Bachelorarbeit

Entwurf und Implementierung einer Daten-Pipeline zur Aufbereitung von IoT Daten in der Fertigung

Design and Implementation of a Data Pipeline for Processing IoT Data in Manufacturing

Tobias **Weiß**

Industrie-4.0-Informatik
Fakultät Elektrotechnik, Medien und Informatik
Ostbayerische Technische Hochschule Amberg-Weiden

Bachelorarbeit

Entwurf und Implementierung einer Daten-Pipeline zur Aufbereitung von IoT Daten in der Fertigung

Design and Implementation of a Data Pipeline for Processing IoT Data in Manufacturing

Tobias **Weiß**

Industrie-4.0-Informatik
Fakultät Elektrotechnik, Medien und Informatik
Ostbayerische Technische Hochschule Amberg-Weiden

1. Prüfer:	Prof. Dr.-Ing. Christoph P. Neumann
2. Prüfer:	Prof. Wolfgang Schindler
Externer Betreuer:	Dr. Daniel Pohl
Ausgabetag:	04. Oktober 2022
Abgabetag:	03. März 2023

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

Name und Vorname
des Studenten: **Weiß, Tobias**

Studiengang: **Industrie-4.0-Informatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Entwurf und Implementierung einer Daten-Pipeline zur Aufbereitung von IoT Daten in der Fertigung

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 03. März 2023

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Bachelorarbeit Zusammenfassung

Student:	**Weiß, Tobias**
Studiengang:	Industrie-4.0-Informatik
Aufgabensteller, Professor:	Prof. Dr.-Ing. Christoph P. Neumann
Durchgeführt in Firma:	up2parts GmbH
Betreuer in Firma:	Dr. Daniel Pohl
Ausgabedatum: 04. Oktober 2022	Abgabedatum: 03. März 2023

Titel:

Entwurf und Implementierung einer Daten-Pipeline zur Aufbereitung von IoT Daten in der Fertigung

Zusammenfassung

Ziel dieser Arbeit ist es, ein Konzept für eine Daten-Pipeline zu entwickeln, mit der IoT-Daten aus der Fertigung verarbeitet und in ein bestehendes System integriert werden können. Für dieses Konzept wurde zunächst eine Literaturrecherche im Bereich IoT sowie Big Data durchgeführt. Aufbauend auf den Erkenntnissen der Literaturrecherche wurde ein eigenes Konzept erstellt und dieses anhand eines Prototyps validiert. Exemplarisch wurde eine Pipeline implementiert, die aus dem Datenstrom mehrerer CNC-Maschinen die Bearbeitungszeiten einzelner NC-Programme extrahiert. Diese Bearbeitungszeiten werden anschließend mit Metadaten aus der Fertigungsplanung verknüpft und gespeichert.

Schlüsselwörter

Industrie 4.0, IIoT, IoT, Daten-Pipeline, Pipeline, Big Data, Smart Factory

Title:

Design and Implementation of a Data Pipeline for Processing IoT Data in Manufacturing

Abstract

This thesis aims to develop a concept for a data pipeline with which IoT data from manufacturing can be processed and integrated into an existing system. For this concept, a literature research in the field of IoT as well as Big Data was initially conducted. Based on the findings of the literature research, an own concept was created and validated using a prototype. As an example, a pipeline was implemented that extracts the processing times of individual NC programs from the data stream of several CNC machines. These machining times are then linked to metadata from production planning and stored.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Hinführung . . . 1
1.2 Relevanz . . . 2
1.3 Anwendungsfälle . . . 3
1.4 Beschreibung der Problemstellung . . . 4

2 Methodik **6**

3 Stand der Technik **7**
3.1 Daten-Pipeline Architekturen . . . 7
3.1.1 Batch-Architektur . . . 7
3.1.2 λ-Architektur . . . 9
3.1.3 κ-Architektur . . . 13
3.2 Technische Grundlagen . . . 14
3.2.1 MQTT . . . 14
3.2.2 AMQP . . . 16
3.2.3 Apache Kafka . . . 21
3.3 Machine Data Connector . . . 22
3.4 Verwandte Arbeiten . . . 23

4 Konzeption **25**
4.1 Ausgangslage . . . 25
4.2 Beschreibung des Datenstroms der MDCs . . . 26
4.3 Analyse des Datenstroms . . . 29
4.4 Architektur der Daten-Pipeline . . . 33
4.5 Klassen zum Verarbeiten der Datenströme . . . 36
4.5.1 Die Klasse MdcStreamProcessor . . . 37
4.5.2 Die Klasse Pipeline . . . 38
4.5.3 Die Klasse RuntimeExtractor . . . 39
4.5.4 Die Klasse ManufacturingMachine . . . 40
4.5.5 Die Klasse Enricher . . . 42
4.5.6 Die Klasse Publisher . . . 44

5 Ergebnisse **46**
5.1 Test mit einer realen Zerspannung . . . 46
5.2 Durchsatz der Daten-Pipeline . . . 48
5.3 Auslastung . . . 50

6 Ausblick und Diskussion **52**
6.1 Bewertung der Architektur 52
6.2 Offene Punkte 53
6.3 Weitere Ideen 53

7 Zusammenfassung **55**

8 Literatur **56**

Abkürzungen

Abkürzung	Bedeutung
AMQP	Advanced Message Queuing Protocol
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CNC	Computerized Numerical Control
HTTP	Hypertext Transfer Protocol
IIC	Industry IoT Consortium
IIRA	Industrial Internet Reference Architecture
IoT	Internet der Dinge, englisch Internet of Things
KI	Künstliche Intelligenz
M2M	Machine-to-Machine
MDC	Machine Data Connector
MQTT	ursprünglich für Message Queuing Telemetry Transport
NC-Programm	Programm für eine CNC-Maschine
OASIS	Organization for the Advancement of Structured Information Standards
OPC UA	Open Platform Communications Unified Architecture
QoS	Quality of Service
RAMI4.0	Referenzarchitekturmodell Industrie 4.0
REST	Representational State Transfer
SQL	Structured Query Language
TCP	Transmission Control Protocol
UC	Use Case
VDI	Verein Deutscher Ingenieure
XML	Extensible Markup Language

1. Einleitung

Dieses Kapitel dient der Einführung in die Thematik. Dazu werden zunächst der Hintergrund und die Relevanz der Arbeit erläutert. Anschließend wird die Problemstellung beschrieben, die im Rahmen dieser Arbeit bearbeitet wird.

1.1 Hinführung

Up2parts bietet Softwarelösungen für die Digitalisierung der Fertigung. Der Schwerpunkt liegt dabei auf der Fertigungsplanung. Mit Hilfe von Künstlicher Intelligenz (kurz KI) können aus 3D-Modellen automatisiert Fertigungspläne erstellt werden [1]. Die 3D-Modelle liegen dabei in Form von CAD-Dateien (aus dem Englischen für Computer Aided Design [2]) vor. Für die Fertigungsplanung ist insbesondere die Abschätzung der Bearbeitungszeiten für einzelne Arbeitsschritte von Interesse. Auch für die Abschätzung dieser Bearbeitungszeiten bietet up2parts KI-basierte Softwarelösungen an [1].

Unter dem Begriff Internet der Dinge (englisch Internet of Things, kurz IoT) versteht man die Idee, dass neben Computern und mobilen Endgeräten auch beliebige physische Objekte über Sensoren und Aktoren in das Internet eingebunden werden. Dadurch können auch diese zu Konsumenten oder Anbietern von digitalen Diensten werden [3]. Im hier gewählten Beispiel werden also die Fertigungsmaschinen (CNC-Maschinen, kurz für Computerized Numerical Control) einer Produktion zu Anbietern ihrer Daten. Das IoT bietet in diesem Zusammenhang die Möglichkeit, automatisiertes Feedback aus der Fertigung zu erhalten. So können Abweichungen zwischen der Fertigungsplanung und dem tatsächlichen Fertigungsprozess erfasst werden. Auf dieser Basis können die zuvor beschriebenen KI-Modelle validiert und optimiert werden.

Um diese Verbindung zwischen Fertigungsplanung und tatsächlicher Fertigung herzustellen, müssen die gesammelten Maschinendaten von den Maschinen in die Cloud transportiert werden. Dort können diese Daten weiterverarbeitet, mit zusätzlichen Informationen angereichert und schließlich genutzt werden. Um diese Übertragung der Maschinendaten in die Cloud zu gewährleisten, wird im Rahmen dieser Bachelorarbeit das Konzept einer Daten-Pipeline entwickelt. Diese Pipeline soll die gesammelten

Daten zuverlässig in die Cloud transportieren, aufbereiten, mit zusätzlichen Metadaten anreichern, persistent speichern und schließlich anderen Diensten zur Verfügung stellen. In den folgenden Abschnitten werden die Anforderungen und Ziele näher erläutert.

1.2 Relevanz

Neben dem bereits erwähnten Szenario zur Verbesserung der Fertigungsplanung gibt es noch viele weitere Bereiche, in denen das IoT von Nutzen sein kann. Diese werden im Folgenden kurz vorgestellt.

Ein weiteres Beispiel aus der Fertigung ist die Überwachung von Maschinen in einer Produktionsanlage. Durch den Einsatz vernetzter Sensoren und die Analyse der Daten können Muster erkannt werden, die darauf hindeuten, dass eine Maschine bald ausfallen könnte. Diese Daten können genutzt werden, um mögliche Ausfälle frühzeitig zu erkennen und durch gezielte Wartungsarbeiten zu verhindern. Dadurch können die Kosten für Ausfallzeiten gesenkt und die Effizienz der Produktion gesteigert werden. [4–6]

Der Einsatz von vernetzten Sensoren in der Landwirtschaft ist ein weiteres Beispiel. Durch die Überwachung verschiedener Parameter wie Bodenfeuchte oder Temperatur können Landwirte bessere Entscheidungen über die Bewässerung und den Einsatz von Düngemitteln treffen. Dies kann zu höheren Erträgen und einer besseren Ressourcennutzung führen. [7]

IoT-Daten sind auch für die Entwicklung neuer Produkte von großer Bedeutung. Mit Hilfe von Daten aus bestehenden Produkten können beispielsweise Rückschlüsse auf das Nutzungsverhalten der Kunden gezogen werden. Dadurch können Kundenbedürfnisse besser erfasst werden. Für die Entwicklung neuer Produkte können diese Daten genutzt werden, so dass die Produkte den Bedürfnissen der Kunden besser entsprechen. [8]

Insgesamt haben IoT-Daten das Potenzial, eine Vielzahl von Branchen zu revolutionieren. Sie können Unternehmen dabei unterstützen, bessere Entscheidungen zu treffen, effizienter zu arbeiten und bessere Produkte anzubieten. Mit der Weiterentwicklung und dem vermehrten Einsatz von IoT-Technologien wird die Bedeutung von IoT-Daten weiter zunehmen. Unternehmen, die in der Lage sind, diese Daten effektiv zu nutzen,

werden einen Wettbewerbsvorteil haben [6, 9]. Um diese Daten effektiv nutzen zu können, ist in allen genannten Beispielen eine zuverlässige Weiterverarbeitung dieser Daten notwendig. Um dies zu gewährleisten, wird im Rahmen dieser Bachelorarbeit ein Konzept vorgestellt, wie solche Daten aufbereitet und in ein bestehendes System integriert werden können. Als Beispiel werden für diese Bachelorarbeit Daten von CNC-Maschinen verwendet. Um diese zu erfassen wird der sogenannte Machine Data Connector (kurz MDC) der Frima DMG Mori verwendet. Dieser wird in Abschnitt 3.3 näher beschrieben.

1.3 Anwendungsfälle

Im Folgenden werden die Anwendungsfälle (engl. Use Case, kurz UC), die durch die zu entwickelnde Daten-Pipeline abgedeckt werden sollen, grob dargestellt. Eine detaillierte Beschreibung der einzelnen Anwendungsfälle findet sich im Anhang.

Anwendungsfall 1 (UC-1): Bearbeitungszeit extrahieren
Dieser Anwendungsfall beschreibt, wie das System die Bearbeitungszeit für ein konkretes NC-Programm (Programm für eine CNC-Maschine) aus dem Datenstrom des MDCs extrahiert. Der Anwendungsfall beginnt, sobald ein Mitarbeiter ein NC-Programm auf einer CNC-Maschine startet, und endet, sobald die Bearbeitung abgeschlossen ist. Dabei ist nur die Zeit interessant in der die Maschine tatsächlich ein Programm bearbeitet, Zeiten in denen die Maschine pausiert war, werden abgezogen.

Anwendungsfall 2 (UC-2): Maschine wechselt den Modus
Dieser Anwendungsfall beschreibt das Systemverhalten, wenn ein Mitarbeiter den Modus einer Maschine wechselt. Da in diesem Beispiel nur die Bearbeitungszeiten der NC-Programme relevant sind, muss dem System mitgeteilt werden, wenn sich die Maschine im manuellen Modus befindet. In diesem Fall werden keine Bearbeitungszeiten extrahiert.

Anwendungsfall 3 (UC-3): Neues NC-Programm geladen
Das System speichert den Namen des aktuell geladenen NC-Programms, um eine Zuordnung zu den extrahierten Bearbeitungszeiten zu ermöglichen. Über den Namen des NC-Programms ist es möglich, einer Kalkulation im bestehenden System eine tatsächliche Bearbeitungszeit zuzuordnen.

Anwendungsfall 4 (UC-4): Bearbeitungszeit mit Metadaten anreichern und speichern
Sobald eine Bearbeitungszeit für ein NC-Programm extrahiert wurde, wird diese mit zusätzlichen Metadaten aus der Fertigungsplanung angereichert und diese Informationen persistent gespeichert. Metadaten sind in diesem Zusammenhang die zugrundeliegende Kalkulation, das zugrundeliegende CAD-Modell, die Maschine auf der die Bearbeitung stattfand sowie die Nummer der Aufspannung.

Anwendungsfall 5 (UC-5): Ergebnisse abfragen
Dieser Anwendungsfall beschreibt, wie ein anderer Dienst (z.B. zum Trainieren eines KI-Modells) die ermittelten Informationen von diesem Dienst abfragen kann. Dabei soll es möglich sein, nach dem Namen des NC-Programms, der zugrundeliegenden Kalkulation und dem zugrundeliegenden CAD-Modell zu filtern.

1.4 Beschreibung der Problemstellung

Abbildung 1.1 zeigt die Aufgaben der zu entwickelnden Daten-Pipeline. Die Daten-Pipeline erhält als Eingabe einen Datenstrom, der die Ereignisse der CNC-Maschinen enthält. Daraus extrahiert sie im ersten Schritt den Namen des NC-Programms und die Bearbeitungszeit für dieses Programm. Im nächsten Schritt werden diese Informationen mit Metadaten aus der Produktionsplanung angereichert. Metadaten sind in diesem Zusammenhang, wie im Anwendungsfall UC-4 beschrieben, die Information, auf welcher CNC-Maschine die Bearbeitung stattgefunden hat, welche Kalkulation dieser Bearbeitung zugrunde liegt, welches CAD-Modell der Kalkulation zugrunde liegt und, falls ein Bauteil in mehreren Aufspannungen gefertigt wird, um welche Aufspannung es sich handelt.

Im Rahmen dieser Bachelorarbeit soll ein Konzept für eine Daten-Pipeline entwickelt werden, welche die oben beschriebenen Anwendungsfälle abdeckt. Des Weiteren soll gezeigt werden, wie eine solche Pipeline in ein bestehendes System von Microservices integriert werden kann. Um dieses Konzept zu validieren, wird im Rahmen dieser Bachelorarbeit ein Softwareprototyp entwickelt, der die grundlegenden Funktionen dieser Daten-Pipeline implementiert.

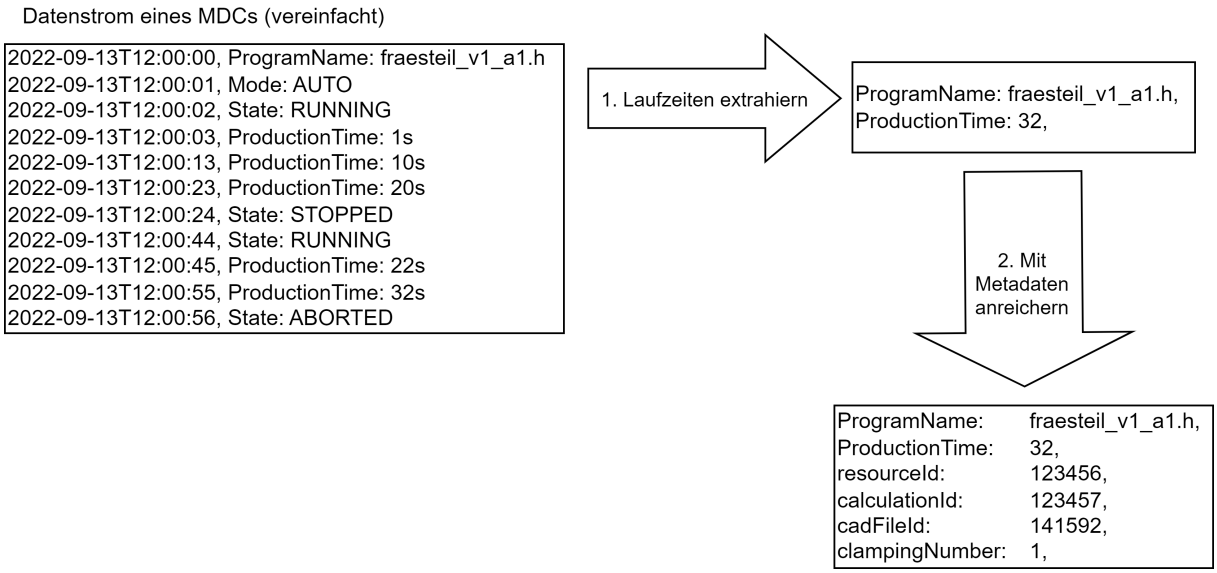

Abbildung 1.1: Schematischer Abfolge der einzelnen Schritte in der Daten-Pipeline (Quelle: eigene Darstellung)

2. Methodik

In diesem Kapitel wird die Struktur dieser Arbeit beschrieben. Außerdem wird die Vorgehensweise bei der Erstellung des Softwareprototyps beschrieben.

Diese Arbeit beschreibt zunächst die Motivation sowie die Relevanz des Themas. Anschließend wird ein konkretes Beispiel vorgestellt, für das im Rahmen dieser Arbeit ein Softwareprototyp entwickelt werden soll. Nach der Definition der Anwendungsfälle für diesen Prototypen folgt ein Kapitel über technische Grundlagen und verwandte Arbeiten. Anschließend wird die Konzeption und Implementierung des Softwareprototyps am konkreten Beispiel gezeigt. Kapitel fünf beschreibt die Ergebnisse dieser Arbeit. Abschließend werden die Ergebnisse diskutiert, offene Punkte benannt und weitere Ideen zur Verbesserung vorgestellt.

Die Konzeption und Umsetzung des Softwareprototyps gliedert sich wie folgt. Zunächst wird die Ausgangssituation beschrieben. Darauf aufbauend wird ein Konzept erstellt, wie die anfallenden Daten verarbeitet werden sollen. Für die Umsetzung dieses Konzepts werden geeignete Technologien ausgewählt. Abschließend wird für ausgewählte Teile des Systems eine konkrete Umsetzung gezeigt.

3. Stand der Technik

In diesem Kapitel soll der aktuelle Stand der Technik dargestellt werden. Darüber hinaus sollen die verschiedenen Technologien und Architekturen im Bereich der Datenverarbeitung kurz vorgestellt werden. Zunächst sollen die verschiedenen Architekturen von Daten-Pipelines vorgestellt werden.

3.1 Daten-Pipeline Architekturen

Im Folgenden sollen verschiedene Architekturen für Daten-Pipelines vorgestellt werden. Außerdem soll auf deren Unterschiede, sowie Vor- und Nachteile eingegangen werden.

3.1.1 Batch-Architektur

Die historisch älteste Architektur für die automatisierte Datenverarbeitung ist die Stapelverarbeitung (englisch batch). Soll, wie im vorliegenden Fall, ein Datenstrom auf diese Weise verarbeitet werden, müssen zunächst alle eintreffenden Ereignisse persistent gespeichert werden. In regelmäßigen Abständen wird dann die eigentliche Verarbeitung gestartet. Dies geschieht in der Regel nachts oder am Wochenende, um die Systembelastung zu Spitzenzeiten möglichst gering zu halten. Die Ergebnisse der Verarbeitung werden dann typischerweise in einer Datenbank gespeichert, um sie anderen Diensten zur Verfügung zu stellen. Diese Funktionsweise ist in Abbildung 3.1 dargestellt. [10, 11]

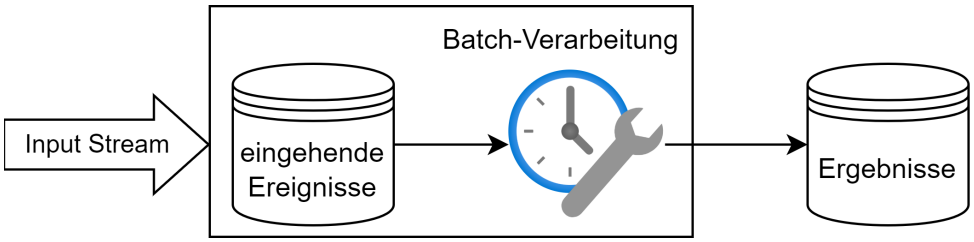

Abbildung 3.1: Aufbau einer Batch-Architektur (Quelle: eigene Darstellung nach [11])

Hadoop wird häufig für die Implementierung von Batch-Architekturen verwendet. Hadoop ist ein verteiltes System, das es ermöglicht, das sogenannte MapReduce-Verfahren effizient auf mehrere Rechner zu verteilen [12]. MapReduce verarbeitet eine Menge von Schlüssel-Wert-Paaren zu einer neuen Menge von Schlüssel-Wert-Paaren

[13]. Dabei werden die folgenden drei Phasen durchlaufen, die auch in Abbildung 3.2 dargestellt sind:

1. **Map Phase:** `map(input_key, input_value) –> list(output_key, intermediate_value)`
   Eine Funktion `map` bildet ein Schlüssel-Wert Paar `(input_key, input_value)` auf ein neues Schlüssel-Wert Paar `(output_key, intermediate_value)` mit Zwischenergebnissen ab [13]. Die Funktion `map` kann dabei vom Nutzer definiert werden [13]. Hadoop ermöglicht es diese Funktion parallel auf mehreren Rechnern eines Clusters mit jeweils einem Teil des Datensatzes auszuführen [12].
2. **Shuffle Phase:**
   Die Zwischenergebnisse werden innerhalb des Clusters neu verteilt, so dass alle Zwischenergebnisse mit dem gleichen Schlüssel auf dem gleichen Rechner bearbeitet werden [13]. Diese Funktion wird vom zugrundeliegenden System übernommen und muss nicht vom Anwender definiert werden.
3. **Reduce Phase:** `reduce(output_key, list(intermediate_value)) –> list(out_value)`
   Eine Funktion `reduce` kombiniert alle Werte `list (intermediate_values)` eines Schlüssels `output_key` und gibt eine Menge von Ergebniswerten zurück. In den meisten Fällen gibt die Reduce Funktion dabei nur einen Wert pro Schlüssel zurück, theoretisch können aber beliebig viele Werte zurückgegeben werden. [13]

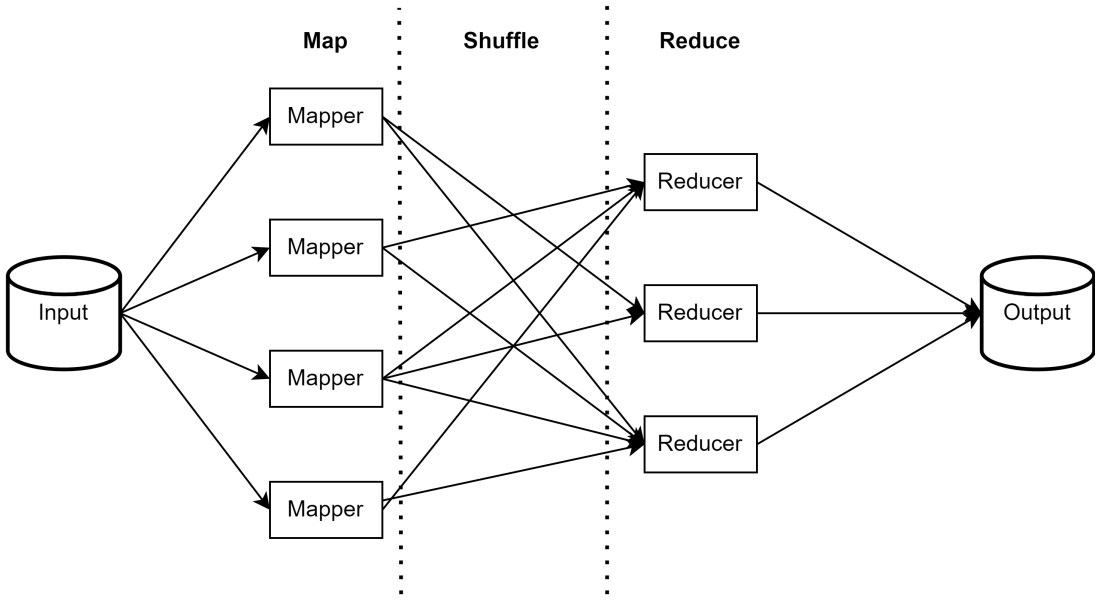

Abbildung 3.2: Die Phasen Map, Shuffle und Reduce (Quelle: eigene Darstellung nach [14])

In dem hier vorgestellten Anwendungsfall würde die Datenverarbeitung mit Hilfe einer Batch-Architektur wie folgt aussehen. Zunächst werden alle Ereignisse aus der Fertigung aufgezeichnet. Jede Nacht wird dann ein Batch-Job gestartet, der aus allen bis dahin gesammelten Daten die Laufzeiten der einzelnen NC-Programme extrahiert und diese Ergebnisse in einer Datenbank ablegt.

Ein Vorteil dieser Architektur ist, dass es keine Probleme gibt, wenn Nachrichten in der falschen Reihenfolge eintreffen. Da alle Daten zunächst gespeichert werden, können sie vor der Verarbeitung nach dem angegebenen Zeitstempel sortiert werden. Da alle Ereignisse zunächst gespeichert werden, ist auch ein Absturz während der Verarbeitung unkritisch, der Batch-Job kann einfach mit den gespeicherten Daten neu gestartet werden. Dies macht die Architektur sehr robust gegenüber Fehlern.

Ein Nachteil dieser Architektur ist, dass die berechneten Daten immer erst mit einer gewissen Zeitverzögerung zur Verfügung stehen. Um dieses Problem zu verringern, kann natürlich das Zeitintervall zwischen den Batch-Jobs verkürzt werden. Damit wird aber der Vorteil der Nutzung ungenutzter Ressourcen (nachts oder am Wochenende) zunichte gemacht.

3.1.2 λ-Architektur

Um die im vorherigen Kapitel 3.1.1 beschriebenen Nachteile der Batch-Architektur zu beheben, stellte Nathan Marz 2011 in seinem Blog-Post mit dem Titel *How to beat the CAP theorem*, die sogenannte λ-Architektur vor [15].

Der größte Nachteil der Batch-Architektur besteht darin, dass die Datenverarbeitung nicht in Echtzeit erfolgen kann. Dabei wird zwischen harter und weicher Echtzeit unterschieden. Unter harter Echtzeit versteht man die Anforderung an ein System, innerhalb einer vorgegebenen Zeit ein Ergebnis zu berechnen. Liegt das Ergebnis nicht innerhalb dieser Zeit vor, wird dies als Fehler gewertet. Unter weicher Echtzeit versteht man dagegen, dass alle Eingaben schnell genug verarbeitet werden. Eine Überschreitung der Zeitvorgabe ist hier unkritisch. Die Zeitvorgaben sind hier als Richtlinie zu verstehen [16]. Wenn im Folgenden von Echtzeit gesprochen wird, ist immer weiche Echtzeit gemeint.

Um auch Echtzeitdaten zur Verfügung zu haben, baut die λ-Architektur auf der Batch-Architektur auf und ergänzt diese um eine weitere Schicht. Diese Aufteilung ist in Abbildung 3.3 dargestellt. Die Funktionsweise sowie die Verantwortlichkeiten der einzelnen Schichten werden im Folgenden beschrieben. [17]

Die erste Schicht ist die Batch-Schicht. Die Funktionsweise dieser Schicht ist analog zur Batch-Architektur. Auch hier werden alle eingehenden Daten zunächst persistent gespeichert und dann in regelmäßigen Abständen von einem Batch-Job verarbeitet. Dabei ist zu beachten, dass der Batch-Job bei jeder Ausführung den gesamten Datensatz verarbeitet. Die Ergebnisse werden in der sogenannten Batch View zur Verfügung gestellt. Wie bereits in Abschnitt 3.1.1 beschrieben, führt dies zu sehr zuverlässigen Ergebnissen, die jedoch nicht aktuell sind. [17]

Um dieses Problem zu lösen, führt die λ-Architektur die sogenannte Speed-Schicht ein. Diese erhält wie die Batch-Schicht alle neu eintreffenden Ereignisse als Eingabe. Im Gegensatz zur Batch-Schicht führt diese Schicht jedoch keine Operation auf allen Daten aus, sondern verwendet das aktuelle Event nur, um das letzte Ergebnis zu aktualisieren. Auf diese Weise ist es möglich, die Daten in Echtzeit zu verarbeiten. Die Ergebnisse werden dann in der sogenannten Realtime View zur Verfügung gestellt. [17]

Daneben gibt es noch eine dritte Schicht, die sogenannte Serving-Schicht. Nathan Marz und James Warren beschreiben diese Schicht in ihrem *Buch Big Data: Principles and best practices of scalable real-time data systems* als eine Datenbank, die die Ergebnisse der Batch-Schicht verwaltet. Dabei werden die Daten nach jedem Durchlauf der Batch-Schicht ersetzt. Die Serving-Schicht stellt dem Benutzer nur die aktuellen Ergebnisse der Batch-Schicht zur Verfügung. Die korrekte Zusammenführung der Ergebnisse aus der Speed- und der Batch-Schicht liegt in der Verantwortung des Anwenders [17]. Es gibt jedoch auch Erweiterungen der λ-Architektur, bei denen die Serving-Schicht auch die Aufgabe der Zusammenführung der Ergebnisse aus Batch- und Speed-Schicht übernimmt (siehe Abbildung 3.3). In diesem Fall kann der Benutzer die aktuellen Ergebnisse direkt von der Serving-Schicht anfordern und muss sie nicht selbst kombinieren [18].

Kurz lässt sich die Funktionsweise der λ-Architektur auf folgende drei Gleichungen zusammenfassen:

- $batch\ view = funktion(alle\ Daten)$
- $realtime\ view = funktion(realtime\ view, neues\ Event)$
- $ergebnisse = funktion(reatlitem\ view, batch\ view)$ [17]

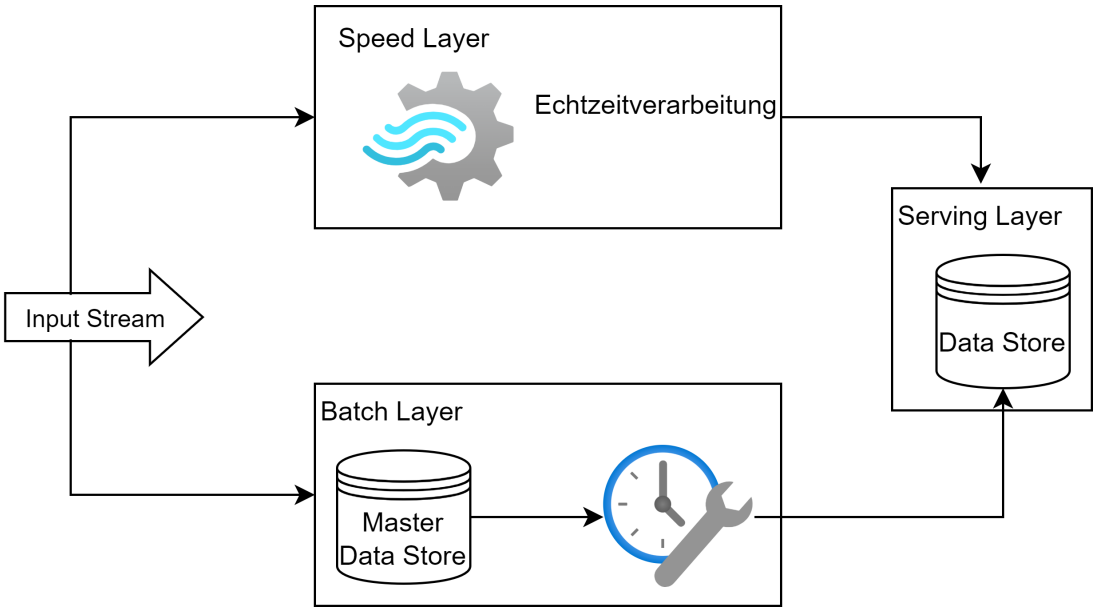

Abbildung 3.3: Aufbau einer λ-Architektur (Quelle: eigene Darstellung nach [18])

Für die beiden Verarbeitungsschichten der λ-Architektur werden typischerweise unterschiedliche Technologien bei der Implementierung verwendet. Wie bereits in Abschnitt 3.1.1 zur Batch-Architektur erwähnt, wird für die Batch-Schicht der λ-Architektur häufig Hadoop eingesetzt. In der Speed-Schicht werden häufig spezielle Streaming-Technologien wie Apache Storm, Apache Spark Streaming oder Apache S4 eingesetzt. [15, 18]

Ein Vorteil dieser Architektur ist, dass die Ergebnisse der Speed-Schicht nicht mehr benötigt werden, sobald die entsprechenden Daten von der Batch-Schicht verarbeitet wurden. Das bedeutet, dass nach jedem Durchlauf der Batch-Schicht die Speed-Schicht mit exakten Werten aktualisiert werden kann. Dadurch können Fehler, die bei der Echtzeitverarbeitung auftreten, korrigiert werden. [17]

Die Implementierung einer inkrementellen Lösung für die Speed-Schicht ist oft deutlich komplexer als eine Batch-Lösung. Dadurch, dass alle Daten in regelmäßigen Abständen von der Batch-Schicht verarbeitet werden, bietet die λ-Architektur die

Möglichkeit, in der Speed-Schicht nur eine einfachere Abschätzung zu implementieren. Auf diese Weise können aktuellere Ergebnisse als mit einer reinen Batch-Lösung erzielt werden, ohne dass eine aufwändige exakte Echtzeitlösung entwickelt werden muss. Die λ-Architektur verbindet somit die Robustheit von Batch-Systemen mit den Geschwindigkeitsvorteilen von Streaming-Architekturen. [17]

Ein weiterer Vorteil ist, dass alle eingehenden Ereignisse zunächst in der Batch-Schicht gespeichert werden. Sollte sich im Laufe der Zeit etwas an der Berechnungslogik ändern, sind alle alten Ereignisse immer noch in der Batch-Schicht gespeichert und können einfach mit der neuen Logik erneut verarbeitet werden. Dadurch ist es möglich, Änderungen an der Berechnungslogik ohne Datenverlust vorzunehmen. [19]

Neben den genannten Vorteilen hat diese Architektur aber auch Nachteile. Zum einen führt die doppelte Verarbeitung der Daten in der Batch- und der Speed-Schicht zu einer höheren Komplexität. Es müssen zwei Systeme entwickelt, gewartet und getestet werden, die die gleichen Aufgaben auf unterschiedliche Weise lösen. Diese Aufteilung in Speed- und Batch-Schicht erschwert auch eine eventuelle Fehlersuche. [19]

Durch diese Aufteilung ist auch die Erweiterbarkeit eingeschränkt. Soll die Pipeline um eine Funktion erweitert oder eine Anpassung vorgenommen werden, müssen beide Schichten angepasst werden. Dies führt wiederum zu erhöhtem Aufwand und erhöhter Fehleranfälligkeit. [19]

Dieser Nachteil kann zwar mit Frameworks wie Summingbird umgangen werden, dafür müssen aber an anderer Stelle Abstriche gemacht werden [20]. Mit Summingbird ist es möglich, Code zu schreiben, der sowohl für die Batch-Schicht als auch für die Speed-Schicht geeignet ist. Summingbird nutzt dazu die MapReduce-Funktionen von Hadoop für die Batch-Verarbeitung und Storm für die Stream-Verarbeitung [21].

Eine weitere Herausforderung dieser Architektur besteht darin, die in der Batch-Schicht und der Speed-Schicht berechneten Ergebnisse korrekt zusammenzuführen. Beispielsweise werden ältere Ergebnisse aus der Speed-Schicht hinfällig, sobald ein entsprechendes Ergebnis aus der Batch-Schicht vorliegt. Die Sicherstellung dieser Funktion stellt eine Herausforderung dar. [19]

3.1.3 κ-Architektur

In seinem Artikel *Questioning the Lambda Architecture* beschreibt Jay Kreps 2014, die Probleme der im vorherigen Kapitel 3.1.2 beschriebenen λ-Architektur. Außerdem stellt er darin eine alternative Architektur vor, die er κ-Architektur nennt. [19]

Die Grundidee besteht darin, dass in der Speed- und in der Batch-Schicht grundsätzlich die gleiche Berechnungslogik verwendet wird. Außerdem werden alle Ergebnisse, die bereits in der Speed-Schicht berechnet wurden, später in der Batch-Schicht erneut berechnet. Die κ-Architektur besteht somit nur aus zwei Schichten. Die Speed-Schicht übernimmt, wie bereits in der λ-Architektur, die Berechnung der Ergebnisse aus einem Datenstrom. Im Gegensatz zur λ-Architektur werden jedoch alle eingehenden Daten in der Speed-Schicht persistent gespeichert. Dieses Speichern ermöglicht es, alle Ereignisse bei Bedarf erneut zu verarbeiten. Die Ergebnisse werden wie in der λ-Architektur an die Serving-Schicht übergeben. Diese verwaltet die Ergebnisse und stellt sie anderen Diensten zur Verfügung. Diese Architektur ist ebenfalls in Abbildung 3.4 beschrieben. [19]

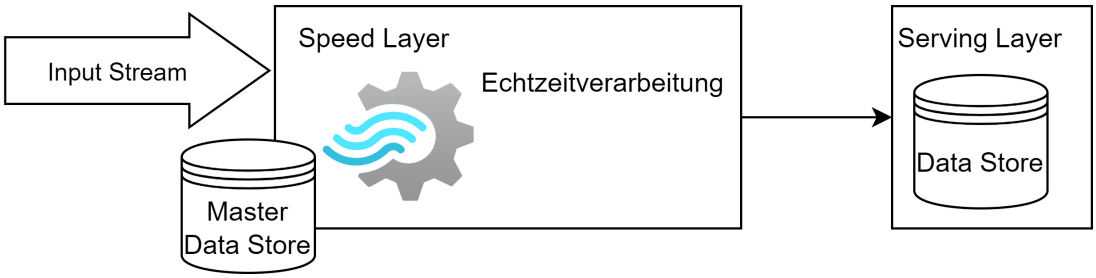

Abbildung 3.4: Aufbau einer κ-Architektur (Quelle: eigene Darstellung nach [18])

Zur Verwaltung des Datenstroms und zum Speichern der eingehenden Daten wird in der κ-Architektur häufig Apache Kafka verwendet. Auf Kafka wird in Abschnitt 3.2.3 näher eingegangen. Für die Verarbeitung werden häufig die gleichen Technologien wie bereits in der Speed-Schicht der λ-Architektur verwendet, also beispielsweise Apache Storm, Apache Spark Streaming oder Apache S4. [18]

Der große Vorteil dieser Architektur ist, dass nur eine Berechnungslogik entwickelt und gepflegt werden muss. Dies vereinfacht die Entwicklung und Wartung erheblich. Außerdem werden im Gegensatz zur λ-Architektur die gespeicherten Daten nicht regelmäßig neu verarbeitet, sondern nur dann, wenn sich Änderungen in der Berechnungslogik ergeben. In diesem Fall wird eine neue Instanz der Pipeline gestartet. Diese verarbeitet zunächst den gespeicherten Datenstrom, bis sie zu den aktuellen

Daten gelangt. Dann kann die alte Instanz der Pipeline beendet werden und die neue übernimmt die Arbeit. [19]

3.2 Technische Grundlagen

In den folgenden Abschnitten werden einige Technologien aus dem Bereich der Datenverarbeitung vorgestellt. Dabei wird besonders auf die im Rahmen dieser Bachelorarbeit verwendeten Technologien eingegangen.

3.2.1 MQTT

Das Protokoll wurde 1999 von IBM unter dem Namen Message Queueing Telemetry Transport (kurz MQTT) entwickelt und wird heute von der Organization for the Advancement of Structured Information Standards (kurz OASIS) standardisiert. Seit der Version 3.1.1 definiert OASIS, dass MQTT kein Akronym ist, da die Bezeichnung Message Queueing irreführend ist. Im Gegensatz zu vielen anderen Protokollen definiert MQTT keine Nachrichtenwarteschlange.

Doch was ist das MQTT-Protokoll und wofür kann es verwendet werden? In der offiziellen Dokumentation der MQTT Version 3.1.1 heißt es dazu:

> „MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.“ [22, Abstract]

Es handelt sich also um ein Client-Server-Protokoll zum Transport von Nachrichten nach dem Publish-Subscribe-Pattern. Es wurde entwickelt, um leicht, offen, einfach und unkompliziert implementierbar zu sein, heißt es weiter. Diese Eigenschaften machen es ideal für die Kommunikation zwischen Maschinen (englisch Machine to Machine oder kurz M2M) und im IoT-Umfeld, wo es oft auf eine geringe Codebasis ankommt und die Netzwerkbandbreite begrenzt ist. [23]

Der Ablauf der Kommunikation über MQTT ist in Abbildung 3.5 dargestellt. Ein oder mehrere MQTT-Clients veröffentlichen Nachrichten (englisch publish) und senden

diese an einen Server, der im MQTT-Kontext als Broker bezeichnet wird. Neben dem Inhalt der Nachricht sendet der Client auch die Information, auf welchem Topic die Nachricht veröffentlicht werden soll. Ein Client kann nun ein solches Topic beim Broker abonnieren (englisch subscribe). Wird nun eine neue Nachricht auf diesem Topic veröffentlicht, so erhalten alle Clients, die das entsprechende Topic abonniert haben, diese Nachricht vom Broker zugestellt. [23]

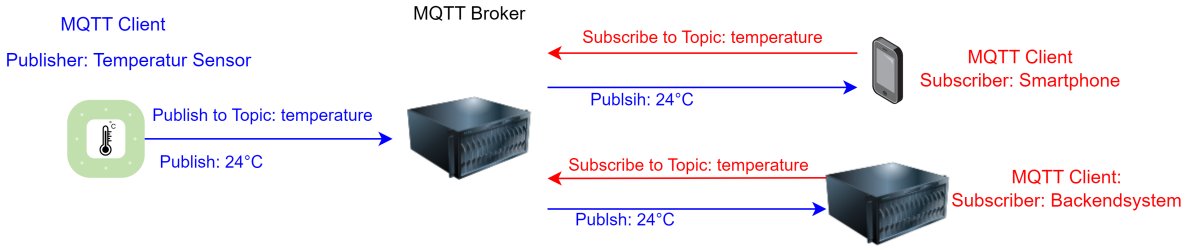

Abbildung 3.5: Nachrichtenaustausch über MQTT (Quelle: eigene Darstellung nach [23])

Darüber hinaus bietet MQTT die Möglichkeit, sowohl beim Publishen einer Nachricht als auch beim Subscriben ein sogenanntes Quality of Service Level (kurz QoS) festzulegen, mit dem Nachrichten übertragen werden sollen. Das angegebene QoS-Level beim publishen bezieht sich dabei auf die Übertragung der Nachricht vom Publisher zum Broker. Das angegebene QoS-Level beim Subscriben bezieht sich auf die Übertragung der Nachrichten vom Broker zum Subscriber. [23]

Die niedrigste Stufe ist QoS 0, die oft auch als „fire and forget“ bezeichnet wird [24]. Bei diesem QoS-Level wird die Nachricht genau einmal gesendet, es wird nicht sichergestellt, dass die Nachricht auch beim Empfänger ankommt. Dies ist auch in Abbildung 3.6 dargestellt.

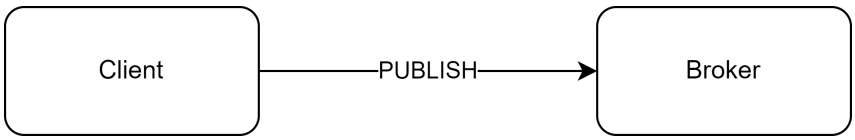

Abbildung 3.6: Übertragung einer Nachricht mit QoS 0
(Quelle: eigene Darstellung nach [24])

Die nächsthöhere QoS-Stufe ist QoS 1, die sicherstellt, dass eine Nachricht auch beim Empfänger ankommt. Wenn dieser eine Nachricht empfängt, antwortet er mit einer sogenannten PUBACK-Nachricht, die die Packet-ID der ursprünglichen Nachricht enthält. Durch diese Rückmeldung weiß der Sender, dass seine Nachricht auch empfangen wurde. Erhält er diese PUBACK-Nachricht nicht, so kann die ursprüngliche Nachricht erneut gesendet werden. Auf diese Weise kann sichergestellt werden, dass

eine Nachricht auch beim Empfänger ankommt. Dieses Verhalten ist auch in Abbildung 3.7 dargestellt. Unter bestimmten Umständen kann dieses Verfahren jedoch dazu führen, dass Nachrichten mehrfach empfangen werden. [24]

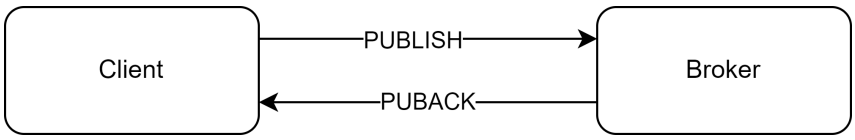

Abbildung 3.7: Übertragung einer Nachricht mit QoS 1
(Quelle: eigene Darstellung nach [24])

Um zu verhindern, dass Nachrichten mehrfach zugestellt werden, gibt es die QoS-Stufe 2, die sicherstellt, dass eine Nachricht genau einmal beim Empfänger ankommt. Der Ablauf der Kommunikation mit QoS 2 ist in Abbildung 3.8 dargestellt. Wird eine Nachricht mit QoS 2 empfangen, antwortet der Empfänger mit einer sogenannten PUBREC-Nachricht. Diese enthält die Packet-ID der empfangenen Nachricht. Erhält der Sender diese PUBREC-Nachricht nicht innerhalb einer definierten Zeit, so sendet er seine Nachricht erneut, diesmal jedoch mit gesetztem Duplicated-Flag (kurz DUP). Erhält der Sender die PUBREC Nachricht, antwortet er mit einer sogenannten PUBREL Nachricht. Diese enthält ebenfalls die Packet-ID der ursprünglichen Nachricht. Schließlich antwortet der Empfänger auf die PUBREL Nachricht mit einer sogenannten PUBCOMP Nachricht und die Übertragung ist abgeschlossen. [24]

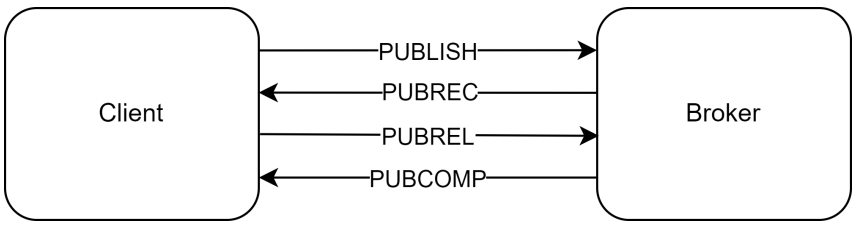

Abbildung 3.8: Übertragung einer Nachricht mit QoS 2
(Quelle: eigene Darstellung nach [24])

3.2.2 AMQP

Das Advanced Message Queueing Protocol (kurz AMQP) ist ein nachrichtenorientiertes Middleware-Protokoll, das zur Kommunikation zwischen verschiedenen Anwendungen oder Systemen eingesetzt werden kann. Dabei gibt es sogenannte Publisher, die Nachrichten versenden und sogenannte Consumer, die Nachrichten empfangen. Die Kommunikation zwischen Publisher und Consumer läuft über einen sogenannten Broker, der die Nachrichten verteilt. [25]

Ein Publisher sendet die Nachrichten an einen sogenannten Exchange. Die Subscriber erhalten die Nachrichten über sogenannte Queues, die an die Exchanges angebunden sind. Dabei gibt es verschiedene Arten von Exchanges, die die Nachrichten unterschiedlich verteilen. Gibt es für eine Queue mehrere Consumer, so werden die Nachrichten der Queue nach dem Round-Robin-Verfahren auf die Consumer verteilt. [25]

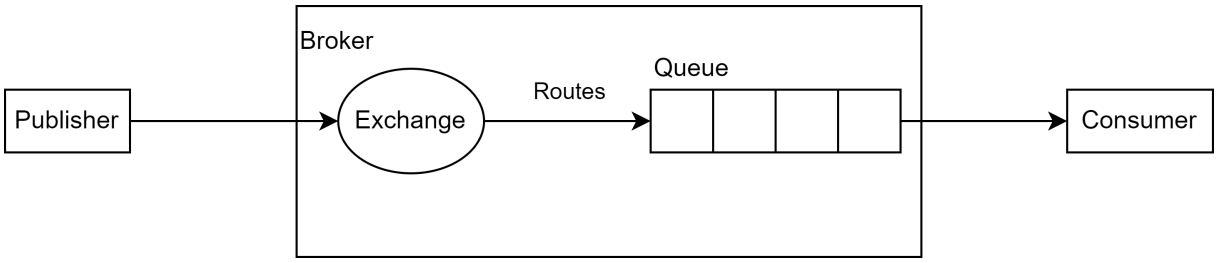

Abbildung 3.9: Kommunikation zwischen Publisher und Consumer über AMQP (Quelle: eigene Darstellung nach [25])

Der erste Exchange-Typ ist der Direct Exchange. Bei diesem Exchange Typ werden die Nachrichten über den sogenannten Routing Key auf die Queues verteilt. Dabei wird eine Queue über einen Routing Key an einen Exchange gebunden. Eine Message wird genau dann an diese Queue weitergeleitet, wenn der Routing Key der Message mit dem der Queue übereinstimmt. Dabei kann eine Queue auch mit mehreren Routing Keys an einen Exchange gebunden sein. Die Abbildung 3.10 zeigt eine beispielhafte Konfiguration mit drei Queues. In diesem Beispiel sollen Log-Meldungen je nach Typ an unterschiedliche Queues geliefert werden. Die sogenannte `Error–Queue` soll dabei nur Meldungen mit dem Routing Key `log.error` erhalten, während die `Production–Log–Queue` neben Fehlermeldungen auch Warnungen erhalten soll. Zusätzlich gibt es in diesem Beispiel noch die `Debug–Log–Queue`, die neben Fehlern und Warnungen auch Info-Logs erhalten soll. [26]

Daneben gibt es noch den Exchange-Typ Fanout. Bei diesem Typ werden alle Nachrichten unabhängig vom Routing Key an alle angeschlossenen Queues verteilt. Mit diesem Exchange können sehr einfach Broadcasts realisiert werden. Die Abbildung 3.11 zeigt ein Beispiel, bei dem alle eingehenden Nachrichten mit Hilfe eines Fanout-Exchange an alle angeschlossenen Queues weitergeleitet werden. In diesem Beispiel erhalten die Queues eins bis drei jeweils alle Nachrichten. [25]

Der Topic-Exchange funktioniert ähnlich wie der Direct-Exchange. Auch hier werden die Nachrichten über Routing Keys an die Queues weitergeleitet. Im Gegensatz zum Direct-Exchange können hier jedoch neben dem gesamten Routing Key auch Platzhal-

ter verwendet werden. Einer dieser Platzhalter ist das Symbol `*`, das innerhalb eines Routing Keys genau eine Hierarchieebene ersetzt. Wie in Abbildung 3.12 zu sehen ist, können mit `log.*` alle Routing Keys abonniert werden, die auf der ersten Hierarchieebene den Eintrag `log` haben und genau eine weitere Hierarchieebene enthalten. Neben dem Symbol `*` kann auch das Symbol `#` als Platzhalter verwendet werden. Es ersetzt im Gegensatz zum `*`-Symbol beliebig viele Hierarchieebenen. Wird eine Queue mit dem Routing Key `#` an einen Topic-Exchange gebunden, so werden alle Nachrichten unabhängig vom Routing Key an diese Queue weitergeleitet. In diesem Fall verhält sich der Topic-Exchange gegenüber dieser Queue wie ein Fanout-Exchange. [26]

Ein weiterer Exchange-Typ ist der Headers-Exchange. Bei diesem werden, ähnlich wie beim Direct-Exchange, nur bestimmte Nachrichten an die Queues weitergeleitet. Dazu wird jedoch nicht der Routing Key verwendet, sondern ein oder mehrere Header-Attribute. Werden mehrere Header-Attribute angegeben, kann zusätzlich festgelegt werden, ob eine Nachricht weitergeleitet werden soll, sobald eines der Attribute übereinstimmt, oder ob alle Attribute übereinstimmen müssen. Das in Abbildung 3.13 dargestellte Beispiel demonstriert die Funktionsweise des Headers-Exchange. Das Attribut `x–match` wird verwendet, um anzugeben, ob die Header-Attribute mit einem logischen Und oder einem logischen Oder verknüpft werden sollen. `"x–match": "any"` steht für eine Oder-Verknüpfung, `"x–match": "all"` für eine Und-Verknüpfung. In diesem Beispiel werden also an `Queue 1` und `Queue 3` alle Nachrichten weitergeleitet, die das Header-Attribut `"h1": "header1"` oder `"h2": "header2"` enthalten. An `Queue 2` werden nur Nachrichten weitergeleitet, die sowohl das Attribut `"h1": "header1"` als auch das Attribut `"h2": "header2"` enthalten. [26, 27]

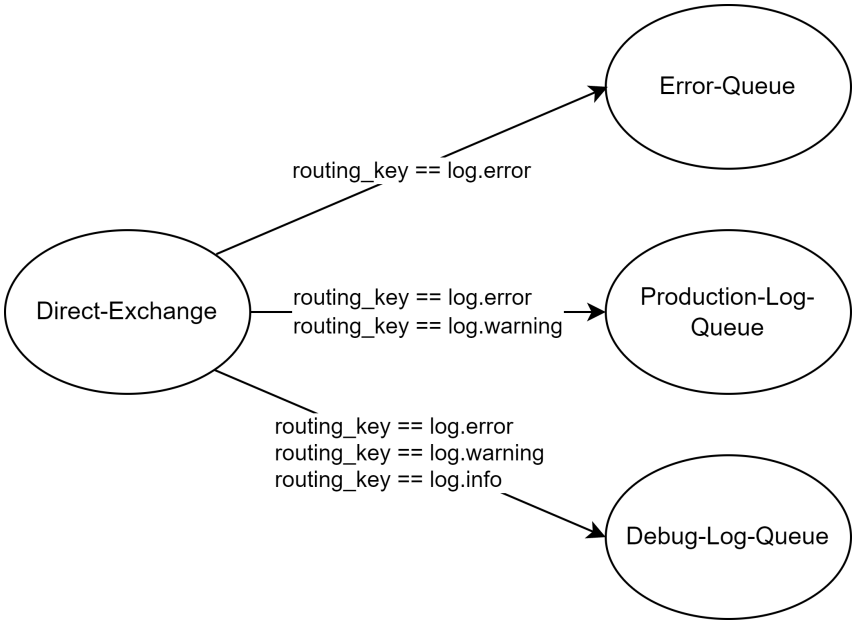

Abbildung 3.10: Verteilung der Nachrichten mit Hilfe eines Direct Exchanges (Quelle: eigene Darstellung nach [26])

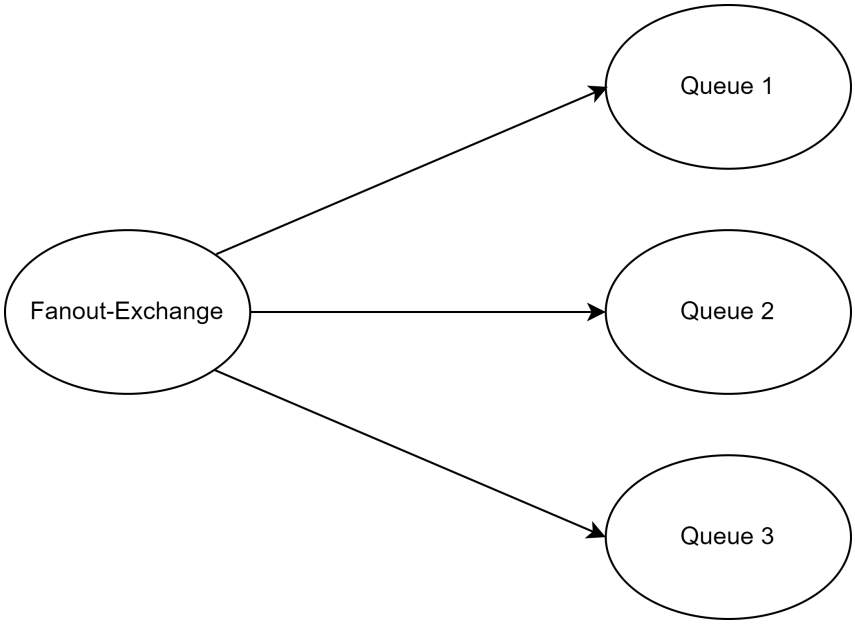

Abbildung 3.11: Verteilung der Nachrichten mit Hilfe eines Fanout Exchanges (Quelle: eigene Darstellung nach [25])

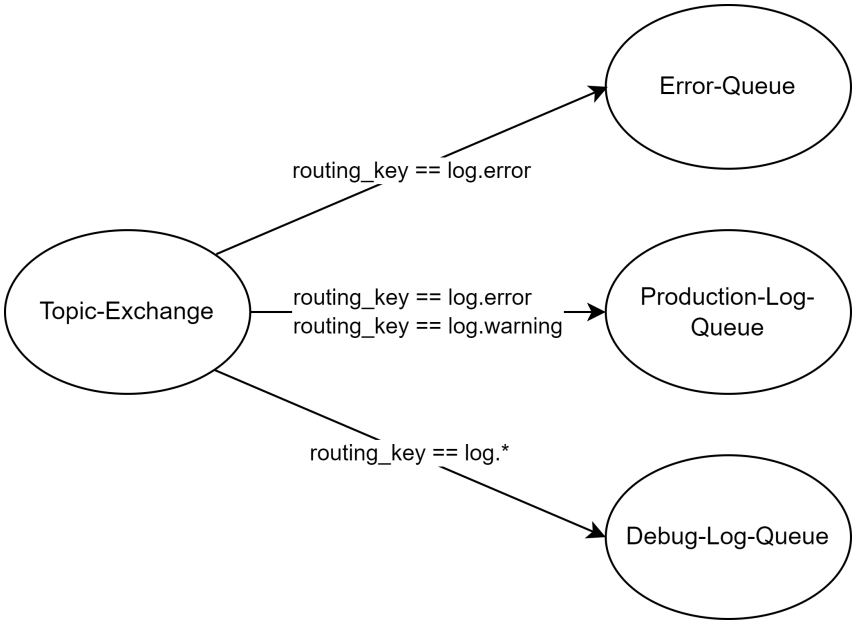

Abbildung 3.12: Verteilung der Nachrichten mit Hilfe eines Topic Exchanges (Quelle: eigene Darstellung nach [26])

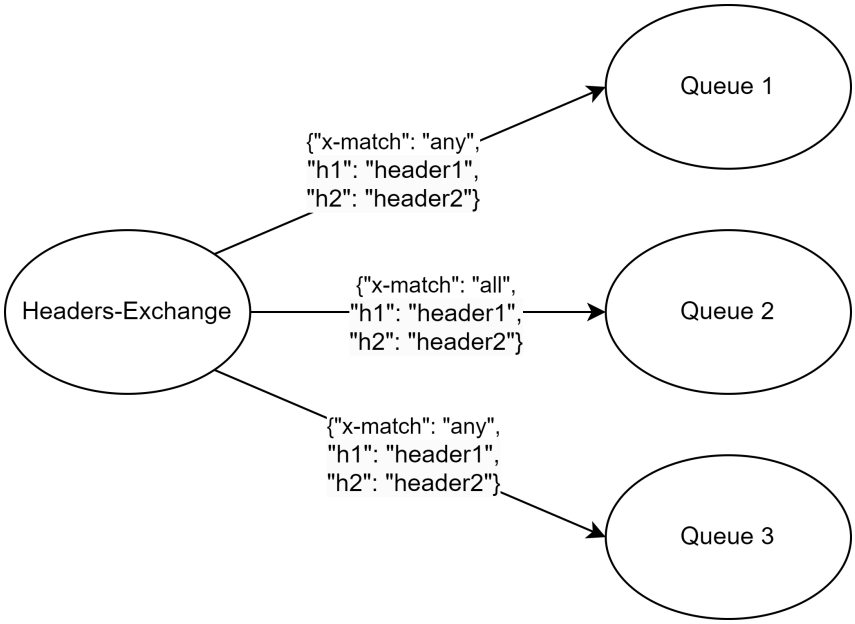

Abbildung 3.13: Verteilung der Nachrichten mit Hilfe eines Headers Exchanges (Quelle: eigene Darstellung nach [27])

3.2.3 Apache Kafka

Kafka ist eine Open-Source-Software der Apache Software Foundation zur Verarbeitung von Datenströmen. Ursprünglich wurde Kafka 2011 von LinkedIn als Message Queue entwickelt. Seit 2012 wird es von der Apache Software Foundation gepflegt und weiterentwickelt. Mittlerweile ist Kafka eine verteilte Streaming-Plattform. [28]

Kafka ist ein verteiltes System, das aus Servern und Clients besteht. Diese kommunizieren über ein sehr leistungsfähiges TCP-Netzwerkprotokoll (kurz für Transmission Control Protocol). Kafka kann auf physischer Server-Hardware, auf virtuellen Maschinen und als Container in Cloud-Umgebungen eingesetzt werden. [28]

Dabei bietet Kafka drei Hauptfunktionalitäten, die verwendet werden können:

1. Lesen und Schreiben von Event Streams, sowie der kontinuierliche Import und Export von Daten aus anderen Systemen. [28]
2. Konsistentes Speichern der Datenströme [28]
3. Das Verarbeiten von Datenströmen, sowohl in Echtzeit als auch im Nachhinein [28]

Eine Nachricht wird im Kafka Umfeld Event genannt. Ein Event besteht dabei aus den Elementen Key, Value und Timestamp. Ein Event könnte zum Beispiel folgendermaßen aussehen:

- **Event Key**: „Alice“
- **Event Value**: „zahlte 200€ an Bob“
- **Event Timestamp**: „2022-12-24T12:00“

Darüber hinaus gibt es die sogenannten Producer, die Ereignisse senden, und die Consumer, die Ereignisse lesen bzw. empfangen und verarbeiten. Diese sind in Kafka komplett voneinander entkoppelt. Einzelne Events werden in sogenannten Topics verwaltet und gespeichert. Für das oben genannte Beispiel Event könnte es das Topic „Zahlungen“ geben. In diesem Topic werden alle Events gespeichert, die Zahlungen von einem Benutzer an einen anderen Benutzer enthalten. In Kafka können sowohl mehrere Producer in ein Topic schreiben, als auch mehrere Consumer aus einem Topic lesen. Im Gegensatz zu vielen anderen Messaging-Systemen werden Events in Kafka nicht aus dem System gelöscht, nachdem sie von einem Consumer verarbeitet wurden. Dadurch ist es möglich, Events erneut zu verarbeiten. [28]

Die Topics sind wiederum partitioniert. Das bedeutet, dass ein Topic auf mehrere Broker Instanzen verteilt werden kann. Durch diese Verteilung der Daten kann eine höhere Skalierbarkeit erreicht werden, da sowohl auf mehrere Partitionen gleichzeitig geschrieben, als auch von mehreren Partitionen gleichzeitig gelesen werden kann. Kafka stellt dabei sicher, dass Ereignisse mit dem gleichen Event Key immer auf der gleichen Partition gespeichert werden. Im obigen Beispiel würden also alle Zahlungen von „Alice“ auf der gleichen Partition, innerhalb der Topic „Zahlungen“ gespeichert werden. Auch beim Lesen stellt Kafka sicher, dass alle Events einer Partition vom gleichen Consumer verarbeitet werden. Diese Aufteilung eines Topics in mehrere Partitionen ist auch in der Abbildung 3.14 dargestellt. [28]

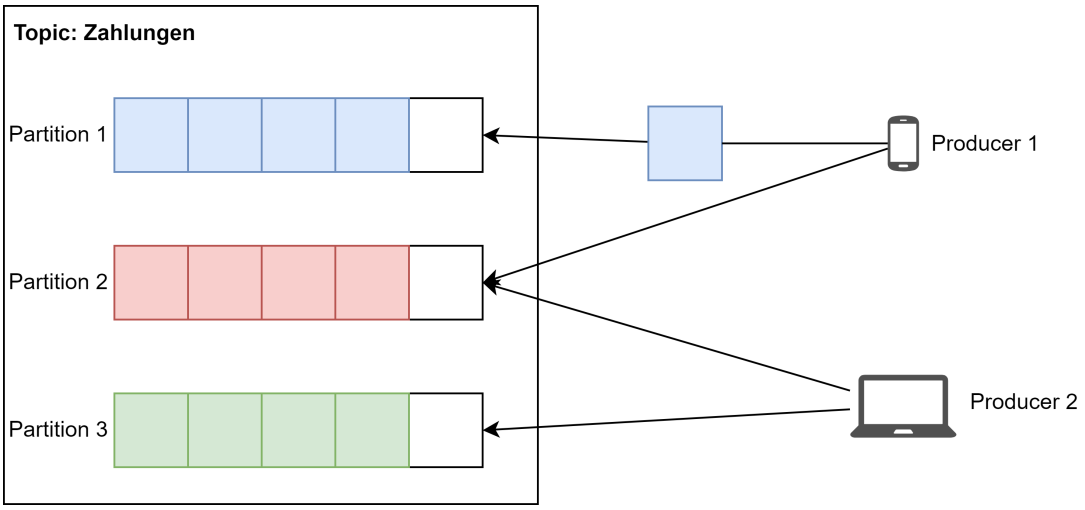

Abbildung 3.14: Aufteilung von Events auf mehrere Partitionen innerhalb eines Topics. Gleiche Farben stehen in dieser Abbildung für gleiche Event Keys (Quelle: eigene Darstellung nach [28])

Um das System hochverfügbar und fehlertolerant zu machen, bietet Kafka die Möglichkeit, Replikationen von Topics zu erstellen. Dies bedeutet, dass eine Kopie aller Daten eines Topics auf einem oder mehreren anderen Brokern existiert. Dadurch bleibt das System auch bei Ausfall eines Brokers funktionsfähig. [28]

3.3 Machine Data Connector

Um die Daten einer CNC-Maschine auszulesen und weiterzuleiten, wird im Rahmen dieser Bachelorarbeit der sogenannte Machine Data Connector (kurz MDC) der Firma DMG Mori verwendet. Seine Eigenschaften werden im Folgenden kurz vorgestellt.

Der MDC bietet die Möglichkeit, Daten von CNC-Maschinensteuerungen auszulesen. Dabei unterstützt der MDC Steuerungen verschiedener Hersteller, darunter Siemens,

Heidenhain und FANUC [29]. Er kann verschiedene Informationen auslesen, wie zum Beispiel das Laden eines neuen NC-Programms, das Starten der Bearbeitung, das Stoppen der Maschine und vieles mehr. Die von einem MDC erfassten Ereignisse werden im Abschnitt 4.2 näher erläutert.

Um die gesammelten Informationen weiterverarbeiten zu können, bietet der MDC verschiedene Protokolle an, um die Informationen weiterzugeben. Unterstützt werden die Protokolle MTConnect, MQTT sowie OPC UA. [30]

Das MTConnect Protokoll basiert auf dem Hypertext Transfer Protocol (kurz HTTP). Über einen HTTP-Request können Daten angefordert werden, die im Format der Extensible Markup Language (kurz XML) zurückgegeben werden. Das MTConnect Protokoll wird von der Association for Manufacturing Technology (kurz AMT) gepflegt und verwaltet. [31]

OPC UA ist ein Kommunikationsstandard, der von der OPC Foundation entwickelt wird. OPC UA steht für Open Platform Communications Unified Architecture. Auch OPC UA wird zum Datenaustausch zwischen Maschinen und Anlagen verwendet. OPC UA definiert dafür zwei Protokolle für den Datenaustausch zwischen Clients und Servern. Zum einen ein ressourcenoptimiertes binäres Protokoll und zum anderen ein auf Webservices basierendes Protokoll. [32]

Das MQTT-Protokoll wurde bereits in Abschnitt 3.2.1 beschrieben. Dieses wird auch für den entwickelten Softwareprototypen verwendet. Der Aufbau der Events sowie deren Bedeutung wird in Abschnitt 4.2 beschrieben.

3.4 Verwandte Arbeiten

Nachdem in den vorherigen Abschnitten bereits einige verwandte Arbeiten aus dem Bereich Datenverarbeitung, sowie dabei verwendete Technologien vorgestellt wurden. Werden in diesem Abschnitt verwandte Arbeiten genannt und beschrieben, die sich mit dem Aufbau von IoT-Systemen beschäftigen.

Für die Umsetzung eines IoT-Systems in der Produktion wurden bereits mehrere Referenzarchitekturen vorgestellt. Eine davon ist die Industrial Internet Reference Architecture (kurz IIRA) des Industry IoT Consortiums (kurz IIC). Sie beschreibt Anforderungen und Funktionen eines IoT-Systems aus verschiedenen Blickwinkeln

[33]. Eine ähnliche Vorgehensweise beschreibt der Verein Deutscher Ingenieure e.V. (kurz VDI) in seinem Status Report mit dem Titel *Referenzarchitekturmodell Industrie 4.0* (kurz RAMI4.0) [34]. Beide Arbeiten beschreiben die Vorgehensweise und mögliche Herausforderungen bei der Implementierung eines IoT-Systems im industriellen Kontext. Beide sind jedoch eher als Leitfaden zu verstehen und geben keine konkrete Umsetzungsstrategie vor.

Eine konkretere Umsetzungsstrategie stellt IBM mit der *IBM Industrie 4.0 Architecture* vor. Diese besteht aus zwei Schichten. Einer sogenannten Edge-Schicht, die sich nahe an der tatsächlichen Fertigung befindet und für die Kommunikation mit den einzelnen Maschinen zuständig ist. Daneben gibt es die sogenannte Plattformschicht. Diese erhält Informationen aus der Edge-Schicht und kann darauf aufbauend Analysen durchführen und Prozesse in der Edge-Schicht steuern. Die Kommunikation zwischen den Schichten erfolgt dabei über klar definierte Schnittstellen. [35]

Darüber hinaus gibt es eine Vielzahl weiterer Referenzarchitekturen, die wiederum spezielle Anforderungen abdecken. Einen guten Überblick über verschiedene Architekturen gibt der Artikel von Moghaddam et al. *Reference Architectures for Smart Manufacturing: A Critical Review*. [36]

Neben diesen beschriebenen Referenzarchitekturen, die eine allgemeine Lösungsstrategie für IoT-Anwendungsfälle beschreiben, finden sich auch Arbeiten, die Architekturen an konkreten Beispielen zeigen. So zeigen Farooqui et al. in *Towards data-driven approaches in manufacturing: an architecture to collect sequences of operations* einen Ansatz, wie mit Hilfe von Event Streams Daten gesammelt und verarbeitet werden können. Als Beispiel wird dafür eine automatisierte Roboterzelle verwendet [37]. Ein weiteres Beispiel ist die in *A Data-Centric Internet of Things Framework Based on Azure Cloud* vorgestellte Architektur. Dort wird ein zweischichtiges Framework beschrieben, mit dem ein IoT-System realisiert werden kann [38].

In diesem Kapitel wurden verschiedene Konzepte und Technologien aus dem Bereich IoT und Daten-Pipelines vorgestellt. Darauf aufbauend wird in den folgenden Kapiteln ein eigenes Konzept vorgestellt, mit dem die im Kapitel 1 beschriebenen Anforderungen erfüllt werden können.

4. Konzeption

In den folgenden Abschnitten wird ein Konzept vorgestellt, wie die im Abschnitt 1.3 beschriebenen Anwendungsfälle umgesetzt werden können. Dabei wird zunächst die bestehende Ausgangssituation beschrieben. Anschließend werden die von den MDCs gelieferten Daten analysiert und schließlich ein Lösungsansatz vorgestellt.

4.1 Ausgangslage

Für einen früheren Test wurde bereits ein System aufgebaut, bei dem drei Fertigungsmaschinen mit einem MDC ausgestattet wurden. Diese MDCs wurden so konfiguriert, dass sie auftretende Ereignisse mit Hilfe von MQTT verteilen. Als Broker wird RabbitMQ mit dem MQTT-Plugin verwendet. Diese Konfiguration ermöglicht es, Nachrichten über MQTT zu empfangen und diese mit AMQP weiter zu verteilen. MQTT Topics werden in diesem Szenario mit einem AMQP Topic Exchange verbunden. Für einen Consumer ist es dann möglich, über AMQP Queues an bestimmte Topics zu binden. Der Name des MQTT Topics, auf dem eine Nachricht veröffentlicht wurde, wird dabei zu einem AMQP Routing Key. Bei MQTT wird zur hierarchischen Trennung von Topics das Symbol `/` verwendet, während bei AMQP Routing Keys zur Trennung das Symbol `.` verwendet wird. Aus einem MQTT Topic `DMG/PRODUCTION/TIMES.ACT_CYCLE` wird also der AMQP Routing Key `DMG.PRODUCTION.TIMES.ACT_CYCLE`. [39]

Der RabbitMQ Broker wird nicht selbst installiert, sondern es wird die gemanagte Cloud-Lösung cloudAMQP verwendet. CloudAMQP bietet einen verwalteten RabbitMQ Broker an, der über die Oberfläche von cloudAMQP konfiguriert werden kann [40]. Dadurch entfällt der Aufwand für die Installation und Verwaltung eines eigenen Brokers. Diese Ausgangssituation ist auch in Abbildung 4.1 dargestellt. Die MDCs senden ihre Ereignisse über MQTT an einen RabbitMQ Broker in der Cloud. Dieser verteilt die empfangenen Nachrichten mit Hilfe von AMQP an andere Dienste.

In den folgenden Abschnitten wird beschrieben, wie darauf aufbauend ein System entwickelt wird, um die Bearbeitungszeiten einzelner NC-Programme aus dem Datenstrom zu extrahieren, aufzubereiten und anderen Diensten zur Verfügung zu stellen.

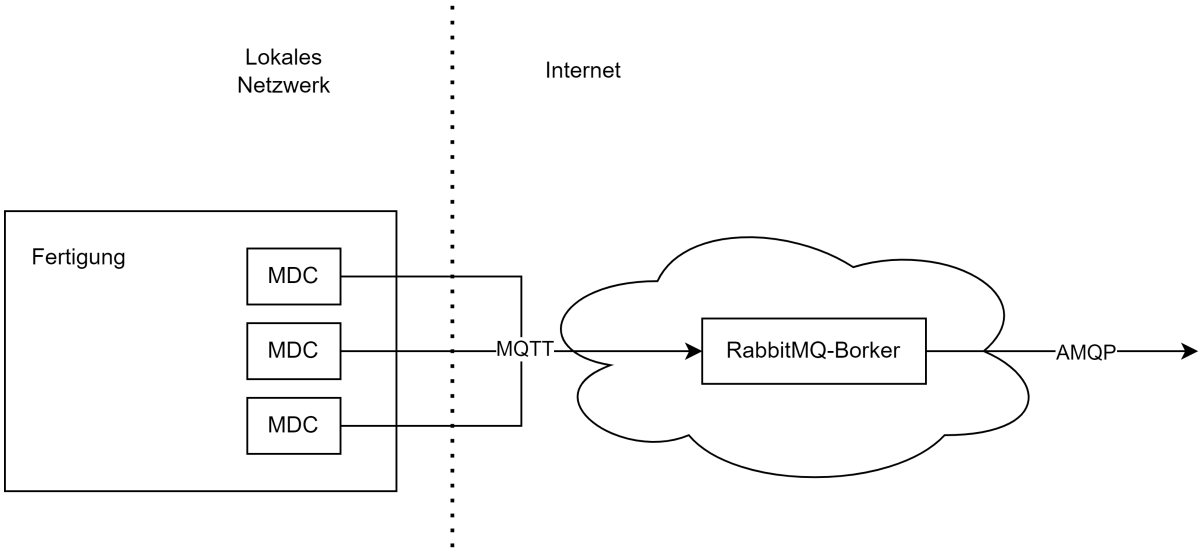

Abbildung 4.1: Ausgangslage zu Beginn der Bachelorarbeit (Quelle: eigene Darstellung)

4.2 Beschreibung des Datenstroms der MDCs

Alle Maschinenereignisse werden vom MDC über MQTT verteilt. Um die Ereignisse unterscheiden zu können, werden verschiedene Topics verwendet. Der erste Teil dieses Topics ist immer für die Identifikation der Maschine vorgesehen. Das bedeutet, dass jede Maschine bzw. jeder MDC seine Ereignisse in einem eigenen Topic veröffentlicht [30].

Zur Strukturierung der Nachrichten verwendet der MDC die sogenannten Cumolocity-Templates [30]. Diese definieren Formate, in denen der Inhalt von MQTT-Nachrichten strukturiert werden kann. Eine so strukturierte Nachricht beginnt immer mit einer Nummer. Diese Zahl gibt an, welches Template zur Strukturierung der Nachricht verwendet wurde. Die Templates definieren dabei mehrere Felder innerhalb einer Nachricht, wobei die einzelnen Felder durch ein Komma voneinander getrennt sind. Für den hier dargestellten Anwendungsfall sind insbesondere die beiden Templates 200 für Messergebnisse und 400 für aufgetretene Ereignisse relevant. Der Aufbau dieser Templates ist in den Tabellen 4.1 und 4.2 dargestellt [41].

200,<fragment>,<series>,<value>,<unit>,<time>	
200	templateID für Messungen
<fragment>	Um welche Art der Messung handelt es sich?
<series>	Gehört diese Messung zu einer Serie an Messungen?
<value>	Der Messwert
<unit>	Die Einheit in der der Messwert angegeben ist
<time>	Der Zeitpunkt der Messung

Tabelle 4.1: Aufbau des Cumolocity Templates 200

400,<type>,<text>,<time>	
400	templateID für Ereignisse
<type>	Um welche Art von Ereignis handelt es sich?
<text>	Ein frei definierbarer Text für das Ereignis
<time>	Der Zeitpunkt zu dem das Ereignis aufgetreten ist

Tabelle 4.2: Aufbau des Cumolocity Templates 400

Im Folgenden werden die wichtigsten Ereignisse die ein MDC liefert kurz beschrieben.

Aktuelle Bearbeitungszeit:
Beispiel:
`200,DMG.PRODUCTION.TIMES.ACT_CYCLE,DEFAULT,3.2,s,2022−09−08T07:33:14.330+02:00`
Auf diesem Topic werden die aktuellen Bearbeitungszeiten veröffentlicht. Die Bearbeitungszeit ist dabei in Sekunden angegeben. Wird eine neue Bearbeitung gestartet beginnt diese Zeit wieder bei null [30]. Als Vorlage wird das Cumolocity Template 200 (Measurement) verwendet [41].

Aktueller Zustand der Maschine:
Beispiel: `400,DMG.NC.PROG_STATE_1,"RUNNING",2022−09−08T07:44:23.051+02:00`
Auf diesem Topic werden Änderungen des Maschinenstatus veröffentlicht. Dieser zeigt an, ob eine Maschine gerade ein NC-Programm abarbeitet, angehalten ist oder die Bearbeitung beendet hat [30]. Als Vorlage wird hier das Cumolocity Template 400 (Event) verwendet [41]. Mögliche Werte sind

- `"RUNNING"`: Ein Programm wird aktiv ausgeführt
- `"STOPPED"`: Die Bearbeitung wurde pausiert
- `"ABORTED"`: Die Bearbeitung wurde beendet

Aktueller Betriebsmodus:
Beispiel: `400,DMG.NC.OP_MODE_1,"AUTO",2022−09−08T07:52:59.108+02:00`
Nachrichten auf diesem Topic zeigen Änderungen des Betriebsmodus einer Maschine an [30]. Auch hier wird das Cumolocity Template 400 (Event) verwendet [41]. Folgende Betriebsmodi können dabei angenommen werden:

- `"AUTO"`: Automatische Abarbeitung eines NC-Programms [30]
- `"MDA"`: (Manual Data Automatic) Die Maschine arbeitet einen Satz, oder eine Folge von Sätzen ab, die über die Bedientafel eingegeben wurden [30]
- `"JOG"`: (Jogging) Maschine wird manuell über die Richtungstasten, bzw. das Handrad gesteuert [30]

Name des NC-Programms:
Beispiel: `400,DMG.NC.PROG_NAME_1,"<programName>",2022−09−08T13:37:46.505+02:00`
Ein Ereignis auf diesem Topic zeigt an, dass ein neues NC-Programm geladen wurde. Anstelle von `<programName>` steht abhängig von der verwendeten Maschinensteuerung der Name des geladenen NC-Programms oder der gesamte Pfad zu diesem Programm [30]. Auch für dieses Ereignis wird das Cumolocity Template 400 (Event) verwendet [41]. Mit Hilfe des NC-Programmnamens kann eine Bearbeitung später einer Kalkulation zugeordnet werden.

Teilezähler:
Beispiel:
`200,DMG.PRODUCTION.PARTS.ACTUAL_1,DEFAULT,37,Parts,2022−09−07T07:47:31.451+02:00`
Bei manchen Maschinen ist es möglich zu definieren wie oft eine Bearbeitung durchgeführt werden soll. Ein Ereignis auf diesem Topic zeigt an, dass eine solche Bearbeitung fertiggestellt wurde, indem der Wert des Events um eins erhöht wird. In diesem Beispiel wurde die Bearbeitung zum 37 Mal fertiggestellt. Neben diesem Event gibt es ein weiteres, welches anzeigt wie oft diese Bearbeitung insgesamt durchgeführt werden soll [30]. Für dieses Ereignis wird das Cumolocity Template 200 (Measurement) verwendet [41].

Neben den beschriebenen Ereignissen liefert der MDC noch einige weitere, wie beispielsweise Alarmmeldungen, Änderungen der Vorschubgeschwindigkeit, Zustand der Signalleuchten, und viele weitere [30]. Diese sind allerdings für die Bestimmung der Bearbeitungszeit eines Bauteil nicht relevant. Deshalb wird an dieser Stelle nicht weiter darauf eingegangen.

4.3 Analyse des Datenstroms

Im Folgenden soll der vom MDC gesendete Datenstrom für eine Maschine analysiert werden. Insbesondere soll untersucht werden, welche Ereignisse zur Identifizierung einer abgeschlossenen Bearbeitung verwendet werden können.

Auf den ersten Blick ist ersichtlich, dass der Maschinenstatus verwendet werden kann. Nachdem eine `"ABORTED"` Nachricht gesendet wurde, springt die Programmlaufzeit auf Null, bevor nach einiger Zeit eine neue Bearbeitung beginnt. Damit können jedoch nicht alle Bearbeitungszeiten identifiziert werden. Wie in Abbildung 4.2 zu sehen ist, gibt es auch Stellen, an denen die Laufzeit auf Null springt, ohne dass vorher eine `"ABORTED"` Nachricht gesendet wurde. Dies ist der Fall, wenn mehrere identische Teile nacheinander bearbeitet werden. Das Ende einer solchen Bearbeitung wird durch die Erhöhung des Teilezählers um eins angezeigt (siehe Abbildung 4.2).

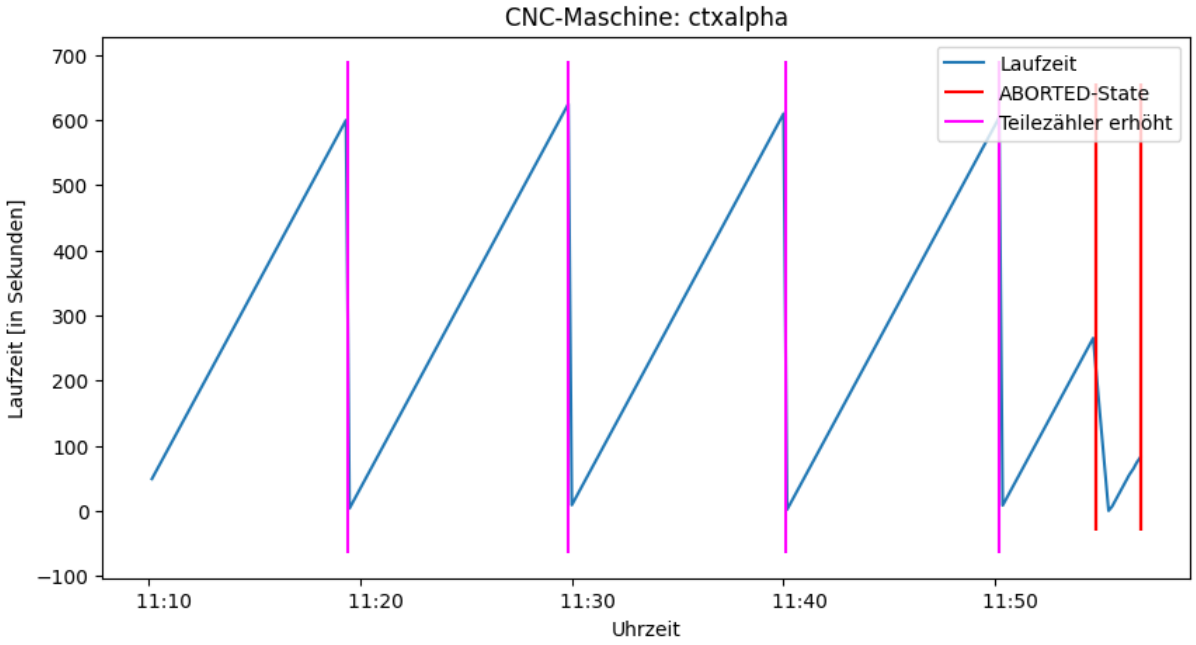

Abbildung 4.2: Laufzeiten für einzelne NC-Programme mit eingeblendeten ABORTED und Teilezähler Nachrichten. RUNNING und STOPPED Nachrichten wurden in dieser Darstellung zur besseren Übersicht weggelassen (Quelle: eigene Darstellung)

Das Ereignis mit der aktuellen Bearbeitungszeit kann nicht für die Laufzeitanalyse verwendet werden. In der MDC-Dokumentation heißt es dazu, dass dieses Ereignis die Laufzeit des aktiven Programms liefert. Die Zeit, in der das Programm gestoppt war, wird abgezogen [30]. Bei der Analyse des Datenstroms für die bereits angeschlossenen CNC-Maschinen fiel auf, dass bei einer Maschine die Zeiten aus diesem Ereignis nicht mit den zugehörigen Zeitstempeln übereinstimmen. Zwischen zwei Zeitstempeln

liegen ca. zehn Sekunden, während das Ereignis die Laufzeit nur um 5 Sekunden erhöht (siehe Abbildung 4.3 und Quellcode 4.1). Ob dieses Verhalten auf einen Fehler im MDC oder in der Maschinensteuerung zurückzuführen ist, konnte nicht abschließend geklärt werden. Durch die im Abschnitt 5.1 beschriebene Testbearbeitung konnte jedoch bestätigt werden, dass die im Laufzeitereignis angegebenen Programmlaufzeiten für diese Maschine falsch sind und die über die Zeitstempel ermittelten Bearbeitungszeiten den tatsächlichen Zeiten entsprechen. Daher werden für die Berechnung der Laufzeiten die Zeitstempel verwendet, wie im Folgenden beschrieben.

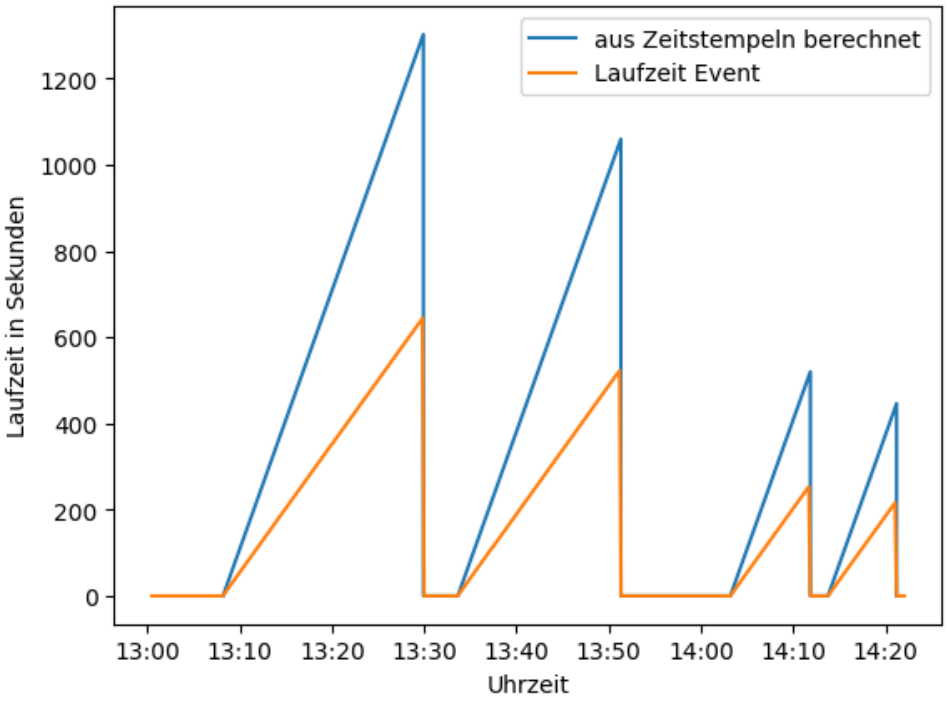

Abbildung 4.3: Unterschied zwischen dem Laufzeit Event und der aus den Zeitstempeln berechneten Laufzeit (Quelle: eigene Darstellung)

```
200,DMG.PRODUCTION.TIMES.ACT_CYCLE,DEFAULT,3,s,2022-11-16T15:13:50.373+01:00
200,DMG.PRODUCTION.TIMES.ACT_CYCLE,DEFAULT,8,s,2022-11-16T15:14:00.452+01:00
```

Quellcode 4.1: Die Laufzeit wird von Event eins auf Event zwei um fünf Sekunden erhöht, obwohl zwischen den beiden Zeitstempeln über zehn Sekunden liegen

Die Laufzeit eines NC-Programms errechnet sich also aus der Zeitdifferenz zwischen Bearbeitungsbeginn und Bearbeitungsende. Außerdem müssen die Zeiten, in denen die Maschine angehalten wurde, abgezogen werden. Die Laufzeit eines NC-Programms wird also vereinfacht wie folgt berechnet:

$$Laufzeit_{NcProgramm} = \sum_{i=1}^{n} (t_{Stopp\ i} - t_{Start\ i})$$

Es werden also die Zeiten zwischen Start und Stopp einer Verarbeitung berechnet und aufsummiert. Die Zeit $t_{Stop\ n}$ ist der Zeitpunkt, zu dem eine Bearbeitung beendet wurde, d.h. der Zeitstempel einer `"ABORTED"`-Nachricht oder einer Nachricht, in der der Teilezähler erhöht wurde. Im einfachsten Fall wird die Bearbeitung nicht gestoppt und es gilt $n = 1$. In diesem Fall ist die Bearbeitungszeit die Differenz zwischen dem Ende der Bearbeitung und dem Beginn der Bearbeitung. Es kann auch vorkommen, dass mehrere Start- oder Stoppereignisse hintereinander auftreten. Diese werden nur berücksichtigt, wenn zuvor ein Ereignis des anderen Typs aufgetreten ist. Eine vereinfachte Funktion, die die Laufzeiten aus einer Liste von Paaren aus Ereignistyp und Zeitstempel berechnet, ist im Quellcode 4.2 zu sehen.

```python
def getRuntimeFrom(eventTypesAndTimestamps):
    runtime = 0
    lastStart = None
    extractedRuntimes = []
    for eventType, timestamp in eventTypesAndTimestamps:
        if eventType == "RUNNING" and not lastStart:
            lastStart = timestamp
        elif eventType == "STOPPED" and lastStart:
            runtime += timestamp - lastStart
            lastStart = None
        elif eventType == "ABORTED":
            if lastStart:
                runtime +=  timestamp - lastStart
            extractedRuntimes.append(runtime)
            lastStart = None
            runtime = 0
    return extractedRuntimes
```

Quellcode 4.2: Vereinfachte Berechnung von Bearbeitungszeiten aus einer Liste von Events und Zeitstempeln (Beispielcode in Python)

In Abbildung 4.4 ist die zeitliche Abfolge der Nachrichten bei einer Bearbeitung dargestellt. Anhand dieses Beispiels soll demonstriert werden wie die Laufzeit berechnet wird.

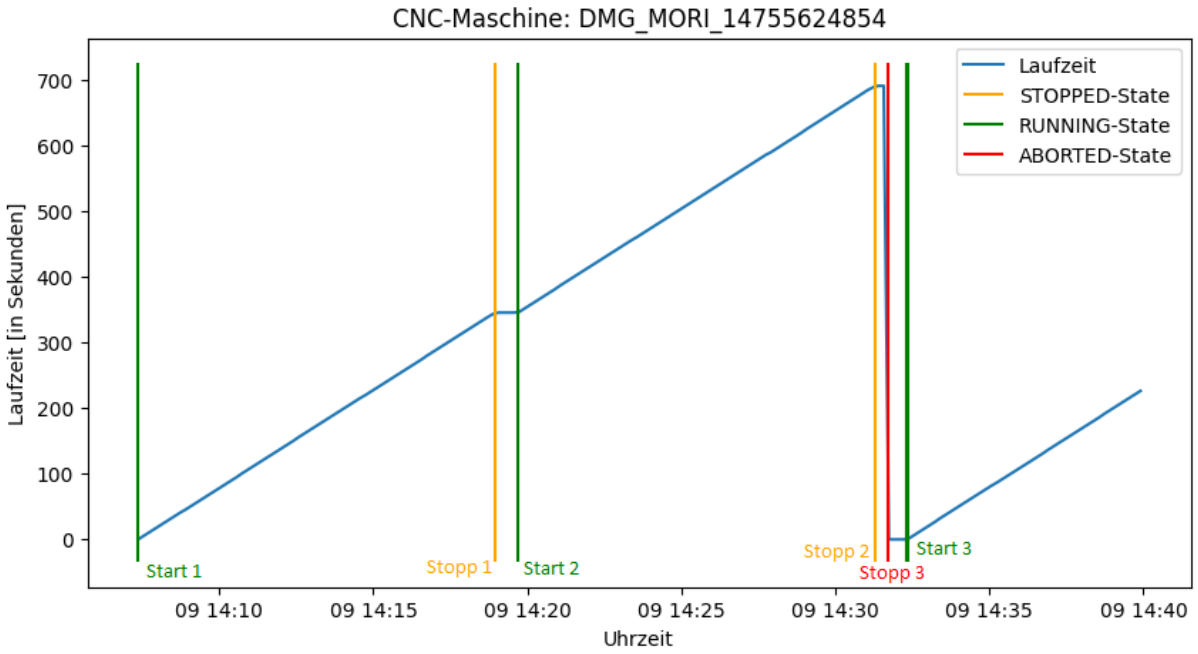

Abbildung 4.4: Laufzeit eines einzelnen NC-Programms mit eingeblendeten RUNNING, STOPPED und ABORTED Nachrichten (Quelle: eigene Darstellung)

$t_{Start\ 1} = 14{:}07{:}20$ Uhr (Running-Event)
$t_{Stopp\ 1} = 14{:}18{:}58$ Uhr (Stopped-Event)
$t_{Start\ 2} = 14{:}19{:}42$ Uhr (Running-Event)
$t_{Stopp\ 2} = 14{:}31{:}18$ Uhr (Stopped-Event)
$t_{Stopp\ 3} = 14{:}31{:}41$ Uhr (Aborted-Event)

$$Laufzeit_{NcProgramm} = t_{Stopp\ 1} - t_{Start\ 1} + t_{Stopp\ 2} - t_{Start\ 2} =$$
$$= 14{:}18{:}58\text{h} - 14{:}07{:}20\text{h} + 14{:}31{:}18\text{h} - 14{:}19{:}42\text{h} =$$
$$= 11{:}38\text{min} + 11{:}36\text{min} =$$
$$= 23{:}14\text{min}$$

Der Zeitstempel $t_{Stopp\ 3}$ wird in der Berechnung nicht berücksichtigt, da wie vorher beschrieben nur Stopp-Events berücksichtigt werden, wenn sie auf ein Start-Event folgen. Da $t_{Stopp\ 3}$ auf ein Stopp-Event $(t_{Stopp\ 2})$ folgt, wird $t_{Stopp\ 3}$ nicht berücksichtigt.

Um die Laufzeiten für einzelne NC-Programme extrahieren zu können, müssen also Folgende Ereignisse erfasst werden:

1. Laden eines neuen NC-Programms
2. Start der Bearbeitung (Maschinenzustand wechselt auf `"RUNNING"`)
3. Pausieren der Bearbeitung (Maschinenzustand wechselt auf `"STOPPED"`)
4. Ende der Bearbeitung (Maschinenzustand wechselt auf `"ABORTED"` oder der Teilezähler wird erhöht)

4.4 Architektur der Daten-Pipeline

Auch wenn für die aktuellen Anforderungen keine Echtzeitfähigkeit erforderlich ist, basiert die entwickelte Pipeline-Architektur auf der κ-Architektur. Sollten in Zukunft Anwendungsfälle hinzukommen, die Echtzeitfähigkeit erfordern, kann die bestehende Architektur verwendet werden und es muss keine neue entwickelt werden. Als Messaging-System wird Apache Kafka verwendet. Für die eigentliche Datenverarbeitung wurde ein Microservice in Python entwickelt. Dieser liest die Daten aus einem Kafka-Topic, führt die Verarbeitung durch und legt die Ergebnisse schließlich in einer Datenbank ab. Auf einzelne Details der Architektur wird im Folgenden näher eingegangen.

Um die von einem MDC gesammelten Daten in die Cloud-Umgebung zu bekommen, unterstützt der MDC die Übertragungsprotokolle OPC UA, MTConnect sowie MQTT [30]. Da MQTT bereits für ein bestehendes System im Unternehmen verwendet wurde, werden für die im Folgenden vorgestellte Architektur, wie bereits im Abschnitt 4.1 beschrieben, MQTT und AMQP zur Übertragung verwendet.

Die Kombination von RabbitMQ und dem zugehörigen MQTT-Plugin ermöglicht es, Daten über MQTT zu empfangen und über das AMQP-Protokoll weiterzuleiten [39, 42]. Ein Vorteil von AMQP gegenüber MQTT ist, dass mehrere Consumer mit einer Queue verbunden werden können. Die Nachrichten in dieser Queue werden dann im Round-Robin-Verfahren auf die Consumer verteilt. Auf diese Weise ist es möglich, durch Hinzufügen weiterer Consumer ein skalierbares System zu schaffen. Bei MQTT würden

alle Consumer eine Nachricht erhalten, wenn sie das entsprechende Topic abonniert haben [39]. Auch mit MQTT ist eine skalierbare Verarbeitung möglich, indem ein Consumer nicht alle Topics abonniert, sondern verschiedene Topics auf verschiedene Consumer verteilt, dies bedeutet aber einen höheren Konfigurationsaufwand und weniger Flexibilität [23].

Wie in Abbildung 4.5 dargestellt, findet die Datenverarbeitung nicht direkt nach dem Empfang der Daten statt. Ein Dienst (hier `Receiver` genannt) empfängt die Daten über AMQP, wandelt die unstrukturierten Daten in ein strukturiertes Datenformat um und schreibt sie in ein Kafka-Topic. Ein Vorteil dieses Ansatzes ist, dass Kafka sicherstellt, dass alle Nachrichten mit dem gleichen Schlüssel auf der gleichen Partition und damit beim gleichen Consumer landen [43]. In dem hier vorgestellten Beispiel wird als Schlüssel die Kennung der CNC-Maschine verwendet. Das bedeutet, dass alle Nachrichten einer Maschine automatisch von einer Instanz verarbeitet werden. Dies ist notwendig, da Informationen nur aus der Abfolge von Nachrichten einer Maschine gewonnen werden können, nicht aber aus einzelnen unabhängigen Nachrichten. Dieses Verhalten kann auch mit MQTT bzw. AMQP nachgebildet werden, allerdings müssten dann wieder verschiedene Instanzen verschiedene Topics bzw. Routingkeys abonnieren, was einen erhöhten Konfigurationsaufwand bedeutet. Durch die Verwendung von Kafka können alle Consumer ein Kafka-Topic abonnieren und die Last wird automatisch auf die Consumer verteilt, da jede zum Topic gehörende Partition von einem Consumer gelesen wird [43]. Ein weiterer Vorteil der Verwendung von Kafka ist, dass Kafka die Daten persistent speichert. Dies bedeutet, dass im Falle einer Änderung in der Berechnungslogik, die bereits gelesenen Daten einfach erneut gelesen und somit mit der neuen Logik analysiert werden können [43].

Die eigentliche Verarbeitung der Daten erfolgt in einem weiteren Dienst, der eigentlichen Daten-Pipeline. Dieser Dienst ist in Abbildung 4.5 durch einen roten Kasten gekennzeichnet. Der Kern dieser Bachelorarbeit beschäftigt sich mit der Implementierung dieses Subsystems. Die Pipeline ist so aufgebaut, dass die Datenverarbeitung weitgehend unabhängig von den anderen Systemen genutzt werden kann. Wie in Abbildung 4.5 zu sehen ist, erhält die Pipeline im Prototypen die zu verarbeitenden Daten von einem Kafka-Topic. Ohne große Anpassungen kann die Pipeline aber auch direkt über MQTT übertragene Events verarbeiten. Diese Implementierung kann auch für die Batch-Verarbeitung genutzt werden, indem z.B. alle eingehenden Events zunächst in einer Datenbank gespeichert werden und einmal täglich alle gespeicherten

Events durch die Pipeline verarbeitet werden. Gleiches gilt für ausgehende Daten. Für den Prototyp wurde entschieden, die verarbeiteten Daten, also die extrahierten Laufzeiten, in einer PostgreSQL-Datenbank zu speichern. Denkbar wäre aber z.B. auch eine Architektur, bei der diese Daten in ein neues Kafka-Topic geschrieben und so an andere Dienste verteilt werden.

Außerdem bietet die Pipeline eine REST-Schnittstelle für andere Dienste an. REST steht dabei für „Representational State Transfer“. Darunter versteht man eine auf HTTP basierendes, Zustandsloses Kommunikationskonzept zwischen mehreren Diensten [44]. Über diese Schnittstelle ist es anderen Diensten möglich, die Ergebnisse der Pipeline abzufragen. Für den Softwareprototyp können auch die notwendigen Metadaten, wie z.B. die Verknüpfung des NC-Programmnamens mit der zugrundeliegenden Kalkulation, über die REST-Schnittstelle eingepflegt werden.

Für den Einsatz in einem produktiven Umfeld werden außerdem weitere Dienste benötigt. Zum einen ein Dienst zur Verwaltung der Kalkulationen. Von diesem Dienst werden die Informationen, die mit den Laufzeiten verknüpft werden sollen, verwaltet. Also beispielsweise zugrundeliegendes CAD-Modell, kalkulierte Laufzeit und Name des NC-Programms. Zukünftig sollen von diesem Dienst die benötigten Informationen über eine REST-Schnittstelle abgefragt werden. Für den Prototypen wird davon ausgegangen, dass diese Informationen in einer Datenbanktabelle vorliegen und mit Hilfe der Structured Query Language (kurz SQL) abgefragt werden können.

Außerdem soll ein weiterer Dienst die extrahierten und angereicherten Daten verwenden um ein Machine-Learning Modell zu trainieren und dadurch die vorhergesagten Bearbeitungszeiten zu verbessern. Da diese Systeme aktuell noch nicht existieren sind diese in Abbildung 4.5 nur gestrichelt angedeutet. Diese sind kein Bestandteil dieser Bachelorarbeit und dienen in Abbildung 4.5 nur zum besseren Verständniss des Ziels.

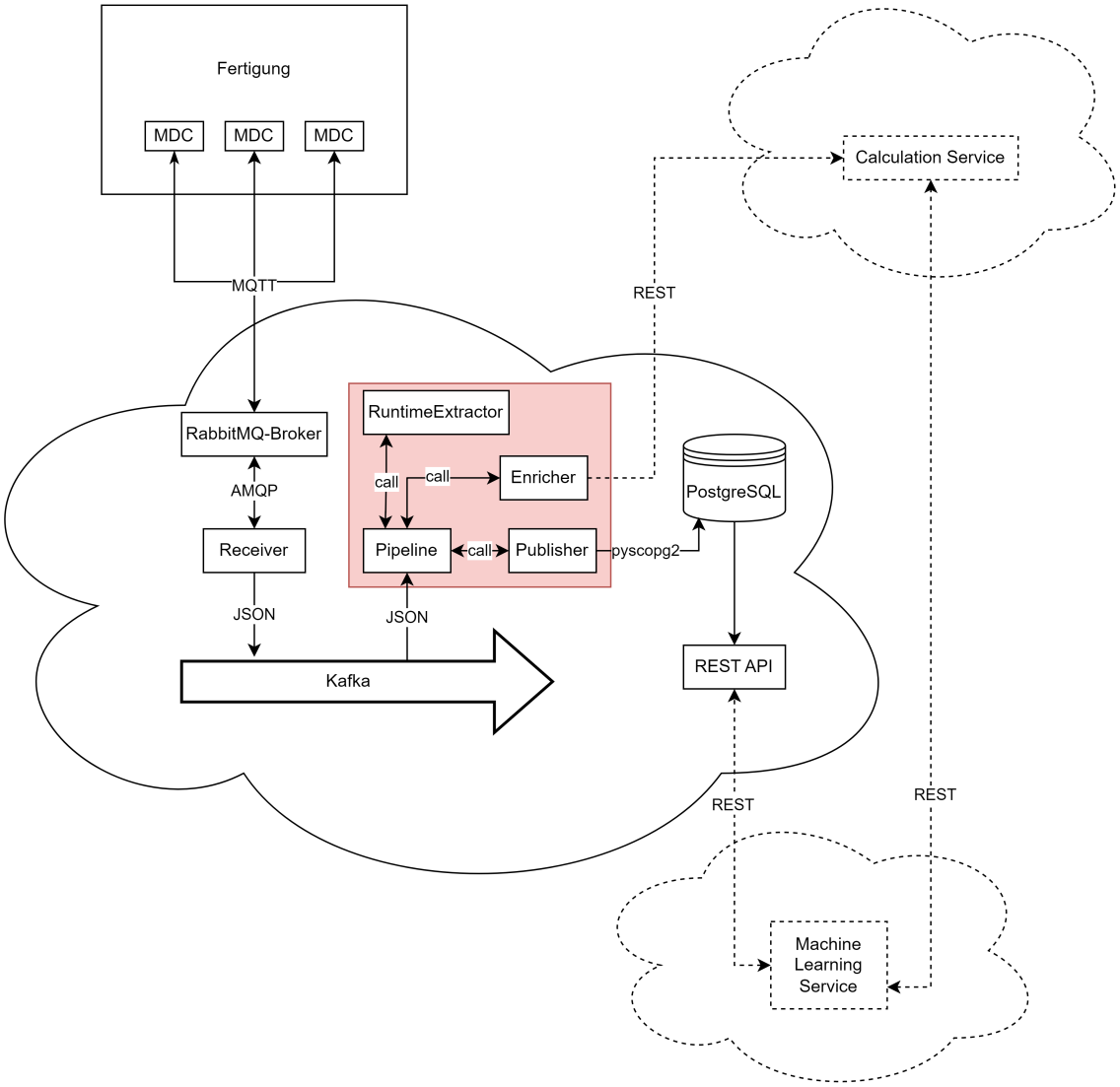

Abbildung 4.5: Aufbau der Architektur im Groben

4.5 Klassen zum Verarbeiten der Datenströme

Die Abbildung 4.6 zeigt die wichtigsten Klassen der Daten-Pipeline sowie deren Methoden und Attribute. Außerdem sind die Beziehungen zwischen den einzelnen Klassen ersichtlich. Das in Abbildung 4.6 dargestellte Klassenkartendiagramm entspricht dabei dem in Abbildung 4.5 rot hinterlegten Subsystem. Im Folgenden werden die wichtigsten Klassen näher beschrieben.

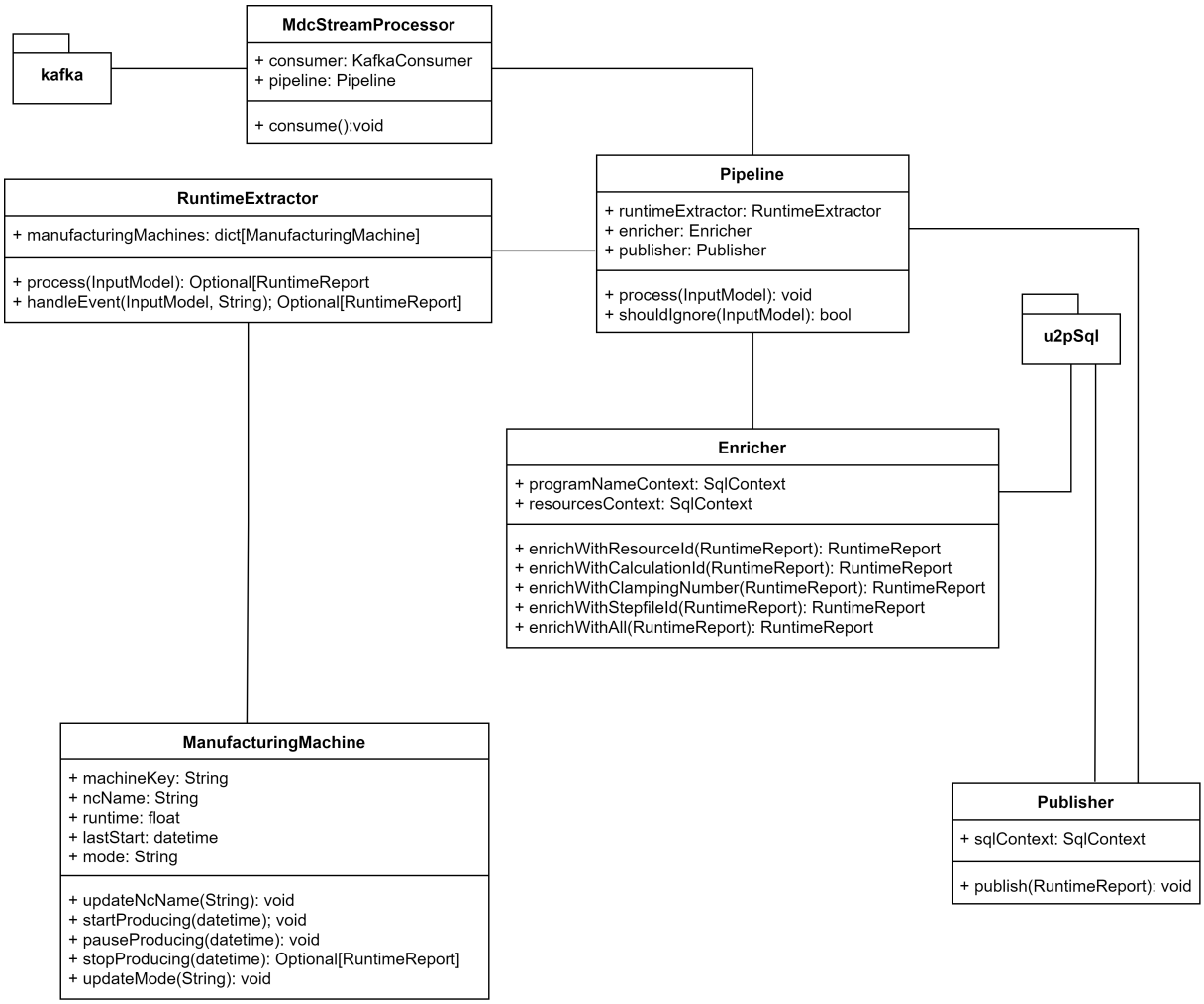

Abbildung 4.6: Darstellung der verwendeten Klassen innerhalb des Systems (Quelle: eigene Darstellung)

4.5.1 Die Klasse MdcStreamProcessor

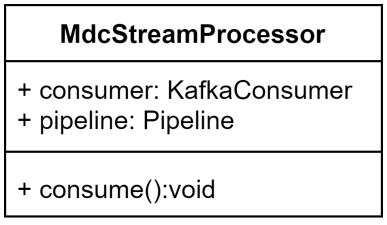

Abbildung 4.7: Die Klasse MdcStreamProcessor
(Quelle: eigene Darstellung)

Diese Klasse wird verwendet, um neue Ereignisse, die über Kafka gesendet werden, zu empfangen und an die eigentliche Datenverarbeitung weiterzuleiten. Zu diesem Zweck enthält die Klasse die beiden Attribute `consumer` vom Typ `KafkaConsumer` und `pipeline` vom Typ `Pipeline`. Der `consumer` kann verwendet werden, um Nachrichten aus einem Kafka-Topic zu lesen. Für die Weiterverarbeitung der Nachrichten ist die `pipeline` zuständig, an die die empfangenen Nachrichten weitergeleitet werden.

Durch die Trennung von Nachrichtenempfang und -verarbeitung ist ein späterer Wech-

sel des verwendeten Message Queue Systems sehr einfach möglich. Soll statt Kafka ein anderes System verwendet werden, muss lediglich eine neue Klasse `MdcStreamProcessor` entwickelt werden, der Rest des Systems bleibt davon unberührt. Denkbar wäre z.B. eine Implementierung, die Nachrichten direkt über MQTT empfängt.

Auch eine Verwendung in einem Batchverarbeitungssystem ist so möglich. Anstelle der Klasse `MdcStreamProcessor` wird eine neue Klasse verwendet, die die Daten nicht von einem Kafka-Topic liest, sondern beispielsweise aus einer Datenbank. Der Rest des Systems bleibt dabei unverändert.

4.5.2 Die Klasse Pipeline

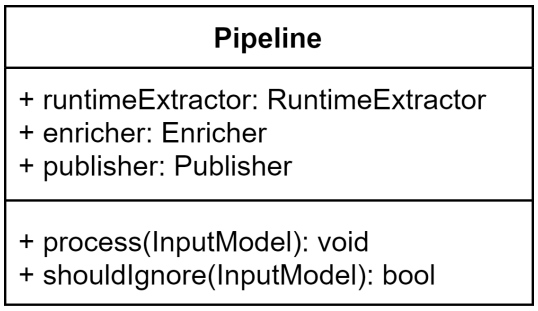

Abbildung 4.8: Die Klasse Pipeline (Quelle: eigene Darstellung)

Innerhalb dieser Klasse werden die einzelnen Schritte der Daten-Pipeline sowie deren Reihenfolge definiert. Für den hier vorgestellten Prototypen besteht die Pipeline aus den Schritten `runtimeExtractor`, um die Programmlaufzeiten aus dem Datenstrom zu extrahieren, `enricher`, um die erhaltenen Laufzeiten mit Metadaten anzureichern, sowie dem Schritt `publisher`, um die Ergebnisse in einer relationalen Datenbank abzulegen. Sollen in Zukunft zusätzliche Schritte, wie z.B. eine visuelle Darstellung der gesammelten Daten, eingefügt werden, so erfolgt die Einbindung des neuen Schrittes ebenfalls in der Klasse `Pipeline`.

Die Methode `process` wird zur Verarbeitung eines eingehenden Ereignisses verwendet. Der Input-Parameter dieser Methode ist vom Typ `InputModel`. Ein `InputModel` ist eine in ein Objekt gepackte Nachricht, die empfangen wurde. Innerhalb von `process` wird zunächst die Methode `shouldIgnore` aufgerufen. Diese prüft, ob die aktuell zu bearbeitende Nachricht für die Berechnung der Laufzeit benötigt wird, wenn nicht, gibt sie `False` zurück und die Methode `process` wird beendet. Auf diese Weise werden zunächst alle eingehenden Nachrichten an die `Pipeline` übergeben und erst hier gefiltert, welche Nachrichten benötigt werden. Sollen zu einem späteren Zeitpunkt auch andere Daten erfasst und dafür andere Nachrichtentypen verarbeitet werden, muss lediglich die Methode `shouldIgnore` angepasst werden.

Wird die Nachricht jedoch zur Berechnung benötigt, wird sie an die Methode `process` der Klasse `RuntimeExtractor` übergeben. Die Klasse `RuntimeExtractor` übernimmt die Extraktion der Bearbeitungszeiten für die einzelnen NC-Programme aus den eingehenden Ereignissen. Konnte eine Bearbeitungszeit extrahiert werden, gibt diese Methode ein Objekt vom Typ `RuntimeReport` zurück. Ein `RuntimeReport` ist ein Datencontainer, der zunächst den Namen des NC-Programms, die berechnete Bearbeitungszeit und den Routing Key der CNC-Maschine enthält.

Wird vom `RuntimeExtractor` ein `RuntimeReport` zurückgegeben, so wird dieser an die Methode `enrichWithAll` der Klasse `Enricher` übergeben. Diese ist dafür verantwortlich, den `RuntimeReport` mit zusätzlichen Metadaten aus der Fertigungsplanung anzureichern.

Abschließend wird der `RuntimeReport` an die Methode `publish` der Klasse `Publisher` übergeben. Diese speichert den empfangenen Datensatz und stellt ihn so anderen Diensten zur Verfügung.

4.5.3 Die Klasse RuntimeExtractor

Die zentrale Klasse für die Extraktion der Bearbeitungszeiten aus den eingehenden Nachrichten ist die Klasse `RuntimeExtractor`. Diese Klasse verwaltet intern alle bekannten CNC-Maschinen und verarbeitet die eingehenden Nachrichten.

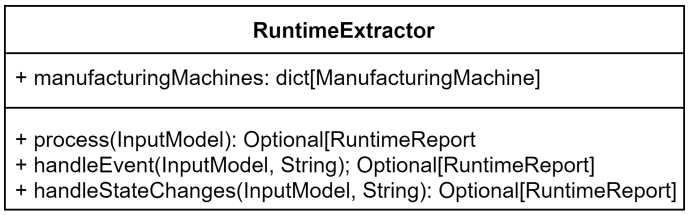

Abbildung 4.9: Die Klasse RuntimeExtractor
(Quelle: eigene Darstellung)

Erhält die Klasse `Pipeline` eine neue Nachricht von einer der angeschlossenen CNC-Maschinen, so ruft sie die Methode `process` der Klasse `RuntimeExtractor` auf. Dort wird zunächst geprüft, von welcher CNC-Maschine die Nachricht kommt und ob diese bereits bekannt ist. Ist die CNC-Maschine dem `RuntimeExtractor` noch nicht bekannt, so wird eine neue Instanz der Klasse `ManufacturingMachine` erzeugt und diese als Schlüssel-Wert-Paar in der Variablen `manufacturingMachines` gespeichert. Der Schlüssel ist das Top-Level-Topic der MQTT-Nachricht, d.h. die Maschinenkennung des MQTT-Topics.

Anschließend wird die Nachricht zusammen mit der Maschinenkennung an die Methode `handleEvent` übergeben. Ist die CNC-Maschine bereits bekannt, wird direkt die Methode `handleEvent` aufgerufen. Diese Methode sorgt dafür, dass je nach Art der

eingehenden Nachricht die entsprechende Methode im zugehörigen Objekt der Klasse `ManufacturingMachine` aufgerufen wird. Auf diese Weise erhält die `ManufacturingMachine` den aktuellen Zustand der realen CNC-Maschine. Beispielsweise wird der Name des aktuellen NC-Programms gesetzt oder der Maschinenmodus geändert. Außerdem wird der Bearbeitungszustand der Maschine aktualisiert, wofür zusätzlich die Methode `handleStateChanges` verwendet wird. Diese sorgt dafür, dass in der `ManufacturingMachine` die entsprechenden Methoden zum Starten, Beenden und Pausieren einer Bearbeitung mit den entsprechenden Zeitstempeln der Nachricht aufgerufen wird.

Liefert die in der `ManufacturingMachine` aufgerufene Methode ein `RuntimeReport`-Objekt zurück, d.h. konnte eine Laufzeit aus dem bisherigen Datenstrom extrahiert werden, so wird dieses Objekt auch von den Methoden `handleEvent` und `process` an die `Pipeline` zurückgegeben. Auf diese Weise erhält die `Pipeline` die extrahierten Bearbeitungszeiten und kann diese weiterverarbeiten.

4.5.4 Die Klasse ManufacturingMachine

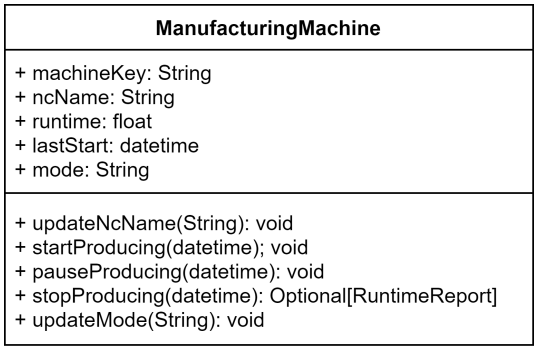

Abbildung 4.10: Die Klasse ManufacturingMachine (Quelle: eigene Darstellung)

Die Klasse `ManufacturingMachine` stellt ein virtuelles Abbild einer realen CNC-Maschine dar. Dabei wird der aktuelle Bearbeitungszustand der Maschine durch die beiden in Abbildung 4.11 dargestellten Zustandsautomaten repräsentiert.

Der erste Zustandsautomat ist für den aktuellen Bearbeitungszustand zuständig. Mögliche Zustände in diesem Zusammenhang sind `RUNNING`, wenn die Maschine gerade ein Werkstück bearbeitet, `PAUSED`, wenn die aktuelle Bearbeitung pausiert ist und `STOPPED`, wenn die aktuelle Bearbeitung abgeschlossen ist. Zum Auslösen der Zustandswechsel stellt die Klasse `ManufacturingMachine` die Methoden `startProducing`, `pauseProducing` und `stopProducing` zur Verfügung. Alle diese Methoden erhalten als Eingabeparameter ein Objekt vom Typ `datetime`, also einen Zeitstempel mit Datum und Uhrzeit. Diese Zeitstempel stammen aus dem ursprünglichen MDC-Ereignis und werden zur Berechnung der Laufzeiten verwendet, wie im Abschnitt 4.3 beschrieben.

Da nur die Zeiten relevant sind, in denen sich die CNC-Maschine im Automatikbetrieb befindet, verwaltet die Klasse `ManufacturingMachine` in einem weiteren Zustandsautomaten den aktuellen Modus der Maschine. Mögliche Zustände sind `AUTO`, wenn sich die Maschine im Automatikbetrieb befindet, bzw. `JOG` und `MDA`, wenn die Maschine manuell bzw. halbautomatisch gesteuert wird. Um diesen Zustandswechsel herbeizuführen, stellt die Klasse `ManufacturingMachine` die Methode `updateMode` zur Verfügung. Diese erhält als Eingabeparameter einen `String`, der den aktuellen Modus repräsentiert.

Beim Wechsel in den Zustand `RUNNING`, d.h. beim Aufruf der Methode `startProducing`, wird der übergebene Zeitstempel in der Variablen `lastStart` gespeichert. Wird anschließend in den Zustand `PAUSED` gewechselt, wird die Zeitdifferenz zwischen dem in `lastStart` gespeicherten Zeitstempel und dem übergebenen Zeitstempel berechnet und zur Variablen `runtime` addiert. Danach wird der Wert aus `lastStart` gelöscht. Auf diese Weise wird verhindert, dass Zeiten mehrfach addiert werden, wenn die Methode `pauseProducing` mehrmals hintereinander aufgerufen wird. Wechselt die Maschine in den Zustand `STOPPED`, wird zunächst, genau wie beim Wechsel in den Zustand `PAUSED`, die Zeitdifferenz zwischen dem übergebenen Zeitstempel und `lastStart`, falls vorhanden, berechnet und zu `runtime` addiert. Anschließend wird geprüft, ob sich die Maschine gerade im Modus `AUTO` befindet, falls ja, wird die berechnete Laufzeit zusammen mit dem Namen des aktuellen NC-Programms als `RuntimeReport` zurückgegeben. Befindet sich die Maschine nicht im `AUTO`-Modus, wird nichts zurückgegeben. Abschließend werden in beiden Fällen die Variablen `runtime` und `lastStart` zurückgesetzt. Der Zyklus kann für die nächste Bearbeitung von vorne beginnen.

Wird auf der CNC-Maschine ein neues NC-Programm geladen, so wird dies über die Methode `updateNcName` der Klasse `ManufacturingMachine` mitgeteilt. Diese speichert den Namen des aktuellen NC-Programms in der Variablen `ncName`. Der Name des NC-Programms wird zusammen mit der berechneten Bearbeitungszeit im `RuntimeReport` zurückgegeben.

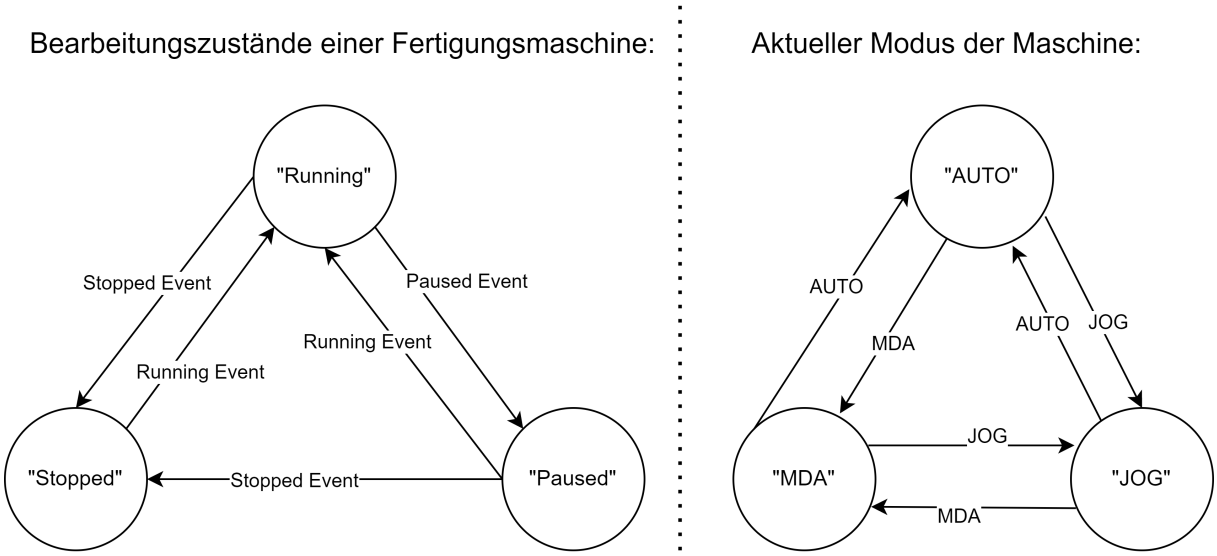

Abbildung 4.11: Zustandsautomaten einer virtuellen Fertigungsmaschine (Quelle: eigene Darstellung)

4.5.5 Die Klasse Enricher

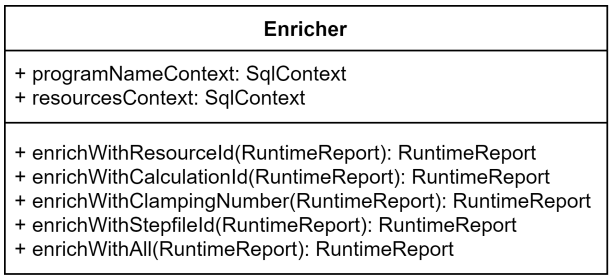

Abbildung 4.12: Die Klasse Enricher (Quelle: eigene Darstellung)

Die Klasse `Enricher` ist dafür zuständig, die erhaltenen `RuntimeReport`s, die aus dem Namen des NC-Programms, der zugehörigen Laufzeit und dem Routing Key der CNC-Maschine bestehen, mit weiteren Metadaten aus der Fertigungsplanung anzureichern. Diese Metadaten sollen folgende Fragen beantworten:

- Auf welcher Maschine wurde das Bauteil bearbeitet?
- Welche Kalkulation liegt der Bearbeitung zugrunde?
- Welche Aufspannung wurde bearbeitet?
- Welches CAD-Modell liegt der Bearbeitung zugrunde?

Um diese Fragen beantworten zu können, muss eine Verknüpfung zwischen dem `routing_key`, unter dem die Nachrichten der Maschine veröffentlicht werden, und der `resource_id`, unter der die Maschine im bestehenden System verwaltet werden, hergestellt werden. Da diese Verknüpfung im bestehenden System noch nicht existiert, wurde für die prototypische Implementierung eine Datenbanktabelle mit genau diesen Einträgen angelegt (siehe `resources` Tabelle in Abbildung 4.13).

Außerdem muss eine Verknüpfung zwischen einem NC-Programm und einer Kalkulation hergestellt werden können. Da die Erstellung von NC-Programmen noch nicht Bestandteil der up2parts Cloud ist, existiert diese Verknüpfung im bestehenden System noch nicht. Um eine Kalkulation innerhalb der up2parts Cloud zu identifizieren, kann die `calculation_id` verwendet werden. Komplexere Bauteile können oft nicht in einer Aufspannung hergestellt werden, sondern müssen in mehreren Aufspannungen gefertigt werden. Aus diesem Grund wird zusätzlich zur `calculation_id` auch die Nummer der Aufspannung gespeichert (`clamping_number`). Für eine Implementierung in das bestehende System müssen zumindest diese Informationen zur Verfügung stehen. Für den im Rahmen dieser Arbeit entstandenen Prototypen wird davon ausgegangen, dass eine Datenbanktabelle mit diesen Informationen existiert, auf die zugegriffen werden kann (siehe Tabelle `program_names` in Abbildung 4.13).

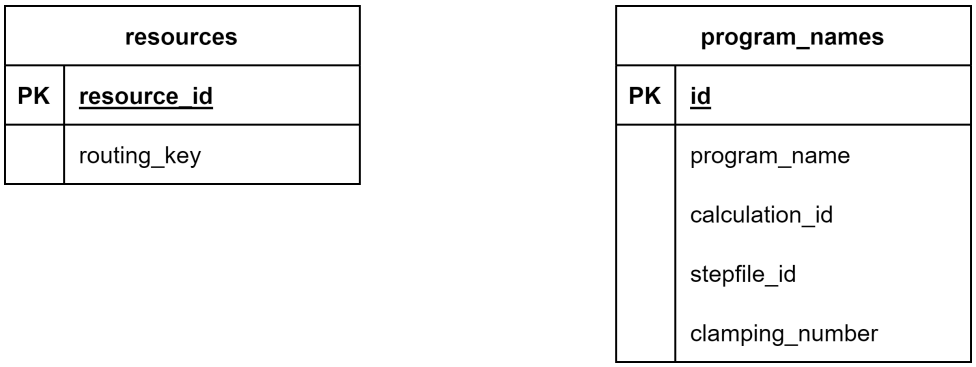

Abbildung 4.13: Schema der benötigten Tabellen (Quelle: eigene Darstellung)

Um die extrahierten Bearbeitungszeiten mit zusätzlichen Metadaten anreichern zu können, hält die Klasse `Pipeline` eine Instanz der Klasse `Enricher`. Sobald die `Pipeline` eine extrahierte Bearbeitungszeit erhält, ruft sie die Methode `enrichWithAll` der Klasse `Enricher` auf und übergibt den `RuntimeReport`. Die Methode `enrichWithAll` holt mit Hilfe des `sqlContext` die Metadaten aus der Datenbank und fügt sie dem `RuntimeReport` hinzu. Anschließend wird der `RuntimeReport` mit den angehängten Metadaten zurückgegeben. Die Klasse `Enricher` bietet zusätzlich die Methoden `enrichWithResourceId`, `enrichWithCalculationId`, `enrichWithClampingNumber` und die Methode `enrichWithStepfileId`, um nur einzelne Metadaten aus der Datenbank abzufragen. Diese werden jedoch nur intern in der Methode `enrichWithAll` verwendet.

Für eine produktive Implementierung würden diese Metadaten wahrscheinlich von einem anderen Dienst bereitgestellt werden. Es ist z.B. denkbar, dass diese Informationen über eine REST-Schnittstelle bezogen werden können. In diesem Fall muss die Enricher-Klasse so angepasst werden, dass anstelle einer Datenbankabfrage eine

Anfrage an diese REST-Schnittstelle gestellt wird. Die restliche Implementierung des Systems muss nicht angepasst werden.

4.5.6 Die Klasse Publisher

runtimes	
PK	id
	nc_program_name
	runtime
	machine_key
	calculation_id
	stepfile_id
	resource_id
	clamping_number

Abbildung 4.14: Die Tabelle runtimes (Quelle: eigene Darstellung)

Um einen Nutzen aus den gesammelten Daten ziehen zu können, müssen diese, wie im Anwendungsfall UC-4 beschrieben, gespeichert werden. Für den Prototyp wird dazu eine relationale Datenbank verwendet. Als Datenbankmanagementsystem wird für den Prototyp PostgreSQL verwendet. Die extrahierten und mit Metadaten angereicherten Ergebnisse werden in einer Tabelle gespeichert. Das verwendete Tabellenschema ist in Abbildung 4.14 dargestellt.

In dieser Tabelle werden folgende Daten gespeichert. Eine eindeutige `id`, die als Primärschlüssel dient. Über diese `id` können Einträge an anderer Stelle referenziert werden. In der Spalte `nc_program_name` wird der Name des NC-Programms gespeichert, zu dem der Eintrag gehört. Dieser kann in der Tabelle auch mehrfach vorkommen. Da die meisten Werkstücke mehr als einmal gefertigt werden, gibt es auch mehrere Einträge in der Datenbank. Die Spalte `runtime` enthält die extrahierte Bearbeitungszeit für diesen Datensatz. Die Spalte `machine_key` enthält den für die Maschine verwendeten Routing Key. Die Spalte `calculation_id` enthält einen Fremdschlüssel, der auf die entsprechende Kalkulation im bestehenden System verweist. Über diesen Fremdschlüssel ist es später möglich, die kalkulierte Bearbeitungszeit mit der tatsächlichen Bearbeitungszeit zu vergleichen. In der Spalte `stepfile_id` wird ein Fremdschlüssel gespeichert, der auf ein CAD-Modell verweist. Die Spalte resource_id enthält den Fremdschlüssel, der auf die verwendete CNC-Maschine im bestehenden System verweist. Diese ist z.B. auch in einer Kalkulation hinterlegt, so dass überprüft werden kann, ob die Bearbeitung tatsächlich auf der Maschine durchgeführt wurde, für die die Kalkulation erstellt wurde. Schließlich wird auch die Nummer der Aufspannung in der Spalte `clamping_number` gespeichert. Mit dieser Information können verschiedene Aufspannungen eines Bauteils unterschieden werden.

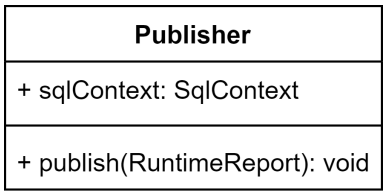

Abbildung 4.15: Die Klasse Publisher (Quelle: eigene Darstellung)

Um die Ergebnisse in der Datenbank speichern zu können, hält die Klasse `Pipeline` eine Instanz der Klasse `Publisher`. Sobald die angereicherten Ergebnisse in der Klasse `Pipeline` vorliegen, ruft diese die Methode `publish` der Klasse `Publisher` auf und übergibt die angereicherten Ergebnisse. Diese Methode speichert die übergebenen Ergebnisse mit Hilfe des `sqlContext` in der Datenbank.

Die Ablage der Daten in einer Datenbank ist nur ein Beispiel für die Umsetzung im Softwareprototyp. Es sind auch Anwendungsfälle denkbar, bei denen die Ergebnisse in einem neuen Kafka-Topic veröffentlicht oder über ein anderes Messaging-System verteilt werden. Um dies zu realisieren, muss eine neue Klasse `Publisher` entwickelt werden, die die Ergebnisse an ein Messaging-System übergibt, anstatt sie in einer Datenbank zu speichern. Alle anderen Klassen müssen dafür nicht geändert werden.

5. Ergebnisse

In diesem Kapitel werden die Ergebnisse der Bachelorarbeit vorgestellt. Zunächst wird auf die Ergebnisse einer Testfertigung eingegangen. Anschließend werden die zu verarbeitenden Datenmengen sowie der Durchsatz der vorgestellten Pipeline analysiert.

5.1 Test mit einer realen Zerspannung

Um die Funktionsweise der Daten-Pipeline zu validieren, wurde am 16.11.2022 eine Testfertigung durchgeführt. Dabei wurde das in Abbildung 5.1 dargestellte Bauteil auf einer mit einem MDC ausgestatteten Fräsmaschine (CNC-Fräsmaschine: DMG Mori DMC 1850 V, Steuerung: Heidenhain TNC 640) gefertigt. Bei dem Bauteil handelt es sich um ein Testteil, das für Versuche im Bereich der automatischen CAM-Programmierung (aus dem Englischen für Computer Aided Manufacturing) erstellt wurde. Die für diesen Versuch verwendeten NC-Programme wurden dabei automatisch erzeugt.

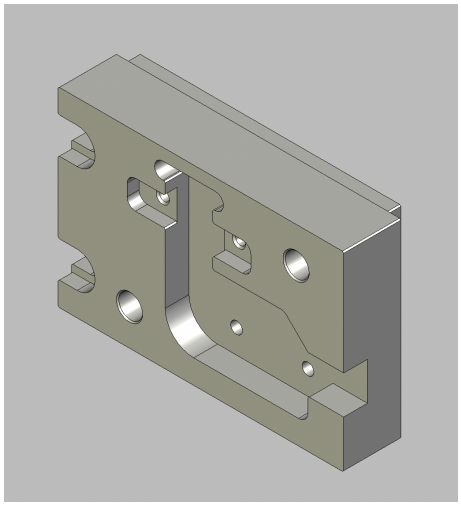

Abbildung 5.1: 3D CAD-Modell des zu fertigenden Testbauteils (Quelle: [45])

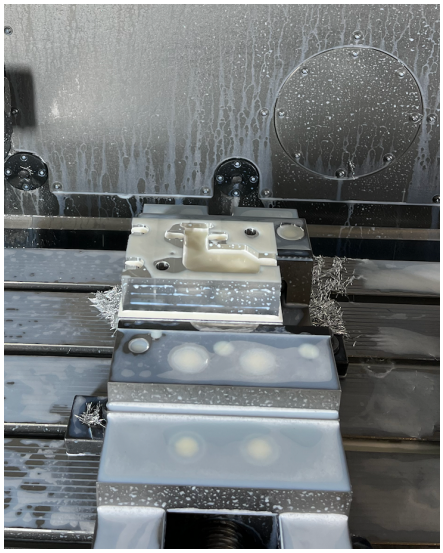

Abbildung 5.2: Das Bauteil während der Bearbeitung (Quelle: [45])

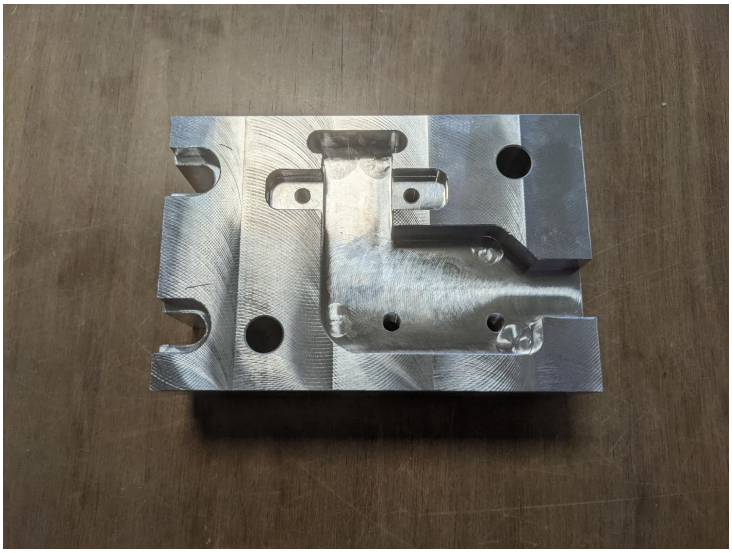

Abbildung 5.3: Das fertige Bauteil (Quelle: [45])

Ziel dieses Experiments war es, einerseits die mit dem Prototyp ermittelten Bearbeitungszeiten zu validieren und andererseits die Unterschiede zwischen den simulierten und den tatsächlichen Bearbeitungszeiten zu untersuchen.

Die Bearbeitung des in Abbildung 5.1 dargestellten Bauteils erfolgt in zwei Aufspannungen. In der ersten Aufspannung wird die in den Abbildungen 5.1 - 5.3 zu sehende Vorderseite bearbeitet. In der zweiten Aufspannung wird die nicht sichtbare Rückseite bearbeitet.

Im Rahmen dieses Tests sollten vor allem zwei Fragen geklärt werden. Zum einen: Welche ermittelte Bearbeitungszeit entspricht der tatsächlichen Bearbeitungszeit? Die Bearbeitungszeit aus dem entsprechenden Ereignis oder die aus den Zeitstempeln ermittelte Bearbeitungszeit (vergleiche Abschnitt refsec:analyse)? Zum anderen: Wie stark weicht die tatsächliche Bearbeitungszeit von der simulierten Bearbeitungszeit ab?

Vergleicht man die tatsächliche Bearbeitungszeit mit der Bearbeitungszeit im entsprechenden Ereignis des MDCs, so stellt man fest, dass die tatsächliche Bearbeitungszeit doppelt so lang ist wie die im Ereignis erfasste Bearbeitungszeit. Die aus den Zeitstempeln berechnete Bearbeitungszeit stimmt mit der tatsächlichen Bearbeitungszeit überein. Diese Abweichung ist vermutlich auf einen Fehler in der Software des MDCs oder der Maschinensteuerung zurückzuführen, was jedoch im Rahmen dieser Bachelorarbeit nicht abschließend geklärt werden konnte. Aufgrund dieses Fehlers wird, wie bereits im Abschnitt 4.3 beschrieben, die Bearbeitungszeit für die Implementierung des Softwareprototyps aus den Zeitstempeln berechnet.

Die Versuchsergebnisse sind in den Tabellen 5.1 für die erste Aufspannung und 5.2 für die zweite Aufspannung dargestellt. Der Unterschied zwischen dem ersten und zweiten Durchlauf jeder Aufspannung besteht darin, dass im ersten Durchlauf jeweils manuell eingegriffen und mit reduziertem Vorschub gearbeitet wurde, um Beschädigungen an Maschine und Werkzeug auszuschließen. Beim zweiten Durchlauf wurde nicht manuell eingegriffen, weshalb die Bearbeitung hier schneller erfolgte. In allen Fällen wurde die Bearbeitung nicht pausiert, weshalb für die Berechnung der Bearbeitungszeit nur der Start- und der Endzeitpunkt relevant sind.

Erwartungsgemäß sind die von der CNC-Steuerung simulierten Bearbeitungszeiten kürzer als die tatsächlichen. Dies liegt daran, dass bei der Simulation von einer

konstanten Vorschubgeschwindigkeit ausgegangen wird und die Beschleunigungs- und Abbremszeiten der Maschine nicht berücksichtigt werden. Zukünftig sollen diese Abweichungen mit Hilfe der Pipeline automatisiert ausgewertet werden.

Art des Events	Zeitstempel 1. Lauf	Zeitstempel 2. Lauf
Start:	14:08:12	14:33:46
Ende:	14:29:54	14:51:18

Laufzeit aus Zeitstempeln:	1302s	1052s
Laufzeit aus MDC Event:	644s	522s

Simulierte Laufzeit:	976s

Tabelle 5.1: Ergebnisse der ersten Aufspannung

Art des Events	Zeitstempel 1. Lauf	Zeitstempel 2. Lauf
Start:	15:03:09	15:13:44
Ende:	15:11:49	15:21:10

Laufzeit aus Zeitstempeln:	520s	446s
Laufzeit aus MDC Event:	253s	216s

Simulierte Laufzeit:	393s

Tabelle 5.2: Ergebnisse der ersten Aufspannung

5.2 Durchsatz der Daten-Pipeline

In diesem Abschnitt soll untersucht werden, wie viele Ereignisse die Pipeline verarbeiten kann und wo dabei die Grenzen liegen. In diesem Zusammenhang wird nur der Durchsatz der eigentlichen Verarbeitungslogik untersucht. Es wird also nicht untersucht, welche zusätzlichen Einschränkungen bei der Übertragung der Daten über die Netzwerkverbindungen auftreten.

Um dies zu untersuchen, wurde ein Kafka-Topic mit 100.000 Maschinenereignissen gefüllt. Die verwendeten Ereignisse stammen von den angeschlossenen Testmaschinen. Es handelt sich also nicht um künstlich erzeugte Daten, sondern die verwendeten Daten sind auf realen Maschinen aufgetreten. Anschließend wurde die Pipeline gestartet und gemessen, wie lange es dauert, diese 100.000 Nachrichten zu verarbeiten. Diese Vorgehensweise wurde mit insgesamt 600.000 verschiedenen aufgezeichneten Maschinenereignissen wiederholt, um ein möglichst aussagekräftiges Ergebnis zu erhalten.

Die Ergebnisse sind in den Abbildungen 5.4 und 5.5 dargestellt. Zur Verarbeitung von 100.000 Ereignissen benötigt die Pipeline zwischen 8,83 und 11,41 *Sekunden*, der Mittelwert liegt bei ca. 9,69 *Sekunden*. Im Durchschnitt kann die Pipeline mit der für die Untersuchung verwendeten Hard- und Softwarekonfiguration also ca. 10.318 *Nachrichten/Sekunde* verarbeiten.

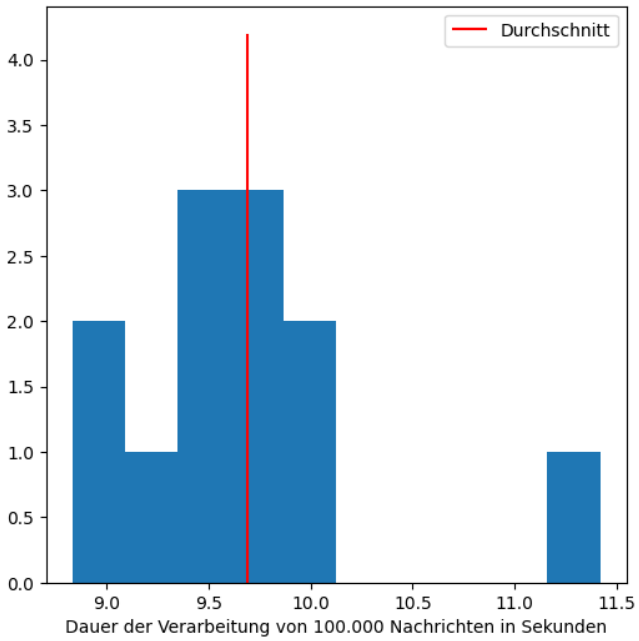

Abbildung 5.4: Verteilung der Dauer der Verarbeitung von 100.000 Nachrichten mit Hilfe der Pipeline (Quelle: eigene Darstellung)

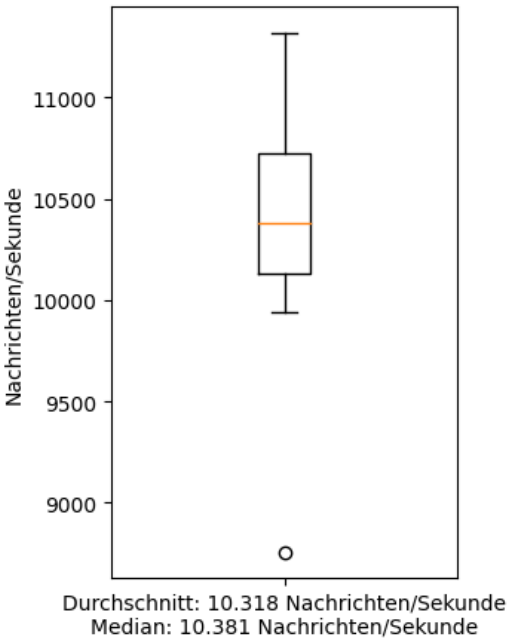

Abbildung 5.5: Verteilung des Datendurchsatz der Pipeline (Quelle: eigene Darstellung)

Es ist zu beachten, dass diese Messungen auf Standardhardware in einem Docker-Container durchgeführt wurden. Der Versuch wurde auf einem Microsoft Surface Book 2, mit Intel Core i7-8650U Prozessor und 16GB Arbeitsspeicher durchgeführt. Den Docker-Containern standen dabei 6 Prozessorkerne und 7GB Arbeitsspeicher zur Verfügung. Mit leistungsfähigerer Serverhardware sind hier vermutlich kürzere Verarbeitungszeiten zu erwarten. Außerdem liefen sowohl die Pipeline-Instanz als auch Kafka auf demselben Rechner, in einem Produktivsystem müssten die Nachrichten vom Kafka-Broker über eine Netzwerkverbindung übertragen werden, weshalb hier ein geringerer Durchsatz zu erwarten ist. Außerdem wurde in diesem Experiment nur der Durchsatz der Verarbeitungslogik betrachtet. Auch die Übertragung der Ereignisse von den Maschinen in eine Cloud-Architektur kann die Durchsatzleistung weiter einschränken.

5.3 Auslastung

In diesem Abschnitt wird analysiert, wie viele Ereignisse ein MDC im Durchschnitt und maximal erzeugt. Außerdem wird untersucht, wie viele CNC-Maschinen an die Pipeline angeschlossen werden können, wenn man den im vorherigen Abschnitt 5.2 beschriebenen Durchsatz berücksichtigt.

Für diese Untersuchung wurden im Zeitraum vom 30.11.2022 bis zum 23.01.2023 mehr als 680.000 Ereignisse auf den drei angeschlossenen CNC-Maschinen aufgezeichnet. Anschließend wurde für jede Minute in diesem Zeitraum gezählt, wie viele Ereignisse pro Maschine und insgesamt aufgetreten sind. Dabei wurden nur Zeitintervalle berücksichtigt, in denen mindestens ein Ereignis auftrat. Zeiten, in denen keine Ereignisse auftraten, z.B. am Wochenende oder über die Weihnachtsfeiertage, wurden also nicht berücksichtigt. Insgesamt wurden auf diese Weise ca. 24.000 Zeitintervalle von einer Minute betrachtet. Die Verteilung der Anzahl der Ereignisse pro Minute für alle drei CNC-Maschinen ist in Abbildung 5.6 dargestellt. Für die meisten Zeitintervalle treten insgesamt weniger als 50 *Ereignisse/Minute* auf, der Durchschnitt liegt bei $28,21$ *Ereignissen/Minute*. Es gibt aber auch Ausreißer nach oben, so treten in einem betrachteten Zeitintervall mehr als 560 *Ereignisse/Minute* auf.

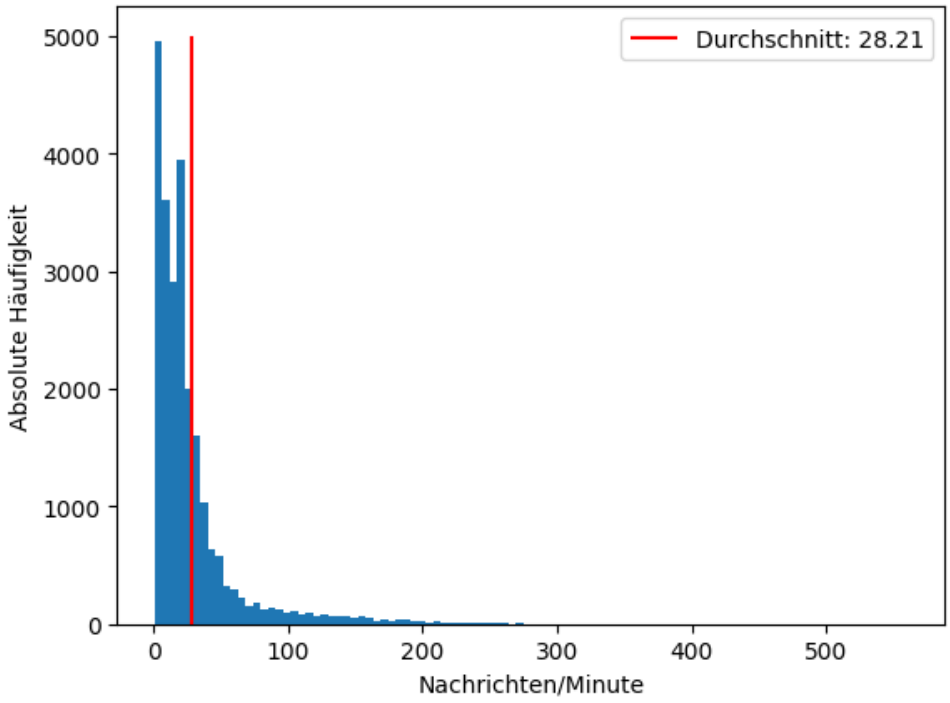

Abbildung 5.6: Verteilung von aufretenden Ereignissen/Minute für alle drei angeschlossenen CNC-Maschinen

Die Zeitintervalle mit einer auffälligen Anzahl von Ereignissen wurden anschließend näher betrachtet. In all diesen Zeitintervallen war die Mehrzahl der Nachrichten ein Ereignis, das eine Änderung der Vorschubgeschwindigkeit anzeigte. In diesem Zeitraum wurde also oft der Vorschub manuell verändert, was zu sehr vielen Ereignissen führt.

Die Verteilung der Ereignisse pro Minute für die einzelnen Maschinen ist in Abbildung 5.7 dargestellt. Die Titel der einzelnen Diagramme entsprechen den Namen der CNC-Maschinen. Es handelt sich um eine CNC-Fräsmaschine (DMC 1850 V) und zwei CNC-Drehmaschinen (CTX Alpha und CTX Beta). Hier fällt auf, dass sowohl der Median als auch der Mittelwert der Meldungen pro Minute bei der Fräsmaschine niedriger sind als bei den beiden Drehmaschinen. Auch die Ausreißer nach oben sind bei den Drehmaschinen deutlich höher als bei der Fräsmaschine. Insgesamt kann man sagen, dass eine CNC-Maschine im Durchschnitt etwa 10 *Ereignisse/Minute* erzeugt (Median). Wobei in 75 % der Fälle weniger als 20 *Ereignisse/Minute* erzeugt werden. Im ungünstigsten Fall können aber auch mehr als 500 *Ereignisse/Minute* an einer CNC-Maschine auftreten.

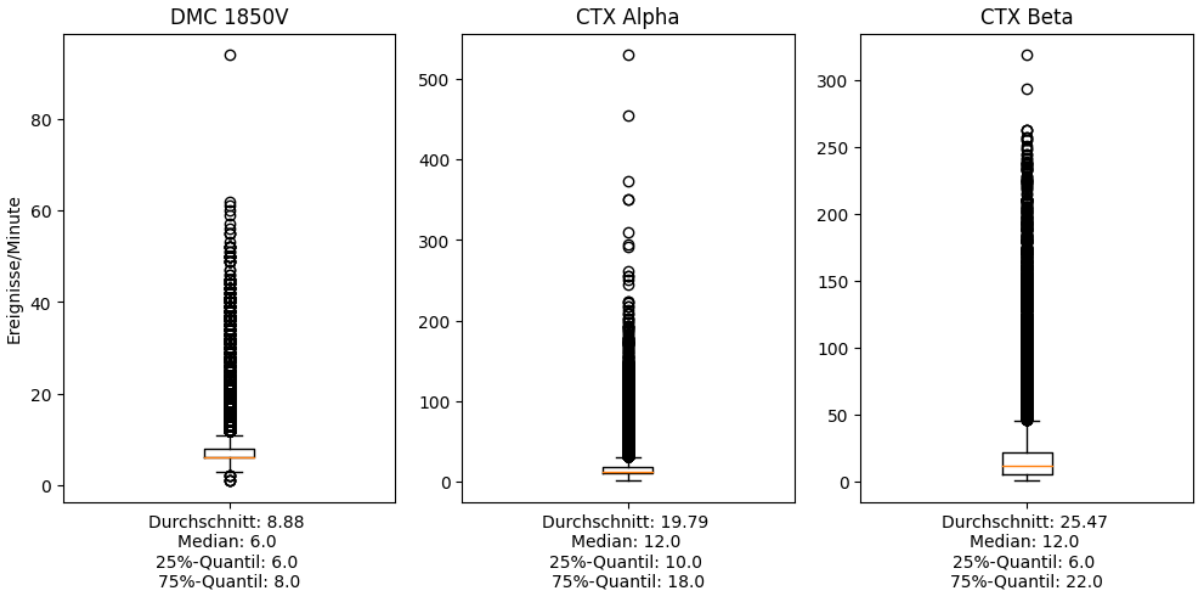

Abbildung 5.7: Verteilung von Ereignissen/Minute für die einzelnen Maschinen

6. Ausblick und Diskussion

Abschließend soll im Folgenden darauf eingegangen werden, welche Punkte bei der Entwicklung der beschriebenen Pipeline nicht berücksichtigt werden konnten, wo noch Verbesserungspotential besteht und wo Probleme bei der Umsetzung liegen. Außerdem soll ein Ausblick gegeben werden, was in Zukunft mit der vorgeschlagenen Architektur realisiert werden kann.

6.1 Bewertung der Architektur

Wie bereits im Abschnitt 5.2 beschrieben, ist es mit der vorgeschlagenen Architektur möglich, auf Standard-Hardware in einem Docker-Container mit einer Instanz ca. 10.000 *Ereignisse/Sekunde* zu verarbeiten. Wie im Abschnitt 5.3 beschrieben, liefert ein MDC im Durchschnitt ca. 20 *Ereignisse/Minute*, so dass es theoretisch möglich ist, mit einer einzigen Instanz ca. 30.000 CNC-Maschinen zu verwalten. Dabei ist jedoch zu berücksichtigen, dass ein MDC im ungünstigsten Fall mehr als 500 *Ereignisse/Minute* sendet. In diesem Fall kann die vorgestellte Architektur mit einer Instanz also nur die Ereignisse von ca. 1.200 MDCs verarbeiten.

Außerdem ist zu beachten, dass bei diesen Berechnungen nur der Durchsatz der Pipeline berücksichtigt wird, nachdem sich die Ereignisse bereits in der Kafka-Topic befinden. Der Durchsatz der verwendeten Übertragungswege in die Cloud wird nicht berücksichtigt. Wie viele Nachrichten pro Minute der RabbitMQ Broker verarbeiten kann, muss in weiteren Tests noch untersucht werden. Der mögliche Durchsatz der Klasse `Receiver` wurde ebenfalls nicht untersucht. Also wie viele Nachrichten pro Minute über AMQP empfangen und in die Kafka-Topic geschrieben werden können. Auch dies muss in weiteren Tests untersucht werden. Es ist davon auszugehen, dass in diesem Szenario die Übertragung der Nachrichten über Netzwerkverbindungen einen Flaschenhals darstellt, nicht jedoch die Verarbeitung der Nachrichten.

Wie im Abschnitt 4.4 beschrieben, kann diese Architektur skaliert werden, indem weitere Instanzen der `Pipeline` und weitere Kafka-Partitionen hinzugefügt werden. Dies wurde jedoch noch nicht in der Praxis getestet und muss in weiteren Experimenten getestet werden. Alle bisherigen Experimente wurden mit einer Pipeline-Instanz und einer Kafka-Partition durchgeführt.

6.2 Offene Punkte

Die verwendeten MDCs unterstützen, wie in Abschnitt 3.3 beschrieben, neben dem in diesem Prototyp verwendeten MQTT-Protokoll auch die Protokolle OPC UA und MTConnect [30]. Da, wie in Abschnitt 4.1 beschrieben, die MDCs mit MQTT bereits für eine andere Anwendung eingesetzt wurden, wurde MQTT auch für diesen Prototypen verwendet. Ob eines der beiden anderen Protokolle für den gegebenen Anwendungsfall Vorteile bietet, muss im Rahmen weiterer Untersuchungen noch geprüft werden.

Darüber hinaus werden Metadaten, die zuvor in einer Datenbanktabelle gespeichert wurden, zur Anreicherung der extrahierten Laufzeiten verwendet. Für die Verwendung im bestehenden System müssen diese Metadaten von einem weiteren Dienst bereitgestellt und verwaltet werden. Dieser Dienst muss für einen produktiven Einsatz noch entwickelt werden. Ein weiterer Punkt bei der Anreicherung mit Metadaten ist, dass ein NC-Programm in der Regel mehr als einmal auf einer CNC-Maschine abgearbeitet wird. Folglich werden auch mehrere Bearbeitungszeiten für ein NC-Programm extrahiert. Sollen nun alle diese extrahierten Bearbeitungszeiten mit Metadaten angereichert werden, so wird in der aktuellen Implementierung für jede einzelne Bearbeitungszeit eine SQL-Abfrage an die Datenbank gestellt. Die Metadaten werden also für ein NC-Programm mehrfach abgefragt. An dieser Stelle könnten die Metadaten für NC-Programme lokal zwischengespeichert werden, um die Datenbankabfragen bzw. zukünftig Anfragen an andere Dienste zu reduzieren.

Ein weiterer Punkt, der in dieser Bachelorarbeit nicht berücksichtigt wurde, ist die Behandlung von Nachrichten, die in der falschen Reihenfolge eintreffen. Das heißt, wenn der Zeitstempel einer empfangenen Nachricht vor dem Zeitstempel der zuvor verarbeiteten Nachricht liegt. Die aktuelle Implementierung geht davon aus, dass die Nachrichten in der richtigen Reihenfolge eintreffen. Eine mögliche Lösung wäre, die Nachrichten vor der Verarbeitung in einem Puffer zu speichern und immer die Nachricht mit dem frühesten Zeitstempel aus dem Puffer zu verarbeiten.

6.3 Weitere Ideen

Neben dem in Kapitel 1 beschriebenen Nutzen kann die entwickelte Pipeline-Architektur auch für andere Problemstellungen eingesetzt werden. Im Folgenden werden weitere Ideen beschrieben, die in Zukunft mit Hilfe der Daten-Pipeline realisiert werden können.

Wenn auch die Erstellung des NC-Programms in den eigenen Händen liegt, kann man neben der Bearbeitungszeit für das gesamte NC-Programm auch Bearbeitungszeiten für einzelne Bearbeitungsschritte erhalten. In den NC-Programmen können sogenannte Operator Messages platziert werden. Diese dienen eigentlich dazu, den Bediener einer CNC-Maschine über ein bestimmtes Ereignis in einem automatisch abgearbeiteten NC-Programm zu informieren und werden zu diesem Zweck auf dem Display der CNC-Maschine angezeigt [46]. Der in dieser Bachelorarbeit verwendete MDC gibt diese Operator Messages weiter [30]. Wenn die Erstellung des NC-Programms nun in den eigenen Händen liegt, können vor und nach einem Bearbeitungsschritt spezielle Operator Messages gesetzt werden. Die Pipeline kann dann diese Nachrichten auswerten und so die Bearbeitungszeit für diesen Bearbeitungsschritt berechnen. Diese kann dann wiederum zur Verbesserung der vorhergesagten Bearbeitungszeiten verwendet werden.

Neben den Zeitpunkten, zu denen ein Programm gestartet, gestoppt oder beendet wurde, liefert der MDC auch eine Reihe weiterer Informationen, wie z.B. die Änderung des Vorschubs oder der Spindeldrehzahl [30]. Auch diese Werte können für zukünftige Berechnungen von Interesse sein und könnten in ähnlicher Weise verarbeitet werden. Insbesondere in Kombination mit dem vorherigen Punkt, den Bearbeitungszeiten für einzelne Bearbeitungsschritte, können diese Werte interessant sein. So können z.B. Zusammenhänge zwischen Spindeldrehzahl, Vorschub und Bearbeitungsschritt hergestellt werden. Auch diese Informationen können in Zukunft von Nutzen sein.

Darüber hinaus kann mit Hilfe der Daten aus den MDCs auch die Auslastung der Fertigung überwacht werden. Mit Hilfe von Ereignissen kann beispielsweise überwacht werden, welche Maschinen lange Stillstandszeiten haben oder welche Maschinen am meisten ausgelastet sind. Diese Informationen können z.B. bei der Einplanung von Fertigungsaufträgen oder bei der Entscheidung über die Anschaffung neuer Maschinen genutzt werden.

7. Zusammenfassung

Ziel der Arbeit war es, eine Daten-Pipeline zu entwerfen, mit der IoT-Daten aus dem Kontext der industriellen Fertigung verarbeitet werden können. Betrachtet wurden dabei die Daten mehrerer CNC-Maschinen aus der Fertigung. Die Pipeline sollte aus diesem Datenstrom die Bearbeitungszeiten einzelner NC-Programme extrahieren und diese mit Metadaten aus der Fertigungsplanung verknüpfen.

Aufbauend auf den in Kapitel 3 vorgestellten Konzepten wurde entschieden, eine Pipeline nach der κ-Architektur zu entwerfen, die in der Lage ist, die Daten ohne Verzögerung zu verarbeiten. Für den Transport der Daten aus der Fertigung in die Cloud-Umgebung wurden die Protokolle MQTT und AMQP verwendet. Als Messaging-System wurde Apache Kafka eingesetzt. Damit wurden die eingehenden Daten verwaltet und persistent gespeichert. Für die Verarbeitung der Daten wurde ein Microservice in Python entwickelt. Dieser extrahiert die Laufzeiten der einzelnen NC-Programme aus dem Datenstrom, verknüpft diese mit Metadaten aus der Fertigungsplanung und legt die Ergebnisse schließlich in einer Datenbank ab. Zusätzlich wurde eine REST-Schnittstelle entwickelt, mit deren Hilfe die Ergebnisse gefiltert und von anderen Diensten abgefragt werden können.

In einem Experiment wurde der Durchsatz der entwickelten Pipeline gemessen. Bei der zu erwartenden Datenmenge, die eine CNC-Maschine im Durchschnitt liefert, können mit der vorgestellten Implementierung bis zu 500 CNC-Maschinen von einer Instanz der Pipeline verwaltet werden. Außerdem wurde eine Möglichkeit vorgestellt, die Pipeline durch die Verwendung mehrerer Instanzen zu skalieren.

8. Literatur

[1] up2parts GmbH. „Simplify manufacturing - Mit KI-basierter Software die Fertigung digital optimieren.“ (o.D.), Adresse: https://up2parts.com/ (zuletzt aufgerufen am 16. 02. 2023).

[2] R. Koller, *CAD: automatisiertes Zeichnen, Darstellen und Konstruieren*. Springer Berlin, Heidelberg, 1989.

[3] E. Fleisch und F. Thiesse, „Internet der Dinge,“ in *N. Gronau et al. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik – Online-Lexikon*, Berlin : GITO, 2019. Adresse: https://wi-lex.de (zuletzt aufgerufen am 18. 02. 2023).

[4] A. Kanawaday und A. Sane, „Machine learning for predictive maintenance of industrial machines using IoT sensor data,“ in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, S. 87–90. DOI: 10.1109/ICSESS.2017.8342870.

[5] M. Keith, „IoT #LikeABosch — in einem Fertigungswerk,“ in *Digitalisierung im Mittelstand: Trends, Impulse und Herausforderungen der digitalen Transformation*, H. R. Fortmann, Hrsg. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, S. 255–267, ISBN: 9783658292911. DOI: 10.1007/978-3-658-29291-1_24.

[6] M. Gajjar, S. Sigg, B. Litz und A. P. Baruah, „Intelligentes IoT: Erkenntnisse aus IoT-Daten durch Machine Learning,“ in *Künstliche Intelligenz: Mit Algorithmen zum wirtschaftlichen Erfolg*, P. Buxmann und H. Schmidt, Hrsg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, S. 139–148, ISBN: 9783662617946. DOI: 10.1007/978-3-662-61794-6_8.

[7] R. Dagar, S. Som und S. K. Khatri, „Smart Farming – IoT in Agriculture,“ in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2018, S. 1052–1056. DOI: 10.1109/ICIRCA.2018.8597264.

[8] M. Helmold, „Lean Management in der Produktentwicklung,“ in *Kaizen, Lean Management und Digitalisierung: Mit den japanischen Konzepten Wettbewerbsvorteile für das Unternehmen erzielen*. Wiesbaden: Springer Fachmedien Wiesbaden, 2021, S. 113–118, ISBN: 9783658323424. DOI: 10.1007/978-3-658-32342-4_11.

[9] E. Fleisch, M. Weinberger und F. Wortmann, „Geschäftsmodelle im internet der dinge,“ *Schmalenbachs Zeitschrift für betriebswirtschaftliche Forschung*, Jg. 67, S. 444–465, 2015.

[10] P. Robert und M. Stephen, „Batch processing,“ in *An Executive's Guide to Information Technology : Principles, Business Models, and Terminology.* Cambridge University Press, 2007, ISBN: 9780521853361. Adresse: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=194291&site=ehost-live.

[11] M. Sarda. „Batch Processing Vs Stream Processing.“ (2020), Adresse: https://k21academy.com/microsoft-azure/data-engineer/batch-processing-vs-stream-processing/ (zuletzt aufgerufen am 14. 02. 2023).

[12] Apache Software Foundation. „Apache Hadoop 3.3.4.“ (o. D.), Adresse: https://hadoop.apache.org/docs/stable/index.html (zuletzt aufgerufen am 11. 01. 2023).

[13] J. Dean und S. Ghemawat, „MapReduce: Simplified Data Processing on Large Clusters,“ in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004, S. 137–150.

[14] W. Sleeman und B. Krawczyk, „Imbalanced Big Data Oversampling: Taxonomy, Algorithms, Software, Guidelines and Future Directions,“ 2021.

[15] N. Marz. „How to beat the CAP theorem.“ (2011), Adresse: http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html (zuletzt aufgerufen am 10. 11. 2022).

[16] *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ISBN: 978-3-540-27416-2. DOI: 10.1007/3-540-27416-2_1.

[17] J. Warren und N. Marz, *Big Data: Principles and best practices of scalable realtime data systems*. Simon und Schuster, 2015, ISBN: 9781617290343.

[18] B. Schröter. „Echtzeitanalysen & Streamingarchitekturen.“ (2019), Adresse: https://www.cologne-intelligence.de/blog/echtzeitanalysen-streamingarchitekturen (zuletzt aufgerufen am 27. 01. 2023).

[19] J. Kreps. „Questioning the Lambda Architecture.“ (2014), Adresse: https://www.oreilly.com/radar/questioning-the-lambda-architecture/ (zuletzt aufgerufen am 10. 11. 2022).

[20] J. Lin, „The lambda and the kappa,“ *IEEE Internet Computing*, Jg. 21, Nr. 05, S. 60–66, 2017.

[21] O. Boykin, S. Ritchie, I. O'Connell und J. Lin, „Summingbird: A framework for integrating batch and online MapReduce computations,“ *Proceedings of the VLDB Endowment*, Jg. 7, S. 1441–1451, 08/2014. DOI: 10.14778/2733004.2733016.

[22] OASIS. „MQTT Version 3.1.1.“ (2015), Adresse: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (zuletzt aufgerufen am 21. 12. 2022).

[23] OASIS. „MQTT: The Standard for IoT Messaging.“ (2022), Adresse: https://mqtt.org/ (zuletzt aufgerufen am 07. 12. 2022).

[24] HiveMQ GmbH. „Quality of Service (QoS) 0,1, & 2 MQTT Essentials: Part 6.“ (2015), Adresse: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ (zuletzt aufgerufen am 04. 01. 2023).

[25] VMware, Inc. „AMQP 0-9-1 Model Explained.“ (o. D.), Adresse: https://www.rabbitmq.com/tutorials/amqp-concepts.html (zuletzt aufgerufen am 17. 01. 2023).

[26] VMware, Inc. „AMQP 0-9-1 Model Explained.“ (o. D.), Adresse: https://www.rabbitmq.com/getstarted.html (zuletzt aufgerufen am 17. 01. 2023).

[27] B. Kundu. „Headers Exchange in AMQP - RabbitMQ.“ (2020), Adresse: https://jstobigdata.com/rabbitmq/headers-exchange-in-amqp-rabbitmq/ (zuletzt aufgerufen am 17. 01. 2023).

[28] Apache Software Foundation. „Kafka 3.3 Documentation.“ (o. D.), Adresse: https://kafka.apache.org/documentation/ (zuletzt aufgerufen am 06. 01. 2023).

[29] DMG MORI Software Solutions GmbH. „Highest Level of Security IoTconnector Compatible with Different Open Protocols Offered as Standard.“ (2021), Adresse: https://www.dmgmori.co.jp/en/trend/detail/id=5501 (zuletzt aufgerufen am 17. 01. 2023).

[30] DMG MORI Software Solutions GmbH, *Benutzerhandbuch - Machine Data Connector (MDC)*, 2020.

[31] MTConnect Institute, *MTConnect Standard Part 1.0 - Fundamentals Version 1.4.0*, 2018. Adresse: https://docs.mtconnect.org/MTC_Part1_0_OverviewAndFundamentals1_4_0.pdf (zuletzt aufgerufen am 17. 01. 2023).

[32] Unified Architecture Core - UA, *UA Part1: Overview and Concepts*, OPC Foundation, Scottsdale, USA, 2022. Adresse: https://reference.opcfoundation.org/Core/Part1/v105/docs/ (zuletzt aufgerufen am 17. 01. 2023).

[33] I. I. Consortium, „The Industrial Internet Reference Architecture,“ 2022. Adresse: https://www.iiconsortium.org/wp-content/uploads/sites/2/2022/11/IIRA-v1.10.pdf (zuletzt aufgerufen am 20. 02. 2023).

[34] P. Adolphs und U. e. a. Epple, „Statusreport: Referenzarchitekturmodell Industrie 4.0 (RAMI4. 0),“ *VDI/VDE*, 2015.

[35] IBM. „Internet of Things reference architecture.“ (o.D.), Adresse: https://www.ibm.com/cloud/architecture/architectures/iotArchitecture/reference-architecture (zuletzt aufgerufen am 20. 02. 2023).

[36] M. Moghaddam, M. N. Cadavid, C. R. Kenley und A. V. Deshmukh, „Reference Architectures for Smart Manufacturing: A Critical Review,“ *Journal of Manufacturing Systems*, Jg. 49, S. 215–225, 2018, ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2018.10.006.

[37] A. Farooqui, K. Bengtsson, P. Falkman und M. Fabian, „Towards data-driven approaches in manufacturing: an architecture to collect sequences of operations,“ *International Journal of Production Research*, Jg. 58, Nr. 16, S. 4947–4963, 2020. DOI: 10.1080/00207543.2020.1735660.

[38] Y. Liu, K. Akram Hassan, M. Karlsson, Z. Pang und S. Gong, „A Data-Centric Internet of Things Framework Based on Azure Cloud,“ *IEEE Access*, Jg. 7, S. 53 839–53 858, 2019. DOI: 10.1109/ACCESS.2019.2913224.

[39] E. Gündoğdu. „RabbitMQ MQTT & AMQP.“ (2019), Adresse: https://medium.com/@gundogdu.emre/rabbitmq-mqtt-amqp-a1c915ecc1c2 (zuletzt aufgerufen am 15. 02. 2023).

[40] 84codes AB. „CloudAMQP.“ (o. D.), Adresse: https://www.cloudamqp.com/docs/index.html (zuletzt aufgerufen am 04. 01. 2023).

[41] Cumolocity GmbH. „MQTT Static templates.“ (2022), Adresse: https://cumulocity.com/guides/reference/smartrest-two/#mqtt-static-templates (zuletzt aufgerufen am 09. 11. 2022).

[42] VMware, Inc. „RabbitMQ.“ (2022), Adresse: https://www.rabbitmq.com/ (zuletzt aufgerufen am 07. 12. 2022).

[43] Apache Software Foundation. „APACHE KAFKA.“ (2022), Adresse: https://kafka.apache.org/ (zuletzt aufgerufen am 07. 12. 2022).

[44] M. Masse, *REST API design rulebook: designing consistent RESTful web service interfaces*. O'Reilly Media, Inc., 2011.

[45] up2parts, *up2parts interne Dokumentation der Testfertigung*, 2022.

[46] M. Lynch. „Six Times to Include Messages in CNC Programs.“ (2021), Adresse: https://www.mmsonline.com/articles/six-times-to-include-messages-in-cnc-programs (zuletzt aufgerufen am 16. 02. 2023).

Abbildungsverzeichnis

1.1 Schematische Abfolge der einzelnen Schritte in der Daten-Pipeline . . . 5

3.1 Aufbau einer Batch-Architektur . . . 7
3.2 Die Phasen Map, Shuffle und Reduce . . . 8
3.3 Aufbau einer λ-Architektur . . . 11
3.4 Aufbau einer κ-Architektur . . . 13
3.5 Nachrichtenaustausch über MQTT . . . 15
3.6 Übertragung einer Nachricht mit QoS 0 . . . 15
3.7 Übertragung einer Nachricht mit QoS 1 . . . 16
3.8 Übertragung einer Nachricht mit QoS 2 . . . 16
3.9 Kommunikation zwischen Publisher und Consumer über AMQP . . . 17
3.10 Verteilung der Nachrichten mit Hilfe eines Direct Exchanges . . . 19
3.11 Verteilung der Nachrichten mit Hilfe eines Fanout Exchanges . . . 19
3.12 Verteilung der Nachrichten mit Hilfe eines Topic Exchanges . . . 20
3.13 Verteilung der Nachrichten mit Hilfe eines Headers Exchanges . . . 20
3.14 Aufteilung von Events auf mehrere Partitionen innerhalb eines Topics . 22

4.1 Ausgangslage zu Beginn der Bachelorarbeit . . . 26
4.2 Laufzeiten für einzelne NC-Programme mit eingeblendeten ABORTED und Teilezähler Nachrichten . . . 29
4.3 Unterschied zwischen dem Laufzeit Event und der aus den Zeitstempeln berechneten Laufzeit . . . 30
4.4 Laufzeit eines einzelnen NC-Programms mit eingeblendeten RUNNING, STOPPED und ABORTED Nachrichten . . . 32
4.5 Aufbau der Architektur im Groben . . . 36
4.6 Darstellung der verwendeten Klassen innerhalb des Systems . . . 37
4.7 Die Klasse MdcStreamProcessor . . . 37
4.8 Die Klasse Pipeline . . . 38
4.9 Die Klasse RuntimeExtractor . . . 39
4.10 Die Klasse ManufacturingMachine . . . 40
4.11 Zustandsautomaten einer virtuellen Fertigungsmaschine . . . 42
4.12 Die Klasse Enricher . . . 42

4.13 Schema der benötigten Tabellen 43
4.14 Die Tabelle runtimes 44
4.15 Die Klasse Publisher 45

5.1 3D CAD-Modell des zu fertigenden Testbauteils 46
5.2 Das Bauteil bei während der Bearbeitung 46
5.3 Das fertige Bauteil 46
5.4 Verteilung der Dauer der Verarbeitung von 100.000 Nachrichten mit Hilfe der Pipeline 49
5.5 Verteilung des Datendurchsatz der Pipeline 49
5.6 Verteilung von aufretenden Ereignissen/Minute für alle drei angeschlossenen CNC-Maschinen 50
5.7 Verteilung von Ereignissen/Minute für die einzelnen Maschinen 51

Tabellenverzeichnis

4.1 Aufbau des Cumolocity Templates 200 27
4.2 Aufbau des Cumolocity Templates 400 27

5.1 Ergebnisse der ersten Aufspannung 48
5.2 Ergebnisse der ersten Aufspannung 48

Quellcodeverzeichniss

4.1 Die Laufzeit wird von Event eins auf Event zwei um fünf Sekunden erhöht, obwohl zwischen den beiden Zeitstempeln über zehn Sekunden liegen 30
4.2 Vereinfachte Berechnung von Bearbeitungszeiten aus einer Liste von Events und Zeitstempeln (Beispielcode in Python) 31

Anhang

Anwendungfälle

Bearbeitungszeiten Extrahieren	
Kennung:	UC-1
Kurzbeschreibung:	Dieser Anwendungsfall beschreibt, wie das System die Bearbeitungszeiten für eine Bearbeitung aus dem IoT-Datenstrom extrahiert.
Vorbedingungen:	• Ein NC-Programm wurde geladen • Die Maschine befindet sich im Automatik-Modus
Nachbedingungen:	
Normaler Ablauf:	1. Dieser Anwendungsfall beginnt, wenn ein Mitarbeiter an einer Maschine die Bearbeitung eines Bauteils startet. 2. Das System erkennt die gestartete Bearbeitung 3. Das System wartet auf weitere Ereignisse (siehe Ablaufvarianten) 4. Das System erkennt das Ende der Bearbeitung und berechnet die für die Bearbeitung benötigte Zeit 5. Ende
Ablauf Varianten: 3a)	Bearbeitung wird pausiert: 1. Das System erkennt, dass die aktuelle Bearbeitung pausiert wurde 2. Das System erkennt, dass die Bearbeitung fortgesetzt wird. 3. Die Zeit in der die Bearbeitung pausiert war, wird von der ermittelten Bearbeitungszeit abgezogen 4. Weiter im normalen Ablauf bei 3.

3b)	Maschine wechselt in manuellen Modus: 1. Das System erkennt, dass die Maschine in den manuellen Modus gewechselt hat 2. Das System verwirft die bisher berechnete Bearbeitungszeit 3. Ende

Maschine wechselt den Modus	
Kennung:	UC-2
Kurzbeschreibung:	Dieser Anwendungsfall beschreibt, wie das System reagiert wenn die Maschine vom manuellen Modus in den automatik Modus wechselt, bzw. umgekehrt.
Vorbedingungen:	
Nachbedingungen:	
Normaler Ablauf:	1. Dieser Anwendungsfall beginnt, wenn ein Mitarbeiter an einer Maschine den Modus wechselt. 2. Das System speichert den gewählten Modus 3. Ende
Ablauf Varianten:	

Neues NC-Programm geladen	
Kennung:	UC-3
Kurzbeschreibung:	Dieser Anwendungsfall beschreibt, wie das System reagiert wenn ein neues NC-Programm auf eine Maschine geladen werden.
Vorbedingungen:	
Nachbedingungen:	
Normaler Ablauf:	1. Dieser Anwendungsfall beginnt, wenn ein Mitarbeiter an einer Maschine eine neues NC-Programm lädt. 2. Das System speichert den Namen des geladenen NC-Programms 3. Ende
Ablauf Varianten:	

Extrahierte Bearbetiungszeiten mit Metadaten anreichern und speichern	
Kennung:	UC-4
Kurzbeschreibung:	Dieser Anwendungsfall beschreibt mit welchen Metadaten die extrahierten Bearbeitungszeiten angereichert werden. Außerdem werden die erhaltenen Ergebnisse gespeichert.
Vorbedingungen:	• Eine Bearbeitungszeit wurde extrahiert
Nachbedingungen:	
Normaler Ablauf:	1. Dieser Anwendungsfall beginnt, wenn das System eine Bearbeitungszeit extrahiert hat 2. Das System fügt die Information an auf welcher Maschine die Bearbeitung stattfand 3. Das System fügt die Infromation an welche Kalkulation zu dieser Bearbeitung gehört 4. Das System fügt die Information an welches CAD-Modell zu dieser Bearbeitung gehört 5. Das System speichert die ermittelte Bearbeitungszeit mit den entsprechenden Metadaten persistent ab 6. Ende
Ablauf Varianten:	

Bearbeitungszeiten für ein NC-Programm erhalten	
Kennung:	UC-5
Kurzbeschreibung:	Dieser Anwendungsfall beschreibt das Verhalten des Systems, wenn ein anderer Dienst die Bearbeitungszeiten für ein bestimmtes NC-Programm erhalten möchte.
Vorbedingungen:	
Nachbedingungen:	
Normaler Ablauf:	1. Dieser Anwendungsfall beginnt, wenn ein anderer Dienst nach den Laufzteiten für ein bestimmtes NC-Programm fragt. Der Name des NC-Programms wird dabei übergeben. 2. Das System sucht in der Datenbank nach allen Bearbeitungszeiten für dieses NC-Programm 3. Das system gibt eine Liste von Bearbeitungszeiten zurück 4. Ende
Ablauf Varianten: 1a)	Filtern nach Kalkulation ID: 1. Anstelle des Namens des NC-Programms wird die ID der dazugehörigen Kalukulation übergeben 2. Das System sucht in der Datenbank nach allen Bearbeitungszeiten für diese Kalkulations ID 3. Weiter im normalen Ablauf bei 3.
1b)	Filtern nach CAD-Modell: 1. Anstelle des Namens des NC-Programms wird die ID des zugehörigen CAD-Modells übergeben 2. Das System sucht in der Datenbank nach allen Bearbeitungszeiten für dieses CAD-Modell 3. Weiter im normalen Ablauf bei 3.

2a)	Keine Ergebnisse gefunden 1. Das System sucht in der Datenbank nach den entsprechenden Bearbeitungszeiten. Allerdings werden keine Ergebnisse gefunden 2. Das System gibt eine leere Liste zurück 3. Ende