Machine-Learning“ Techniken oder eine Teil-Autonomie durch intelligente Agenten für die Überwachung zum Einsatz kommen.

Literaturverzeichnis

[1] : *Amazon Web Services AWS – Server Hosting & Cloud Services.* https://aws.amazon.com/de/

[2] : *Apache Kafka.* https://kafka.apache.org/documentation/

[3] : *Architecture - Amazon Timestream.* https://docs.aws.amazon.com/timestream/latest/developerguide/architecture.html

[4] : *Athena.* https://eu-central-1.console.aws.amazon.com/athena/home?region=eu-central-1#/landing-page

[5] AWS Industrial Applications with Siemens LOGO!

[6] : *AWS IoT Core-Funktionen – Amazon Web Services.* https://aws.amazon.com/de/iot-core/features/

[7] : *Connecting Devices to AWS IoT - AWS IoT Core.* https://docs.aws.amazon.com/iot/latest/developerguide/iot-connect-devices.html

[8] : *Device Communication Protocols - AWS IoT Core.* https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

[9] : *Documentation.* https://grafana.com/docs/

[10] : *IoT Connected Devices by Vertical 2030.* https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/

[11] : *Overview - Spark 3.3.0 Documentation.* https://spark.apache.org/docs/3.3.0/

[12] Predictive Equipment Health for Utilities.

[13] : *Regional Industrial Internet of Things Market Size 2017-2025.* https://www.statista.com/statistics/1102164/global-industrial-internet-of-things-market-size/

[14] : *Welcome to Apache Cassandra's Documentation! | Apache Cassandra Documentation.* https://cassandra.apache.org/doc/latest/

[15] : *What Is Amazon Kinesis Data Firehose? - Amazon Kinesis Data Firehose.* https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html

[16] : *What Is Amazon Managed Grafana? - Amazon Managed Grafana.* https://docs.aws.amazon.com/grafana/latest/userguide/what-is-Amazon-Managed-Service-Grafana.html

[17] : *What Is Amazon S3? - Amazon Simple Storage Service.* https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[18] : *What Is Amazon Timestream? - Amazon Timestream.* https://docs.aws.amazon.com/timestream/latest/developerguide/what-is-timestream.html

[19] : *What Is AWS Glue? - AWS Glue.* https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html

[20] : *What Is AWS IoT? - AWS IoT Core.* https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

[21] Al-Gumaei, Khaled ; Muller, Arthur ; Weskamp, Jan N. ; Longo, Claudio S. ; Pethig, Florian ; Windmann, Stefan: Scalable Analytics Platform for Machine Learning in Smart Production Systems. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Zaragoza, Spain : IEEE, September 2019, S. 1155–1162. – ISBN 978-1-72810-303-7

[22] Armbrust, Michael ; Fox, Armando ; Griffith, Rean ; Joseph, Anthony D. ; Katz, Randy ; Konwinski, Andy ; Lee, Gunho ; Patterson, David ; Rabkin, Ariel ; Stoica, Ion ; Zaharia, Matei: A View of Cloud Computing. In: *Communications of the ACM* 53 (2010), April, Nr. 4, S. 50–58. – ISSN 0001-0782, 1557-7317

[23] Bök, Patrick-Benjamin ; Noack, Andreas ; Müller, Marcel ; Behnke, Daniel: *Computernetze und Internet of Things: technische Grundlagen und Spezialwissen.* Wiesbaden [Heidelberg] : Springer Vieweg, 2020 (Lehrbuch). – ISBN 978-3-658-29408-3

[24] Brown, Mat: *Learning Apache Cassandra: Build an Efficient, Scalable, Fault-Tolerant, and Highly-Available Data Layer into Your Application Using Cassandra.* Birmingham, UK : Packt Publishing, 2015. – ISBN 978-1-78398-921-8

[25] Bruns, Ralf ; Dunkel, Jürgen: *Event-Driven Architecture.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2010 (Xpert.Press). – ISBN 978-3-642-02438-2 978-3-642-02439-9

[26] Buckel, Thomas: Zum Potential von Event-Driven Architecture für komplexe Unternehmensnetzwerke. In: *Multikonferenz Wirtschaftsinformatik 2012 : Tagungsband der MKWI 2012* (2012)

[27] Byzek, Yeva: Best Practices for Developing Apache Kafka® Applications on Confluent Cloud.

[28] Byzek, Yeva: Optimizing Your Apache Kafka® Deployment.

[29] Carpenter, Jeff ; Hewitt, Eben: *Cassandra: The Definitive Guide.* Second edition. Sebastopol, CA : O'Reilly Media, Inc, 2016. – ISBN 978-1-4919-3366-4

[30] Cerquitelli, Tania ; Pagliari, Daniele J. ; Calimera, Andrea ; Bottaccioli, Lorenzo ; Patti, Edoardo ; Acquaviva, Andrea ; Poncino, Massimo: Manufacturing as a Data-Driven Practice: Methodologies, Technologies, and Tools. In: *Proceedings of the IEEE* 109 (2021), April, Nr. 4, S. 399–422. – ISSN 0018-9219, 1558-2256

[31] Chambers, Bill ; Zaharia, Matei: *Spark: The Definitive Guide: Big Data Processing Made Simple.* First edition. Sebastapol, CA : O'Reilly Media, 2018. – ISBN 978-1-4919-1221-8

[32] D'silva, Godson M. ; Khan, Azharuddin ; Gaurav ; Bari, Siddhesh: Real-Time Processing of IoT Events with Historic Data Using Apache Kafka and Apache Spark with Dashing Framework. In: *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT).* Bangalore : IEEE, Mai 2017, S. 1804–1809. – ISBN 978-1-5090-3704-9

[33] Dunkel, Jürgen (Hrsg.) ; Eberhart, Andreas (Hrsg.) ; Fischer, Stefan (Hrsg.) ; Kleiner, Carsten (Hrsg.) ; Koschel, Arne (Hrsg.): *Systemarchitekturen für verteilte Anwendungen: Client-Server, Multi-Tier, SOA, Event-Driven Architectures, P2P, Grid, Web 2.0.* München : Hanser, 2008. – ISBN 978-3-446-41321-4

[34] Ferrari, Paolo ; Sisinni, Emiliano ; Depari, Alessandro ; Flammini, Alessandra ; Rinaldi, Stefano ; Bellagente, Paolo ; Pasetti, Marco: On the Performance of Cloud Services and Databases for Industrial IoT Scalable Applications. In: *Electronics* 9 (2020), September, Nr. 9, S. 1435. – ISSN 2079-9292

[35] Greer, Christopher ; Burns, Martin ; Wollman, David ; Griffor, Edward: Cyber-Physical Systems and Internet of Things / National Institute of Standards and Technology. Gaithersburg, MD, März 2019 (NIST SP 1900-202). – Forschungsbericht. – NIST SP 1900–202 S

[36] Hedtstück, Ulrich: *Complex Event Processing: Verarbeitung von Ereignismustern in Datenströmen.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2020. – ISBN 978-3-662-61575-1 978-3-662-61576-8

[37] Heinrich, Berthold ; Glöckler, Michael ; Linke, Petra: *Grundlagen Automatisierung: Sensorik, Regelung, Steuerung.* Aufl. 2014. Wiesbaden : Springer Fachmedien Wiesbaden GmbH, 2015. – ISBN 978-3-658-05961-3

[38] Helu, Moneer ; Sprock, Timothy ; Hartenstine, Daniel ; Venketesh, Rishabh ; Sobel, William: Scalable Data Pipeline Architecture to Support the Industrial Internet of Things. In: *CIRP Annals* 69 (2020), Nr. 1, S. 385–388. – ISSN 00078506

[39] Herterich, Matthias M. ; Uebernickel, Falk ; Brenner, Walter: *Industrielle Dienstleistungen 4.0.* Wiesbaden : Springer Fachmedien Wiesbaden, 2016 (Essentials). – ISBN 978-3-658-13910-0 978-3-658-13911-7

[40] Hüning, Felix: *Embedded Systems für IoT.* Berlin [Heidelberg] : Springer Vieweg, 2019. – ISBN 978-3-662-57900-8

[41] Illa, Prasanna K. ; Padhi, Nikhil: Practical Guide to Smart Factory Transition Using IoT, Big Data and Edge Analytics. In: *IEEE Access* 6 (2018), S. 55162–55170. – ISSN 2169-3536

[42] John, Karl-Heinz ; Tiegelkamp, Michael: *SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen.* 4., neubearb. Aufl. Berlin Heidelberg : Springer, 2009 (VDI-Buch). – ISBN 978-3-642-00268-7

[43] Kaestner, Florian ; Kuschnerus, Dirk ; Spiegel, Christoph ; Janssen, Benedikt ; Huebner, Michael: Design of an Efficient Communication Architecture for Cyber-Physical Production Systems. In: *2018 IEEE 14th International Conference*

on Automation Science and Engineering (CASE). Munich, Germany : IEEE, August 2018, S. 829–835. – ISBN 978-1-5386-3593-3

[44] Karau, Holden ; Konwinski, Andy ; Wendell, Patrick ; Zaharia, Matei: *Learning Spark*. First edition. Beijing ; Sebastopol : O'Reilly, 2015. – ISBN 978-1-4493-5862-4

[45] Koutanov, Emil: *Effective Kafka: A Hands-on Guide to Building Robust and Scalable Event-Driven Applications with Code Examples in Java*. Victoria, British Columbia : Leanpub, 2020. – ISBN 9798628558515

[46] Langmann, Reinhard: *Vernetzte Systeme für die Automatisierung 4.0: Bussysteme - Industrial Ethernet - Mobile Kommunikation - Cyber-Physical Systems*. München : Hanser, 2021. – ISBN 978-3-446-46939-6

[47] Leang, Bunrong ; Ean, Sokchomrern ; Ryu, Ga-Ae ; Yoo, Kwan-Hee: Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Environment. In: *Sensors* 19 (2019), Januar, Nr. 1, S. 134. – ISSN 1424-8220

[48] Lee, Edward: The Past, Present and Future of Cyber-Physical Systems: A Focus on Models. In: *Sensors* 15 (2015), Februar, Nr. 3, S. 4837–4869. – ISSN 1424-8220

[49] Martin, Cristian ; Garrido, Daniel ; Diaz, Manuel ; Rubio, Bartolome: From the Edge to the Cloud: Enabling Reliable IoT Applications. In: *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*. Istanbul, Turkey : IEEE, August 2019, S. 17–22. – ISBN 978-1-72812-888-7

[50] Marwedel, Peter: *Eingebettete Systeme: Grundlagen eingebetteter Systeme in cyber-physikalischen Systemen*. 2. Auflage. Wiesbaden [Heidelberg] : Springer Vieweg, 2021 (Lehrbuch). – ISBN 978-3-658-33436-9

[51] Melzer, Ingo ; Eberhard, Sebastian: *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*. 4. Aufl. Heidelberg : Spektrum Akad. Verl, 2010. – ISBN 978-3-8274-2549-2

[52] Michelson, Brenda: Event-Driven Architecture Overview / Patricia Seybold Group. Boston, MA, Februar 2006 (681). – Forschungsbericht. – 681 S

[53] Nakagawa, Elisa Y. ; Antonino, Pablo O. ; Schnicke, Frank ; Capilla, Rafael ; Kuhn, Thomas ; Liggesmeyer, Peter: Industry 4.0 Reference Architectures: State of the Art and Future Trends. In: *Computers & Industrial Engineering* 156 (2021), Juni, S. 107241. – ISSN 03608352

[54] Nguyen-Hoang, Phuc ; Vo-Tan, Phuoc: Development An Open-Source Industrial IoT Gateway. In: *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*. Ho Chi Minh City, Vietnam : IEEE, September 2019, S. 201–204. – ISBN 978-1-72815-009-3

[55] Osorio, Alfonso ; Calle, Maria ; Soto, Jose D. ; Candelo-Becerra, John E.: Routing in LoRaWAN: Overview and Challenges. In: *IEEE Communications Magazine* 58 (2020), Juni, Nr. 6, S. 72–76. – ISSN 0163-6804, 1558-1896

[56] Papp, Stefan ; Weidinger, Wolfgang ; Meir-Huber, Mario ; Ortner, Bernhard ; Langs, Georg ; Wazir, Rania: *Handbuch Data Science: mit Datenanalyse und Machine Learning Wert aus Daten generieren.* München : Hanser, 2019. – ISBN 978-3-446-45710-2

[57] Pedone, G. ; Mezgár, I.: Model Similarity Evidence and Interoperability Affinity in Cloud-Ready Industry 4.0 Technologies. In: *Computers in Industry* 100 (2018), September, S. 278–286. – ISSN 01663615

[58] Pierleoni, Paola ; Concetti, Roberto ; Belli, Alberto ; Palma, Lorenzo: Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison. In: *IEEE Access* 8 (2020), S. 5455–5470. – ISSN 2169-3536

[59] Plenk, Valentin: *Grundlagen der Automatisierungstechnik kompakt.* Wiesbaden : Springer Vieweg, 2019 (Lehrbuch). – ISBN 978-3-658-24469-9 978-3-658-24468-2

[60] Rupprecht, Bernhard ; Trunzer, Emanuel ; Konig, Simone ; Vogel-Heuser, Birgit: Concepts for Retrofitting Industrial Programmable Logic Controllers for Industrie 4.0 Scenarios. In: *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*. Valencia, Spain : IEEE, März 2021, S. 1034–1041. – ISBN 978-1-72815-730-6

[61] Sahal, Radhya ; Breslin, John G. ; Ali, Muhammad I.: Big Data and Stream Processing Platforms for Industry 4.0 Requirements Mapping for a Predictive Maintenance Use Case. In: *Journal of Manufacturing Systems* 54 (2020), Januar, S. 138–151. – ISSN 02786125

[62] Salituro, Eric: *Learn Grafana 7.0: A Beginner's Guide to Getting Well Versed in Analytics, Interactive Dashboards, and Monitoring.* Birmingham Mumbai : Packt, 2020. – ISBN 978-1-83882-658-1

[63] Sari, Alparslan ; Lekidis, Alexios ; Butun, Ismail: Industrial Networks and IIoT: Now and Future Trends. In: Butun, Ismail (Hrsg.): *Industrial IoT*. Cham : Springer International Publishing, 2020, S. 3–55. – ISBN 978-3-030-42499-2 978-3-030-42500-5

[64] Schlick, Jochen ; Stephan, Peter ; Loskyll, Matthias ; Lappe, Dennis: Industrie 4.0 in der praktischen Anwendung. In: Vogel-Heuser, Birgit (Hrsg.) ; Bauernhansl, Thomas (Hrsg.) ; ten Hompel, Michael (Hrsg.): *Handbuch Industrie 4.0 Bd.2*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2017, S. 3–29. – ISBN 978-3-662-53247-8 978-3-662-53248-5

[65] Steen, Maarten van ; Tanenbaum, Andrew S.: *Distributed Systems*. Third edition, Version 3.01. Erscheinungsort nicht ermittelbar : Maarten van Steen, 2017. – ISBN 978-1-5430-5738-6

[66] Taha, Walid M. ; Taha, Abd-Elhamid M. ; Thunberg, Johan: *Cyber-Physical Systems: A Model-Based Approach*. Cham : Springer, 2021. – ISBN 978-3-030-36070-2 978-3-030-36073-3

[67] Veneri, Giacomo ; Capasso, Antonio: *Hands-on Industrial Internet of Things: Create a Powerful Industrial IoT Infrastructure Using Industry 4.0*. Birmingham : Packt Publishing, 2018. – ISBN 978-1-78953-722-2

[68] Yi, Shanhe ; Li, Cheng ; Li, Qun: A Survey of Fog Computing: Concepts, Applications and Issues. In: *Proceedings of the 2015 Workshop on Mobile Big Data*. Hangzhou China : ACM, Juni 2015, S. 37–42. – ISBN 978-1-4503-3524-9

A Anhang

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original