FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen, Campus Jülich

Fachbereich 09 - Medizintechnik und Technomathematik
Studiengang Angewandte Mathematik und Informatik

Seminararbeit

Konzeption und Umsetzung einer Streaming-Architektur für das Echtzeit-Streaming von Bildern mit Redpanda

Autor:
Helena Rickmann, 3589243

Betreuer:
Prof. Dr. rer. nat. Martin Reißel
M.Sc. Alexander Mattern

Aachen, 15. Dezember 2024

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Konzeption und Umsetzung einer Streaming-Architektur für das Echtzeit-Streaming von Bildern mit Redpanda

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Helena Rickmann

Aachen, den 15.12.2024

Unterschrift der Studentin / des Studenten

Zusammenfassung

Im Rahmen dieser Seminararbeit wurde eine Streaming-Architektur für das Streaming von Bildern mit Redpanda entwickelt. Anhand dieser Architektur wurde untersucht, inwiefern sich Redpanda, eine Event-Streaming-Plattform, zum Streaming von Bildern in Echtzeit eignet.
Bei der Implementierung dieser Architektur wurde der Fokus neben der Echtzeitfähigkeit auf die Skalierbarkeit und die Benutzerfreundlichkeit gelegt. Dafür wurde eine Kommunikationsarchitektur und ein -protokoll für eine einheitliche Kommunikation zwischen den Clients entworfen. Diese erleichtert es Nutzern, neue Clients an die Architektur anzuschließen oder sie auszutauschen, ohne Einschränkungen in der Lauffähigkeit des Systems in Kauf nehmen zu müssen.
Bei Testungen der Echtzeitfähigkeit der implementierten Architektur wurde bei einer geringen Anzahl Clients eine durchschnittliche Bildübertragungsdauer von 95 ms für 14,4 MB große Bilder erreicht. Bei einer Verfünffachung der Anzahl Consumer stieg die Bildübertragungszeit um 50 % auf 150 ms. Die mit Redpanda implementierte Architektur ist also dazu geeignet, Bilder in Echtzeit zu versenden und zeigt dabei im Rahmen der Testungen eine gute Skalierbarkeit.

Inhaltsverzeichnis

1 Einleitung

Am Fraunhofer Institut für Produktionstechnologie (IPT) wird an der effizienten Gestaltung von Qualitätsmanagement in der Produktion geforscht. Dabei bietet die Digitalisierung und Vernetzung der Produktion die Möglichkeit, mithilfe von Datenanalysen die Qualität von Produkten sowie Prozessen zu bewerten [1]. Die Auswertung von Daten aus der Prozessüberwachung mit Sensoren, die Temperaturen, Vibrationen oder Ähnliches messen, erlaubt eine Vorhersage der Produktqualität und eventuelles Eingreifen während der Produktion. Mithilfe einer visuellen Kontrolle hingegen kann schnell und kontaktlos die Qualität fertiger Produkte bestimmt werden. Die visuelle Kontrolle mittels Bildverarbeitung ist besonders gut für die Defekterkennung bei Bauteilen geeignet [2]. Um dies am Beispiel von Batteriezellen zu zeigen, wurde am Fraunhofer IPT ein Demonstrator entwickelt, welcher durch Auswertung von Bildern von Batteriezellen die Qualität der Schweißnähte dieser bewertet.
Die Architektur des Demonstrators basiert derzeit auf einem synchronen, anfragebasierten Modell. Bei diesem Modell werden die Anfragen eines Frontends, ein Bild aufzunehmen und zu analysieren, durch eine Pipeline bestehend aus einem ML-Modell und diversen Servern und Socket-Servern zu einem Rechner geleitet. Das Bild wird durch diese Pipeline wieder zurückgeleitet, bis das Frontend schließlich das ausgewertete Bild erhält und anzeigt. Dies erschwert die Erweiterbarkeit des Demonstrators, da jede Komponente eng mit der anderen verknüpft ist. Eine solche Architektur wäre in realen Produktionssystemen nicht umsetzbar. Aus diesem Grund wird in dieser Seminararbeit eine eventgesteuerte Architektur mit der Event-Streaming-Plattform Redpanda für den Demonstrator implementiert. Die Übertragung der Bilder, die in Form von Events versendet werden sollen und jeweils 14,4 MB betragen, stellt dabei eine Herausforderung in Bezug auf die Echtzeitfähigkeit der Datenübertragung mit Redpanda dar. Daher wird in dieser Seminararbeit anhand der implementierten Architektur untersucht, inwiefern sich die Plattform Redpanda zum Streamen von Bildern in Echtzeitsystemen eignet.
Zunächst werden in Kapitel 2 die Rolle von Echtzeit-Datenübertragung in der Produktion und die Grundlagen der Funktionsweise Redpandas erklärt. Um die Eignung Redpandas für das Streaming von Bildern in Echtzeit-Systemen zu untersuchen, wurde in Kapitel 3 zunächst eine Anforderungsanalyse an die Architektur durchgeführt. Daraufhin wurde ein Prototyp entwickelt, welcher die Abläufe in den verschiedenen Komponenten des Demonstrators mit echten Daten simuliert. Dies wurde genauer in Kapitel 4 beschrieben. Mit diesem Prototypen wurden verschiedene Tests durchgeführt (Kapitel 5), um eventuelle Einschränkungen Redpandas zu identifizieren, welche in Abschnitt 5.2 diskutiert werden.

2 Theoretische Grundlagen

In den folgenden Abschnitten werden die Bedeutung von Echtzeit-Datenübertragung in Produktionssystemen, sowie die Funktionsweise von Event-gesteuerten Architekturen am Beispiel von Redpanda erläutert. Außerdem wird Echtzeit für den Anwendungsfall des Demonstrators definiert.

2.1 Echtzeit-Datenübertragung in der Produktion

In der Produktion spielt die Verarbeitung von Echtzeitdaten zur Qualitätssicherung eine große Rolle. Die rechtzeitige Erkennung von Auffälligkeiten in Sensordaten kann etwa genutzt werden, um Störungen oder Ausfälle in Maschinen vorherzusagen und diese zu reparieren oder zu ersetzen, bevor dadurch die Produktqualität beeinträchtigt wird [3]. Aber auch in der Fertigung von Produkten, die hohen Qualitäts- und Sicherheitsstandards unterliegen, werden Echtzeit-Sensordaten genutzt, um den Produktionsprozess zu überwachen und bei kritischen Abweichungen sofort anzupassen [4]. Auch in der visuellen Qualitätskontrolle kann eine Echtzeit-Datenübertragung und -verarbeitung nötig sein, etwa für die Defekterkennung bei Rollformverfahren mit Vorschubgeschwindigkeiten, die oft über 10 m/s betragen [5].
Die oben genannten Anwendungsfälle, welche sich stark in ihren Anforderungen an die Übertragungs- und Reaktionsgeschwindigkeit unterscheiden, sind Systeme mit weichen Echtzeitanforderungen. Diese werden von H. Wörn und U. Brinkschulte definiert als Systeme, in denen sich der Mittelwert der Antwortzeit sich in einem festgelegten, akzeptablen Rahmen befindet [6]. Für den konkreten Anwendungsfall des Demonstrators wurde dieser Rahmen auf unter 200 ms für die Bildübertragungsdauer festgelegt.

2.2 Event-gesteuerte Architekturen

Event-gesteuerte Architekturen übermitteln ihre Nachrichten nach dem Publish/Subscribe-Prinzip. Events werden sequentiell für einen gewissen Zeitraum gespeichert, um in diesem Zeitraum die Zustellung der Events zu garantieren. Das erlaubt die unabhängige und asynchrone mehrmalige Verarbeitung des selben Events durch unterschiedliche Empfänger [7].

Event-gesteuerte Architekturen sind aufgrund ihrer Skalierbarkeit und der losen Kopplung der Komponenten gut geeignet, um Sensordaten in Produktionssystemen in Form von Events zu verwalten und zuverlässig an die richtigen Empfänger weiterzugeben [8]. Populäre Event-gesteuerte Architekturen sind dabei Apache Kafka [9], RabbitMQ [10], oder Google Pub/Sub [11]. In den folgenden Abschnitten werden die Grundlagen der Funktionsweise solcher Event-gesteuerter Architekturen am Beispiel von Redpanda erklärt.
Redpanda ist eine auf C++ basierende Event-Streaming-Plattform, die mit dem Ziel entwickelt wurde, eine schnellere, leichtere und einfacher bedienbare Alternative zu der Plattform Apache Kafka darzustellen [9, 12]. Dabei wurde die Architektur stark an die Apache Kafkas angelehnt und kompatibel zur Kafka API entwickelt, über die Clients mit der jeweiligen Plattform kommunizieren [13].

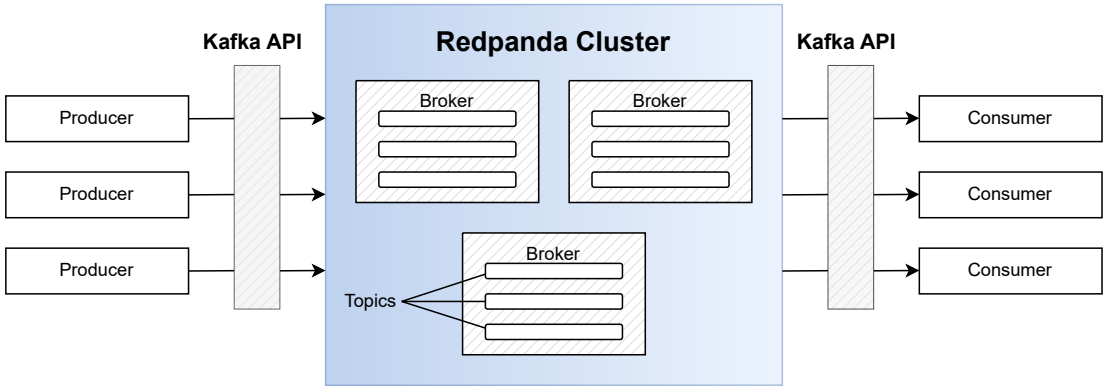


Abbildung 2.1: Redpanda Architektur und Kommunikation mit Clients über Kafka API

2.2.1 Broker, Topics und Partitionen

Ein Redpanda-Cluster besteht aus einem oder mehreren Brokern, welche Nachrichten und deren Schlüssel sequentiell in Topics organisiert speichern (s. Abbildung 2.1). Die Log-Dateien, auf denen die Nachrichten eines Topics gespeichert werden, werden Partitionen genannt. Die Nutzung mehrerer Partitionen für ein Topic ermöglicht gleichzeitige Schreib- und Lesezugriffe auf Nachrichten eines Topics, sofern sie auf verschiedenen Partitionen sind [14, 15]. Wenn ein Topic mehrere Partitionen hat, werden Nachrichten ohne Schlüssel zyklisch auf die Partitionen verteilt. Haben zwei Nachrichten eines Topics jedoch denselben Schlüssel, werden sie immer auf die selbe Partition geschrieben [16].

2.2.2 Producer und Consumer

Client-Anwendungen kommunizieren über die Kafka API mit Redpanda-Brokern (s. Abbildung 2.1). Clients, welche Nachrichten in ein Cluster schreiben, werden Producer genannt, während lesende Clients Consumer genannt werden. Ein Producer muss beim Senden einer

Nachricht das Topic festlegen, auf dem die Nachricht veröffentlicht wird, und ein Consumer kann durch Abonnieren eines Topics dessen Nachrichten lesen. Die Entkopplung von Producern und Comsumern ermöglicht das asynchrone, mehrmalige Lesen und Verarbeiten von Nachrichten durch Consumer [14, 17].

2.2.3 Schema Registry

Nachrichten werden von einem Producer serialisiert, um in Form von Bytes auf den Redpanda-Broker geschrieben und von Consumern gelesen zu werden. Mithilfe von Schemata, die die Struktur und Codierung einer Nachricht beschreiben, können Nachrichten von Clients in das korrekte Format (de-)serialisiert werden, ohne dass dadurch Abhängigkeiten unter Clients entstehen [18].
In einem Schema Registry, einer Anwendung zugehörig zu dem Redpanda-Cluster, werden Schemata registriert, bevor sie zum Serialisieren einer Nachricht verwendet werden. Dort wird ihnen eine ID zugewiesen, die zusammen mit der serialisierten Nachricht auf den Redpanda-Broker geschrieben wird. Mithilfe dieser ID kann ein Consumer dann von dem Schema Registry das passende Schema abrufen und damit die Nachricht deserialisieren. Mit der Nutzung des Schema Registry wird außerdem Abwärtskompatibilität der Schemata und damit Kompatibilität mit veralteten Clients sichergestellt, da die Registrierung eines Schemas mit inkompatiblen Veränderungen zu einer Fehlermeldung führt.

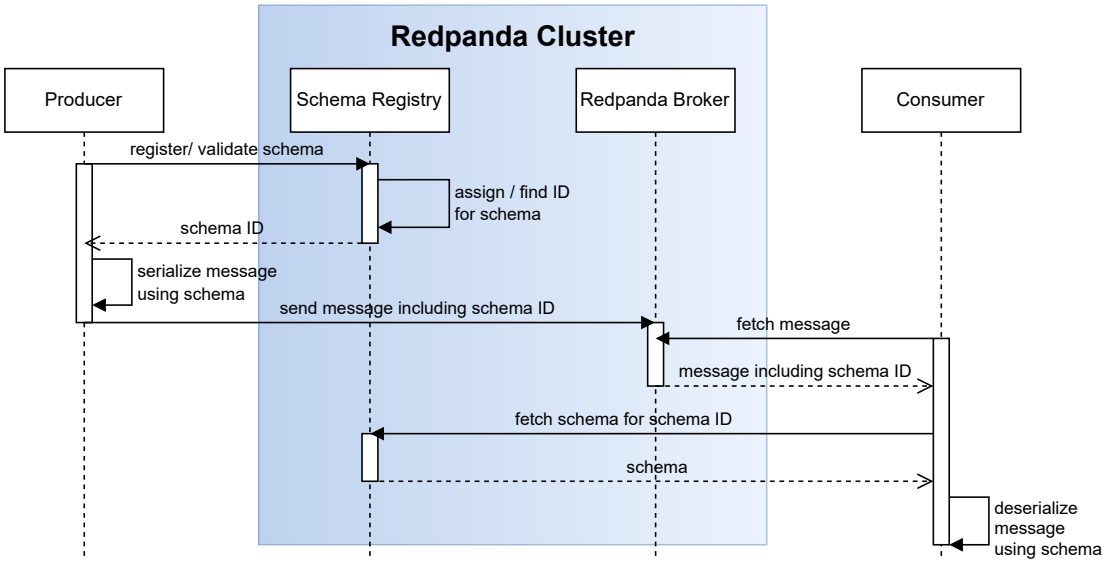


Abbildung 2.2: Serialisieren und Deserialisieren einer Nachricht mit Schema

3 Anforderungen

Damit eine Streaming-Architektur mit Redpanda für den Einsatz in dem Demonstrator und darüber hinaus auch in echten Produktionssystemen geeignet ist, müssen einige Anforderungen erfüllt werden, die im folgenden Abschnitt definiert werden.

Bildübertragung in Echtzeit Es muss möglich sein, mit der Redpanda-Streaming-Architektur Bilder in Echtzeit zu versenden. Mit der Definition von Echtzeit aus Abschnitt 2.1 bedeutet das konkret, Nachrichten mit Bildern von 14,4 MB durchschnittlich in unter 200 ms von einem Producer zu einem Consumer zu übertragen. Die Übertragungszeit der Bilder von einem lokalen Rechner, der an eine Kamera angeschlossen ist, zu dem in einer Cloud gehosteten Redpanda-Cluster kann allerdings abhängig vom Netzwerk stark variieren, und lässt keine Aussage über die Eignung Redpandas zum Echtzeit-Streaming von Bildern zu. Daher wird hier von optimalen Bedingungen ausgegangen, bei denen der Rechner, welcher die Bilder versendet, Redpanda sowie Bilder empfangende Consumer auf der selben Maschine laufen.

Skalierbarkeit Die Clients beschränken sich für den Anwendungsfall des Demonstrators bis auf Weiteres auf eine Kamera, die über einen RaspberryPi mit Redpanda kommuniziert, ein ML-Modell, welches die Bilder auswertet und ein Frontend, welches die Bilder und die Auswertungen anzeigt. Die Architektur soll skalierbar sein, es soll möglich sein, weitere Komponenten, wie einen Datenbankservice, welcher Bilder speichert, oder weitere ML-Modelle anzuschließen, ohne dadurch die Echtzeitfähigkeit zu verlieren.

Benutzerfreundlichkeit Die implementierte Architektur soll benutzerfreundlich sein, indem Consumer und Producer austauschbar sind und das Entfernen oder Hinzufügen eines Services kein Neu-Aufsetzen von Redpanda oder den anderen Services erfordert. Es soll außerdem eine einheitliche Struktur für Nachrichten und Topics geben, die es vereinfacht, Producer oder Consumer in das System zu integrieren.

4 Umsetzung

Für die Umsetzung der Kommunikation zwischen den Clients und Redpanda in dieser Bild-Streaming-Architektur wurden eine Kommunikationsarchitektur und ein -protokoll ausgearbeitet (s. Abschnitt 4.1). Außerdem wurde festgelegt, welche Daten Clients erzeugen oder benötigen und wie die Logik für die Verarbeitung von Nachrichten innerhalb der Clients aussieht (s. Abschnitt 4.2). Anschließend wurde ein Redpanda-Cluster aufgesetzt und konfiguriert und die Clients implementiert (s. Abschnitt 4.3).

4.1 Kommunikationsarchitektur

Um eine einheitliche Kommunikation zu gewährleisten, in der Services einfach integriert oder entfernt werden können, müssen die Schnittstellen eindeutig definiert und möglichst selbsterklärend sein. Des Weiteren sollten jeder Nachricht die notwendigen Informationen beigefügt sein, um selbsterklärend bezüglich des Inhalts zu sein. Dies trägt zur Erfüllung der Anforderungen an die Benutzerfreundlichkeit und Skalierbarkeit (s. Abschnitt 3) bei und ist wichtig für eine Eignung für reale Anwendungsfälle in der Produktion. Um eine möglichst effiziente Kommunikation sicherzustellen, wurde sich bei der Definition einer Kommunikationsarchitektur an bereits existierenden Ausarbeitungen zu Topic-Namensräumen und Event-Streaming-Formaten für Nachrichten orientiert, was in den folgenden Abschnitten näher erläutert wird.

4.1.1 Namenskonvention von Topics

Die Struktur der Topics wurde in Anlehnung an die Sparkplug Specification [19] erstellt, welche für MQTT-Infrastrukturen ausgearbeitet wurde. Diese sieht vor, den Namen eines MQTT-Topics folgendermaßen aufzubauen:
`namespace/group_id/message_type/edge_node_id/device_id` [19]
Die durch Schrägstriche getrennten Elemente stellen hierbei ein Spezialisierungen dar; es kann sowohl das vollständige Topic mit der Spezialisierung `device_id` (deutsch: Geräte-ID) abonniert werden, sodass nur Nachrichten empfangen werden, die dieses Gerät betreffen, wenn allerdings nur `namespace/group_id/message_type/edge_node_id` ohne die Spezialisierung `device_id` abonniert wird, werden die Nachrichten aller `device_id`s empfangen.

Diese Art von Generalisierung ist bei Redpanda-Topics nicht möglich, trotzdem ist eine Einteilung in `namespace` (deutsch: Namensraum) und `message_type` (deutsch: Nachrichtentyp) sinnvoll. `namespace` kann hier genutzt werden, um zwischen Produktionsclustern zu unterscheiden, der Demonstrator als Produktionscluster mit dem Namen "Intelligent Quality Platform" nutzt also den namespace "iqp".
Für eine Aufteilung nach `message_type` wurden die zwei Kategorien `command` (deutsch: Befehl) und `data` (deutsch: Daten) nach der Sparkplug Specification identifiziert, die dritte Kategorie `status` (deutsch: Status) findet hier keinen Nutzen, da jeder Service austauschbar ist und unabhängig von dem Status der anderen ist. Die Topic-Namen ergeben sich aus der Zusammensetzung von namespace und message_type.
So wird für den Demonstrator das Topic "iqp-command" für Befehle genutzt, und das Topic "iqp-data" für Daten. Bei den Daten kann es sich dabei beispielsweise um ein aufgenommenes Bild, oder die Auswertung eines ML-Modells handeln, bei den Befehlen etwa darum, eine Bildaufnahme auszulösen (siehe Abbildung 4.1).

4.1.2 Nachrichtenstruktur

Aufgrund der groben Aufteilung der Topics ist es wichtig, dass jede Nachricht alle nötigen Metadaten enthält, um sich selbst erklären zu können, und dass Services in der Lage sind, für sie relevante Nachrichten zu erkennen. Bei der Strukturierung der Nachrichten wurde sich an dem "Data Point Message Type" von M. Zeng et al. [20] orientiert. So wurden folgende Attribute definiert, die zusammen eine ausreichende Beschreibung für jeden Datenpunkt darstellen:

Attribut	Erläuterung
`source`	Herkunft der Daten
`time`	Zeitpunkt der Erstellung
`context`	beschreibender Name der Daten
`data`	eigentliche Nutzdaten
`flags`	zusätzliche Infos

Tabelle 4.1: Attribute einer Nachricht

Eine Nachricht, in der ein Bild versendet wird, beinhaltet mit der in Tabelle 4.1 vorgestellten Struktur beispielsweise als `source` die Kamera, `time` der Zeitpunkt der Bildaufnahme, und als `context` die Beschreibung "image". `data` beinhaltet das mit Base64 codierte Bild und `flags` zusätzliche Infos oder Hinweise zu dem Bild.
Für Nachrichten, die Befehle übermitteln, wird die gleiche Struktur genutzt, jedoch ohne das Attribut `data`. Der Name des Befehls befindet sich in dem Attribut `context`. So kann die Relevanz jeder Nachricht für den jeweiligen Service anhand des `context` bestimmt werden, unabhängig davon, ob ein Befehl oder Daten übermittelt werden.

Die Nutzung eines Schemas für die Serialisierung und Deserialisierung von Nachrichten (s. Abschnitt 2.2.3) erleichtert es Producern, Nachrichten in diesem Format zu produzieren und vereinfacht die Versionierung, sollte die Nachrichtenstruktur zu einem späteren Zeitpunkt erweitert werden. Consumern wird es so außerdem erleichtert, die Nachrichten zu verarbeiten, da sie nach einer erfolgreichen Deserialisierung sicher sein können, dass die Nachricht die erwarteten Attribute enthält.

4.2 Logik für die Kommunikation der Clients

Die Architektur besteht aus mehreren einzelnen Services, die alle über Redpanda miteinander kommunizieren. Um die Logik innerhalb der Services für die Kommunikation und die Verarbeitung von Nachrichten implementieren zu können, wurden zunächst die verschiedenen Services und ihre Aufgaben definiert, um daraus abzuleiten, welche Daten sie jeweils benötigen oder erzeugen. Dies wurde in Abbildung 4.1 visualisiert.

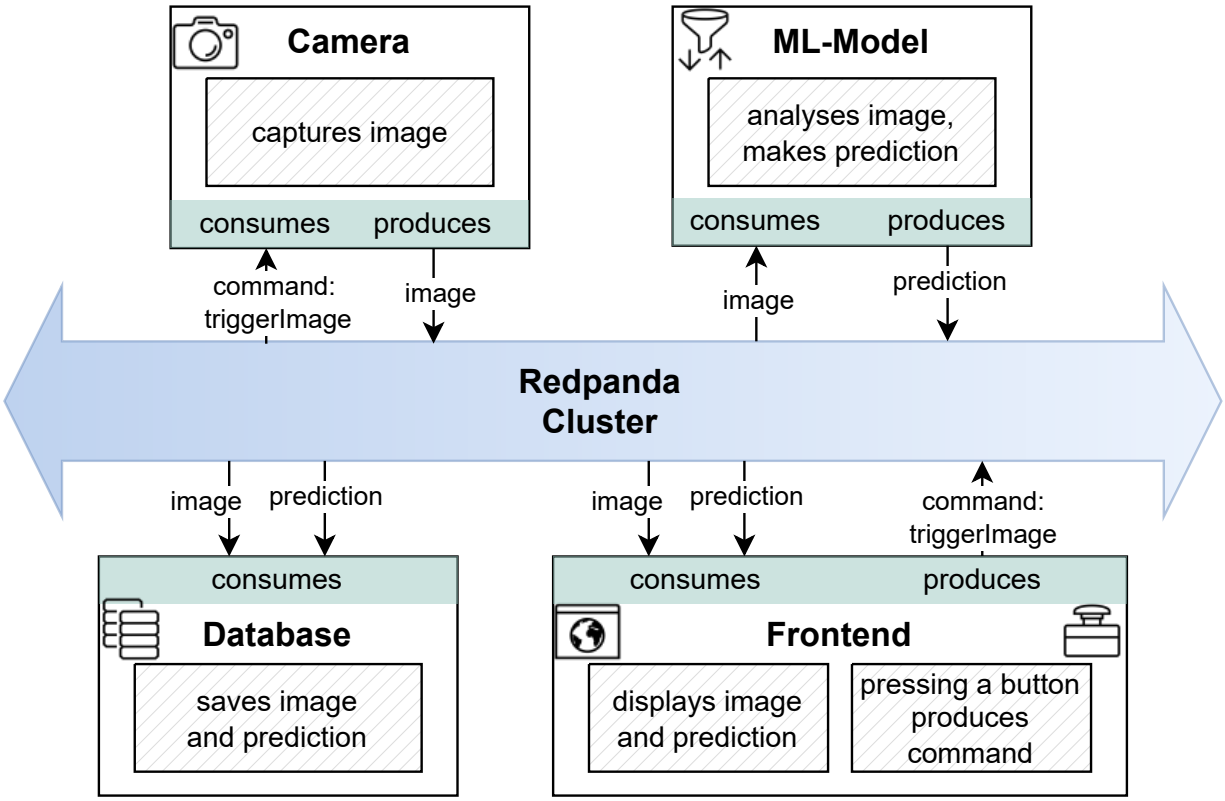


Abbildung 4.1: Funktion sowie Ein- und Ausgangsdaten der Services

Wie in Abbildung 4.1 zu sehen ist, wird ein Bild aufgenommen und zu dem Redpanda-Cluster gesendet, wenn der die Kamera steuernde Rechner einen entsprechenden Befehl erhält.

Diesen Befehl erhält der Rechner, indem ein Consumer, welcher das Topic “iqp-command” abonniert hat, eine Nachricht mit “triggerImage” als `context` empfängt. Von welchem Service diese Nachricht kommt, spielt keine Rolle, in diesem Fall ist es das Frontend nach einem Knopfdruck. Wenn eine solche Nachricht von dem Kamera-Consumer empfangen wird, wird eine Funktion aufgerufen, in der ein Bild aufgenommen wird, welches von einem Producer in eine Nachricht verpackt auf das Topic “iqp-data” mit “image” als `context` gesendet wird. Dabei wird jeder Nachricht ein einzigartiger Schlüssel zugewiesen. Services, welche Bilder verarbeiten, haben derweil mit Consumern das Topic “iqp-data” abonniert und überprüfen jede Nachricht darauf, ob das Attribut `context` den Wert “image” enthält. Im Falle eines ML-Modells, bei welchem eine Prediction (deutsch: Vorhersage) immer zu einem spezifischen Bild gehört, wird der Schlüssel der Nachricht, mit der das Bild gesendet wurde, auch wieder als Schlüssel für die Nachricht mit der Prediction verwendet. Die Nachricht mit der Prediction, welche mit dem `context` “prediction” gekennzeichnet wird, kann dann, beispielsweise von einem Frontend oder einer Datenbank, mithilfe des Schlüssels dem richtigen Bild zugeordnet werden.

4.3 Implementierung

In diesem Kapitel werden die Details der Implementierung und der Konfiguration Redpandas und der Clients näher beleuchtet.

4.3.1 Verwendete Technologien und Bibliotheken

Die Client-Logiken, die Kommunikation mit Redpanda sowie die Verarbeitung von Bildern wurde in Python implementiert. Hierfür wurden verschiedene Bibliotheken und Tools genutzt.

Producer und Consumer Die Implementierung der Clients, welche Nachrichten an Redpanda schreiben (Producer) und von Redpanda lesen (Consumer), wurde mit der Bibliothek confluent-kafka-python umgesetzt. Diese ist ein Wrapper für die Bibliothek librdkafka, welche eine C-Implementierung von Kafka-Clients bereitstellt [21]. Durch den Einsatz von confluent-kafka-python konnte im Vergleich zu der alternativen Bibliothek kafka-python, einer reinen Python-Implementierung [22], eine Verkürzung der Bildübertragungszeit von 4000 ms auf 200 ms erreicht werden. Confluent-kafka-python besitzt, im Gegensatz zu pykafka [23], einer weiteren, auf librdkafka aufbauenden Bibliothek, Avro-Schema-Serialisierer und -Deserialisierer. [21].

(De-)Serialisierung von Nachrichten mit Schema Für die Serialisierung und Deserialisierung von Nachrichten wurden die Avro-Schema-Serialisierer und -Deserialisierer der Bibliothek confluent-kafka-python genutzt. Die Serialisierung von Nachrichten erfordert ein Schema, welches in JSON (JavaScript Object Notation) definiert wird. Bei Nachrichten, die mit dem Serialisierungssystem Apache Avro serialisiert wurden, wird das Schema mitgesendet, was zwar einen leichten Overhead produziert, aber gleichzeitig dafür sorgt, dass jedes Programm die Nachricht deserialisieren kann [24]. Die Validierung des mitgesendeten Schemas erfolgt über das Redpanda Schema Registry (s. Abschnitt 2.2.3).

Bildverarbeitung Für die Konvertierung von Bildern in Bytes, welche als .bmp-Dateien bereitstehen, wurde die Bildverarbeitungs- und Computer Vision-Bibliothek OpenCV-Python benutzt, welche ebenfalls ein Wrapper für eine schnelle C/C++-Implementierung ist [25]. Die Bilder, welche als Bytes zur Verfügung stehen, werden mit dem Codierungsverfahren Base64 in einen String umgewandelt, um so als Nutzdaten einer Nachricht hinzugefügt zu werden.

4.3.2 Implementierungsdetails

Bei der Konfiguration der Producer und Consumer mussten einige Anpassungen gemacht werden, um möglichst geringe Latenzen und eine effiziente Übertragung von Nachrichten ermöglichen zu können.
Bei Producern und Consumern wurde die maximale Anzahl von Bytes, die auf einmal verschickt werden dürfen, auf 20 MB gesetzt, damit Nachrichten nicht aufgeteilt werden müssen, wenn sie zu dem Redpanda Broker oder von dort zum Consumer geschickt werden [26, 27].
Ein Prozess, der ebenfalls das Senden von Nachrichten beeinflusst, ist Batching. Beim Batching werden ausgehende Nachrichten gesammelt, bis die Größe der Nachrichten zusammen einen Schwellwert erreicht oder ein Zeitpunkt überschritten wurde, bevor sie zusammen abgeschickt werden [17]. Batching wurde bei dieser Implementierung deaktiviert, wodurch jede Nachricht sofort gesendet wird. Dies hat allerdings auch zur Folge, dass keine Komprimierung von Nachrichten stattfindet, da diese lediglich auf volle Batches angewandt wird. Bei dem Demonstrator vergehen aber möglicherweise mehrere Sekunden oder Minuten, bevor der Befehl für eine weitere Bildaufnahme gegeben wird, weshalb es sinnvoll ist, kein Batching zu betreiben, um die Latenz der einzelnen Nachrichten zu verringern, da nicht auf weitere Nachrichten gewartet werden muss.
Auch in dem Redpanda-Cluster mussten Anpassungen gemacht werden, um mit großen Nachrichten umgehen zu können. Beispielsweise die Aufbewahrungsregelung für Nachrichten in dem Redpanda Cluster sieht standardmäßig vor, dass Nachrichten sieben Tage lang gespeichert und dann die ältesten gelöscht werden, um Speicherplatz für neue Nachrichten freizugeben [28]. Die Größe der hier verschickten Nachrichten sorgt dafür, dass der Speicherplatz deutlich schneller voll wird, daher wurde festgelegt, dass maximal 1 GB Nachrichten

gespeichert werden, was bei einer Nachrichtengröße von ca. 14,5 MB etwa 71 Nachrichten sind. Mit dem Anschluss eines Datenbank-Services, wie in Abbildung 4.1 gezeigt, könnte eine Archivierung der Daten umgesetzt werden.
Die Anzahl der Partitionen für das Topic “iqp-data” wurde bei einer einzigen Partition belassen. Aus diesem Grund ist nur ein Broker notwendig. Die Entscheidung, Topics nicht zu partitionieren, wurde getroffen, da es keine Consumer-Gruppe mit mehr als einem Consumer gibt, welche von mehreren Partitionen profitieren könnte. Stattdessen muss jeder einzelne Consumer jede Nachricht lesen, was auch auf einer Partition ohne Performanceeinbußen möglich ist.

5 Tests und Evaluation

Um festzustellen, ob es mit der implementierten Redpanda-Streaming-Architektur möglich ist, bei der Übertragung von Bildern eine durchschnittliche Übertragungszeit von unter 200 ms zu erreichen (s. Abschnitt 3), wurde die Bildübertragung bei unterschiedlichen Belastungen getestet. Die Testfälle wurden konzipiert, um zusätzlich zur grundsätzlichen Echtzeitfähigkeit der Bildübertragung auch eine Aussage über die Skalierbarkeit dieser Architektur treffen zu können.

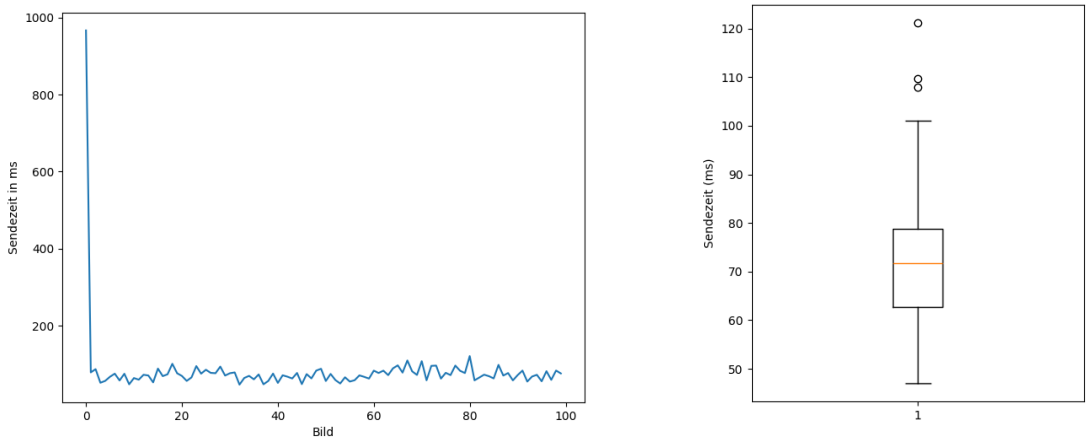
5.1 Beschreibung der Testfälle

Es wurden zwei Testfälle festgelegt, in denen jeweils in direkter Folge 100 Bilder mit einer Größe von 14,4 MB von einem Producer in dem in Abschnitt 4.1 festgelegten Nachrichtenformat auf das Topic "iqp-data" gesendet werden. Die Consumer, welche diese Nachrichten empfangen, messen dabei die Übertragungsdauer von dem Producer über Redpanda zu dem jeweiligen Consumer. Dies wird in den folgenden Abschnitten "Bildübertragungsdauer" genannt und wird aus der Differenz zwischen Empfangszeitpunkt und dem Zeitstempel der Nachricht berechnet. Der Zeitstempel markiert den Zeitpunkt, an dem die Nachricht von dem Producer erstellt wurde.
Zusätzlich wird von dem Producer die Dauer jedes Schreibvorgangs auf Redpanda aufgezeichnet, das wird im Folgenden "Sendedauer" genannt. Beide Messungen werden anschließend visualisiert.

Testfall 1: Ein Consumer Im ersten Testfall wurden die 100 Nachrichten von einem einzelnen Consumer empfangen. So kann überprüft werden, ob eine Echtzeit-Bildübertragung mit dieser Architektur grundsätzlich möglich ist.

Testfall 2: Fünf Consumer Im zweiten Testfall wurden die Nachrichten von fünf Consumern gleichzeitig empfangen. Mit diesem Test soll untersucht werden, wie sehr sich die Bildübertragungsdauer an den einzelnen Consumer erhöht, wenn Redpanda mehr parallele Lesezugriffe bewältigen muss. Die Auswirkungen der erhöhten Anzahl an Lesezugriffen auf die Dauer des Sendevorgangs werden ebenfalls beobachtet. So kann eine Aussage über die Skalierbarkeit der Architektur im Rahmen der Testungen getroffen werden.

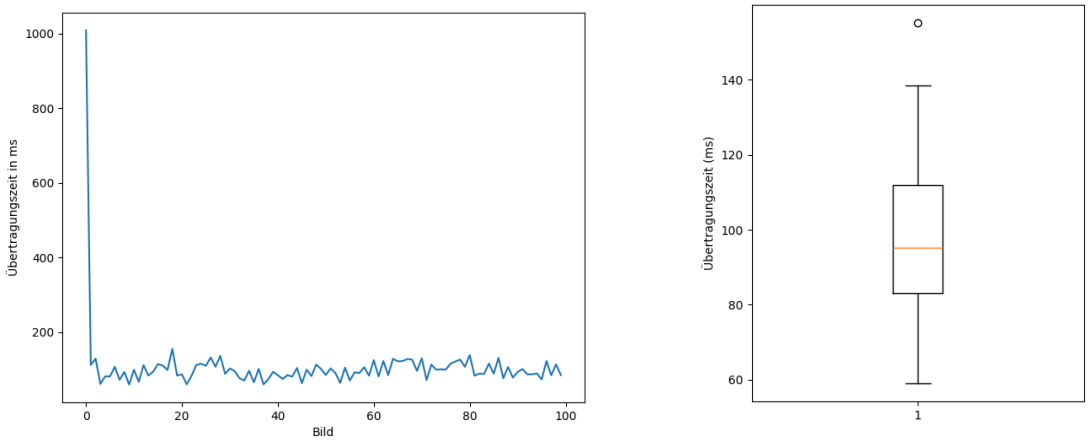
5.2 Testergebnisse und Evaluation



(a) Testfall 1: Sendedauer - Rohdaten (Liniendiagramm)

(b) Testfall 1: Sendedauer - bereinigt (Boxplot)

Abbildung 5.1: Testfall 1: Sendedauer - Rohdaten und bereinigt



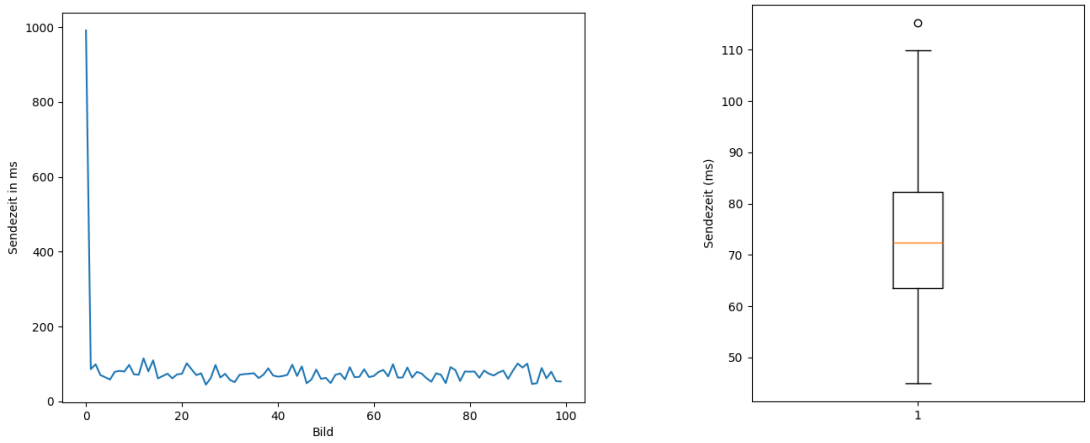
(a) Testfall 1: Bildübertragungsdauer - Rohdaten (Liniendiagramm)

(b) Testfall 1: Bildübertragungsdauer - bereinigt (Boxplot)

Abbildung 5.2: Testfall 1: Bildübertragungsdauer - Rohdaten und bereinigt

Die Rohdaten der Messungen des Testfalls 1 (s. Abbildung 5.2) zeigen, dass die Übertragungsdauer des ersten Bildes etwa 1000 ms beträgt. Die Visualisierung der Sendedauer, die für das erste Bild eine ähnlich hohe Dauer aufweist (s. Abbildung 5.1), legt nahe, dass der

Schreibvorgang des Producers diese Zeit beansprucht. Das könnte daran liegen, dass der Producer nach der Erstellung, aber vor dem Senden der ersten Nachricht, eine Verbindung zu dem Redpanda-Broker herstellen muss. Diese Vermutung wird durch die Tatsache gestützt, dass die Übertragungszeit für alle weiteren Nachrichten weniger als 200 ms beträgt.
Die Auswertung der bereinigten Messdaten des Testfalls 1 (s. Abbildung 5.2) zeigt, dass der Mittelwert der Bildübertragungszeit mit etwa 95 ms weniger als die Hälfte des festgelegten Grenzwerts von 200 ms beträgt. Auch die maximalen Ausreißer liegen unter diesem Schwellenwert. Das deutet darauf hin, dass es grundsätzlich möglich ist, mit der Redpanda-Streaming-Architektur Bilder in Echtzeit zu übertragen.



(a) Testfall 2: Sendedauer - Rohdaten (Liniendiagramm)

(b) Testfall 2: Sendedauer - bereinigt (Boxplot)

Abbildung 5.3: Testfall 2: Sendedauer - Rohdaten und bereinigt

Die Rohdaten des Testfalls 2 weisen bei der Übertragungs- und Sendedauer die selbe Auffälligkeit auf wie die des Testfalls 1, was die zuvor genannte Vermutung weiter bekräftigt. Die bereinigten Daten zur Sendedauer dieses Testfalls (s. Abbildung 5.3) zeigen keine signifikante Steigerung im Vergleich zu denen des Testfalls 1 und befinden sich in beiden Fällen bei durchschnittlich 73 ms. Dies deutet darauf hin, dass viele parallele Lesezugriffe die Dauer des Sendevorgangs nicht oder nur unwesentlich beeinflussen.
Die Analyse der bereinigten Daten zur Bildübertragungsdauer (siehe Abbildung 5.4) zeigt, dass der Durchschnittswert für alle Consumer, die in dieser Messung parallel Nachrichten empfangen haben, zwischen 140 ms und 160 ms liegt. Im Vergleich zu Testfall 1 mit einem einzelnen Consumer entspricht dies einer Steigerung der durchschnittlichen Bildübertragungszeit um 50 %. Das ist erwartungsgemäß, da Redpanda zusätzliche Ressourcen benötigt, um die Bilder gleichzeitig an mehrere Consumer zu verteilen. Die maximalen Ausreißer in den Messungen der Bildübertragungszeit liegen knapp über 220 ms, dennoch erfüllt die Übertragungszeit auch in diesem Testfall mit mehreren parallelen Consumern die

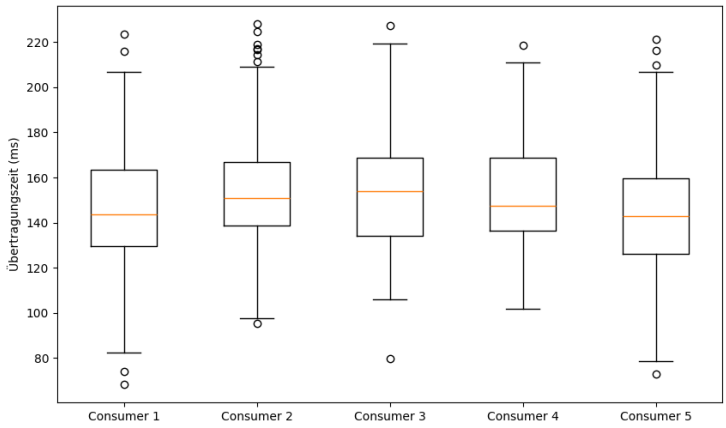


Abbildung 5.4: Testfall 2: Bildübertragungsdauer für alle Consumer - bereinigt (Boxplot)

in Abschnitt 3 definierte Anforderung zur weichen Echtzeit, da sich die durchschnittliche Übertragungszeit unter 200 ms befindet.
Insgesamt weisen die Testergebnisse darauf hin, dass die implementierte Architektur mit Redpanda grundsätzlich dazu geeignet ist, Bilder in weicher Echtzeit zu versenden. Die Architektur zeigt im Rahmen der Testbedingungen eine gute Skalierbarkeit, da bei einer höheren Anzahl Consumer keine überproportionalen Verluste in der Übertragungsgeschwindigkeit auftreten.
Die zu erwartende Anzahl Consumer für den Betrieb des Demonstrators entspricht etwa der des Testfalls 2. Allerdings wird der Demonstrator voraussichtlich nicht mit solch einer hohen Bilderrate betrieben, sondern eher genutzt, um einzelne Bilder zu versenden, was das System entlastet und somit Kapazitäten für weitere Auslastungen neben dem Bildversand freilässt. Andererseits wurden diese Tests unter optimalen Bedingungen ohne Einschränkungen durch das Netzwerk, welches die Clients mit Redpanda verbindet, durchgeführt. Daher ist im Betrieb des Demonstrators abhängig von der Netzwerkverbindung mit variierender Bildübertragungsdauer zu rechnen und die Echtzeitfähigkeit kann nicht garantiert werden.

6 Fazit

Ziel dieser Seminararbeit war es, eine Bild-Streaming-Architektur mit Redpanda zu entwickeln, und anhand dieser zu untersuchen, inwiefern sich Redpanda zum Echtzeit-Streaming von Bildern eignet. Dabei sollte die Implementation der Architektur auch die in Kapitel 3 gestellten Anforderungen an Skalierbarkeit und Benutzerfreundlichkeit erfüllen.
Um die Architektur benutzerfreundlich zu machen, wurde eine Kommunikationsarchitektur entworfen, welche ein einheitliches Nachrichtenformat festlegt. Mit diesem Format, dessen Einhaltung durch die Verwendung eines Avro-Schemas für die Serialisierung der Nachrichten sichergestellt wird, enthält jede Nachricht die nötigen Informationen, um die enthaltenen Nutzdaten zu beschreiben. Über das Redpanda Schema Registry werden die Versionen des Nachrichtenformats verwaltet, um trotz möglicher Erweiterungen Kompatibilität mit veralteten Clients zu gewährleisten. Dies verbessert außerdem die Skalierbarkeit, da das Hinzufügen neuer Komponenten mit erweiterter Nachrichtenstruktur die Funktionsfähigkeit anderer nicht beeinträchtigen.
Die Tests der implementierten Architektur zeigen, dass es möglich ist, mit Redpanda Bilder in Echtzeit zu versenden. Als Grenzwert für die Bildübertragungsdauer wurde in Kapitel 3 ein Durchschnitt von 200 ms festgelegt. Dies wurde in einem Test mit einem Producer und einem Consumer mit einer durchschnittlichen Bildübertragungszeit von 95 ms deutlich unterschritten. Ein zweiter Test mit dem Ziel, die Skalierbarkeit der Architektur zu untersuchen, bei dem Bilder von der fünffachen Menge an Consumern empfangen wurden, zeigte einen Anstieg der durchschnittlichen Bildübertragungszeit um 50 % auf 150 ms. In beiden Fällen erfüllt die implementierte Architektur die Anforderungen zur Echtzeit und zeigt dabei eine gute Skalierbarkeit, da die Bildübertragungsdauer unterproportional zur Anzahl Consumer ansteigt.
Für den Einsatz in dem Demonstrator, der die Anzahl der Clients aus dem zweiten Test vermutlich nicht oder nicht stark überschreiten wird, ist die Architektur geeignet. Es ist jedoch zu erwähnen, dass die Testfälle, welche eine gute Echtzeitfähigkeit der Architektur zeigen, unter optimalen Bedingungen ohne Einschränkungen durch das Netzwerk zwischen den Clients und Redpanda durchgeführt wurden. Daher sollte im Betrieb des Demonstrators mit dieser Architektur abhängig von der Netzwerkverbindung mit höheren Latenzen in der Bildübertragung gerechnet werden.
Die im Test verwendeten Bilder mit einer Größe von 14,4 MB besitzen bereits eine hohe Auflösung. Daher wäre es sinnvoll zu prüfen, ob durch eine schnelle Bildkomprimierung mittels effizienter Algorithmen die Bildübertragungszeit noch weiter reduziert werden kann. Des Weiteren ist noch nicht bekannt, welche Datendurchsatzrate oder Anzahl von Clients die Architektur hinsichtlich ihrer Echtzeitfähigkeit maximal bewältigen kann. Eine

weiterführende Untersuchung könnte aufzeigen, unter welchen Lasten die Architektur leistungsstark bleibt und würde dazu beitragen, geeignete Anwendungsfälle für derartige Architekturen besser eingrenzen zu können.

A Literaturverzeichnis

[1] Fraunhofer IPT, "Qualitätsmanagement – Datengetriebene Qualitätssicherung von Produktionsprozessen." https://www.ipt.fraunhofer.de/de/angebot/strategieplanung-umsetzung/qualitaetsmanagement.html, abgerufen am 20.11.2024.

[2] I. Effenberger, H. Eigenbrod, A. Frommknecht, C. Jauch, J. Denecke, and M. Huber, "Qualitätssicherung in der Produktion," in *Handbuch Industrie 4.0* (M. Ten Hompel, B. Vogel-Heuser, and T. Bauernhansl, eds.), pp. 1–17, Berlin, Heidelberg: Springer Berlin Heidelberg, 2023.

[3] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, July 2018.

[4] "5G-Technologie." https://www.ipt.fraunhofer.de/de/angebot/digitalisierung/5g.html#cop1, abgerufen am 09.12.2024.

[5] K. Salonitis, J. Paralikas, and G. Chryssolouris, "Roll Forming," in *CIRP Encyclopedia of Production Engineering* (L. Laperrière and G. Reinhart, eds.), pp. 1076–1079, Berlin, Heidelberg: Springer, 2014.

[6] H. Wörn and U. Brinkschulte, "Echtzeitprogrammierung," in *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen*, pp. 317–342, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[7] R. Sahal, J. G. Breslin, and M. I. Ali, "Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case," *Journal of Manufacturing Systems*, vol. 54, pp. 138–151, Jan. 2020.

[8] R. Manchana, "Event-Driven Architecture: Building Responsive and Scalable Systems for Modern Industries," *International Journal of Science and Research (IJSR)*, vol. 10, pp. 1706–1716, Jan. 2021.

[9] "Apache Kafka." https://kafka.apache.org/, abgerufen am 03.12.2024.

[10] "RabbitMQ: One broker to queue them all | RabbitMQ." https://www.rabbitmq.com/, abgerufen am 09.12.2024.

[11] "Pub/Sub für Anwendungs- und Datenintegration." https://cloud.google.com/pubsub, abgerufen am 09.12.2024.

[12] "What is Redpanda? | Redpanda." https://www.redpanda.com/what-is-redpanda, abgerufen am 02.12.2024.

[13] "Redpanda | Redpanda vs Kafka." https://www.redpanda.com/compare/redpanda-vs-kafka, abgerufen am 03.12.2024.

[14] "Introduction to Redpanda | Redpanda Self-Managed." https://docs.redpanda.com/current/get-started/intro-to-events/, abgerufen am 25.11.2024.

[15] "Manage Topics | Redpanda Self-Managed." https://docs.redpanda.com/current/develop/config-topics/, abgerufen am 25.11.2024.

[16] "How Redpanda Works | Redpanda Self-Managed." https://docs.redpanda.com/current/get-started/architecture/, abgerufen am 05.12.2024.

[17] "Configure Producers | Redpanda Self-Managed." https://docs.redpanda.com/current/develop/produce-data/configure-producers/, abgerufen am 25.11.2024.

[18] "Redpanda Schema Registry | Redpanda Self-Managed." https://docs.redpanda.com/current/manage/schema-reg/schema-reg-overview/, abgerufen am 03.12.2024.

[19] Sparkplug Specification Project Team, "Sparkplug 3.0.0: Sparkplug Specification," pp. 18–30, Oct. 2022.

[20] M. Zeng, M. Rudack, M. Moser, S. Ulrich, A. Gannouni, A. Abdelrazeq, A. Bührig-Polaczek, and R. H. Schmitt, "IESF: Interval Event Streaming Format for the Data Lake of Production," in *2023 Eighth International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 159–166, Sept. 2023.

[21] "confluent-kafka-python/README.md at master · confluentinc/confluent-kafka-python." https://github.com/confluentinc/confluent-kafka-python/blob/master/README.md, abgerufen am 03.12.2024.

[22] "kafka-python: Pure Python client for Apache Kafka." https://github.com/dpkp/kafka-python, abgerufen am 09.12.2024.

[23] "pykafka: Full-Featured Pure-Python Kafka Client." https://github.com/Parsely/pykafka, abgerufen am 09.12.2024.

[24] "Apache Avro™ 1.12.0 Documentation." https://avro.apache.org/docs/1.12.0/, abgerufen am 04.12.2024.

[25] "OpenCV: Introduction to OpenCV-Python Tutorials." https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html, abgerufen am 03.12.2024.

[26] "Kafka producer configuration reference | Confluent Documentation." https://docs.confluent.io/platform/current/installation/configuration/producer-configs.html#max-request-size, abgerufen am 09.12.2024.

[27] "Kafka Consumer configuration reference for Confluent Platform | Confluent Documentation." https://docs.confluent.io/platform/current/installation/configuration/consumer-configs.html, abgerufen am 09.12.2024.

[28] "Manage Disk Space | Redpanda Self-Managed." https://docs.redpanda.com/current/manage/cluster-maintenance/disk-utilization/, abgerufen am 09.12.2024.

B Tabellenverzeichnis

C Abbildungsverzeichnis