FernUniversität in Hagen

Lehrgebiet Datenbanken und Informationssysteme

Masterarbeit

Data Mesh: Lösungsansätze für Data Engineering Pipelines

Vorgelegt von

Sabine Krause, Matrikel-Nr. 6343694

Hagen, 28.05.2024

Prüfer:	Prof. Dr. Uta Störl
Betreuer:	Maximilian Plazotta (Universität Regensburg)
	Prof. Dr.-Ing. habil. Meike Klettke (Universität Regensburg)

Für Danny,
Felix und Carla.

Für Enya.

Abstract

Aufgrund des zunehmenden Aufkommens an Daten, die in Umfang und Vielfalt die Leistungsfähigkeit gängiger Systeme und Organisationen überfordern, gibt es einen Paradigmenwechsel beim bisherig zentralisierten Ansatz der analytischen Datenverarbeitung. Unter dem Begriff *Data Mesh* wird ein dezentraler Weg beschritten, der sowohl technisch als auch organisational neue Anforderungen an Unternehmen stellt. Dabei werden Konzepte, wie *Domain Driven Design* und *Product Thinking* in den Vordergrund zukünftiger Entwicklungen gestellt. Die Umsetzung der Data Mesh-Prinzipien erfolgt von Unternehmen zu Unternehmen individuell. Insbesondere die Data Engineering Pipeline, die in bisherigen Architekturen als zentrale Ader alle Speichersysteme miteinander verband, durchläuft den Paradigmenwechsel unter vielen Herausforderungen. Diese Herausforderungen sollen in der vorliegenden Arbeit näher beleuchtet werden. Dabei stellt sich die Frage, welchen Einfluss die Data Mesh-Prinzipien auf die Transformationsprozesse ausüben. Ein möglicher Lösungsansatz für die Abbildung von Data Mesh auf die Data Engineering Pipeline soll anhand einer Beispiel-Implementierung erarbeitet und bewertet werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

API	*Application Programming Interface*
BARC	*Business Application Research Center*
BI	*Business Intelligence*
BMDV	*Bundesministeriums für Digitales und Verkehr*
DWH	*Data Warehouses*
ELT	*Extract, Load, Transform*
ETL	*Extract, Transform, Load*
GDWS	*Generaldirektion Wasserstraßen und Schifffahrt*
ITZBund	*Informations Technik Zentrum Bund*
JSON	*Java Script Object Notation*
KPI	*Key Performance Indikator*
ML	*Machine Learning*
NHN	*Normalhöhennull*
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
PNP	*Pegelnullpunkt*
REST	*Representational State Transfer*
RPC	*Remote Procedure Call*
SLA	*Service Level Agreement*
SLO	*Service Level Objective*
SOAP	*Simple Object Access Protocol*
SRE	*Site Reliability Engineering*
URL	*Uniform Ressource Locator*
WSV	*Wasserstraßen- und Schifffahrtsverwaltung des Bundes*

Selbständigkeitserklärung

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

1 Einleitung

Mit dem Begriff *Data Mesh* wird ein fundamentaler Paradigmenwechsel eingeleitet, der die Art, wie mit Daten umgegangen wird, auf ein neues Fundament stellt. Data Mesh veranlasst Unternehmen ihre bisherigen Organisationsstrukturen hinsichtlich der bisher gelebten Daten-Strategie zu überdenken, wobei die Einflüsse des *Data Driven Designs* und des *Product Thinking* einen starken Einzug in die Architektur-Planungen erhalten.

Data Mesh stützt sich auf ausgewählte Prinzipien, deren Umsetzung weder fest definiert noch technologisch untermauert werden. Eine Blaupause für die Datenarchitektur sucht man vergebens. Unternehmen, die den Schritt wagen, ihre Datenarchitektur im Sinne eines Data Mesh zu implementieren, stellen sich der Herausforderung, ein Umdenken in den eigenen Reihen voranzutreiben. Die bisher „nebenbei“ gewonnen Analysedaten werden als wertvoller Rohstoff betrachtet und in den Verantwortungsbereich einer Fach-Domäne gelegt.

Aufgrund des Umfangs dieses Themenbereichs, klammert die nachfolgende Arbeit die organisationalen Herausforderungen aus, und blickt stattdessen aus einer technischen Perspektive auf die Datenarchitektur. Im Fokus steht die Data Engineering Pipeline unter dem Einfluss von Data Mesh. Dazu werden in einem ersten Schritt die Grundlagen von Data Mesh betrachtet (Kap. 2) worauf im Anschluss der analytische Datenverarbeitungsprozess beleuchtet wird (Kap. 3). Anhand einer qualitativen Inhaltsanalyse mittels Literaturrecherche, werden die Anforderungen an Pipelines durch Data Mesh zusammengetragen (Kap. 4). Am Ende des Kapitels werden die gewonnen Erkenntnisse als Charakteristiken für den Lösungsansatz zusammengefasst. Kapitel fünf setzt sich mit der Anwendung des Lösungsansatzes auseinander. Dazu werden eine ausgewählte technologische Basis sowie ein geeigneter Anwendungsfall beschrieben. Anschließend wird die, anhand der Charakteristiken implementierte Lösung, vorgestellt. Es folgt eine Bewertung aufgrund derer sich ein Ergebnis ableiten lässt (Kap. 6). Die Arbeit schließt mit einem Fazit und Ausblick auf weitere Fragen zur Data Mesh Thematik ab (Kap. 7).

1.1 Problemstellung / Motivation

Durch das Voranschreiten der Digitalisierung in Industrie und täglichem Leben, sehen sich Unternehmen von einer stetig zunehmenden Anzahl an Daten konfrontiert. Mit Aufkommen neuer Technologien zur Datengenerierung, der Verarbeitung von Daten und deren Speicherung, werden Forderungen nach einem effizienteren Umgang mit den Daten deutlicher. Die Komplexität in der Datenverwaltung gewinnt stetig hinzu, da Daten nicht mehr nur als

„Nebenprodukt“ betrachtet, sondern auch als treibender Faktor im Bereich der Datenanalyse und der Künstlichen Intelligenz wahrgenommen werden.

Vor diesem Hintergrund finden neue Ansätze im Umgang mit den Daten Einzug in die IT-Landschaft der Unternehmen. Traditionelle Konzepte, wie die Zentralisierung der Datenhaltung, werden aufgebrochen wobei dem Lebenszyklus und den Eigenschaften der Daten mehr Beachtung geschenkt wird. Faktoren, wie das Datenvolumen (*Volume*), dessen Variantenreichtum (*Variety*) sowie die Beweglichkeit (*Velocity*) der Daten, sind nur einige der charakteristischen Merkmale, die unter dem Begriff „Big Data“ zusammengefasst werden (Gartner, 2024). Aber auch weitere Aspekte, wie der unternehmensstrategische Wert (*Value*) sowie die Widerspruchsfreiheit der Daten (*Veracity*), werden zunehmend in diesem Zusammenhang aufgeführt (Ishwarappa & J., 2015). Große Aufmerksamkeit erfahren zudem auch Aspekte der Skalierbarkeit, wobei die Herkunft der Daten und Ihre Verbindung zueinander eine wesentliche Rolle spielen; ungeachtet dessen, wo die Daten tatsächlich liegen.

Ein Ansatz, diese Entwicklung bestmöglich zu unterstützen sowie die zunehmende Komplexität beherrschbar zu machen, wird durch den Begriff „Data Mesh“ beschrieben.

Das Data Mesh Konzept wurde erstmalig 2019 durch Zhamak Dehghani bekannt gemacht. Als Expertin für verteilte Systeme und Datenarchitektur in großen Unternehmen, beschreibt sie ein sozio-technisches Konzept, welches Daten dezentralisiert und in autonomen Domänen organisiert (Dehghani, 2019).

Ein großer Anteil der Komplexität ist auf die Aufteilung von Daten in operative und analytische Daten zurückzuführen. Analytische Daten werden gesammelt und in operative Daten transformiert. Letztere trainieren Machine-Learning-Modelle, welche als intelligente Dienste wieder in die operativen Systeme einfließen (vgl. Dehghani, 2023, S. 143). Demzufolge existiert eine gewisse Zusammengehörigkeit zwischen den analytischen und operativen Daten, die über eine fragile Integrationsarchitektur via Data Engineering Pipelines verwaltet wird.

Andere Probleme adressieren die Organisationsstruktur der Unternehmen. Zentrale Datenteams, werden der aufkommenden Datenflut und -geschwindigkeit sowohl aufgabentechnisch aber auch fachlich nicht mehr gerecht, so dass sich Engpässe (engl. *Bottlenecks*) bilden, die den gesamten Datenverarbeitungsprozess ausbremsen.

Data Mesh bietet einen guten Ansatz für die Datenverarbeitung, um die Herausforderungen an die analytische Datenwelt beherrschbar zu machen. In der Praxis stellen sich jedoch zahlreiche Herausforderungen, die sowohl technisch als auch organisatorisch zu verbuchen sind. Gründe sind die vorherrschenden Marktstellungen von Anbietern zentralisierter Lösungen, die große

Investitionen in die Bereitstellung ihrer Modelle getätigt haben. Zudem werden bewährte Software-Entwicklungs-Praktiken, wie Testen und Refactoring, nicht ohne Weiteres unterstützt, die aber wichtige Techniken sicherer, operativer Systeme darstellen (vgl. Dehghani, 2023, Vorwort von M. Fowler).

In dieser Masterarbeit soll die Frage beantwortet werden, wie sich das Konzept von Data Mesh auf den Datenverarbeitungsprozess auswirkt und wie es sich auf die bisherige Pipeline-Verarbeitung der Daten abbilden lässt.

1.2 Zielsetzung

Data Mesh adressiert zwei wesentliche Herausforderungen, der sich Unternehmen mit Aufkommen von Big Data stellen müssen:

Zum einen handelt es sich um die Vermeidung von Bottlenecks in den Verarbeitungsschritten, und zum anderen um die Aufbereitung der Daten auf eine Art, dass alle beteiligten Instanzen mit ihnen arbeiten können (*Data as a Product*). Bekannte Konzepte des *Domain-Driven-Designs* sowie des *Product Thinkings* werden fokussiert. In diesem Zusammenhang werden Domänen-Teams neu zusammengesetzt, die sich um die Verwaltung eines „Datenproduktes" kümmern (*Data Ownership*).

Diese Arbeit setzt ihren Schwerpunkt auf Betrachtung der Pipelines innerhalb des Data Engineering Prozesses. Eine Pipeline transportiert die Daten aus deren Quellen in ein Zielsystem. Die Daten durchlaufen dabei verschiedene Stationen, in denen sie extrahiert, bereinigt, je nach Zweck transformiert und schließlich dem Zielsystem bereitgestellt werden. Unterstützt werden diese Schritte durch eine Sammlung an Tools und Prozessen.

Das Ziel dieser Masterarbeit besteht in der Entwicklung von Lösungsansätzen für eine Data Engineering Pipeline nach den Prinzipien von Data Mesh. Dazu soll geprüft werden, ob sich das dezentralisierte Datenplattformkonzept von Data Mesh sowie alle weiteren Prinzipien effizient auf eine Data Engineering Pipeline abbilden lassen.

Es stellt sich die folgende Frage, die im Laufe der Masterarbeit beantwortet werden soll:

- Welche Herausforderungen gibt es bei der Übertragung des Data Mesh Konzepts auf eine Data Engineering Pipeline?

Die ausgearbeiteten Herausforderungen und Erkenntnisse dienen der Beantwortung der zweiten Forschungsfrage, die als Kernfrage der Arbeit verstanden werden kann:

- Wie lässt sich das Data Mesh Konzept auf eine Data Engineering Pipeline abbilden?

Die Literatur gibt kaum Anhaltspunkte über die Voraussetzungen, die Data Mesh an eine Data Engineering Pipeline stellt. Der Praxisleitfaden des Digitalverbands Bitkom aus dem Jahr 2022 verdeutlicht, dass es hier noch viele offene Fragestellungen gibt (Bitkom e.V., 2022).

Diese Umstände bieten einen guten Grund dafür, das Themenfeld im Rahmen einer Masterarbeit mittels qualitativer Inhaltsanalyse als Basis für diese Thematik auszuarbeiten.

1.3 Methode

Zur Beantwortung der Forschungsfrage, wie sich das Data Mesh Konzept auf die Data Engineering Pipeline abbilden lässt, erfolgt zu Beginn eine Auseinandersetzung mit den Eigenschaften von Data Mesh. Mittels einer qualitativen Inhaltsanalyse anhand einer Literaturrecherche, wird auf folgende Aspekte eingegangen:

- Vier Grundprinzipien von Data Mesh
- Mehrwert von Data Mesh
- Abgrenzung zu früheren Ansätzen des analytischen Datenmanagements

Ebenso werden der analytische Datenverarbeitungsprozess sowie die einzelnen Stationen einer Data Engineering Pipeline näher betrachtet.

Ziel dieses Vorgehens ist es eine theoretische Basis aufgrund von Literatur und verwandten Arbeiten zu schaffen. Als Ausgangsmaterial dienen auch Praxisbeispiele und Studien verschiedener Umsetzungsprojekte.

Sind die Grundlagen gelegt, soll untersucht werden, welche Herausforderungen Data Mesh an die Data Engineering Pipeline stellt.

Die qualitative Inhaltsanalyse bietet sich als geeignete Forschungsmethode an, da aufgrund des neuartigen Ansatzes der Thematik vermehrt theoretische Konzepte vorliegen. Die meisten Herausforderungen zeichnen sich in der Gestaltung der Plattform und der Automatisierung der Governance ab. Bisherige Lösungen zur Umsetzung basieren auf lose zusammengesteckte Komponenten, die individuell aufeinander abgestimmt werden. Automatisierung ist zudem stark auf den Software-Entwicklungsprozess fokussiert (Niehoff, 2023). Weitere Bedenken werden zudem im Change Management, in der Abwendung vom Spezialistentum hin zum Generalisten-Pool, der doppelten Datenhaltung sowie in isolierten Infrastruktur-Komponenten, die ebenfalls zur Dopplung führen, wahrgenommen (vgl. Goedegebuure et al., 2023, S. 16).

Das Business Application Research Center (BARC) hat im Rahmen einer europaweiten Anwenderstudie zum Thema Data Mesh, ein zunehmendes Interesse an der Thematik

festgestellt (BARC, 2023). Aus dem Ergebnis der mehr als 300 befragten Personen geht hervor, dass 88% einen dezentralen Ansatz zur Organisation von Daten befürworten, jedoch nur 7 % der Unternehmen, die Verantwortung für Datenbestände auch tatsächlich auf die Fachbereiche übertragen möchten.

Im Rahmen der Arbeit wird kein Fokus auf eine spezielle Branche gelegt, somit bietet dieses Vorgehen eine neutrale Perspektive auf die Besonderheiten von Data Mesh.

Im Anschluss werden Charakteristiken ermittelt, um die Sinnmäßigkeit und Qualität des Lösungsansatzes zu bewerten. Nach Testung des Lösungsansatzes, werden die gewonnen Ergebnisse diskutiert in einem ausführlichen Fazit wiedergegeben.

Es folgt ein Ausblick auf offene Fragestellungen, die eine Grundlage für weitere wissenschaftliche Ausarbeitungen darstellen können.

2 Grundlagen von Data Mesh

Der Begriff „Data Mesh“ beschreibt den dezentralen, soziotechnischen Entwurf einer Datenarchitektur, deren Schwerpunkt auf der Bereitstellung, dem Zugriff sowie der Verwaltung analytischer Daten im großen Unternehmensumfeld liegt (Dehghani, 2023). Durch die Zunahme an Bedeutung für wichtige strategische und technologische Ausrichtungen der Unternehmen, werden analytische Daten von ihrem Status als „Nebenprodukt“ der operativen Daten hin zu entscheidungsbildenden „Datenprodukten“ aufgewertet. Die Bezeichnung „Data as a Product“ kennzeichnet somit den ersten Teil des Data Mesh-Begriffs. Die zweite Komponente versinnbildlicht die Organisation und Verwaltung der Daten in Form eines losen, zusammengesetzten Netzwerks (engl. *Mesh*). Die Daten werden in sog. Daten-Domänen organisiert, die dezentral und autonom miteinander interagieren, wobei die Kommunikation zwischen den Domänen über entsprechende Schnittstellen und Richtlinien erfolgt.

Eine nicht unwesentliche Rolle spielt der Mensch in dieser Architektur. Data Mesh veranlasst sowohl ein Umdenken in den technischen Lösungen als auch in sozialen Strukturen eines Unternehmens. Die Art und Weise, wie mit analytischen Daten verfahren wird, wird überdacht und neuorganisiert.

2.1 Die vier Grundprinzipien

Das Data Mesh Konzept befasst sich mit der Anpassung der Datenarchitektur an die Herausforderungen des technischen Fortschritts. Dabei berücksichtigt es Veränderungen in der Datenlandschaft, die zunehmende Anzahl von Datenquellen, die Vielfalt von Anwendungsfällen und Nutzern sowie die Anforderungen an die Reaktionsgeschwindigkeit bei Veränderungen (Dehghani, 2020).

Diese Aspekte werden bei Data Mesh auf Basis von vier Grundprinzipien behandelt:

- Domain Ownership
- Data as a Product
- Self-Serve Data Platform
- Federated Computational Governance

Jede dieser Prinzipien bietet Lösungen an, die neuen Herausforderungen, sowohl technologisch als auch organisatorisch, auf die Datenlandschaft von Unternehmen abzubilden. Insbesondere soll die große Kluft zwischen operativen und analytischen Daten beseitigt werden.

Während **operative Daten** einen transaktionalen Charakter haben, durch Geschäftsfunktionen und Microservices generiert und oft in großen Datenbanksystemen gespeichert werden, handelt es sich bei **analytischen Daten** um eine zeitliche und aggregierte Sicht auf die Fakten eines Unternehmens. Analytische Daten können im Laufe der Zeit modelliert werden und retrospektive und zukunftsorientierte Erkenntnisse liefern (z. B. für ML-Trainingsmodelle, Analyseberichte).

Domain Ownership

Dieses Prinzip besagt, dass analytische Daten der fachlichen Domäne zugeordnet werden, die das meiste Wissen über diese Daten besitzt. Daraus folgt, dass analytische Daten dezentralisiert verwaltet werden. Sie werden entweder der Datenquelle, die sie generiert oder der Domäne des Hauptnutzers zugeordnet. Die Motivation für diese Zuordnung liegt in folgenden Punkten (vgl. Dehghani, 2023, S. 48):

- Bereitstellung der Daten ist entsprechend dem Unternehmenswachstum skalierbar.
- Optimierung für den kontinuierlichen Wandel dadurch, dass sich Veränderungen in den jeweiligen Domänen schnell identifizieren lassen.
- Förderung von Agilität und Beseitigung zentralisierter Engpässe bei Datenteams.
- Verbesserte Datenqualität, da Datenherkunft und -nutzung nah beieinander liegen.
- Erhöhung der Resilienz durch Abschaffung aufwendiger Daten-Pipelines.

Data as a Product

Domänenorientierte Daten werden als Produkt und mit hoher Priorität bewertet. Datennutzer (Data Analysten, Data Scientists usw.) wiederum, agieren als Kunden, deren Bedürfnisse erfüllt werden sollen. Hierbei ist wichtig zu erwähnen, dass Daten-Pipelines sowie die Herausforderungen der Speicherstrukturen und -systeme nachgelagert betrachtet werden (Dehghani, 2019). Um die Bereitstellung von Daten zu ermöglichen, bietet das Datenprodukt eine Reihe von explizit definierten und einfach zu verwendenden Schnittstellen.

Folgende Ziele werden durch dieses Prinzip adressiert (vgl. Dehghani, 2023, S. 49):

- Vermeidung von Datensilos durch domänenübergreifende Bereitstellung gezielter Daten für relevante Teams. Auf eine private Datensammlungen wird verzichtet.
- Der einfache Zugriff auf hochwertige Daten wird effizient unterstützt.
- Durch Isolierung der Daten innerhalb der Datendomänen, sind diese resilient gegenüber Veränderungen.
- Bereitstellung über organisatorische Grenzen hinaus schafft höhere Wertschöpfung.

Der Produkt-Gedanke basiert auf den FAIR-Prinzipien[1] (Wilkinson et al., 2016), die als Akronym für die Begriffe Auffindbarkeit (engl. *Findable*), Zugänglichkeit (engl. *Accessible*), Interoperabilität (engl. *Interoperable*) und Wiederverwendbarkeit (engl. *Reusable*) stehen (GO FAIR Initiative, 2016). Data Mesh erweitert diese Prinzipien um weitere Usability-Merkmale. Insgesamt werden folgende Merkmale zugrunde gelegt (Dehghani, 2019):

- Discoverable (Auffindbar)
- Addressable (Adressierbar)
- Trustworthy (Vertrauenswürdig und wahr)
- Self-Describing (Selbsterklärend)
- Interoperable (Interoperable)
- Secure (Sicher)

Self-Serve Data Platform

Mittels dieses Prinzips wird eine Generation der Datenbereitstellung vorangetrieben. Ziel ist es, die Hindernisse bei der Bereitstellung von Daten zu beseitigen und Daten gemäß ihres Lebenszyklus – von der Datenquelle bis zur Nutzung – zugreifbar zu machen. Moderne Plattformdienste sind in der Lage den gesamten Lebenszyklus der Datenprodukte zu managen und diese auffindbar zu machen, wobei sie die Erstellung, das Deployment und die Wartung von Datenprodukten erleichtern. Sie verwalten ein Netz aus miteinander verbundenen Datenprodukten.

Folgende Ziele lassen sich ableiten (vgl. Dehghani, 2023, S. 50):

- Kostensenkung durch den dezentralen Datenbesitz.
- Reduktion der Komplexität des Datenmanagements durch Verringerung des Cognitive Load[2] von Domänenteams.
- Reduktion des Bedarfs spezialisierter Data Engineers durch den Einsatz generalistischer Entwickler.
- Voranschreiten der Automatisierung von Governance-Policies zur Schaffung und Wahrung gemeinsamer Standards und Regeln für Datenprodukte.

[1] Erstmalige Erwähnung der Fachzeitschrift Data Scientist, 2016.
[2] Der Begriff Cognitive Load bezeichnet die kognitive Belastung des Gedächtnisses und damit eine Beanspruchung desselbigen über seine Kapazitätsgrenzen hinweg. Er geht aus einer Theorie von Paul Chandler und John Sweller (Chandler und Sweller, 1991) hervor.

Federated Computational Governance

Um ein Gleichgewicht zwischen Autonomie und Agilität sowie domänenübergreifender Interoperabilität zu schaffen, sind klar definierte Regeln und Verträge für alle Datenprodukte erforderlich. Federated Computational Governance schafft ein Betriebsmodell für Data Governance, welches auf einer föderalen Entscheidungs- und Verantwortungsstruktur basiert. Es setzt sich zusammen aus einem Team aus Domänenvertretern, Experten für Rechtsfragen, Compliance und Sicherheit usw. Dieses Betriebsmodell setzt dabei auf eine in hohem Maße umgesetzte Automatisierung der Policies.

Folgende Ziele lassen sich identifizieren (vgl. Dehghani, 2023, S. 50-51):

- Aggregation und Korrelation von unabhängigen, interoperablen Datenprodukten schaffen einen Mehrwert.
- Vermeidung unerwünschter Folgen der Dezentralisierung von Datenprodukten (Isolation, Inkompatibilität usw.).
- Übergreifende Governance-Anforderungen (Sicherheit, Datenschutz, Compliance usw.) können zentral definiert und überwacht werden.
- Reduktion des manuellen Abstimmungsaufwands zwischen Domänen- und Governance-Team.

2.2 Mehrwert von Data Mesh

Data Mesh greift bisherige Probleme im analytischen Datenmanagement großer und komplexer Organisationen auf und bietet ein Konzept, diese Schwachstellen zu entkräften. Mehrwert verspricht sich Data Mesh in den folgenden Punkten (vgl. Dehghani, 2023, S. 150):

- Reaktion auf Veränderungen hinsichtlich Komplexität, Volatilität und Ungewissheit.
- Umsetzung von Agilität trotz Wachstum.
- Verbesserung des Kosten-Nutzen-Verhältnisses.

Zhamak Dehghani geht von einer „Omnipräsenz von Daten“ aus und beschreibt in Ihrem Werk, dass „*die Komplexität der Geschäftswelt* – kontinuierliches Wachstum, Veränderung und Vielfalt – der Normalzustand ist.“ (Dehghani, 2023, S. 149). Demzufolge liegt der Fokus auf der Reduzierung von „Zentralisierungspunkten“, die Dehghani als prozessualen Flaschenhals identifiziert. Dabei tritt ein zunehmend asynchroner Datenaustausch in den Vordergrund, der bisherige, zur Verlangsamung neigende Synchronisationsaufwände aufbricht. Daten werden vielmehr an dem Ort der Entstehung verarbeitet, was dazu führt, unbeabsichtigte Komplexität bzgl. des Transfers zu beseitigen. An dieser Stelle wird insbesondere die Pipeline-Verarbeitung angesprochen.

Reaktion auf Veränderungen

Eine angepasste Reaktion auf Veränderungen wird durch die veränderte Ausrichtung von Fachlichkeit und Technologie auf die analytischen Daten erreicht. Ein gängiges Mittel zur Beherrschung von Komplexität ist die Aufteilung in kleine handhabbare Teile. Dies erfolgt meist im Bereich der Fachabteilungen. Data Mesh empfiehlt eine solche Aufteilung auch in der IT umzusetzen, wie es bei modernen Unternehmen im Falle der operativen Daten meist praktiziert wird. Analytische Daten sollten demselben Prinzip folgen. Demnach übernimmt jede Abteilung die Verantwortung für ihre analytischen Daten. Aus diesem „Domain Ownership" resultiert eine verteilte Datenarchitektur, in der alle relevanten Datenartefakte von den verantwortlichen Domänen verwaltet werden (vgl. Dehghani, 2023, S. 151).

Weitere Aspekte behandeln die **Integration der operativen und analytischen Welt**, indem beide Welten durch eine neue Struktur miteinander verbunden werden. Ein Geflecht aus Peer-to-Peer-Datenprodukten und -anwendungen vermeidet umfangreiche Feedback-Schleifen dadurch, dass analytische Daten als Produkt bereitgestellt werden und sich bereits an den Domänen orientieren. Das Konzept zentralisierter Pipelines zwischen beiden Welten wird abgeschafft. Die Bereitstellung gestaltet sich eher simpel und unspektakulär in einer einfachen Übertragung von Daten von einer in die andere Welt. Die fachliche Logik und der Code, die für die Umwandlung der operativen Daten in die analytischen Daten erforderlich sind, wird in den Datenprodukten integriert und entsprechend abstrahiert.

Die **Dezentralisierung von Datenänderungen** sorgt dafür, dass die strikte Synchronisierung von Änderungen entfällt. An deren Stelle tritt die Autonomie einer jeden Domäne, auf Grundlage ihrer fachlichen Kenntnisse und Befugnisse, zu agieren. Von einer zentral verwaltenden Komponente wird abgesehen, was dazu führt, dass Domänen die Möglichkeit haben, ihre Daten kontinuierlich weiterzuentwickeln. Voraussetzung hierfür sind wohldefinierte und zugesicherte Verträge (vgl. Dehghani, 2023, S. 154).

Eine **Beseitigung von unbeabsichtigter Komplexität** wird erreicht, indem Datenprodukte angeboten werden, die möglichst keine Pipelines erfordern. Das lässt sich erreichen, indem Datenprodukte zwar eine domänenorientierte Datensemantik kapseln und gleichzeitig entsprechende Schnittstellen anbieten, die die Daten für unterschiedliche Anwendungsfälle und Nutzer bereitstellen (vgl. Dehghani, 2023, S. 154-155).

Agilität trotz Wachstum

An dieser Stelle liegt der Fokus darauf, im Rahmen des Wachstumsprozesses anfallende Abstimmungen und Synchronisierungen zu reduzieren, um agiler und flexibler zu agieren.

Domänen sollen ermächtigt werden, ihre Ergebnisse mit wenigen Abhängigkeiten und möglichst eigenständig zu erzielen.

Unterstützt wird dieser Aspekt durch die **Beseitigung zentralisierter und monolithischer Komponenten**. Dazu gehören Data Warehouses (DWH) und Data Lakes. Die Anzahl der einzubindenden Quellen und der zu bedienen Anwendungsfälle reduziert die Agilität dadurch, dass sie durch ein zentralisiertes Datenteam verwaltet werden. Data Mesh setzt auf einen Peer-to-Peer-Ansatz zur Verknüpfung der Daten auf (vgl. Dehghani, 2023, S. 155-156).

Obwohl moderne technologische Entwicklungen zu einer schnelleren parallelen Verarbeitung von Nachrichten beigetragen haben, wurden interne organisatorische Abstimmungsprozesse nicht betrachtet. Demzufolge bleiben die Resultate abhängig davon, wie Menschen und Teams ihre Aufgaben intern koordinieren. Data Mesh strebt die **Reduzierung des Koordinationsaufwands von Pipelines** an, und spricht sich für explizite Schnittstellen und angepasste Verträge aus (vgl. Dehghani, 2023, S. 156-157).

Eine Reduzierung des Koordinationsaufwands durch Data Governance wird vermieden, indem stark manuell geprägte Prozesse der Data Governance aufgebrochen werden. Durch eine „Automatisierung und Einbettung von *Policies als Code* in einzelne Datenprodukte“ (Dehghani, 2023, S. 157), wird eine in hohem Maße geforderte Automatisierung von Governance-Aspekten angestrebt. Zudem werden zentrale Zuständigkeiten in die Datendomänen hinein übertragen. Dazu werden die Data Product Owner der jeweiligen Domänen mit entsprechenden Zuständigkeiten versehen (vgl. Dehghani, 2023, S. 157-158).

Data Mesh setzt sich zur **Förderung der Autonomie** ein. Ziel ist die Schaffung eines Gleichgewichts zwischen Teamautonomie und Zusammenarbeit mit anderen Teams herzustellen. Zur Entwicklung und Betrieb der Datenprodukte wird Domänenteams durch das Prinzip der Self-Serve Data Platform eine domänenagnostische Datenplattform zur Verfügung gestellt, die es ermöglicht, den Lebenszyklus ihrer Datenprodukte unabhängig zu verwalten (vgl. Dehghani, 2023, S. 159).

Verbesserung des Kosten-Nutzen-Verhältnisses

Data Mesh stellt das Wesen der analytischen Daten in den Vordergrund und verspricht sich dadurch einen schnelleren Zugang zu Informationen, die von strategischem Wert für ein Unternehmen sein können. Demzufolge wird im Vorfeld in die Schaffung eines neuartigen Typs von Datenplattform investiert, der sich jedoch langfristig betrachtet, auszahlen kann. Die zunehmende Komplexität und Vielfalt der Daten erfordert schnellere und angepasste Bereitstellungsmodelle, die durch Konzepte wie Product Thinking und Data-Ownership vorangetrieben werden (vgl. Dehghani, 2023, s. 159).

2.3 Abgrenzung zu bisherigen Datenarchitekturen

Um Data Mesh in seiner Neuartigkeit zu erfassen, ist es erforderlich darzustellen, wie es sich von den bisherigen Datenarchitekturen abgrenzt. Als Reaktion auf neue Nutzungsmodelle (Cloud-Nutzungsmodelle Saas, PaaS, IaaS), hat sich die Art und Weise verändert, wie analytische Daten verarbeitet und verwaltet werden. Diese reichen von der bisherigen Unterstützung zur Optimierung der Geschäftsprozesse (Business Intelligence, BI) bis hin zu durch Machine Learning unterstützen intelligenten Produkten.

Architekturen der ersten Generation

Mitte der 1980er Jahre, zählten Data Warehouses zu den vorrangigen Datenarchitekturen[3]. Deren Hauptaufgabe bestand darin, Daten aus operativen Systemen und externen Quellen (relationale und objekt-orientierte Datenbanken), in Business Intelligence-Systeme (BI-Systeme) zu verlagern und sie für Analysen bzgl. der Unternehmensentwicklung, zugänglich zu machen.

Dabei liegen die Daten in unterschiedlichen Modellen vor, so dass sie zuvor in ein gemeinsames Schema, auf Basis eines mehrdimensionalen Tabellenformats, transformiert werden müssen. Nachdem die Daten in die DWH-Tabellen geladen wurden, erfolgt der Zugriff darauf über SQL-ähnliche Abfragen. Weiterentwicklungen führten zu „Data Marts“, die sich einem speziellen Geschäftsbereich oder einer einzelnen Abteilung widmen. Weitere Entwicklungen haben zu einem Enterprise Data Warehouse geführt, die oft proprietär angeboten werden und teuer in der Nutzung sind (vgl. Laudon et al., 2006, S. 336-337), (vgl. Dehghani, 2023, S. 166-167). Aufgrund des zentralisierten Ansatzes sowie des *Schema-on-Write*-Konzepts[4], welches bei DWHs im Einsatz ist, eignet sich diese Datenarchitektur nur noch bedingt für Big-Data-Projekte. Diese wählen *Schema-on-Read*[5], da aufgrund der Heterogenität der Daten, eine größere Agilität im Umgang mit den Zielsystemen erforderlich wird (vgl. Frick et al., 2021, S. 89-90).

Architekturen der zweiten Generation

Data Lakes beschreiben eine Datenarchitektur, die um 2010 die Herausforderungen der Data Warehouse-Architektur in Hinblick auf die Erfüllung der neuen Verwendungszwecke

[3] Bzgl. der erstmaligen Erwähnung des durch IBM geprägten Begriffs „Data Warehouse“, wird auf Barry Devlin verwiesen, der in seinem Werk die Architektur des Data Warehouses definiert (Devlin, 1980).
[4] Ein Konzept, bei dem die Daten vor Ablage ins DWH in das Format des Zielsystems transformiert werden.
[5] Schema-on-Read bedeutet, dass die Ablage der Daten ohne festes Schema erfolgt. Erst beim Lesen der Daten, wird in das vom Zielsystem benötigte Schema, transformiert.

unterstützt. Erstmalig geprägt wurde der Begriff des Data Lakes von James Dixon (CTO von Pentaho[6]), der auf seinem Blog das Prinzip des Data Lakes vorstellt (Dixon, 2010).

Data Lakes zielen darauf, den Zugriff auf Daten für moderne Data Science-Prozesse sowie die Nutzung der Daten zum Training von Machine Learning-Modellen, effizient zu unterstützen. Um realitätsnahe Entscheidungen treffen zu können, benötigen Data Scientists möglichst unveränderte Daten. Wie auch bei Data Warehouses, werden Daten aus operativen Systemen extrahiert und in einem zentralen Repository abgelegt. Der Unterschied zu diesem Ansatz, liegt jedoch darin, dass die Daten zu diesem Zeitpunkt, kaum bis gar keine Transformation erfahren haben.

Sobald die Daten im Data Lake abgelegt sind, werden aufwendige Pipeline-Prozesse in Gang gesetzt, um die Daten in spezielleren Speicherorganisationen, wie z. B. Data Lakeshores und Data Marts zur Verfügung zu stellen. Unter dem nachgelagerten Transformationsprozesses leidet die Performanz, insbesondere im Hinblick auf das Training der Modelle.

Data Lakes entwickeln sich aufgrund ihrer Komplexität durch unhandliche Pipelines, ungepflegter und zunehmend schlecht auffindbarer Daten, der Erfordernis eines hochspezialisierten Data Engineering Teams sowie der undurchsichtigen Datenherkunft und Abhängigkeiten, zum Verarbeitungs-Engpass.

Architekturen der dritten Generation

Die folgende Generation der Datenarchitekturen, greift die Konzepte Data Warehouse und Data Lake auf und erweitert diese um Funktionalitäten, die die bisherigen Kritikpunkte minimieren soll. Als Lösung dienen multimodale Cloud-Architekturen, die folgende Eigenschaften mit sich bringen (vgl. Dehghani, 2023, S. 169):

- Unterstützung von Echtzeit-Streaming (durch die Kappa-Architektur)
- Einsatz von Frameworks zur Batch- und Stream-Verarbeitung (z. B. Apache Beam)
- Nutzung von Cloud-basierten Managed Services
- Die Vereinigung von Data Warehouse und Data Lake in einer neuen Technologie (z. B. Data Lakehouse)

Das Data Lakehouse hat sich als guter Ansatz etabliert, der das Datenmanagement effizient unterstützen kann. Die Vorteile beschreibt das Unternehmen Databricks[7] auf seiner Website wie folgt:

[6] Die Pentaho Corporation entwickelt BI-Software und wurde 2015 von Hitachi Data Systems aufgekauft.
[7] Databricks ist ein US-amerikanisches Softwareunternehmen, welches cloudbasierte Lösungen zum Aufbau und Verwaltung von Daten für Unternehmen anbietet.

> *„Ein Data Lakehouse ist eine neuartige, offene Datenverwaltungsarchitektur, die die Flexibilität, Kosteneffizienz und Skalierbarkeit von Data Lakes mit Datenverwaltungsfunktionen und ACID-Transaktionen von Data Warehouses kombiniert und so Business Intelligence (BI) und maschinelles Lernen (ML) auf Grundlage aller Daten ermöglicht.“ (Databricks, 2024)*

Ein kleiner Effizienzgewinn lässt sich durch diese Konzepte gewinnen, jedoch lassen sich auch hier noch Limitierungen vorfinden, die mit einer monolithischen Architektur, einer monolithischen Organisation, einem zentralisiertem Data Ownership sowie einer strengen Technologieorientierung einhergehen und die organisatorische Skalierung verhindern (vgl. Dehghani, 2023, S. 181).

3 Nähere Betrachtung des analytischen Datenverarbeitungsprozesses

Wie bereits aus den Grundlagen von Data Mesh in Kapitel 2 entnehmbar, setzt Data Mesh auf die Reduzierung des Koordinationsaufwands durch Pipelines (vgl. Dehghani, 2023, S. 156). Um diesen Aspekt zu untersuchen und den vorgeschlagenen Lösungsweg von Data Mesh vorzustellen, ist es ratsam, den analytischen Datenverarbeitungsprozess näher zu betrachten. In diesem Zusammenhang wird zwischen operativen und analytischen Daten unterschieden.

Operative Daten bilden die Grundlage des Geschäftsbetriebs indem sie den aktuellen Zustand des Unternehmens wiedergeben. Ihre Erfassung, Verarbeitung und Speicherung erfolgt mittels OLTP-Systemen (Online Transaction Processing) in Echtzeit. Durch definierte Schnittstellen auf Basis von z. B. REST-APIs, GraphQL-Schnittstellen, wird ein Zugriff ermöglicht. Neben der Realisierung der Geschäftslogik, werden aus operativen Daten analytische Daten generiert. Man findet operative Daten in Datenbanken oder Stammdaten-Systemen vor (vgl. Dehghani, 2023, S. 53).

Analytische Daten bilden eine zusammengefasste Sicht auf die Geschehnisse und Transaktionsverläufe des Unternehmens. Sie entstehen bisher als „Nebenprodukt“ und sind Input für OLAP-Systeme (Online Analytical Processing), deren Aufgabe in der Verwaltung der analysierenden Daten liegt (vgl. Dehghani, 2023, S. 54). Der Wert analytischer Daten lieg darin, das rückblickende Erkenntnisse gewonnen werden können. Diese dienen zur Evaluation eines Prozesses. Ebenso sind analytische Daten die Basis für zukünftige Entscheidungen. Gespeichert werden analytische Daten in einem Data Warehouse oder Data Lake.

3.1 Data Engineering Pipeline

Die Konzentration von Data Mesh liegt auf den analytischen Daten und wie sie weg, von ihrem Dasein als Nebenprodukt, ins Zentrum dieser Architektur gestellt werden. Eine wesentliche Rolle spielt dabei die Data Engineering Pipeline. Data Mesh sieht an genau dieser Stelle die Einschränkung in der Datenverarbeitung. Diese drückt sich bspw. durch den hohen Koordinationsaufwand und die Verwaltung der Daten durch ein zentrales Daten-Team aus.

Eine Data Engineering Pipeline beschreibt meist einen automatisierten Daten-Workflow, der aus mehreren Schritten besteht. Zentrale Aufgaben sind die Aufnahme der Daten aus mehreren Quellen (Data Ingestion), die Bearbeitung der Daten (Transformation) und die Ausgabe oder das Ablegen den Daten. Dabei lassen sich im Wesentlichen zwei Verarbeitungsstrategien unterscheiden: ELT und ETL.

ETL ist ein Akronym für „extract, transform, load“ und beschreibt den historisch bekanntesten Weg einer Pipeline. Die Daten werden aus ihren Quellen extrahiert (*extract*), nach festgelegte Routinen verarbeitet (*transform*) und anschließend in ein anderes System geladen (*load*). Dabei handelt es sich meist um ein Data Warehouse. ETL-Pipelines arbeiten nach dem *Schema-on-Write*-Konzept (vgl. Freiknecht & Papp, 2018, S. 19).

Eine Umstellung der Aufgaben nach dem **ELT**-Prozess, sorgt dafür, dass Daten erstmal in ihrer Rohfassung abgelegt werden. Erst bei Zugriff auf diese Daten, erfolgt eine Aufbereitung der Daten für das anfragende System. Diese Variante hat sich für Systeme und Anwendungen etabliert, die durch die Verwendung unbehandelter Daten ein besseres Ergebnis erzielen (z. B. ML-Trainingsmodelle, Big Data). Die Ablage erfolgt meist in einem Data Lake. Hier kommt das *Schema-on-Read*-Konzept zum Einsatz (vgl. Freiknecht & Papp, 2018, S. 19).

Zusammen wird der Datenaufnahme-Prozess mittels ELT oder ETL als „Data Ingestion“ bezeichnet. Die Auswahl einer Variante, hängt von mehreren Faktoren ab.

ETL verarbeitet und transformiert Daten bevor sie abgelegt werden. Das reduziert die Datenmenge, nimmt aber auch den Verlust „wertvoller“ Daten in Kauf.

Dadurch dass Daten bei ELT die Daten außerhalb des Speicherorts transformiert werden, ist eine gezielte Anpassung der Daten an die konsumierende Stelle möglich. Der Zugriff erlaubt Flexibilität, Effizienz und Skalierbarkeit und ermöglicht die Verarbeitung sowohl von strukturierten als auch von unstrukturierten Daten. Nachteilig ist hierbei die Menge der Daten, die zusätzliche Herausforderung bei der Suche sowie die Gefahr, ungenutzter Artefakte aufgrund der fehlenden Kenntnis zur Einordnung der Daten.

3.2 Verarbeitungsmethodik und Architektur

Neben der Verarbeitungsstrategie im Pipeline-Prozess, gibt es zwei Ansätze zur Verarbeitung der Daten. Im Wesentlichen wird zwischen Stream- und Batch-Processing unterschieden.

Verarbeitung der Daten

Bim **Batch-Processing** steht die Sammlung der Daten im Vordergrund, worauf eine gemeinsame Verarbeitung folgt. Batch-Verarbeitung erfolgt in regelmäßigen, geplanten Abständen und eignet sich vor allem für Szenarien, die keine Ergebnisse in Echtzeit erfordern. Vielmehr liegt der Fokus hier vielmehr auf der Verarbeitung großer Informationsmengen (Stapelverarbeitung), wobei diese Art der Verarbeitung in Bezug auf die Infrastrukturkosten in der Regel günstiger ausfällt, als Stream-Verarbeitung (vgl. Freiknecht & Papp, 2018, S. 461).

Das **Stream-Processing** agiert Ereignis-basiert, d.h. sobald eine Zustandsveränderung aufgrund eines bestimmten Ereignisses eintritt, erfolgt eine Verarbeitung in Echtzeit. Es beinhaltet die kontinuierliche Einnahme, Verarbeitung und Analyse von Daten in nahezu Echtzeit. Durch den stetig fließenden Datenstrom, können sofortige Einblicke für Analysen gewonnen werden (vgl. Freiknecht & Papp, 2018, S. 461).

Architektur der Verarbeitung

Zur Umsetzung der Verarbeitungsmethodik, unterscheidet man zwischen den Lambda- und der Kappa-Architektur.

Die **Lambda-Architektur** legt ihren Fokus auf die Batch-Verarbeitung. Sie teilt den eingehenden Strom in zwei Pfade: Speed-Layer du Batch-Layer. Der Speed-Layer eignet sich zur Visualisierung von Echtzeit-Daten. Im Batch-Layer werden eingehende Daten gespeichert und um weitere Informationen, z. B. aus einem Data Lake angereichert (vgl. Marz & Warren, 2016, S. 32-39).

Die **Kappa-Architektur** ist eine schmalere Version der Lambda-Architektur. Durch Verzicht auf den Batch-Layer, gewinnt sie an Schnelligkeit und Effizienz. Allerdings sind Datenhistorien eine Herausforderung (durch den fehlenden Batch-Layer), was sich insbesondere bei Schema-Änderungen bemerkbar macht. Dennoch überwiegen die Vorteile. So ist bspw. die Konzentration auf nur einen Layer anstelle von zwei parallel betriebenen Systemen, ein Gewinn. Die Kappa-Architektur eignet sich vor allem für das Stream-Processing (vgl. Dulay & Mooney, 2023, S. 22-24)

3.3 OLAP im Data Mesh

Nachdem die Grundlagen für den analytischen Datenverarbeitungsprozess (Online Analytical Processing, OLAP) vorgestellt wurden, soll nachfolgend erläutert werden, welche Rolle sie im Data Mesh spielen. Data Mesh arbeitet mit analytischen Daten, deren Existenz als „Nebenprodukt“ grundlegend überdacht wurde. Aus diesem Grund werden Verarbeitungsstrategie, -methodik und die Architektur im Data Mesh vorgestellt.

Batch- vs. Stream-Processing

Die Bereitstellung der Daten im Data Mesh kann sowohl in der Batch-Verarbeitung als auch als Stream umgesetzt werden. Es überwiegen jedoch Vorteile für eine stream-basierte

Verarbeitung. Zudem gibt es auch eine Empfehlung für einen event[8]-getriebenen Ansatz, der stream-basiert erfolgt (vgl. Dulay & Mooney, 2023, S. 12-13).

Folgende Vorteile des stream-orientierten Ansatzes gegenüber der Batch-Verarbeitung werden aufgeführt (vgl. Dulay & Mooney, 2023, S. 16-19):

- Streaming unterstützt die Verarbeitung in Echtzeit
- Streaming bietet Vorteile in der Datenoptimierung
- Nutzung von Reverse ETL[9] (rETL)

Im Falle der Realisierung von Data Mesh mittels stream-orientierter Verarbeitung, werden die Daten von der Quelle eingesammelt und verbleiben im Stream, bis ein definiertes Aufbewahrungsende erreicht ist (engl. *Retention time*). Es gibt keinen Ort, an dem die Date abgelegt werden, wie bspw. einem Data Lake oder DWH (vgl. Dulay & Mooney, 2023, S. 15). Während des Verbleibs im Stream, haben die Daten Zugriff auf alle Werkzeuge, die Data Mesh zur Realisierung nutzt (Self-Serve Data Platform, Workflow-Tools, Policies, …).

ETL vs. ELT

Pipelines im Data Mesh, die stream-orientiert arbeiten, verfahren nach der ETL-Strategie. Die Komponente, die die Daten aus einer Quelle abruft, überträgt diese an einen Stream. Um die Daten streamen zu können, müssen diese serialisiert werden. Dieser Zustand wird mittels Transformation erzeugt.

Datenkonsumenten, die Zugriff auf die Daten zwecks weiterer Nutzung und Verarbeitung, wünschen, laden die Daten aus dem Stream in ihre Domäne. Dabei folgen sie dem dafür aufgestellten, föderalen Regelwerk (vgl. Dulay & Mooney, 2023, S. 15).

Kappa vs. Lambda

Übertragen auf die Ebene der Architektur, lässt sich in Bezug auf Data Mesh sagen, dass sich die Kappa-Architektur gegenüber der der Lambda-Architektur durchgesetzt hat (vgl. Dulay & Mooney, 2023, S. 19-25).

Die Lambda-Architektur wurde von Nathan Marz und James Warren (Manning) im Rahmen ihrer Ausführungen zu Big Data vorgestellt und konzentriert sich auf die Umsetzung einer Echtzeit-Verarbeitung innerhalb von Big Data Anwendungen (Marz & Warren, 2016).

[8] Als *Event* wird ein Ereignis bezeichnet, welches durch eine Anwendung erzeugt werden kann. Im Kontext dieser Arbeit, kennzeichnet eine Event eine Nachricht, welche logisch als Record realisiert werden kann und deren wesentliche Bestandteile u.a. ein Schlüssel-/Wert-Paar (engl. *Key-value*) sind (vgl. Dulay & Mooney, 2023, S. 49-51).
[9] Durch die Nutzung von rETL wird die Umkehrung der Datenabstraktion von den analytischen Daten hinzu den operationalen Daten bezweckt, mit dem Ziel, die Kerndefinitionen aus den analytischen Daten in die operationalen Daten zurückfließen zu lassen, wenn diese dort benötigt werden (vgl. Dulay & Mooney, 2023, S. 18).

Aufgrund ihrer Auslegung mittels zweier Layer für die Echtzeit- (*Speed-Layer*) und die Stapelverarbeitung (*Batch-Layer*), erfordert dieser Entwurf zwei zusätzliche Speicherorte: einem Datenspeicher für analytische Daten (z. B. DWH oder Data Lake) sowie einen In-Memory-Store[10] für Streaming-Daten. Ziel und Zweck dieser Aufteilung ist die Umsetzung des **CAP-Theorems**[11] für Big Data Anwendungen.

Der Batch-Layer stellt *Partition Tolerance* sicher, indem horizontale Skalierbarkeit unterstützt wird. Dieser Umstand bildet auch die Grundlage für die *Availability*, die ebenfalls auf den Batch-Layer zurückzuführen ist. Die Konsistenz der Daten ist wiederum Aufgabe des Speed-Layers, der für die Umsetzung des Streamings zuständig ist. Zudem hat er zur Aufgabe, die Transaktionssicherheit und die Verwaltung der verteilten Daten und deren Zuständen sicherzustellen (vgl. Dulay & Mooney, 2023, S. 21). Auffallend bei dieser Architektur ist, dass eine Vielzahl von Daten kopiert werden um die *Availability* umzusetzen.

Adam Bellemare beschreibt in seinem Werk die **Nachteile der Lambda-Architektur** bezogen auf Data Mesh (vgl. Bellemare, 2023, S. 64-65). Diese sollen nachfolgend erläutert werden.

- Nachrichtenproduzenten (Quellsysteme) müssen zwei Code-Pfade verwalten, an die sie ihre Daten versenden: Einen Pfad zum Batch-Data-Store und einen Pfad zum Event-Stream. Bei atomaren commits kann es zur Herausforderung werden, wenn einer der beiden Zielsysteme temporär nicht verfügbar ist. In diesem Fall müssen Daten dupliziert werden, was zu vermeiden wäre.
- In selber Weise haben die Datenkonsumenten zu verfahren, wenn sie Daten von zwei unterschiedlichen Quellen beziehen. Dabei wirft die Konsistenz der Daten Fragen auf, wenn auf die Daten nach längerer Zeit wieder zugegriffen wird. Es ist nicht auszuschließen, dass sich die Datenzustände, aufgrund zwischenzeitlicher Zugriffe, geändert haben.
- Eine weitere Herausforderung stellt sich dadurch, dass die Konsumenten von Batch-Daten und die Konsumenten von Streaming-Daten nicht unbedingt dasselbe Ergebnis erhalten. Die Sicherstellung eines konstanten Status von mehreren multiplen Zielsystemen kann sehr komplex ausfallen.
- Es ist unerlässlich zu prüfen, ob das Datenschema des Batch-Data-Stores und das Datenmodell des Event-Streams synchron zueinander sind.

[10] Dabei handelt es sich um ein Datenbankmanagementsystem, das den Arbeitsspeicher eines Computers als Datenspeicher nutzt und auf die Ablage der Daten auf einem Festplattenspeicher o.ä. verzichtet.
[11] Das CAP-Theorem basiert auf einer Vermutung des amerikanischen Universitäts-Professors, Dr. Eric Brewer und besagt, dass es in einem verteilten System unmöglich ist, gleichzeitig die drei Eigenschaften *Consistency* (Konsistenz), *Availability* (Verfügbarkeit) und *Partition Tolerance* (Ausfalltoleranz) zu garantieren. Es entstand im Jahr 2000 beim *Symposium on Principles of Distributed Computing (PODC)* der University of California, Berkeley (Brewer, 2000).

- Die Aggregation von Datenprodukten in einer Lambda-Architektur, wird als Herausforderung angesehen. Hintergrund dafür ist, dass Datenprodukte ihren eigenen Regelsatz mitbringen und sich vorrangig autonom verwalten. Diese Kontrolle ist nur it sehr hohem Aufwand aufrecht zu erhalten.

Aufgrund der Ausführungen wird deutlich, dass die Lambda-Architektur hinsichtlich der Big Data Anwendungen, eine Vielzahl an Vorteilen bietet. Für Data Mesh stellt sie keine optimale Unterstützung sondern erzeugt stattdessen einen erhöhten Arbeitsaufwand sowie ungewünschte Komplexität.

An dieser Stelle zeichnen sich die **Vorteile der Kappa-Architektur** ab. Kappa ist einer Vereinfachung der Lambda-Architektur mit nur einem Layer, der für das Streamen von Daten vorgesehen ist. Der Fakt, dass nur eine Streaming-Pipeline eingesetzt wird, führt zu einer geringeren Komplexität als es bei Lambda der Fall ist. In dieser Architektur werden Daten in der Pipeline aufbewahrt und an die Konsumenten rücküberführt (vgl. Dulay & Mooney, 2023, S. 23). Durch die fast unbegrenzte Aufbewahrung, können sehr frühe Zustände rekonstruiert und nachvollzogen werden. Eine Zwischenspeicherung und das Kopieren von Daten entfällt.

4 Anforderungen an Pipelines durch Data Mesh

Nachdem im vorherigen Kapitel der technische Aspekt des analytischen Datenverarbeitungsprozess (OLAP) erläutert und erste architektonische Tendenzen für ein Data Mesh identifiziert wurden, wird in diesem Kapitel ein tieferer Einblick auf die Pipeline selbst vermittelt.

Dabei soll die Frage beantwortet werden, welchen Anforderungen das Data Mesh-Prinzip an die Pipeline stellt. Durch die geforderte Reduzierung der zentralen Pipeline-Aktivitäten, entstehen eine Vielzahl von Herausforderungen, die nachfolgend betrachtet werden.

4.1 Ausgangslage und Herausforderungen

Bisher erfolgte die analytische Datenverarbeitung meist in Form zentraler, domänenbasierter Pipelines, bei der Daten von einer Quelle entgegengenommen, geeignet transformiert und schließlich zu einer Quelle transportiert werden. Die nebenbei entstandenen Analysewerte werden an verschiedenen Stellen erzeugt, müssen sich jedoch am Pipeline-Start anstellen um den gewünschten Transformationsprozess bis hin zum Ablageort zu durchlaufen.

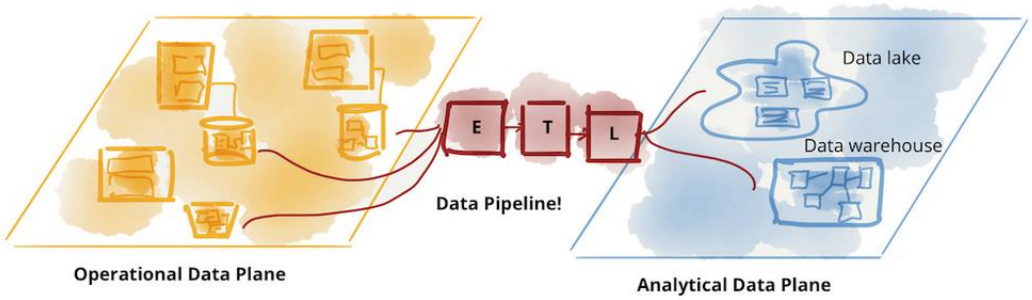

Abbildung 1: Beziehung zwischen operationalen und analytischen Daten (Quelle: Dehghani, 2020).

Eigenschaften zentraler Pipelines

Data Mesh charakterisiert diese zentrale Verarbeitung als klassische Bottleneck-Strategie. Es ist zu erwarten, dass ein Stau an wartenden Daten am Pipeline-Eingang sowie eine entsprechend verzögerte Verarbeitung und Auslieferung der Daten, zwangsläufig zu Performance-Einbußen führt.

Zudem ist eine solche zentrale Pipeline nur schwer *skalierbar*, d.h. möchte man weitere Werte in die Pipeline einfügen, müssen zum einen entsprechende Ports eingerichtet und zum anderen, weitere Prozesse in der gemeinsamen Pipeline-Verarbeitung implementiert werden.

Änderungen in der Unternehmens-Strategie können zu *Anpassungen* der Pipeline führen. Neueingebundene Datenquellen oder die Anfrage anders aufbereiteter Daten, erfordern eine schnelle Reaktion, so dass der Wunsch nach mehr Agilität steigt. Die bisherigen, zentralen

Ansätze resultieren meist darin, dass die Implementierung weiterer Prozesse zur Gewinnung einzelner Analysewerte, eine Änderung der zentralen Pipeline nach sich zieht.

Die Daten werden in verschiedenen Formaten generiert und müssen zur weiteren Verarbeitung und Auswertung entsprechend aufbereitet werden. Entweder passiert dieser Transformationsprozess vor der Speicherung im Zielsystem, wie bspw. einem Data Warehouse oder er erfolgt beim Laden von Rohdaten aus einem Data Lake. Dabei ist nicht ersichtlich, ob alle Nutzertypen bzw. *Zielgruppen* (Data Scientist, App Developer, Data Analyst etc.) angesprochen werden. Ein zentrales, stark ausgelastetes Pipeline Team hat zur Aufgabe, die Ergebnisse bereitzustellen. Von diesem Team wird erwartet, über ein umfangreiches Daten-Wissen zu verfügen. Neben der eigentlichen technischen Arbeit, führt diese Tatsache zu einer enormen kognitiven Auslastung, die oft auf wenigen Schultern verteilt wird.

Analysewerte können schwer kombiniert werden, was die Entstehung von sogenannten Datensilos vorantreibt. Durch die Kombination verschiedener Daten, lassen sich neue Erkenntnisse gewinnen, die einen Mehrwert für verschiedene Domänen liefern könnten. Eine Zunahme an *Datenqualität* wäre zu erwarten.

Zudem steuern die Analysewerte ihren *Lebenszyklus* über die Pipeline. Das bedeutet, dass durch die Pipeline festgelegt ist, woher die Daten entnommen und nach Ihrer Bearbeitung abgelegt werden. Ob aus den Daten noch ein größerer Mehrwert gewonnen werden kann oder diese bereits während des Transformationsprozesses für eine bestimmte Konsumentengruppe einen wesentlichen Nutzen darstellen, ist nicht ersichtlich. Somit entsteht eine gewisse Abhängigkeit von der Pipeline und demnach auch von einem Team, welches eher die technischen Pipeline-Prozesse verantwortet, als die Qualität der Daten.

Die folgende Tabelle fasst die Ausgangslage der zentralen Pipeline-Verarbeitung zusammen.

Stichwort	**Ausgangslage zentraler Pipeline-Verarbeitung**
Performanz	• Warteschlangen an Pipeline-Eingängen und -Ausgängen. • Gesamtsystem verliert an Performanz.
Skalierbarkeit	• Schlechte Skalierung bei Zunahme neuer Komponenten.
Anpassbarkeit	• Modifikationen an Pipeline können nicht isoliert und unabhängig durchgeführt werden.
Zielgruppenansprache	• Keine Erkenntnis darüber, ob alle Benutzertypen angesprochen werden. • Unklar, ob Ergebnisse den Anforderungen der Benutzertypen entsprechen.

Datenqualität	• Schlechte Kombination von Analysewerten. • Mehrere Verarbeitungsschritte erforderlich, bspw. durch die Angleichung der Datenformate und -schemata.
Lebenszyklus der Daten	• Steuerung des Lebenszyklus der Daten über die Pipeline. • Kein ausgewiesenes Pipeline-Team, daher oftmals keine Verantwortlichen für die Daten. • Sinkende Qualität der Daten aufgrund fehlender fachliche Kenntnisse im Pipeline-Team.

Tabelle 1: Ausgangslage zentraler Pipeline-Verarbeitung.

Eigenschaften dezentraler Pipelines (nach dem Data Mesh-Prinzip)

Data Mesh zielt darauf ab, die Nachteile zentralisierter, externer Pipelines zwischen der operativen und der analytischen Datenwelt zu entkräften. Während bisher Daten-Pipelines außerhalb von Datenprodukten verwaltet werden, führt Data Mesh einen „internen" Transformationscode ein. Dieser wird innerhalb eines Datenproduktes implementiert und gekapselt (vgl. Dehghani, 2023, S. 194-195). Dieser Code orientiert sich an dem fachlichen Kontext des Datenproduktes und wird von diesem vollständig gekapselt. Letzen Endes wird er als Pipeline implementiert und ist dem Lebenszyklus des Datenproduktes untergeordnet.

Den bisher genannten Nachteilen zentralisierter Pipelines, stellt sich das Data Mesh-Prinzip mit folgenden Maßnahmen entgegen:

Stichwort	**Vorteile durch dezentrale Pipeline-Verarbeitung**
Performanz	Durch die parallele Verarbeitung lässt sich die Effizienz der Pipeline-Arbeit steigern. Warten an Pipeline-Eingängen und -Ausgängen kann vermieden werden, da die Arbeit nun auf mehrere, domänenabhängige Teams verteilt werden kann.
Skalierbarkeit	Pipeline-Komponenten können unabhängig voneinander skaliert werden. Anpassungen an steigende Datenmengen oder Verarbeitungsanforderungen können unabhängig von anderen Pipeline-Nutzern durchgeführt werden.
Anpassbarkeit	Pipelines sind flexibler, da Änderungen und Erweiterungen an einzelnen Komponenten der Pipeline einfacher umgesetzt werden. Das Gesamtsystem wird nicht beeinflusst, denn es wird auf mehrere Teams bzw. Systeme verteilt.

Zielgruppenansprache	Dezentrale Pipelines ermöglichen es, dass die Datenautonomie in den einzelnen Teams oder Domänen verbleibt. Diese können ihre Datenverarbeitung unabhängig voneinander gestalten und an ihre spezifischen Anforderungen anpassen. Eine individuellere Zielgruppenansprache lässt sich dadurch umsetzen.
Datenqualität	Durch die Flexiblere Pipelineverarbeitung, können Analysewerte besser kombiniert und weiterverarbeitet werden. Demzufolge lässt sich eine bessere Datenqualität erzielen.
Lebenszyklus der Daten	Der Lebenszyklus der Daten hängt nicht allein von der zentralen Pipeline ab. Im Sinne von Data Mesh wird er in den Verantwortungsbereich der Datenprodukte übertragen.

Tabelle 2: Vorteile durch dezentrale Pipeline-Verarbeitung.

Herausforderungen

Auch dezentrale Pipelines stehen einer Reihe Herausforderungen gegenüber. Data Mesh hat diese Herausforderungen identifiziert und stellt mit seinen vier Prinzipien einen Architektur-Ansatz zur Verfügung, dessen Umsetzung und Implementierung noch viel Spielraum bietet.

- Es ist schwierig, die *Kontrolle und Koordination* über die verteilten Pipelines aufrechtzuerhalten. (Diese soll im Data Mesh in den Verantwortungsbereich der Datenprodukte fallen. Dedizierte Rollen kennen die Belange und Herausforderungen, so dass eine angemessene Reaktion ermöglicht wird.)
- Die *Datenintegration und -konsistenz* über die verschiedenen Pipelines müssen sorgfältig geplant werden. (Diese Aufgabe wird an das Datenprodukt und das jeweilige Domain Ownership übertragen.)
- Da die Abstimmung der Komponenten komplexer ist, stellen die Skalierbarkeit und die Leistung eine Herausforderung dar. (Skalierung und Performance fallen in den Aufgabenbereich der Self-Serve Data Platform.)
- In einer dezentralen Architektur ist es schwieriger, die *Sicherheit und Compliance* über die verteilten Pipelines hinweg zu gewährleisten. (Data Mesh verlagert diesen Themenschwerpunkt in das Prinzip Federated Computational Governance.)
- *Betrieb und die Wartung* der einzelnen Pipelines können aufwendiger sein, da mehr Komponenten und Schnittstellen zu verwalten sind. (Data Mesh ordnet diese Aufgaben der Self-Serve Data Platform zu.)

4.2 Einordnung und Einfluss der Pipeline

Nachdem die Pipeline sowohl logisch (Kap. 4.1) als auch technische (Kap. 3) betrachtet wurde, stellt sich die Frage, an welcher Stelle die Data Engineering Pipeline im Data Mesh Konzept einzuordnen ist sowie in welcher Ausprägung sie sich realisieren lässt. Der Fokus von Data Mesh liegt vorranging auf den organisatorischen Aspekten, während der technischen Umsetzung kaum Vorgaben gemacht werden. Betrachtet man dazu die vier Grundprinzipien, so lassen sich folgende Annahmen treffen.

Dehghani platziert die Pipeline im Datenprodukt (vgl. Dehghani, 222, S. 194). An dieser Stelle soll dennoch geprüft werden, welche Rolle sie in den vier Grundprinzipien von Data Mesh spielt bzw. welche Einflüsse sie auf deren Ausgestaltung ausübt.

Domain Ownership

Das Prinzip des *Domain Ownership* spricht vor allem den Lebenszyklus der Daten an. Hier werden konkrete Rollen innerhalb der fachlichen Domänen definiert, die sich hauptsächlich mit den gewonnenen, analytischen Daten auseinandersetzen. Ziel dabei ist die Datenautonomie (sowohl operativer, als auch analytischer Daten) in den Händen der Domänen zu lassen und durch die Bereitstellung dedizierter Schnittstellen, einen Zugriff auf diese zu gewähren.

Die Pipeline nimmt in diesem Prinzip dadurch Einfluss, dass für ihren Aufbau und die Pflege eine eigene Rolle vorgegeben ist: Der *Data Product Developer* arbeitet mit den Anwendungsentwicklern zusammen um Daten aus dem operativen Kontext auf den analytischen Kontext zu übertragen. In seinen Aufgabenbereich fällt auch die Pflege der Transformationslogik (vgl. Dehghani, 2023, S. 368).

Data as a Product

In diesem Prinzip werden die Daten mit dem Product Thinking verknüpft. Beim Product Thinking liegt der Fokus nicht allein auf der Schaffung, Transformation und Bereitstellung von Daten sowie der Einhaltung verschiedener Zeitpläne und Roadmaps. Vielmehr konzentriert man sich auch auf die Belange der Konsumenten. Ziel ist es den Kundenbedürfnissen angepasste Ergebnisse über die gesamte Lebensdauer zu liefern.

An dieser Stelle spielen die Themen Performanz und Anpassbarkeit der Daten sowie die Datenqualität und der Lebenszyklus der Daten eine wichtige Rolle. Durch die Auseinandersetzung mit diesen technischen Belangen der Datenprodukte (Daten, Metadaten, Code und Policy), ist hier entsprechend die Pipeline-Arbeit zu erwarten.

Self-Serve Data Platform

Die Self-Serve Data Platform dient der Koordination der verschiedenen Datendomänen, indem sie u. a. geeignete Mittel zur Suche nach vorhandenen Datenquellen in den verschiedenen Domänen ermöglicht. Zudem verwaltet sie die gesetzten Regeln und Unternehmensrichtlinien. Als wesentlicher Unterschied zu der Vielzahl an bestehenden Daten- und Analyseplattformen gilt, dass Bereitstellung, Zugriff und Nutzung der analytischen Daten auf dezentralem Wege stattfinden (vgl. Dehghani, 2023, S. 91-92). Dabei richtet sich die Plattform an die autonomen Nutzergruppen, wie z. B. Domänenteams, Entwicklern etc.

An dieser Stelle findet eine Auseinandersetzung mit den Themen Skalierbarkeit (der Datenprodukte), Zielgruppenansprache (durch Bereitstellung verschiedener Zugriffsmöglichkeiten) und dem Lebenszyklus der Daten (durch Hinzunahme und Entfernen von Datenprodukten) statt. Wenn auch die technische Transformation an dieser Stelle keine direkte Erwähnung findet, dient die Self-Serve Data Platform dennoch als „Werkzeugkasten" für die Pipeline (vgl. Dehghani, 2023, S. 69).

Die Self-Serve Data Platform spielt eine wesentliche Rolle für Design, Aufbau, Betrieb und Verwaltung von Datenprodukten (vgl. Goedegebuure et al., 2023, S. 13). Sie wird von der Pipeline beeinflusst, indem sie die benötigten Services und Tools zur Unterstützung der Transformationsschritte, bereitstellt.

Federated Computational Governance

Die Self-Serve Data Platform sorgt dafür, dass autonome Datenteams im Hinblick auf die Bereitstellung und Nutzung der Datenprodukte miteinander interagieren können. Dabei trägt jedes Domänenteam die Verantwortung zur Modellierung der Daten. Es benötigt Vertrauen darauf, dass seine Daten eine entsprechende Qualität verfügen sowie sicher, konform und nutzbar sind (vgl. Dehghani, 2023, S. 111).

Mit dem Prinzip Federated Computational Governance werden Regeln und Richtlinien für dieses Zusammenspiel festgelegt, die als Grundlage für einen kontinuierlicher Veränderungsprozess dienen sollen. Die Sicherstellung, dass die definierten Regeln und Policies eingehalten werden, erfolgen weitestgehend automatisiert.

An dieser Stelle werden Regeln zur Zielgruppenansprache definiert und der Lebenszyklus der Daten über entsprechende Vorgaben und Verträge gesteuert. Der Datentransformationsprozess orientiert ich an diesen Regeln. Durch den globalen Charakter dieses Prinzips, ist davon auszugehen, dass diese Regeln auf einem höheren, übergreifendem Abstraktionslevel einzuordnen sind. Es ist auszuschließen, dass die Pipeline diesen Regelsatz beeinflusst.

Zusammenfassung

Wie von Zhamak Dehghani empfohlen, zeichnet sich das Datenprodukt als geeigneter Kandidat für die Verortung der Pipeline ab. Das liegt zum einen daran, dass an dieser Stelle mit den Daten - von der Erzeugung bis hin zur Bereitstellung - gearbeitet wird. Zum anderen greifen die weiteren Prinzipien hauptsächlich den organisatorischen bzw. sozio-technischen Aspekt auf, wobei der Fokus auf dem zwischenmenschlichen Aktionen liegt.

4.3 Nähere Betrachtung des Datenproduktes

Um die Annahme, dass die Pipeline-Arbeit im Datenprodukt durchgeführt wird, zu festigen, wird das Datenprodukt einer näheren Betrachtung unterzogen.

Bisher wurden analytische Daten als Nebenprodukt der operativen Daten und Prozesse eingestuft. Das hat zur Folge, dass diese Daten ohne genauere Kenntnis darüber, was damit gemacht werden kann und welche Zielgruppe oder Zielsysteme davon Nutzen ziehen können, eingesammelt werden. Es folgt ein Prozess der Analyse der gewonnenen Daten und eine umfangreiche Bereinigung und Transformation hin zu einer potentiellen Zielgruppe. Es lässt sich erahnen, dass in diesem Fall eine hohe fachliche Kenntnis seitens des Datenteams vorhanden sein muss, um Entscheidung hinsichtlich der Qualität und der entsprechenden Bereinigungsschritte zu treffen. Zudem kann nicht ausgeschlossen werden, dass der Wert der Daten völlig ausgeschöpft wurde oder sich damit, bspw. in der Aggregation mit anderen Daten, noch weitere Erkenntnisse ableiten würden.

Durch die Einführung des Mesh-Prinzips Data Ownership, wird die Daten-Verantwortlichkeit in die Hände der Fach-Domänen übertragen und somit das zentrale Wissen über die Daten an den Stellen belassen, die sie erzeugen und verwalten. Um jedoch vielen Zielgruppen und Systemen die Arbeit mit den Daten zu ermöglichen, bilden sich neue Herausforderungen hinsichtlich des Zugriffs auf die Daten, sowie der Benutzerfreundlichkeit und der Standardisierung von Datenformaten (vgl. Dehghani, 2023, S. 71).

Einfluss von Product Thinking

An dieser Stelle greift das Prinzip Data as a Product, welches ein Umkehrung vom Prozess-Gedanken hin zum Produkt-Gedanken vollziehen möchte. Um dieses sicherzustellen, müssen Daten klaren Regeln folgen sowie bestimmte Merkmale, die bspw. Benutzerfreundlichkeit, Nützlichkeit und Praktikabilität, aufweisen (vgl. Dehghani, 2023, S. 72). Die kognitive Last des Datenteams wird reduziert und die Datenqualität steigt, da sie nun dort vorgehalten wird, wo die Daten entstehen.

Auf die Benutzerfreundlichkeit, die sich insbesondere auf Personen bezieht, wird in dieser Arbeit nicht eingegangen. Stattdessen sollen die Merkmale definiert werden, die insbesondere für technische Systeme beim Zugriff auf die Daten, relevant sind.

Die folgende Grafik illustriert die Platzierung des Datenproduktes innerhalb einer Domäne und legt damit eine Basis für eine „systemfreundliche“ und ausbaubare Interaktion mit anderen Komponenten: Operationale Funktionen und Daten (O) treffen in der Domäne ein und werden in entsprechenden operationalen Systemen abgelegt. Über eine Schnittstelle (*Input data port*, IDP), gelangen diese Daten kontinuierlich oder prozessabhängig in das Datenprodukt. Hier erfolgt nun die Analyse, Bereinigung und Aufbereitung der Daten. Über einen weitere Schnittstelle (*Output data port*, ODP) können die Daten innerhalb des Datenproduktes verschiedenen Nutzergruppen zugänglich gemacht werden.

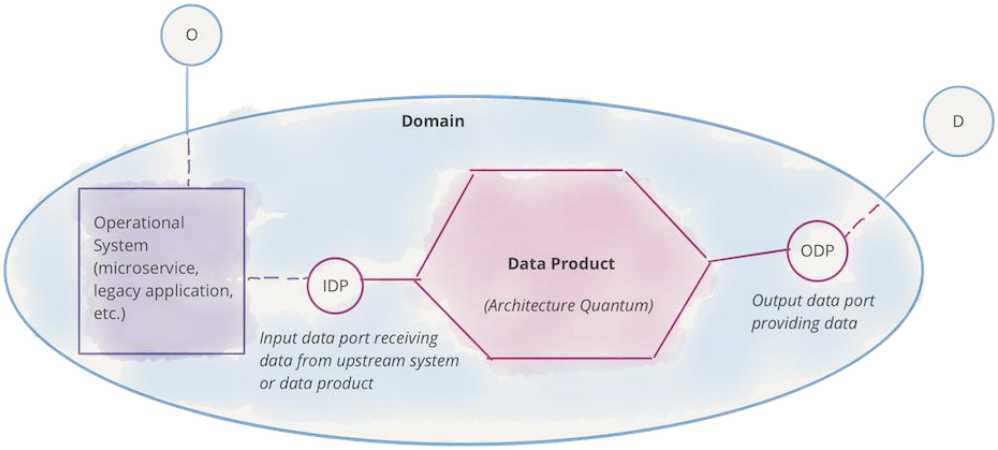

Abbildung 2: Die logische Architektur des Datenproduktes (Quelle: Dehghani, 2020).

Im eigentliche Sinn, impliziert der Datenprodukt-Gedanke die Trennung von Daten und Code. Die Daten werden losgelöst vom Code, der sie erstellt, bearbeitet, verwaltet und ausliefert, betrachtet. Dabei steigt der Gefahr von sogenannten Datensümpfen (engl. *Data Swamps)*, die oft im Data Lake Umfeld aufzufinden sind. Sie beschreiben Datensätze, die verwahrlost und nicht zugänglich sind oder für ihre Nutzer keinen Wert liefern (vgl. Dehghani, 2023, S. 87).

Data Mesh wiederum, betrachtet Daten und Code als architektonische Einheit, bzw. als „eine deploybare Einheit, die strukturell vollständig ist, um ihre Aufgabe zu erfüllen, nämlich die Bereitstellung der qualitativ hochwertigen Daten einer Domäne. Das eine kann ohne das andere nicht existieren.“ (Dehghani, 2023, S. 87). Dazu wird von Zhamak Dehghani der Begriff „Architekturquantum“ eingeführt.

Architekturquantum

Im Falle von Data Mesh ist ein Datenprodukt ein „Architekturquantum“. Das Architekturquantum wird definiert als die Mindestmenge an Komponenten einer Einheit, die an einer Interaktion beteiligt sind. Sie besitzen eine hohe Spezialisierung und verzichten auf nicht notwendige Funktionalität (hohe Kohäsion) (vgl. Ford, et al., 2017, Kap. 4). Bezogen auf Data Mesh beschreibt diese Aussage, dass ein Datenprodukt alle relevanten Komponenten kapselt, die zur Durchführung seiner Aufgabe benötig werden.

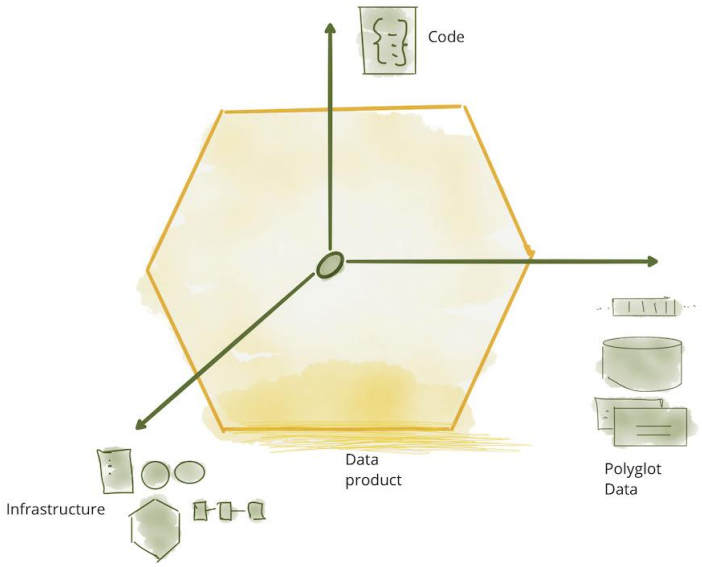

Abbildung 3: Die logische Architektur des Architekturquantums (Quelle: Dehghani, 2020).

Das Architekturquantum stellt die wesentlichen Mittel bereit, um als Datenprodukt funktionstüchtig zu sein. Dabei handelt es sich um die „Dimension, über die ein System skaliert werden kann.“ (Dehghani, 2023, S. 192), denn „Data Mesh skaliert durch das Hinzufügen und Verknüpfen weiterer Datenprodukte“ (Dehghani, 2023, S. 192).

Um zu prüfen, welche Relevanz das Architekturquantum für die Pipeline-Arbeit hat, wird ein Blick auf die Komponenten des Architekturquantums geworfen. Wie bereits erwähnt, zeichnet sich das Datenprodukt durch die Einheit von Daten und Code aus. Diese Aussage realisiert das Architekturquantum durch die folgende 3er-Teilung, die in (vgl. Dehghani, 2023, S. 193) als „Strukturkomponenten“ bezeichnet werden:

- Code
- Daten und Metadaten
- Plattformabhängigkeiten

Code

Data Mesh zielt darauf ab, möglichst viele Arbeiten codebasiert umzusetzen. Dazu zählen die Datentransformation, die Schnittstellengestaltung sowie der Code zur Konfiguration und Ausführung von Struktur- und Verhaltensrichtlinien (engl. *Policies*).

Im Data Mesh wird die **Datentransformation als Code** betrachtet. Im Code ist die Geschäftslogik anzutreffen, deren Aufgabe es ist, Daten zu erstellen, zu verarbeiten und bereitzustellen. Deutlich wird dabei, dass es sich bei einem Datenprodukt um eine aktive Komponente handelt, die einen Lebenszyklus durchläuft. Die bisherige Sicht auf Daten verfolgt einen statischen Ansatz (z. B. Tabellen, Bilder) (vgl. Dehghani, 2023, S. 194).

Damit wird auch schon eine Verbindung zur Pipeline deutlich, denn die Erwähnung eines Lebenszyklus von Daten, beinhaltet auch einen Transformationsprozess.

> *„Datenprodukte transformieren entweder Daten, die sie von einer vorherigen Quelle erhalten, z. B. von ihrem benachbarten operativen System, oder sie erzeugen die Daten selbst. In jedem Fall ist eine analytische Transformation erforderlich, um Daten zu erzeugen und bereitzustellen.“ (Dehghani, 2023, S. 194)*

Dehghani schildert explizit, dass sich in traditionellen Architekturen der Code als „Daten-Pipeline“ außerhalb des Datensatzes befindet (vgl. Dehghani, 2023, S. 194). Der generierte Output wird in externen Systemen, wie bspw. in einem Data Warehouse, abgelegt.

Data Mesh kehrt diesen Ansatz um und verzichtet auf das Konzept einer externen Pipeline. Stattdessen wird ein „interner Transformationscode“ eingeführt. Dehghani lässt offen, ob dieser Code als Pipeline implementiert wird oder nicht (vgl. Dehghani, 2023, S. 194).

In dieser Arbeit wird diese Tatsache aufgegriffen. Sie bildet die Grundlage für die Erarbeitung eines Lösungsansatzes. Dabei wird besonders herausgestellt, dass dieser Code im Kontext des Datenproduktes implementiert und vollständig gekapselt wird. Nur so lässt sich der Lebenszyklus der Daten explizit verwalten.

Der Transformations-/Pipelinecode ist domänenspezifisch[12], während das Datenprodukt domänenagnostisch[13] betrachtet wird.

[12] Domänenspezifisch bedeutet, dass der Code innerhalb einer Domäne an diese Domäne gebunden ist. Er orientiert sich an den fachlichen Inhalten der Domäne. Somit sind in der Domäne enthaltene Funktionen ein Service der Domäne, der nur innerhalb der Domäne genutzt werden kann bzw. einem bestimmten Zweck dient.
[13] Domänenagnostisch ist die Umkehrung von domänenspezifisch und besagt, dass der Code völlig unabhängig vom Domänenkontext angelegt ist. Dabei sollen Basisdienste mit einer Vielzahl von Daten arbeiten können.

Bestandteile des Transformationscodes sind sowohl die *Geschäftslogik*, als auch die *Aggregation* und die *Datenmodelle*. Daneben werden auch noch die automatisierten Tests, mit denen der Code verifiziert wird, als Bestandteil angesehen.

Die Bezeichnung **Schnittstellen als Code** richtet sich an die Ausgestaltung und Umsetzung der Schnittstellen im Datenprodukt. Zum einen bilden sie die Eingangsportale für die Daten in das Datenprodukt, und zum anderen einen Ansprechpunkt, um die bereitgestellten Daten zu konsumieren. Systeme oder Nutzergruppen, die Daten „produzieren“ sowie die Zielsysteme, die die Daten „konsumieren“ müssen darauf vertrauen können, dass sie die Daten entsprechend anliefern bzw. abrufen können. Dieser Umstand entspricht auch dem Service-Gedanken des Datenproduktes.

Letzten Endes bestimmen auch die Schnittstellen über den Lebenszyklus der Daten. Somit kommt diesen eine besondere Bedeutung zu, die im Gegensatz zu den bisherigen Ansätzen steht. Dazu werden sie über explizite Verträge definiert und weisen eine hohe Standardisierung auf. Schnittstellen haben einen domänenagnostischen Charakter, d. h. sie werden nicht in einer zentralen Vorrichtung, wie bspw. einem Data Catalog, vorgehalten. Vielmehr sind sie dem Datenprodukt zugehörig anzusehen. Die API-Lebenszyklen sind unabhängig von den operativen Daten und Pipelines.

Data Mesh grenzt sich vom bisherigen Ansatz dadurch ab, dass bewusst gestaltete Schnittstellen eingesetzt werden, die jedes Datenprodukt individuell verwaltet. Dabei werden die Schnittstellen versioniert und bleiben mit dem übrigen Code des Datenproduktes synchron.

Policy als Code ist eine weitere Kategorie, die im Sinne der Datentransformation zum Einsatz kommt. Policies sind Regelwerke, die darüber bestimmen, wer auf welche Daten zugreifen kann. Hierunter fällt Code, der automatisierte Freigabe- und Zugriffsprozesse ermöglicht. Dabei erfolgt die Überprüfung auf Einhaltung vorhandener Richtlinien meist automatisiert.

Die langen Ausführungen um den Transformations-Code, sind erforderlich, um festzuhalten, welche Ausgangslage der Gestaltung einer Data Engineering Pipeline zugrunde liegt. Des Weiteren wird der Vollständigkeit halber, ein kurzer Blick auf die weiteren Strukturkomponenten des Datenproduktes geworfen.

Daten und Metadaten

An dieser Stelle soll noch einmal der Unterschied zum bisherige Ansatz erwähnt werden. Im Data Mesh ist das Datenprodukt für die Erstellung seiner Metadaten selbst verantwortlich, während in herkömmlichen Architekturansätzen diese Aufgaben oftmals durch ein externes System realisiert. Dieses hat zur Aufgabe, Metadaten zu extrahieren, zu sammeln, auszuwerten und zentral zur Verfügung zu stellen.

Plattformabhängigkeiten

Wenn es auch in der Literatur keine gezielte Aussprache für eine zugrundeliegende Technologie zur Umsetzung von Data Mesh gibt, soll jedoch auch hervorgehoben werden, dass Data Mesh eine gewisse Plattformabhängigkeit besitzt. Das erscheint auch notwendig, um sicherzustellen, dass verschieden Prozesse rund um das Datenprodukt auch tatsächlich umgesetzt werden können. Dabei ist jedoch die Autonomie eines jeden Datenproduktes zu berücksichtigen, ohne dadurch das Anwachsen von kleinen Datensilos voranzutreiben. Diese Abhängigkeit ist für das Datenprodukt von wesentlicher Bedeutung, da es seine Zugriffsmethoden oder die Speicherbedingungen von der zugrundeliegenden Plattform übernimmt.

Dehghani spricht in ihren Arbeiten von den „drei Ebenen der Datenplattform“ (vgl. Dehghani, 2023, S. 201-205). Ihren Ausführungen nach, wird eine „integrierte Plattform“ fokussiert, die „lose integrierte Dienste mit standardisierten und offenen Schnittstellen, […] als eine Kombination aus Eigenentwicklung und Integration von bestimmten Produkten“ miteinander verknüpft (Dehghani, 2023, S. 201). Die Dienste werden auf der Ebene der Dateninfrastruktur, der Datenprodukte und der Mesh-Ebene verteilt.

Ein präzise Ausführung des Ebenen-Gedankens wird in einem Review sog. „Grauer Literatur“ entnommen[14]. Darin wird, bezogen auf das Development, eine Ebene für die Pipeline-Prozesse identifizert (Goedegebuure et al., 2023). Die nachfolgende Gafik implementiert auf der Ebene der Datenprodukte die auf Basis von DDD spezialsierten Analysemodelle. Die darauunter liefgende Ebene, die „Data Product Components“ beinhalten die funktionalen und nichtfunktionalen Anforderungen der Datenprodukte. In diesem Sinne werden verschiedene Pipeline-Typen, wie bspw. Daten Pipelines und ML Training Pipelines erwähnt. An dieser Stelle kommen zudem verschiedene Designpatterns (z. B. *customer-supplier* und *anti-corruoption layer*) im Einsatz. Eine Ebene tiefer, befinden sich die zentral zur Verfügung gestellten *Self-Serve Platform Services*, ein Werkzeugkasten zur Umsetzung des Pipeline-Workflows.

Demzufolge wird an dieser Stelle der Bedarf an einer zugrundeliegenden Plattform deutlich, die sich folgerichtig darunter befindet (Self-Serve Platform Services).

[14] Graue Literatur ist eine Bezeichnung für Publikationen, die weder vom kommerziellen Verlagswesen kontrolliert werden noch im Buchhandel erhältlich sind. Beispiele sind z. B. interne Dokumente, wissenschaftliche Artikel, Konferenzberichte etc.).

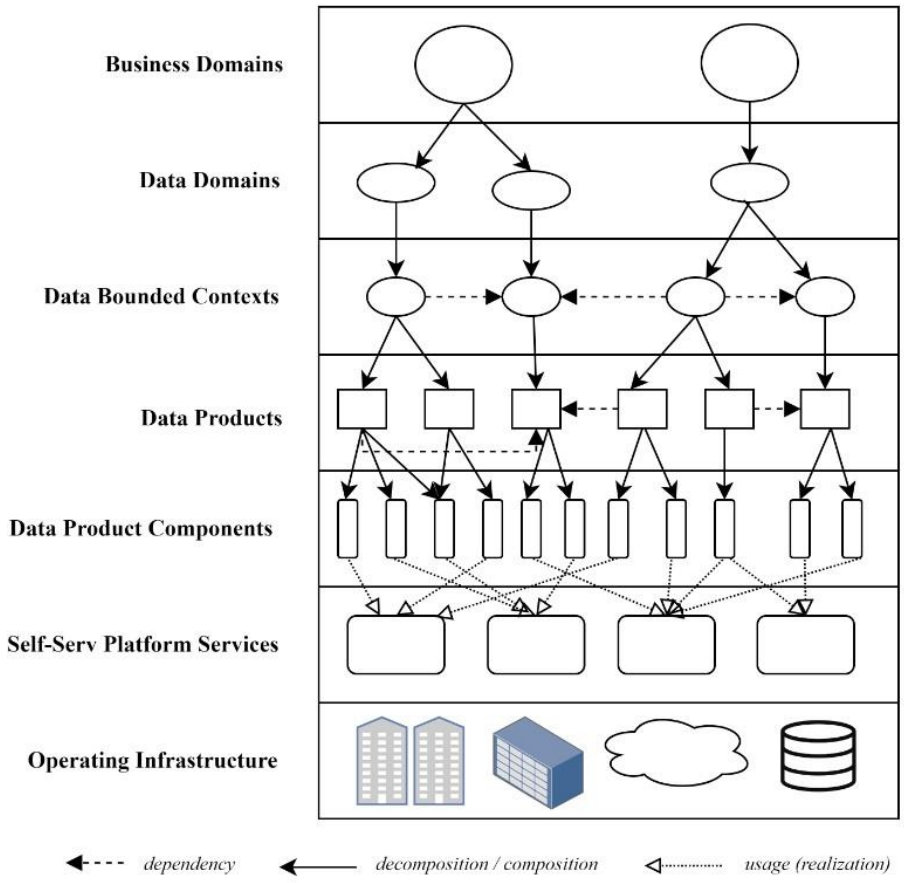

Abbildung 4: Ebenen Modell für das Data Mesh Development (Quelle: Goedegebuure et al., 2023).

4.4 Charakteristiken für einen Lösungsansatz

Zur Entwicklung der Charakteristiken eines Lösungsansatzes für eine Data Engineering Pipeline im Data Mesh, wurde im ersten Schritt untersucht, welchen Einfluss die von Zhamak Dehghani empfohlene Platzierung der Pipeline im Datenprodukt auf die anderen Data Mesh Prinzipien ausübt. Dabei wird deutlich, dass die Pipeline in enger Beziehung zur Self-Serve Data Platform steht. Das Datenprodukt zeichnet sich als geeigneter Ort ab, da dort der Lebenszyklus der Daten, d. h. von der Erstellung, über die Transformation bis hin zur Bereitstellung, durchlaufen wird. Zur Ausübung der Pipeline-Funktionen werden jedoch die Services der zugrundeliegenden Self-Serve Data Platform genutzt.

Um den Lösungsansatz zu konkretisieren, erfolgte eine Auseinandersetzung mit dem kleinsten architektonischen Einheit des Datenproduktes, dem Architekturquantum. Dazu wurden die drei Strukturkomponenten des Architekturquantums näher betrachtet und grundlegende Aufgaben untersucht.

Im Laufe des Kapitels soll zu Beginn dem Transformationsprozess mehr Beachtung geschenkt werden. Im Anschluss werden die angebundenen Datenquellen und Datennutzer sowie deren Anforderungen an die Daten genauer betrachtet. Die gewonnenen Erkenntnisse bilden die Charakteristiken und werden anschließend zusammengefasst wiedergegeben.

4.4.1 Transformationsprozesse

Für die Erstellung eines neuen Analysemodells, durchlaufen die Datenprodukte individuelle Transformationsschritte. Wie bereits im vorherigen Kapitel erwähnt, ist diese Transformation ein „internes Implementierungsdetail“, welches im Datenprodukt implementiert und von selbigem abstrahiert wird. Es ist Aufgabe des Data Product Developers zu entscheiden, wie die Transformationsschritte implementiert werden. Die bisherige Umsetzung der Transformation erfolgt in Daten-Pipelines, die die Daten von der Quelle abgreifen und zu einer Output-Senke transportieren (vgl. Dehghani, 2023, S. 268).

Folgende Möglichkeiten der Implementierung beeinflussen die Gestaltung der Pipeline:

- Programmatische oder nichtprogrammatische Transformation
- Datenflussbasierte Transformation
- Machine Learning als Transformation
- Zeitabhängige Transformation

Programmatische oder nichtprogrammatische Transformation

Innerhalb dieser Kategorie wird zwischen zwei Varianten unterschieden: Die programmatische und die nichtprogrammatische Transformation (vgl. Dehghani, 2023, S. 269).

Die **nichtprogrammatische Transformation** nutzt deklarative Abfragesprachen, wie bspw. SQL, Flux oder GraphQL um die Verarbeitung von Daten durchzuführen. Bei dieser Art der Transformation sind die Filterung und Umwandlung von Daten meist schneller und einfacher zu implementieren, da Daten einer Menge mittels einer Anweisung, komplett erfasst und umgewandelt werden. Die Beschränkung auf die Funktion der Anweisung kann jedoch auch einen Nachteil darstellen. Mit zunehmender Komplexität der Transformation, besteht die Gefahr der mangelnden Modularisierung und die Reduzierung des automatischen Testens.

Programmatische Transformationen verwende Codelogik in Form von Bedingungen und Anweisungen. Umgesetzt wird sie mittels programmatischer Datenverarbeitungs-Frameworks, wie bspw. Apache Beam, Spark etc. Darüber hinaus sind sie zur Einbindung in verschiedene Programmiersprachen, wie Java, Python etc. vorgesehen. Im Gegensatz zur nichtprogrammatischen Variante, bieten diese Frameworks eine Schnittstelle, die die Erstellung und

Ausführung von Pipeline-Workflows unterstützt. Die nachfolgende Tabelle fasst die Eigenschaften noch einmal zusammen.

	Vorteile	**Nachteile**
Nichtprogrammatische Transformation	• einfach • wenig Code	• schlecht modularisierbar • kein autom. Testen • hohe Komplexität • sehr kontextbezogen
Programmatische Transformation	• Modularisierung • automatisierte Tests • erweiterbar	• kompliziert für Data Product Developer mit wenig Code-Erfahrung

Tabelle 3: Programmatische vs. nichtprogrammatische Transformation.

Datenflussbasierte Transformation

Eine Daten-Pipeline zeichnet sich durch eine bestimmte Anzahl von Transformationsschritten aus, die in Form eines gerichteten Graphen durchlaufen werden. Die Schritte kennzeichnen Funktionen, die an entsprechenden Stellen auf die, den Graphen durchlaufenden Daten, angewendet werden. Traditionell werden dieses Schritte in *Integration/Ingestion*, *Datenqualität prüfen*, *Datenbereinigung/Transformation* und *Datenbereitstellung* unterschieden.

Die Pipeline-Tätigkeit spielt sich im Inneren des Datenproduktes ab. Sie ist dem Aufgabenkontext des Datenproduktes zugehörig. In diesem Fall ist die enge Kopplung akzeptabel, da Pipelines an Komplexität verlieren. Die Transformation ist auf den Kontext des Datenproduktes begrenzt und ermöglicht Änderungsprozessen eine hohe Flexibilität. Zwischen den Datenprodukten selbst, finden keine Transformationen statt (vgl. Dehghani, 2023, S. 270-271).

	Vorteile	**Nachteile**
Enge Kopplung der Transformationen im Datenprodukt	• weniger Komplexität • kein Einfluss auf andere Datenprodukte • klare Schnittstellen • Probleme können sofort lokalisiert werden • erweiterbar	• keine Wiederverwendbarkeit aufgrund kontextbezogener Funktionen im Datenprodukt • Pipeline-Schritte sind auf Datenproduktgrenzen beschränkt

Tabelle 4: Eigenschaften der datenflussbasierten Transformation.

Machine Learning als Transformation

Hierbei handelt es sich um eine modellbasierte Transformation, die mittels eines Machine-Learning-Modells oder eines statisches Modells umgesetzt wird. Vorteile liegen im hohen Grad der Automatisierung (vgl. Dehghani, 2023, S. 271-272). Bei dieser Variante werden die Berechnungen im Modell durchgeführt, welche neben der Realisierung als Microservices auch als Datenprodukt umgesetzt werden.

Zeitabhängige Transformation

Die zeitabhängige Transformation befasst sich mit Daten, die sowohl eine zeitliche Spanne beschreiben und dadurch klare Grenzen setzen. Sie wird vorranging bei der Verarbeitung von Zeitreihendaten, bei Datums- und Zeitformatierung, Datenintegration und Datenfusion sowie bei der Verarbeitung von Streaming-Daten eingesetzt.

4.4.2 Datenquellen und Datennutzer

Nachdem die Verarbeitungsmethoden thematisiert wurden, folgt eine Betrachtung der Input- und Output-Ports. Bei Datenprodukten spielt das Teilen, Aggregieren und Bereitstellen der Daten eine wesentliche Rolle um einen größtmöglichen Nutzen anzubieten.

Dehghani erwähnt in Ihrem Werk drei unterschiedliche „Archetypen von Datendomänen“ (vgl. Dehghani, 2023, S. 63-66). Sie unterscheidet die Einordnung der analytischen Daten in

- quellenorientierte Datendomänen,
- aggregierte Datendomänen und
- konsumentenorientierte Datendomänen.

Vor diesem Hintergrund werden die Datenprodukte in die drei Kategorien

- quellenorientierte Datenprodukte,
- aggregierte Datenprodukte und
- konsumentenorientierte Datenprodukte

eingeordnet (vgl. Dehghani, 2023, S. 261).

Quellenorientiert Datenprodukte unterstützen mehrere Datenquellen. Dieses können operative Systeme, andere Datenprodukte sowie lokaler Input sein. Eine präzisere Aussage wird in (Wider et al., 2023) getroffen. Folgende Beschreibung lässt sich entnehmen:

> *„Such source-aligned data products often have only one input port to consume data from the original source systems, but exhibit several output ports.“ (Wider et al., 2023, S. 2)*

Demzufolge wird für die nachfolgende Ausarbeitung diese Annahme zugrunde gelegt und für das quellenbasierte Datenprodukt ein Input-Port und mehrere Output-Ports festgelegt.

Ein Datenprodukt kann seine Werte aus verschiedenen anderen quellenorientierten Datenprodukten speisen. Ist dies vorgesehen, spricht man von **aggregierten Datenprodukten**. Beispielsweise werden Attribute einer Eigenschaft des Datenproduktes aus verschiedenen Quellen zusammengeführt, so dass sich eine Informationen zweckdienlich beieinander abgegriffen werden können. Allerdings empfiehlt Dehghani, diese aggregierte Sicht nicht mit Informationen zu überladen um möglichst alle Facetten einer Eigenschaft des Datenproduktes zu präsentieren. Die Gefahr einer erneuten Zentralisierung von Informationen steigt. Demzufolge wird empfohlen „…, dass die Nutzenden ihre eigenen *zweckdienlichen* Aggregate erstellen und der Versuchung widerstehen, hochgradig wiederverwendbare und anspruchsvolle Aggregate zu verwenden.“ (Dehghani, 2023, S. 66).

Aufgrund der Aussage, dass es viele Datenquellen geben kann, wird davon ausgegangen, dass das aggregierte Datenprodukt mehrere Input-Ports enthalten kann. Über die Anzahl der Output-Ports ist keine Erwähnung ersichtlich.

Konsumentenorientierte Datenprodukte unterscheiden sich von quellenorientierten Daten dadurch, dass sie analytische Daten für einen bestimmten Zweck, bspw. einem speziellen Anwendungsfall, bereitstellen. Durch den Umstand, die Daten für eine gezielte Anforderung aufzubereiten, unterliegt das Datenprodukt größeren Veränderungen, die sich auf seine Struktur auswirken und in dessen Transformationen widerspiegeln (vgl. Dehghani, 2023, S. 66).

Aufgrund seiner Ähnlichkeit zum aggregierten Datenprodukt hinsichtlich des Zusammentragens der Informationen aus verschiedenen Quellen (die im Falle des aggregierten Datenproduktes als quellenorientierte Datenprodukte identifiziert werden), könnte das konsumentenorientierte Datenprodukt die fehlenden Angaben hinsichtlich des Output-Ports liefern. Dadurch, dass es Informationen aus mehreren Datenquellen bezieht, aber nur einen speziellen Anwendungsfall bedient, wird an dieser Stelle festgelegt, dass das konsumentenorientierte Datenprodukt über mehrere Input-Ports und nur einen Output-Port verfügt.

Daten konsumieren

Wie bereits erwähnt, muss ein Datenprodukt über spezielle Input-Ports verfügen um Daten aus einer Datenquelle (bspw. einer Datenbank, einem Service oder anderem Datenprodukt) zu beziehen. Diese Ports können z. B. Domain-Events in Echtzeit von einem Microservice oder in regelmäßigen periodischen Abfragen von einem Upstream-Datenprodukt abrufen.

Eine genaue Spezifikation der APIs erfolgt kontextbezogen, so dass man hier nur erwähnen kann, dass die Input-Ports über eine gewisse Standardisierung bzgl. der zugrundeliegenden

Plattform verfügen müssen. Darunter können z.B. möglichst identische Zugriffsverfahren, eine standardisierte Auswahl an verwendeten Daten-Schemata, eine fortlaufende Versionierung etc. fallen.

Daten bereitstellen

Zur Bereitstellung der Daten können extern adressierte APIs eingesetzt werden, die den Abruf der zur weiteren Nutzung bereitgestellten Daten, ermöglicht. Es wird empfohlen keinen Zugriff auf die eigentlichen Daten zu gewähren, da sich das Datenprodukt, aufgrund geänderter organisatorischer oder unternehmensstrategischer Ausrichtungen auch ändern kann. Über die Schnittstellen wird sichergestellt, dass kein anderer Datenkonsument in den Kontext des liefernden Datenproduktes eingebunden wird und somit dessen Änderungsprozessen unterworfen ist. Wie auch bei den Input-Ports liegen zur Gestaltung und Spezifikation der Output-Ports, explizit definierte Verträge und APIs zugrunde.

Verknüpfung der Input- und Output-Ports

Das Zusammenspiel zwischen Input- und Output-Ports zeichnet sich als ein wesentlicher Bestandteil der Kommunikation zwischen den Datenprodukten aus. Die Ansätze von (Wider et al., 2023) erwähnen einen Einfluss von *Site Reliability Engineering (SRE)*[15] bei der Schaffung der Verträge für den Datenaustausch. Ihren Ausführungen nach, wird dies deutlich bei der Einführung von und *Service Level Agreements (SLA)*[16] und *Service Level Objectives (SLO)*[17] für Datenprodukte.

Da die Ausgestaltung von Schnittstellen nicht im Fokus dieser Arbeit steht, wird sie an dieser Stelle nicht weiter vertieft. Sie ist jedoch Bestandteil der Strukturkomponenten des Architekturquantums, und sollte im Sinne von Policy als Code an dieser Stelle berücksichtigt werden.

Anforderungen der Datennutzer

Data Mesh entsprechend, kommt hier der Service-Gedanke des Product Thinking zum Einsatz. Um das zu bewerkstelligen, gibt es eine Reihe von Entwurfsüberlegungen darüber, welche Form der Datenbereitstellung erfolgen bzw. angeboten werden soll. Nutzer von Daten können sowohl Personen, als auch Anwendungen sein. Zu den Personen zählen Data Analysts, Data Scientists sowie Software-Entwicklerinnen und -Entwickler.

[15] Site Reliability Engineering ist eine Reihe von Grundsätzen und Praktiken, die bei Google entwickelt wurden und erstmalig im Jahr 2003 durch ein Site Reliability Team praktiziert wurden. Es kombiniert Aspekte der Softwaretechnik mit Infrastruktur- und Betriebsanforderungen. Hauptziel ist die Schaffung skalierbarer und hochzuverlässiger Softwaresysteme (Farooqui & Chikoti, 2021).
[16] Ein Service Level Agreement ist ein Vertrag zwischen Anbieter und Kunden, der Konsequenzen für die Erfüllung (oder Nichterfüllung) der darin enthaltenen Vereinbarung über Metriken vorsieht (vgl. Jones et al., 2024).
[17] Ein Service Level Objective ist eine Zielwert innerhalb eines SLA. Gemessen werden bestimmte Metriken, wie bspw. die Verfügbarkeit oder Reaktionszeit (vgl. Jones et al., 2024).

Folgende Anforderungen müssen berücksichtigt werden, damit Nutzer analytischer Daten den meisten Nutzen und den geringsten Aufwand verzeichnen können (vgl. Dehghani, 2023, S. 243-253):

- Multimodale Daten
- Bitemporale Daten
- Unveränderliche Daten
- Read-Only-Zugriff

Multimodale Daten sind Daten, die einmal eingelesen und für mehrere ausgewählte Nutzergruppen zur Verfügung gestellt werden. Mit anderen Worten lässt sich sagen, dass die Semantik der Daten in mehreren syntaktischen Ausprägungen zur Verfügung gestellt werden kann.

Bitemporale Daten beschreiben Daten, die über einen längeren Zeitraum aufgenommen wurden, sog. Längsschnittdaten. Diese Daten liefern Erkenntnisse, die sowohl retrospektiv als auch zukunftsorientiert betrachtet werden. Durch eine kontinuierliche Dokumentation dieser Werte, lassen sich Prognosen und Tendenzen ableiten, aber auch ein Verständnis für vergangene Ereignisse. Es ist daher erforderlich, dass „jedes Datenquantum bitemporale Daten bereitstellt“ (Dehghani, 2023, S. 244).

Die **Unveränderlichkeit** von Daten ist eine Voraussetzung, die sicherstellt, dass das Architekturquantum eine stets konsistente Sicht auf die Daten mehrerer Domänen liefert. Soll bspw. der Output mehrerer Datenprodukte zu einem festgelegten Zeitpunkt abgefragt werden, muss man sich darauf verlassen können, dass die Daten im Vorfeld nicht aktualisiert wurden. Das Festhalten der Werte zu einem bestimmten Zeitpunkt, macht diese Daten reproduzierbar und ermöglicht einen erneuten Abruf zu einem späteren Zeitpunkt.

Der **Read-Only-Zugriff** auf die Daten, stellt sicher, dass die Daten nicht verändert werden und unterstreicht somit die zuvor genannte Forderung der Unveränderlichkeit.

4.4.3 Zusammenfassung der Charakteristiken

Die zuvor beschriebenen charakteristischen Merkmale der Pipeline-Arbeit im Data Mesh, werden in den nachfolgenden Tabellen zusammengefasst.

Jede Tabelle verfügt über einen Titel sowie eine fortlaufende Nummerierung mit einem kennzeichnenden Kürzel vorab, wobei das „C“ eine Abkürzung für „Charakteristik“ darstellen soll. Es folgt eine weitere Titelleiste, die die Spalten *ID*, *Kontext, Quelle, Ergebnis* und *Begründung* beinhaltet. Ein „+“ bei „Ergebnis“ bedeutet, dass das Feld für den Lösungsansatz relevant ist.

Einordnung der Pipeline

Die nachfolgende Tabelle gibt an, wo die Pipeline im Data Mesh eingeordnet werden kann. Die Spalte „Ergebnis“ zeigt auf, welche Entscheidung für diese Arbeit getroffen wurde.

C1: Einordnung der Pipeline				
ID	**Kontext**	**Quelle**	**Ergebnis**	**Begründung**
C1-1	Domain Ownership	Kap. 4.2	-	Aufgrund des organisatorischen Charakters nicht relevant für den Lösungsansatz.
C1-2	Data as a Product	Kap. 4.2	+	Transformationsprozesse sind wesentlicher Bestandteile einer Pipeline.
C1-3	Federated Governance	Kap. 4.2	-	Besitzt einen organisatorischen Charakter und realisiert u.a. ein globales Regelwerk.
C1-4	Self-Serve Data Platform	Kap. 4.2	+	Bildet eine gemeinsame Basis für Pipeline-Tools, Speicherzugriffe etc.

Tabelle 5: Einordnung der Pipeline im Data Mesh.

Charakteristiken der Strukturkomponenten Datenproduktes

Nachdem die Entscheidung auf das Datenprodukt gefallen ist, verdeutlicht die nachfolgende Tabelle den Einfluss der Strukturkomponenten des Datenproduktes auf die Pipeline.

C2: Datenprodukt				
ID	**Kontext**	**Quelle**	**Ergebnis**	**Begründung**
C2-1	Code und Logik - Transformation	Kap. 4.3	+	Datentransformation werden als Code implementiert.
	Code und Logik - Schnittstellen	Kap. 4.3	+	Schnittstellen werden als Code implementiert.
	Code und Logik - Policy als Code	Kap. 4.3	-	Nicht ersichtlich ob Richtlinien innerhalb der Pipeline zu tragen kommen.

C2-2	Daten und Metadaten	Kap. 4.3	-	Diese werden nach wie vor außerhalb der Pipeline implementiert. Es findet sich keine Erwähnung innerhalb der Literatur für eine Platzierung innerhalb der Pipeline.
C2-3	Plattform-abhängigkeit	Kap. 4.3	+	Plattformabhängigkeit ist explizit gewünscht und auch notwendig. Ermöglicht die Nutzung zentraler Pipeline-Tools, Speicherzugriffe etc.

Tabelle 6: Charakteristiken der Strukturkomponenten des Datenproduktes.

Charakteristiken des Transformationsprozesses

Aufgrund der intensiven Auseinandersetzung der Pipeline mit der Transformation von Daten, wird eine Entscheidung hinsichtlich der Transaktionsprozesse getroffen.

C3: Transformationsprozesse				
ID	**Kontext**	**Quelle**	**Ergebnis**	**Begründung**
C3-1	Programmatische oder nichtprogrammatische Transformation	Kap. 4.4.1	+	Je nach Aufgabe und Kontext kann die Implementierung der Pipeline mittels deklarativer Abfragesprachen oder anhand entsprechender Frameworks direkt gecodet werden.
C3-2	Datenflussbasierte Transformation	Kap. 4.4.1	+	Die Pipeline durchläuft die Schritte: Integration/Ingestion, Datenanalyse, Datenbereinigung/Transformation und Datenbereitstellung.
C3-3	Machine Learning Transformation	Kap. 4.4.1	+	Modellbasierte Transformation, die mittels eines Machine-Learning-Modells implementiert wird.

C3-4	Zeitabhängige Transformation	Kap. 4.4.1	+	Die Pipelineschritte sind auf die Verarbeitung von Daten ausgelegt, die konkrete Zeitspannen adressieren (Zeitreihen, Datumsangaben, Streaming-Daten etc.).

Tabelle 7: Charakteristiken für die Transformationsprozesse.

Datenquellen und Datennutzer

Pipeline-Eingang und -Ausgang werden anhand von Datenprodukt-Archetypen klassifiziert. Relevante Datenquellen und Nutzer werden in der nachfolgenden Tabelle zusammengefasst.

C4: Datenquellen und Datennutzer				
ID	**Kontext**	**Quelle**	**Ergebnis**	**Begründung**
C4-1	Quellenorientierte Datenprodukte	Kap. 4.4.2	+	Ein Input-Port, mehrere Output-Ports.
C4-2	Aggregierte Datenprodukte	Kap. 4.4.2	-	Innerhalb der verfügbaren Literatur wurde keine konkrete Aussage vorgefunden.
C4-3	Konsumentenorientierte Datenprodukte	Kap. 4.4.2	+	Mehrere Input-Ports, ein Output-Port.

Tabelle 8: Charakteristiken der Datenquellen und Datennutzer.

Bereitstellung der Daten

Die Anforderungen der Daten durch die Nutzer zeigen auf, welche besonderen Eigenschaften diese für den Wirkbetrieb im Data Mesh benötigt werden.

C5: Anforderungen an die Daten				
ID	**Kontext**	**Quelle**	**Ergebnis**	**Begründung**
C5-1	Multimodale Daten	Kap. 4.4.2	+	Ein und dieselbe Semantik lässt sich in mehreren syntaktischen Ausprägungen beschreiben.

C5-2	Bitemporale Daten	Kap. 4.4.2	+	Ermöglicht die Bewertung von Daten in fest definierten Zeitabschnitten.
C5-3	Unveränderliche Daten	Kap. 4.4.2	+	Unterstützt retrospektive Entscheidungen. Ergebnisse lassen sich kontext- und zeitbezogen nachvollziehen.
C5-4	Read-Only-Zugriff	Kap. 4.4.2	+	Stellt eine gewisse Revisionssicherheit dar und vermeidet die nachträgliche Veränderung der Daten.

Tabelle 9: Charakteristiken zur Bereitstellung der Daten.

5 Vorstellung des Lösungsansatzes

Diese Arbeit zielt darauf ab, die Lösungsansätze für Data Engineering Pipelines zu erarbeiten, die sich an einer Datenarchitektur nach dem Data Mesh-Prinzip orientieren.

Im vorherigen Kapitel wurden entsprechenden Charakteristiken erarbeitet, die ein solcher Lösungsansatz berücksichtigen sollte (Kap. 4.3). Die nachfolgenden Ausführungen sollen nun schildern, wie dieser Lösungsansatz realisiert werden kann. Dazu wird in einem ersten Schritt auf die gewählte, zugrundeliegende Technologie eingegangen (Kap. 5.1).

Um diese Technologie anzuwenden, wird ein angemessener Anwendungsfall herangezogen, der in Kapitel 5.2 beschrieben wird. Die Umsetzung des Lösungsansatzes erfolgt durch die Implementierung des Anwendungsfalls auf Basis der gewählten Technologie in Kap. 5.3.

5.1 Verwendete Technologie

Datenprodukte leben vom Austausch untereinander. Zum einen können Datenprodukte Datenquellen darstellen und zum anderen, auch ein Zielsystem für andere Datenprodukte sein. Der Datenaustausch ist somit ein zentrales Element des Lebenszyklus des Datenproduktes und wird über dessen Ports realisiert. Die Herausforderung dabei ist, dass mit zunehmende Beteiligung von Datenprodukten, der Datenfluss und die Kommunikation der Datenprodukte untereinander, sehr komplex und unüberschaubar werden kann.

Eine Idee aus der Software-Entwicklung besagt, dass das Einfügen einer zusätzlichen Ebene, die Gesamtkomplexität reduzieren kann. Genau diese Aussage wird in dem nachfolgenden Zitat von David John Wheeler verdeutlicht:

> *Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem." (David J. Wheeler[18])*

Die Idee soll in der vorliegenden Arbeit aufgegriffen werden: Der Datenfluss wird auf eine separate Ebene ausgelagert. Diese Ebene soll mittels einer Daten-Streaming-Plattform umgesetzt werden. An dieser Stelle wird die Abhängigkeit zur Plattform deutlich, da das Datenprodukt Zugriff auf Werkzeuge zur Aufgabenrealisierung benötigt. Zudem ist durch die Nutzung gemeinsamer Werkzeuge mit einer hohen Standardisierung von Schnittstellen zu rechnen.

[18] Der britische Computerpionier David J. Wheeler (1927-2004) gilt zusammen mit Maurice Wilkes und Stanley Gill als Entwickler der ersten Subroutine. Von seinem bekanntesten Zitat wird allerdings meistens nur der erste Satz zitiert und das Zitat so verfälscht. Der Vollständigkeit halber, wird das komplette Zitat aufgeführt. Dabei wird offen gehalten, welches „andere Problem" erzeugt wird (vgl. Wikipedia01, 2024).

Daten- und Event-Streaming-Plattformen

Daten- und Event-Streaming-Plattformen haben die Aufgabe, eingehende Nachrichten entgegenzunehmen und an ein Zielsystem weiterzuleiten. Ein Nachrichtenvermittler (engl. *Broker*) übernimmt die Kommunikation zwischen den Systemen. Dabei liegt der Fokus auf der Entkopplung verschiedener Endpunkte, sowohl zeitlich als auch technisch.

Die Plattform verwendet ein definiertes Format, damit alle Akteure in dem System die Nachrichten verstehen können. Zu den Akteuren zählen u. a. die Nachrichtenerzeuger (eng. *Producer*) und die Nachrichtenkonsumenten (engl. *Consumer*). Eingehende Nachrichten werden in sogenannten *Queues*[19] (dt. Warteschlangen) gespeichert.

Es erfolgt ein themen- und inhaltsbasiertes Nachrichten-Routing, bspw. nach dem *Publish-Subscribe-Prinzip*. Demnach werden Nachrichten von einem Producer veröffentlicht, die der Consumer bei Bedarf zu einem gewählten Zeitpunkt beziehen kann.

Vorteile solcher Plattformen sind die einfache Skalierbarkeit durch die Hinzunahme weiterer Nachrichten-Akteure, die Persistenz der Nachrichten, die sonst nach der Zustellung nicht mehr zugreifbar wären sowie die Performanz, durch den Versand von Binärdatenformaten (Trennung von Syntax und Semantik). Letzteres wird nicht von allen Plattformen unterstützt.

Es können unterschiedliche Streaming-Technologien zum Einsatz kommen:

- Log[20]-orientierte Broker
- Queue-orientierte Broker
- Subscription-orientierte Broker

Vergleich von Broker-Services

Zur Auswahl eines Broker-Typs, werden je ein Vertreter des *log-orientierten* Broker-Services und des *queue-orientiert*en Broker-Services miteinander verglichen. Beide Broker-Typen sind in ihrer Technologie weit verbreitet, was sich im Umfang von Dokumentation und Support bemerkbar macht.

Während sich Apache Kafka, eine freie Software der Apache Software Foundation, den log-orientierten Brokern zuordnen lässt, nähert sich das ebenfalls freie RabbitMQ (Fa. VMware) den queue-orientierten Brokern. Der dritte Typ, der *Subscription-orientierte* Broker, kommt in

[19] Eine *Queue* (dt. Warteschlange) ist ein abstrakter Datentyp, der für die Zwischenspeicherung von Objekten in einer bestimmten Reihenfolge verwendet wird. Queues sind spezielle Listen, in denen Elemente an einem Ende eingefügt und am anderen Ende entnommen werden (vgl. Wikipedia02, 2024).
[20] Als *Logfile* (dt. Protokolldatei) wird eine Datei bezeichnet, in der Prozesse, die in einem Computer- oder Netzwerksystem ablaufen, protokolliert werden (vgl. Wikipedia03, 2024).

dieser Arbeit nicht zum Einsatz, da es sich dabei um einen Cloud-native-basierenden Service handelt. Dieser wird aufgrund des Cloud-Bezugs nicht näher betrachtet.

Die nachfolgende Tabelle fasst einige Unterschiede von Kafka und RabbitMQ zusammen:

Anforderung	**Log-orientiert (z. B. Kafka)**	**Queue-orientiert (z. B. RabbitMQ)**
Architektur	• Partitionsbasiertes Design • Pull-Modell • Producer veröffentlichen Nachrichten, die Consumer abonnieren können • Unterstützung von physischer und zeitlicher Entkopplung	• Komplexes Nachrichten-Routing, da viele Regeln mitgesendet werden • Push-Modell • Producer erwartet eine Bestätigung über den Nachrichtenerhalt • Unterstützung von physischer Entkopplung
Nachrichten-bearbeitung	• Nachrichtenverbrauch mittels Offset-Tracker verfolgt • Nachrichten werden gemäß Aufbewahrungsrichtlinie behalten und können von mehreren Konsumenten, mehrmals verarbeitet werden	• Überwachung des Nachrichtenversandes • Löschung der Nachrichten nach Bestätigung des Erhalts • Unterstützung von Nachrichtenprioritäten
Performanz	• Überträgt bis zu Millionen Nachrichten/Sek. in Nahezu-Echtzeit • Performant durch Partitionierung	• Geringe Latenz • Versendet Tausende Nachrichten/Sek. • Benötigt mehrere Broker
Programmier-sprache und Protokoll	• Unterstützt eine Vielzahl von Sprachen • Abwärtskompatibel zu Protokollen (MQTT, STOMP)	• Begrenzte Auswahl von Sprachen • Verwendet Binärprotokoll über TCP

Tabelle 10: Unterschiede zwischen Kafka und RabbitMQ (Quelle: vgl. Amazon, 2024).

Auswahl und Begründung eines Broker-Services

In dieser Arbeit fällt die Auswahl auf eine log-orientierte Daten-Streaming-Plattform zur Realisierung des Data Mesh-Ansatzes. Der Tabelle mit der Gegenüberstellung eines log-orientierten und queue-orientierten Brokers, lassen sich bereits Tendenzen für diese Entscheidung entnehmen, da sie eine gewisse Nähe zu den Prinzipien des Data Mesh aufweisen.

Weit verbreitet ist die Nutzung von Apache Kafka, einer Open Source Software, die unter der Apache 2.0 Lizenz genutzt werden kann.

Kafka ist ein verteilter, partitionierter und replizierender Service für Datenströme. Die Verteilung drückt sich durch den Cluster-Betrieb aus, der eine hohe Fehlertoleranz und Skalierbarkeit ermöglicht. Da der Nachrichtenversand einem Themenschwerpunkt (engl. *Topic*) zugeordnet wird, spricht man an dieser Stelle auch von Partitionierung. Zudem werden die Nachrichten über mehrere Knoten (*Server*) repliziert, was eine gewisse Verfügbarkeit sicherstellt.

Die Auswahl von Kafka wird aufgrund folgender Aspekte begründet:

- Keine Logik auf der Transportebene (weniger Komplexität)
- Themen- und inhaltsbasierte Speicherung der Nachrichten durch Producer
- Abholung und individuelle Aufbereitung der Daten liegt beim Consumer (Pull-Prinzip)
- Ein oder mehrere Consumer können die Nachrichten abonnieren und zu gegebenem Zeitpunkt bzw. mehrmalig und asynchron abrufen (technische und zeitliche Entkopplung)
- Datenvorhaltung für definierte Zeit, kein Löschen und erneutes Versenden der Daten durch den Producer nötig
- Datenautonomie bleibt erhalten

Interne Architektur von Kafka

Wie beim Datenprodukt, können bei Kafka Nachrichtenerzeuger und -nutzer zum Einsatz.

Producer kennzeichnen die Erzeuger-Systeme, die z. B. Datenbanken, Data Warehouses, Data Lakes oder einzelne Dokumente sowie Microservices sein können.

Consumer stellen die Nachrichtenkonsumenten dar. Dabei kann es sich um Microservices, ML-Trainingsmodelle, Systeme für Monitoring etc. handeln.

Topics kennzeichnen die Inhalte über die sich die Erzeuger und Verbraucher austauschen. Sie stellen einen Themenspeicher dar, in den die Erzeugersysteme ihre Daten ablegen, und die Datennutzer ihre Daten beziehen. Innerhalb der Topics kann eine Aufteilung der Nachrichten in Partitionen erfolgen. Der Broker hat die Aufgabe, die Nachrichten auf die einzelnen Partitionen zu verteilen. Der sequentielle Zugriff auf Daten in den Partitionen, ist durch ein Offset (vergleichbar mit einem Index) möglich und wird durch den Consumer gesteuert. Die zuletzt eingefügte Nachricht wird gelesen, was jedoch auch umgekehrt werden kann.

Kafka speichert seine Nachrichten für einen vorab definierten Zeitraum oder bis eine max. Größe des Logs erreicht ist. Somit unterliegt es der Autonomie des Consumers, ob und wann er welche Nachricht aus welcher Partition er beziehen möchte. Ein mehrmaliges Abrufen und die Nutzung in unterschiedlichen Zielsystemen, wird somit effizient unterstützt.

Der *Broker* ist der zugrundeliegende Kafka-Service, der den Nachrichtenversand koordiniert. Er ist der Speicherort für die Topics. Meist werden drei Broker verwendet, davon ist einer der *Leader*, die anderen agieren als *Follower*. Der Leader wird auch als *Bootstrap-Server* bezeichnet, der als zentraler Ansprechpartner innerhalb eines Clusters vorgesehen ist.

Zur Verwaltung und Steuerung eines Kafka-Clusters, d. h. mehrerer Broker, kommt eine weitere Komponente, der *Zookeeper*, zum Einsatz. Der Zookeeper wird dadurch auch als *Cluster-Manager* bezeichnet.

Die nachfolgende Grafik zeigt den internen Aufbau der Architektur.

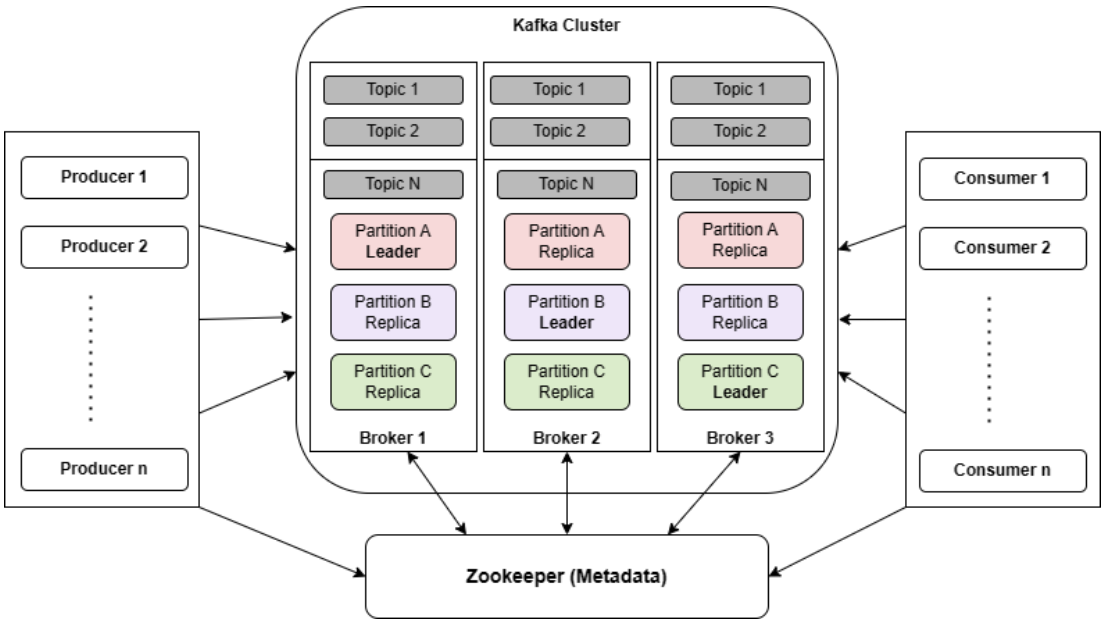

Abbildung 5: Interne Architektur von Apache Kafka (Quelle: Gupta, 2024).

Während der Producer, der Consumer und das Topic ideal auf die Eigenschaften des Datenprodukts abbildbar sind, gibt es für den Broker, den Zookeeper und das Kafka-Cluster eher weniger passende Gegenstücke.

Verwendung einer Schema-Registry

Eine Besonderheit von Apache Kafka ist die Speicherung der Daten in einem schlanken Binärdatenformat. Um den Nachrichtenversand effizient zu transportieren, ist es hilfreich, ihn von seiner Darstellung zu trennen. Der Vorteil davon ist, dass Informationen, die für die Strukturierung der Daten aufgewendet werden, wegfallen können. Durch die Aufspaltung von Syntax und Semantik ist der Code zudem flexibler, was sich für die weitere Verwendung als nützlich erweisen kann.

Kafka setzt dazu das eigens entwickelte Avro-Format ein. Dabei handelt es sich um ein RPC- und Serialisierungs-Framework, welches JSON verwendet um Datentypen und Protokolle zu definieren. Die eigentlichen Daten werden in einem kompakten Binärdatenformat serialisiert.

Diese Eigenschaft kommt den Prinzipien eines Datenproduktes sehr entgegen, da diese das Ziel verfolgen, Daten in ihrer „Rohfassung“ jedem interessierten bzw. registrierten Daten-Konsumenten zur Verfügung zu stellen. Über ein explizit definiertes Schema, lassen sich die Daten dann in die gewünschte Form bringen. Durch die Bereitstellung verschiedener Schemata, kann eine breite Gruppe von Konsumenten bedient werden.

Zur Bereitstellung der verschiedenen Schemata, kann eine *Schema-Registry* verwendet werden. Dabei handelt es sich um eine verteilte Speicherschicht für Schemata, die Kafka als zugrunde liegenden Speichermechanismus nutzt. Die nachfolgende Grafik illustriert die Funktionsweise.

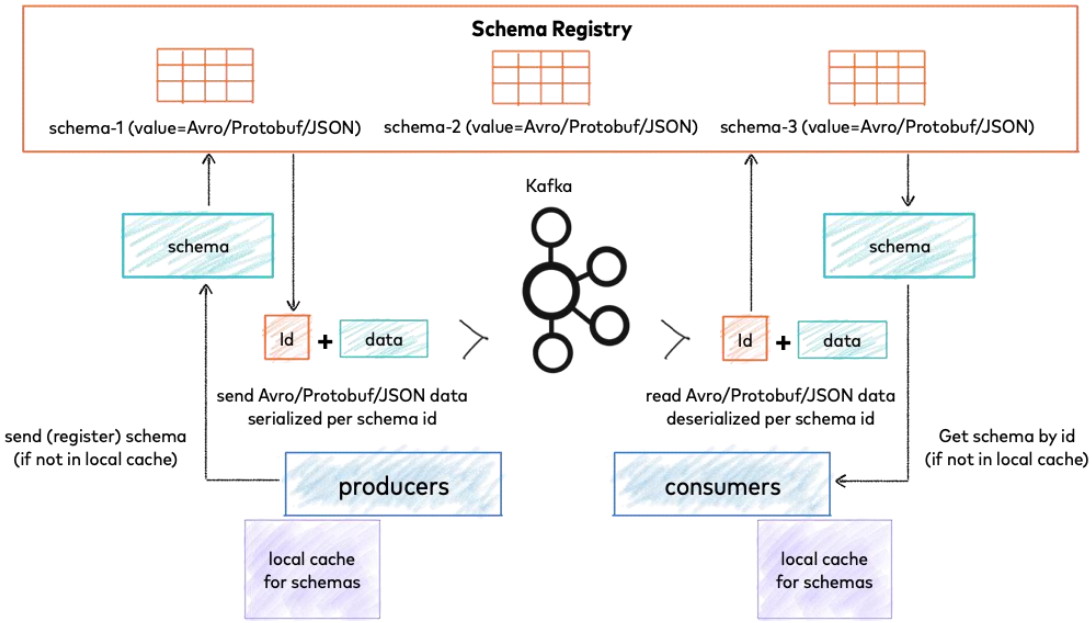

Abbildung 6: Funktionsweise der Schema-Registry (Quelle: Confluent Inc., 2024).

Um Nachrichten losgelöst von ihrem Darstellungsformat zu übertragen, wird die Nachricht von ihrem zugrundeliegenden Datenmodell entkoppelt. Das Schema wird in der Schema-Registry registriert, die daraufhin eine entsprechende ID zurückliefert. Anschließend wird die Nachricht zusammen mit der ID des gewünschten Schemas an ein Topic gesendet/serialisiert. Die Übertragung kann im (sparsamen) Binärdatenformat (bspw. Avro) erfolgen. Der Consumer holt die Nachricht aus der jeweiligen Partition eines Topics ab und konvertiert sie in das gewünschte Format. Dazu erfragt er über die ID bei der Registry das Schema und deserialisiert damit die empfangenen Binärdaten.

5.2 Vorstellung des Anwendungsfalls

Um die zuvor beschriebene Technologie bezüglich ihrer Eignung zum Einsatz im Data Mesh näher zu untersuchen, wird nun ein Anwendungsfall vorgestellt auf Basis dessen eine Implementierung im folgenden Kapitel erarbeitet wird.

Für diese Arbeit wird das Internetangebot „Gewässerkundliches Informationssystem PEGELONLINE der Wasserstraßen und Schifffahrtsverwaltung des Bundes (WSV)“ gewählt, welches von der Generaldirektion Wasserstraßen und Schifffahrt (GDWS) herausgegeben wird (vgl. GDWS, 2024). Die WSV ist ein Geschäftsbereich des Bundesministeriums für Digitales und Verkehr (BMDV), wobei die GDWS eine weitere Untergliederung im Bereich der WSV darstellt (vgl. WSV, 2024).

PEGELONLINE veröffentlicht auf seiner Website tagesaktuelle Rohwerte unterschiedlicher gewässerkundlicher Parameter. Dazu zählen z. B. der Wasserstand, die Wassertemperatur und Flussmenge der Wasserstraßen des Bundes. Dies geschieht bis maximal 30 Tage rückwirkend und ist über eine Reihe von Webservices (z. B. via REST und XML-basiert), die kostenfrei genutzt werden können, abrufbar. Die kostenfreie Bereitstellung erfolgt durch das ITZBund und ist ohne Registrierung nutzbar. Lediglich eine Quellangabe ist bei Verwendung der Services erwünscht.

Datenquellen

Das Internetangebot von PEGELONLINE stellt kontinuierlich Wasserstandsdaten zur Verfügung, die über die verschiedenen Messstellen der Wasserstraßen des Bundes erhoben werden. Die PEGELONLINE REST-API fungiert dabei als einfache Schnittstelle, die unterhalb von *https://www.pegelonline.wsv.de/webservices/rest-api/v2* adressierbar ist.

Anmerkung: Um ein besseres Verständnis der nachfolgenden Ausführungen zu erzielen, wird für Elemente des implementierten Lösungsansatzes ein Monospace-Schriftformat verwendet (Beispiel: `Code`).

Da die API ressourcenorientiert ausgelegt ist, sind eine Reihe von Werten abrufbar, welche über HTTP-URLs eindeutig angesprochen werden. Dabei gibt es keine Vorgabe zur Darstellung. Zur Ausgabe wird derzeitig die JSON-Notation verwendet.

Die für diese Arbeit wesentlichen Ressourcen lauten:

- Station
- Timeseries
- CurrentMeasurement
- GaugeZero (Unterressource)

Die Ressourcen werden kurz beschrieben und anhand ihrer Attribute vorgestellt. Durch die Kombination der Ressourcen lassen sich unterschiedliche Ausgaben erzeugen.

Die Ressource `Station` macht Angaben zur Messstelle und enthält folgende Attribute:

`uuid`	Eindeutige unveränderliche ID
`number`	Pegelnummer
`shortname`	Pegelname (max. 40 Zeichen)
`longname`	Pegelname (max. 255 Zeichen)
`km`	Flusskilometer
`agency`	Wasserstraßen- und Schifffahrtsamt
`longitude`	Längengrad in WGS84 Dezimalnotation
`latitude`	Breitengrad in WGS84 Dezimalnotation
`water`	Angaben zum Gewässer

Tabelle 11: Attribute der Ressource Station (vgl. GDWS - Webservices, 2024).

Die Ressource wird über die folgende Anfrage angesprochen und liefert die aktuellen Messwerte aller Pegelmessstellen an den Wasserstraßen des Bundes:

https://www.pegelonline.wsv.de/webservices/rest-api/v2/stations.json

Um der enormen Masse an Informationen Grenzen zu setzen, wird in dieser Arbeit die Ausgabe auf ein Gewässer beschränkt. Durch das Anhängen des Parameters `waters=RHEIN` an die URL, werden nur die Ergebnisse der 33 Messstellen entlang des Rheins betrachtet.

Zusätzlich sollen die gemessenen Werte an den Messstellen ausgegeben werden. Dazu zählen Angaben zum Messverfahren, die unter dem Parameter `Timeseries` zusammengefasst werden. Der Parameter `Timeseries` ergänzt die Ressource `Station` um folgende Angaben:

`shortname`	Kurzbezeichnung
`longname`	Langbezeichnung
`unit`	Maßeinheit
`equidistance`	Abstand der Messwerte in Minuten.

Tabelle 12: Attribute der Ressource Timeseries (vgl. GDWS - Webservices, 2024).

Über den Parameter `CurrentMeasurement` lassen sich Angaben über den Abgleich der Messwerte mit bestimmten Wasserstands-Metriken abrufen. `CurrentMeasurement` fügt der Anfrage weitere Attribute hinzu:

`timestamp`	Zeitpunkt codiert im ISO_8601 Format.
`value`	Wert als Dezimalzahl in der Einheit, welche durch die `Timeseries` der Station vorgegeben ist.
`stateMnwMhw` und `stateNswHsw`	Diese Attribute sind nur bei Wasserständen vorhanden. Sie setzen den aktuellen Wasserstand entweder mit den mittleren niedrigsten Werten (MNW) und den mittleren höchsten Werten (MHW) in Beziehung (`stateMnwMhw`) oder mit dem höchsten Schifffahrtswasserstand (`stateNswHsw`). Die Attribute `stateMnwMhw` und `stateNswHsw` können verschiedene Werte annehmen, die jedoch hier nicht weiter vertieft werden.

Tabelle 13: Attribute der Ressource CurrentMeasurement (vgl. GDWS - Webservices, 2024).

Der tatsächlich gemessene Wert wird durch die Unterressource `GaugeZero` (Pegelnullpunkt) ausgegeben. Die nachfolgende Tabelle listet deren Attribute auf:

`unit`	Einheit des Pegelnullpunkts (in m über einem Normalhöhennull)
`value`	Höhe als Dezimalwert
`validFrom`	Beginn der Gültigkeit. ISO_8601 Datum.

Tabelle 14: Attribute der Unterressource GaugeZero (vgl. GDWS - Webservices, 2024).

Folgende URL liefert die gewünschten Angaben:

https://www.pegelonline.wsv.de/webservices/rest-api/v2/stations.json?waters=RHEIN&timeseries=W&includeTimeseries=true&includeCurrentMeasurement=true

Entsprechend einem quellenorientierten Datenprodukt, sollen diese Daten einmal eingelesen und mehreren Datennutzern zur Verfügung gestellt werden (vgl. Kap. 4.4.2).

Datennutzer / Zielsysteme

In dieser Arbeit sollen die Daten für einen Data Scientist aufbereitet. Dieser erwartet eine visualisierte Form der Pegelstände an den 33 Messstellen des Rhein-Verlaufs. Zudem sollen auch die Pegelnullpunkte (PNP) über Normalhöhennull (NHN) eingetragen werden, um den An- und Abstieg des Gewässers zu verdeutlichen.

Alternativ bestünde die Möglichkeit, die zugrunde liegenden Daten in anderer Form zu speichern oder auszugeben. So könnte bspw. ein Document Store (z. B. MongoDB) die JSON-Dateien entgegennehmen, aber auch die Speicherung in einer relationalen Datenbank (z. B. PostgreSQL) ist denkbar. Diese Fakten entsprechen dem Data Mesh Ansatz nach polyglotten Daten, d. h. einmal eingelesene Daten einer Vielzahl von Benutzern zur Verfügung zu stellen.

Anwendungs-Szenario

Für diese Arbeit soll folgendes Szenario mittels Implementierung dargestellt werden. Diese Implementierung wird im nachfolgenden Kapitel 5.3 näher erläutert.

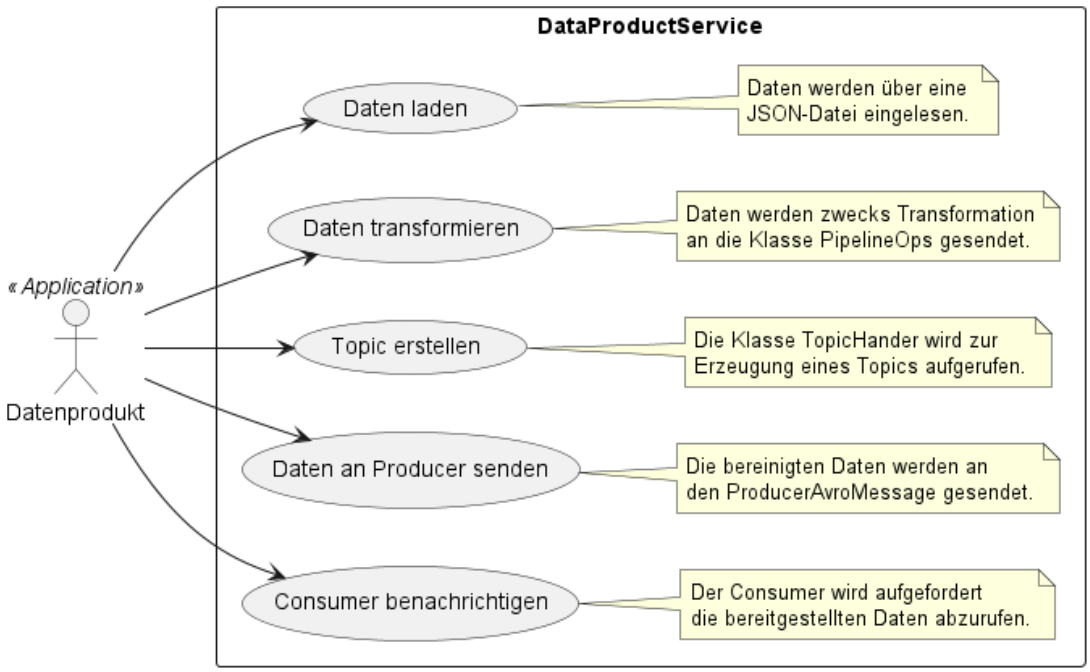

Abbildung 7: Anwendungsfall-Diagramm DataProductService.

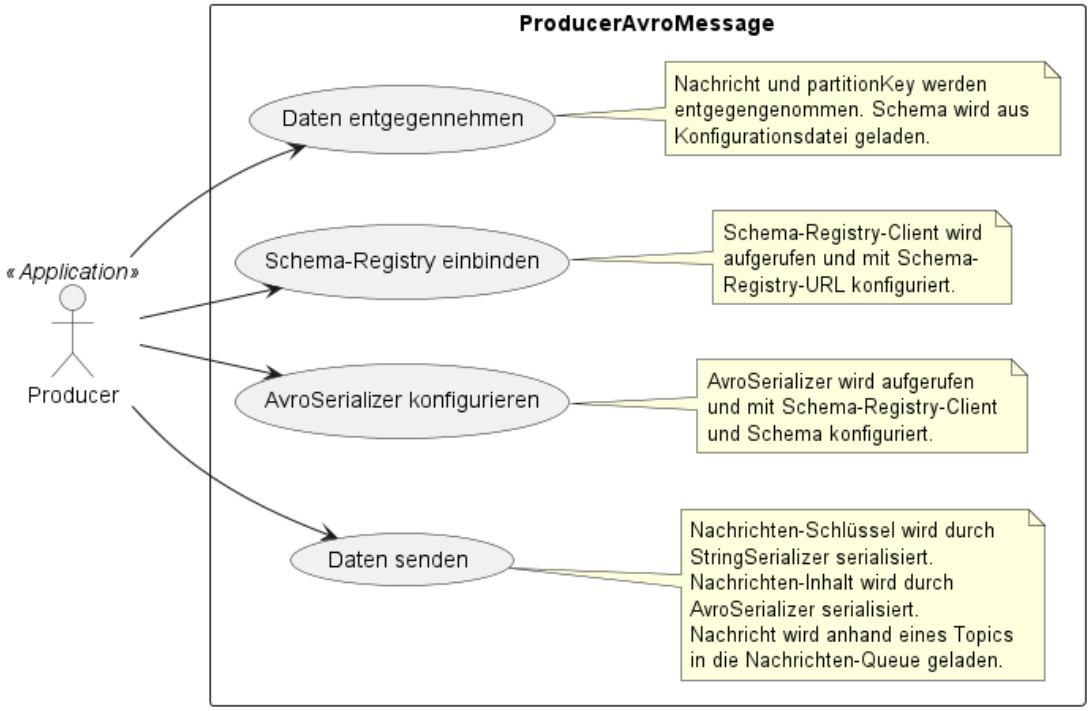

Abbildung 8: Anwendungsfall-Diagramm ProducerAvroMessage.

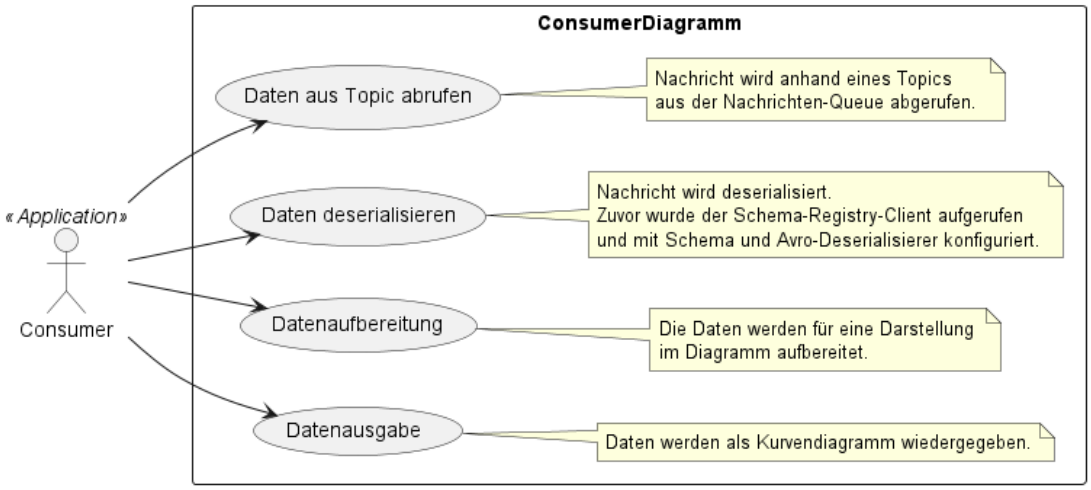

Abbildung 9: Anwendungsfall-Diagramm ConsumerDiagramm.

Datenqualität

Den vorherigen Kapiteln zu entnehmen, wird die Pipeline-Arbeit kontextbezogen innerhalb des Datenproduktes verrichtet. Da Data Mesh u.a. dem Service-Gedanken in den Vordergrund stellt (vgl. Kap. 4.3), wird an dieser Stelle der Qualität der Daten Aufmerksamkeit gewidmet.

Datenqualität basiert nach den Ausführungen von S. McGregor auf den Inhalten der Begriffe *Data Fitness* und *Datenintegrität* (vgl. McGregor, 2022, S. 71-89).

Data Fitness trifft Aussagen zu den folgenden Eigenschaften der Daten:

- Gültigkeit (*Validity*),
- Zuverlässigkeit (*Reliability*) und
- Repräsentativität (*Representativeness*).

Data Fitness behandelt die Frage, inwieweit die Daten für den geplanten Einsatz geeignet sind. Mittels der genannten Eigenschaften, wird die Datentauglichkeit untersucht und entsprechend bewertet. Zur Bewertung ist es notwendig sich im Vorfeld zu überlegen, welchem Zweck die Daten dienen.

Datenintegrität konzentriert sich auf die Frage, ob die Daten die geplanten Analysen unterstützen können. Dabei gibt es verschiedene Aspekte, die zur Bewertung der Integrität zu betrachten sind. Es ist nicht davon auszugehen, dass die Daten bereits alle benötigten Eigenschaften besitzen. Durch die Bewertung werden jedoch Schritte ersichtlich, die im Rahmen der Datenbereinigung durchzuführen sind.

Folgende Merkmale (nach Relevanz) werden betrachtet (vgl. McGregor, 2022, S. 80)[21]:

- Notwendig
    - Datenherkunft (*Of known provenance*)
    - Metadaten (*Well-annotated*)
- Wichtig
    - Zeitgemäß (*Timely*)
    - Vollständig (*Complete*)
    - Hohes Datenvolumen (*High Volume*)
    - Mehrere Darstellungen (*Multivariate*)
    - Atomar (*Atomic*)
- Erreichbar
    - Konsistent (*Consistent*)
    - Fehlerfrei (*Clear*)
    - Mehrdimensional strukturiert (*Dimensionally structured*)

Das Datenprodukt zeichnet sich durch die Bereitstellung qualitativ hochwertiger Daten aus. Die Bewertung der Daten liegt im Aufgabenbereich der im Datenprodukt angesiedelten Rollen (bspw. Data Product Owner, Data Product Developer). Anhand dieser Prüfungen können entsprechende Schritte zur Implementierung eines Bereinigungsprozesses unternommen werden.

[21] Susan McGregor nimmt in ihren Ausführungen Bezug auf ein Werk von Stephen Few, *Now You See It: Simple Visualization Techniques for Quantitative Analysis* (Analytics Press).

Der vorliegende Anwendungsfall aus dem Bereich des gewässerkundlichen Informationssystems PEGELONLINE beantwortet diese Fragestellungen der Datenintegrität wie folgt:

Notwendig, jedoch nicht ausreichend:

- **Of known Pedigree:** Die Frage nach der Datenherkunft und ob diese zuverlässig ist, lässt sich bejahen. Die Daten werden von offiziellen Messstellen erhoben, daher ist anzunehmen, dass deren Bertreiber zum Zwecke ihrer Eignung einer vorherigen Prüfung unterzogen wurden.
- **Well-annotated:** Die Daten sind gut dokumentiert. Dies geht zwar nicht aus dem JSON-Dokument hervor, welches die Daten ausliefert, lässt sich jedoch dem Internetangebot PEGELONLINE im Bereich der Beschreibung der REST-API gut entnehmen. Metadaten lassen sich nicht entnehmen.

Wichtig:

- **Timely:** Die Aktualität der Daten sollte gegeben sein. Durch die Angabe der Messabstände und die kontinuierliche Messung, kann man von einer zeitgemäßen Datenlieferung ausgehen.
- **Complete**: Der Frage nach der Vollständigkeit der Daten kann zugestimmt werden. Alle Werte die erhoben werden, sind weitestgehend vorhanden. Zudem sind alle relevanten Gewässer und Schifffahrtsstraßen Deutschlands aufgeführt.
- **High-volume:** Das Datenvolumen ist recht umfangreich und hoch, da es von einer Vielzahl von Messstellen erhoben wird. Jede Messstelle liefert die verfügbaren Daten nach gleichen Parametern.
- **Multivariate:** Wenn Daten multivariant sind, weisen sie mehrere Attribute oder Merkmale auf, die mit dem Datensatz verbunden sind. Für den Anwendungsfall bedeutet dies, dass die Bereitstellung im JSON-Format noch eine Vielzahl von Darstellungsmöglichkeiten zulässt. Alternative Darstellungsformen im Bildformat (.png) sind möglich, jedoch keine Video- oder Audioformate.
- **Atomar:** Dem Wunsch nach Unveränderlichkeit und Einheitlichkeit der Daten wird nachgekommen. Die Daten werden nach Ihrer Erhebung nicht mehr verändert und liegen eine gewisse Zeitspanne (30 Tage) vor. Eine nachträgliche Modifikation ist nicht vorgesehen.

Erreichbar:

- **Consistence:** Das zugrundeliegende Datenschema sorgt dafür, dass die Daten in der gewünschten Form konsistent erfasst werden.
- **Clear:** Die Daten sind klar und verständlich. Dem JSON-Format verdankend, lässt sich die Bedeutung der Daten, durch die Strukturierung und Namensgebung der Parameter, leicht erschließen. Zudem wird auf die Dokumentation der REST-API auf PEGELONLINE verwiesen, die vorhandene Unklarheiten klären kann (vgl. GDWS - Webservices, 2024).
- **Dimensional:** Die Strukturierung der Daten beinhaltet eine Vielzahl von Dimensionen und fachspezifischen Schachtellungen. Das wird insbesondere durch das JSON-Dokument verdeutlicht. Hier gibt es tiefere Schachtelungen und Array-Elemente um die unterschiedlichen Unterressourcen darzustellen.

Zusammenfassend lässt sich sagen, dass die Prüfung der Datenqualität eine wichtige Vorarbeit leistet, um weitere Schritte des Pipeline-Prozesses zu initiieren. Anhand dieser Analyse können Entscheidungen bzgl. der Datenbereinigung (engl. *Data Cleaning*) und der Datenerweiterung (engl. *Data Augmentation*), d. h. des Umgangs mit fehlenden Werten, getroffen werden.

5.3 Lösung auf Basis der Data Mesh Charakteristiken

Nachdem in den vorangegangenen Kapiteln eine geeignete Technologiegrundlage (vgl. Kap. 4.1) sowie ein passender Anwendungsfall (vgl. Kap. 4.2) geschildert wurden, soll an dieser Stelle eine Zusammenführung dieser Bausteine unter dem Einfluss der Data Mesh-Prinzipien geschaffen werden. Die nachfolgende Grafik veranschaulicht, wie sich die Streaming-Technologie in die Strukturkomponenten des Architekturquantums einordnen lässt.

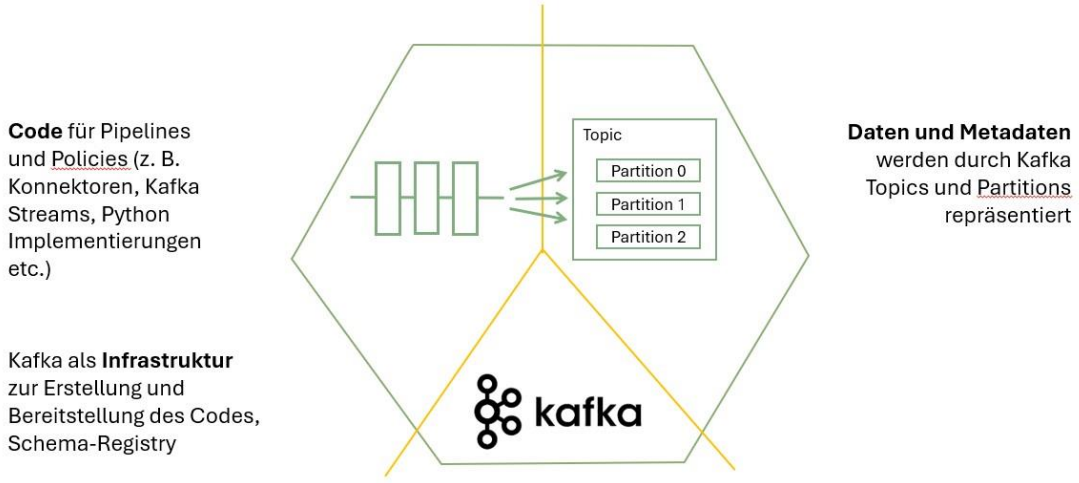

Abbildung 10: Einordnung einer Data-Streaming-Technologie im Architekturquantum.

An dieser Stelle soll noch einmal verdeutlicht werden, dass das Architekturquantum die einfachste Form des Datenproduktes darstellt. Es wird im Wesentlichen in drei Bereiche unterteilt

Der erste Bereich kennzeichnet die Abhängigkeit zur Plattform und beinhaltet Kafka als **Infrastruktur** sowie eine Schema-Registry. Die Plattformabhängigkeit ist explizit gewünscht, da sie die grundlegenden Mechanismen der Speicherung von Daten zur Verfügung stellt.

Der zweite Bereich beinhaltet verschiedene **Codefragmente** für den Pipeline-Workflow. Dort lassen sich z. B. programmierte Regeln, Plugins (Konnektoren, Kafka Streams, …) sowie die Python-Implementierungen einordnen. Entsprechend der Vorgaben von Data Mesh wird auf externe Pipelines verzichtet, daher findet sich diese im Code-Teil des Datenproduktes wieder. Die Pipeline ist kontextspezifisch und wird von Datenprodukt vollständig gekapselt.

Zur Implementierung der Pipeline kommt die Skriptsprache Python zum Einsatz. Die Wahl fällt auf Python, da diese Sprache für den Bereich der Datenanalyse und -transformation eine Vielzahl von Funktionen und Bibliotheken zur Verfügung stellt. Zudem wird sie im Data Science Umfeld gerne aufgrund ihrer zahlreichen Darstellungsmöglichkeiten genutzt.

Folgende Bibliotheken kommen zum Einsatz:

- pandas (Data Science, Datenanalyse)
- kafka-python (Kommunikation mit Apache Kafka)
- yaml, requests, json (Konfiguration, Datenaustausch)
- mathplotlib (Visualisierung)
- schema-registry-client (Kommunikation mit der Confluent Schema-Registry)

Der dritte Bereich enthält **Daten und Metadaten** und befasst sich mit der Datenhaltung und Speicherung, daher wird das Kafka-Topic dort eingeordnet. Das Topic ist das Kernelement des Datenproduktes, da es den wesentlichen Zweck des Datenproduktes wiedergibt.

Mithilfe dieser drei Bereiche, lässt sich das Datenprodukt zielführend mit den gewünschten Komponenten zusammensetzen.

Abgrenzung

Es wird darauf hingewiesen, dass diese Arbeit sich nicht zum Ziel gesetzt hat ein vollständiges Datenprodukt zu implementieren, wie es im Data Mesh vorgesehen ist. Hierzu gehören wesentlich mehr Aspekte, wie bspw. Policy-als-Code, automatisches Testen, die Gestaltung der Schnittstellen-Verträge. All diese Bereiche sind selbst bei (Dehghani, 2023) noch nicht definiert und bilden Aufgabenbereiche für zukünftige Forschungen.

Zudem verfügt jedes Datenprodukt über einen sog. *Sidecar-Prozess*, dessen Ausgestaltung ebenfalls noch völlig offen ist. Der Sidecar-Prozess wurde bisher nicht thematisiert, da er keinen wesentlichen Einfluss auf die Pipeline ausübt. Vielmehr handelt es sich um eine Art „Steuerelement“, welches domänenagnostisch ausgelegt ist und u.a. den Zugriff auf das Datenprodukt, die globalen Governance-Vorgaben und das Suchen und Auffinden des Datenproduktes steuert. Da die Pipeline innerhalb des Datenproduktes verortet ist, soll lediglich dieser Teil im Ansatz ausgearbeitet werden.

Lösungsansatz

Der Lösungsansatz für eine Pipeline nach dem Data Mesh-Prinzip orientiert sich im Wesentlichen an der Entgegennahme, Verarbeitung, Bereitstellung sowie dem Abruf der Daten durch den Consumer. Die nachfolgende Grafik illustriert das Vorgehen, wie diese Schritte auf herkömmlichen Wege mittels Kafka umgesetzt werden.

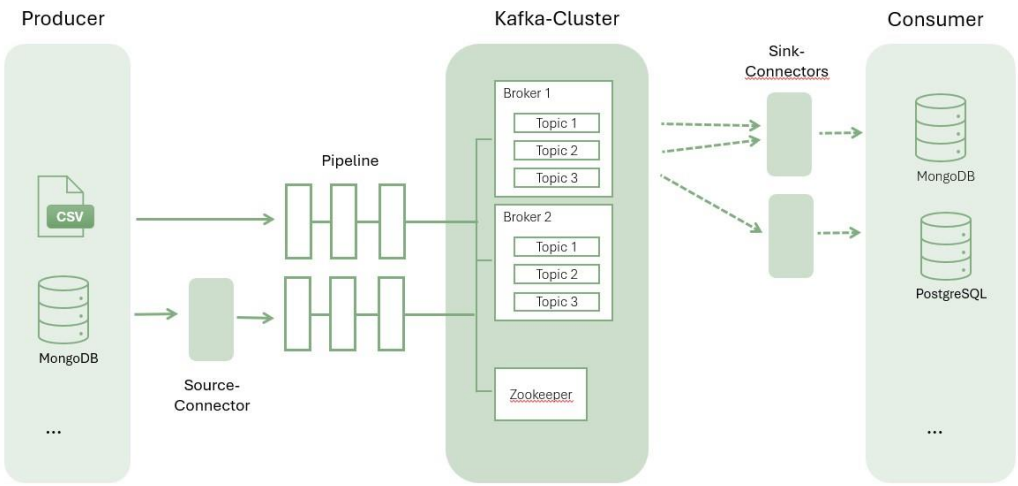

Abbildung 11: Bisherige Kafka-Lösung zum Umgang der Pipeline-Arbeit.

Um den Nachrichtenversand von der Anwendungslogik zu entkoppeln, kommt die Daten-Streaming-Plattform Apache Kafka zum Einsatz.

Der Producer hat die Aufgabe, Daten entgegenzunehmen und an ein Kafka Topic zu senden. Die Daten können in unterschiedlicher Form vorliegen. So können bspw. Daten aus CSV-Dateien eingelesen, aus einer externen Quelle kontinuierlich abgerufen oder auch aus einer Datenbank bezogen werden.

Die Daten liegen in unterschiedlichen Schemata vor. Eine große Herausforderung liegt darin, die Daten derart aufzubereiten, dass sie von einer Vielzahl von Datenkonsumenten genutzt werden können. Zu diesem Zwecke wird die Syntax von der Semantik getrennt, d.h. für die Darstellungsform wird ein Schema definiert. So können die Daten losgelöst von ihrer

Darstellung in einem schlanken Binärdatenformat zur Ablage in ein Topic transportiert werden. Der Konsument kann sich bei Bedarf die gewünschten Daten aus einer ihm zugewiesenen Partition eines bestimmten Topics abrufen. Um das gewünschte Zielformat zu erhalten, erfragt er bei der Schema-Registry anhand einer mitgesendeten Schema-ID das gewünschte Schema. Nach der Deserialisierung der Daten mit diesem Schema, erhält er die gewünschten Daten zur weiteren Verarbeitung.

Diese Beschreibung entspricht der von Kafka zugrunde gelegten Vorgehensweise. Die nachfolgende Grafik zeigt auf, wie dieser Weg für Data Mesh nutzbar gemacht werden kann.

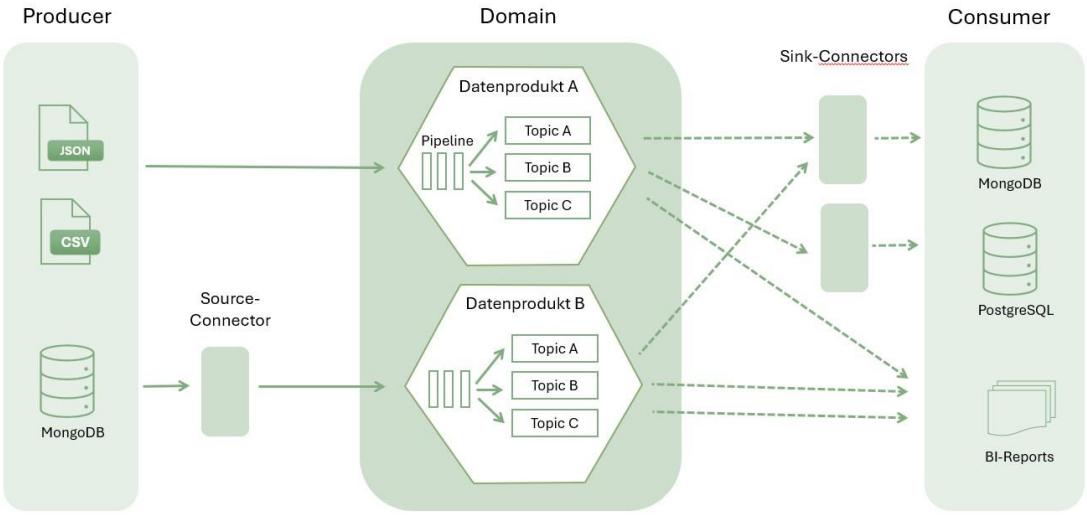

Abbildung 12: Nutzung der Streaming-Technologie zum Aufbau von Data Mesh.

In diesem Lösungsansatz werden die Topics den Datenprodukten zugeordnet. Da die Pipeline kontextspezifisch ausgelegt wird, wird sie in den Datenprodukten platziert. Datenprodukte wiederum, gehören fachlichen Domänen an.

Diese Grafik zeigt ein quellenorientiertes Datenprodukt, daher werden ein Dateneingang und mehrere Datenausgängen dargestellt. In diesem Beispiel entstammen die Objekte Producer, Consumer, Topic und Partition dem Kafka-Kontext. Die interne Verteilung der Topics auf die Kafka-Broker wird in dieser Arbeit nicht thematisiert, da sie nicht im Fokus der Pipeline steht. Hier sei nur erwähnt, dass die Broker aufgrund von Last- und Performance-Kriterien über die Verteilung der Daten entscheiden, wobei die Verwaltung der Broker in einem Kafka-Cluster in den Aufgabenbereich des Cluster-Managers Zookeeper fällt.

Die Implementierung der Datenquellen und -nutzer sowie die Ausgestaltung des Datenproduktes, der Domänen und Pipelines, fällt in den Aufgabenbereich eines Data Product Developers. In dieser Arbeit sollen diese Aufgaben in einer Python-Anwendung zusammengefasst werden.

Übertragung der Lösungsansatzes auf den Anwendungsfall

Der Lösungsansatz wird als skalierbare und containerisierte Anwendung implementiert. Er besteht aus vier eigenständigen Docker-Containern, die jeweils eine bestimmte Aufgabe erfüllen. Durch die Verwendung von Docker wird die Anwendung plattformunabhängig und kann in einfacher Form an verschiedene Systeme ausgeliefert werden.

Der Aufbau gestaltet sich wie folgt:

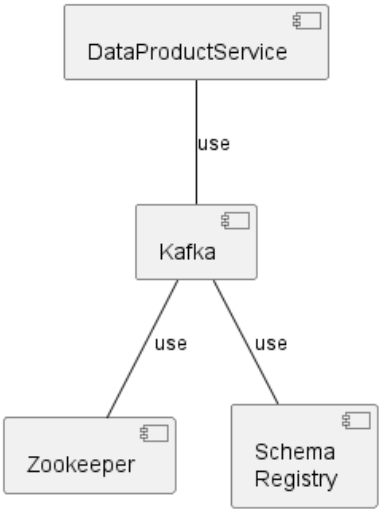

Abbildung 13: Komponenten-Diagramm mit den Containern des Anwendungsfalls.

Der **Container DataProductService** realisiert die Anwendung, die die Services des Datenproduktes zur Verfügung stellt. Die Anwendung steuert die Kommunikation mit den Datenquellen und -nutzern und veranlasst den Pipeline-Prozess. Die Pipeline wird in einer ihrer Klassen implementiert.

Der **Container Kafka** beinhaltet die zugrundeliegende Data-Streaming-Plattform und stellt die relevanten Komponenten Producer, Consumer, Topic und Partition bereit. Daten werden in sog. *Message Queues* (engl. Nachrichten-Warteschlangen) verwaltet.

Der **Container Schema-Registry** dient als zentrales Repository, welches Schemata für strukturierte Datenformate wie Avro, JSON Schema etc. verwaltet und speichert. Die Schema-Registry wird von den Kafka-Producern und -Consumern verwendet um Schemata abzurufen, die während des Nachrichtenversands nicht übertragen werden.

Der **Container Zookeeper** ist ein zentralisierter Service zur Koordination verteilter Systeme. Er fungiert als Cluster-Manager und verwaltet die Konfigurationen der Broker. Er stellt Mechanismen für Synchronisation, Gruppenverwaltung und Hochverfügbarkeit zur Verfügung.

Rollen der Anwendung

Nachfolgend wird die Anwendung DataProductService näher betrachtet. Dazu werden folgende Rollen definiert:

- Nachrichtenerzeuger (`ProducerAvroMessage`, abgeleitet von der Klasse Producer aus dem Package kafka-python)
- Nachrichtenkonsumenten (`ConsumerDiagramm`, abgeleitet von der Klasse Consumer aus dem Package kafka-python)
- Nachricht-Kontext (Klasse `Topic` aus dem Package kafka-python)
- Hilfsklasse zur Erzeugung und Partitionierung von verschiedenem Nachrichten-Kontext (`TopicHandler`)
- Komponente zur Verwaltung von Datenschemata (`Schema-Registry`, aus dem Package confluent-schema-registry)
- Pipeline, die innerhalb des Datenproduktes gekapselt wird (`PipelineOps`)
- Datenprodukt-Service, der alle Komponenten steuert (`DataProductService`)

Das nachfolgende Diagramm zeigt die Beziehungen zwischen den einzelnen Komponenten auf. Im Anschluss folgt eine Beschreibung der Klassen.

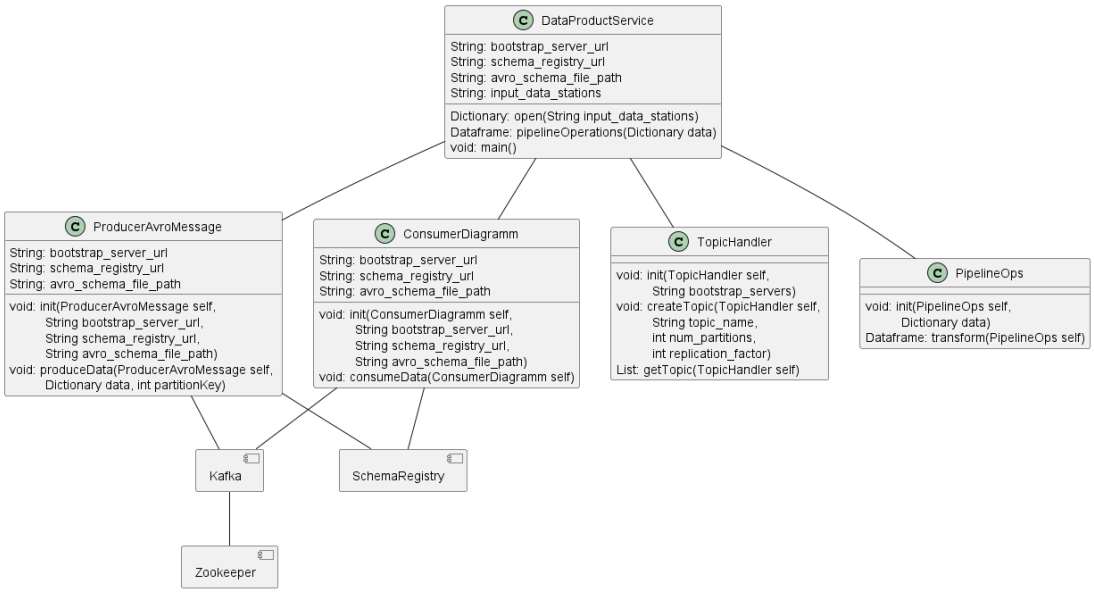

Abbildung 14: Klassendiagramm des Anwendungsfalls.

Wie bereits erwähnt, wird in dieser Arbeit kein vollständiges Datenprodukt geschaffen. Vielmehr soll der Weg von der Erzeugung der Daten bis hin zur Bereitstellung aufgezeigt werden. Aus diesem Grund empfiehlt sich der Aufbau einer Klasse, die den grundlegenden Service realisiert und alle beteiligten Komponenten steuert. Die Klasse `DataProductService` wird zu diesem Zweck implementiert.

`DataProductService` kapselt folgende Schritte:

1. Einlesen von Daten aus einer externen Quelle
2. Erteilen eines Auftrags zur Transformation der Daten
3. Erteilen eines Auftrags zur Erzeugung eines Nachrichten-Kontexts (Topic)
4. Erteilen eines Auftrags zur Bereitstellung der transformierten Daten in einem Topic
5. Aufruf eines Konsumenten, der die Daten abruft[22]

Einlesen der Daten

Als Datenquelle dient der Webservice von PEGELONLINE. Dieser ist als REST-API abrufbar und stellt die Daten der Pegelmessstellen des Rheins im JSON-Format zur Verfügung. Die Werte aktualisieren sich nach einem vorgegebenem Zeitraum, der derzeitig 15 Minuten umfasst. An dieser Stelle können auch andere Datenquellen ausgelesen werden, wie bspw. ein Document Store (z. B. MongoDB) oder eine relationale DB (z. B. PostgreSQL).

Anmerkung: Für die Implementierung des Lösungsansatzes werden die Daten nicht per HTTP-Request von der Website angefordert. In der Demo-Implementierung bezieht die Anwendung ihre Daten aus einer JSON-Datei, die in einem Verzeichnis der Anwendung abgelegt wird. Das Laden der Daten erfolgt somit aus einem lokalen Ablageort der Anwendung.

Transformation der Daten

Die eigentliche Transformation der Daten erfolgt in der Klasse `PipelineOps`. Diese wird vom `DataProductService` instanziiert und bekommt als Parameter den eingelesenen Datensatz im JSON-Format als Dictionary übermittelt. Über den Aufruf der Funktion `transform()`, wird der Pipeline-Prozess gestartet. Mithilfe der importierten Python-Bibliothek pandas, werden die JSON-Records in entsprechenden Dataframe-Objekten gespeichert, die den Zugriff auf einzelne Daten sowie deren Manipulation effizient unterstützen.

Die vorhergehende Analyse der Datenquelle wurde bereits in Kap. 5.2 beschrieben. Die Klasse `PipelineOps` befasst sich nun mit den daraus resultierenden Maßnahmen zur Datenbereinigung. Dazu zählt u.a.

- die Aufteilung der Daten des geladenen Dictionaries in zwei Dataframe-Objekte, da dieses in sich stark geschaltet ist,
- das Entfernen nicht benötigter Spalten, wie z. B. die Angabe von `longname` und `shortname` des Gewässer- oder Messstellennamens,

[22] Der Aufruf des Datenkonsumenten über das Datenprodukt dient nur zu Demozwecken. In der Regel wird der Konsument nicht unaufgefordert benachrichtigt, wenn er sich nicht für ein bestimmtes Topic registriert hat.

- die Umbenennung von Spaltennamen, um ein besseres Verständnis für den Spaltenwert zu erhalten,
- die Formatierung falsch dargestellter Zeichen (z. B. Sonderzeichen, Umlaute) sowie
- das Einfügen zusätzlicher Spalten, die als Ergebnis die Höhe des Wasserstands über dem Meeresspiegel angeben, welche in den Daten nicht erfasst wird[23].

Erzeugung eines Nachrichten-Kontexts

Die Klasse `TopicHandler` ist eine Hilfsklasse, die vom `DataProductService` aufgerufen wird. Beim Aufruf erhält sie die URL des Bootstrap-Servers als Parameter übergeben. Die Hilfsklasse erzeugt durch den Aufruf der Methode `createTopic()` ein entsprechendes Topic. Als Parameter übergibt sie den Namen des Topics, die gewünschte Anzahl an Partitionen sowie den Replikations-Faktor. Das Topic wird als Listenobjekt angelegt, so dass mit einem Aufruf von `createTopic()` auch gleich mehrere Topics angelegt werden können.

Senden und Bereitstellung der Daten

Die von der Klasse `PipelineOps` über die Methode `pipelineOperations()` gelieferten Daten werden im Dataframe-Format zurückgegeben. Zur weiteren Verwendung, werden sie in ein Dictionary-Format konvertiert. Das Dictionary ist eine Ausgangslage für die Klasse `ProducerAvroMessage`. Die Aufgabe dieser Klasse ist es, Daten anhand eines konkreten Topics und einer Kennzeichnung der relevanten Partition (`partitionKey`) an die entsprechende Partition des genannten Topics zu senden.

`ProducerAvroMessage` wird von `DataProductService` instanziiert um Daten für eine bestimmte Konsumentengruppe zur Verfügung zu stellen. Die Klasse wird beim Aufruf mit der Bootstrap-Server-URL, der Schema-Registry-URL sowie dem Datenschema im Avro-Format konfiguriert, die als Parameter beim Klassenaufruf übergeben werden. Deren Methode `produceData()` wird mit dem Dictionary und dem `partitionKey` als Werte vom `DataProductService` aufgerufen.

Jede Nachricht, die der Producer ans Topic sendet, besteht aus einem Schlüssel-Wert-Paar (engl. *Key-value)*. Während der Schlüssel im String-Format serialisiert wird, erfolgt die Serialisierung des Nachrichteninhaltes im Avro-Format. Aus diesem Grund werden entsprechende String- und Avro-Serializer verwendet.

[23] Die Höhe des Wasserstandes über dem Pegelnullpunkt (PNP) wird in cm gemessen. Der PNP ist die Nullstelle an der Pegellatte. Er trifft keine Aussage über die lokale Wassertiefe oder den Höhenbezug des Gewässers zum umliegenden Gelände. Der PNP ist ein fixer Wert, der in Metern über Normalhöhennull (m. ü. NHN) festgelegt wird. Der Wert *m. ü. NHN* kann gleichgesetzt werden mit der Angabe von *Metern über dem Meeresspiegel*. Um die Höhe des Wasserspiegels über dem Meeresspiegel zu erhalten, addiert man den Wasserstand am Pegel mit dem Wert des PNP (GDWS, 2024).

Der Avro-Serializer benötigt zur Konfiguration die Schema-Registry-URL sowie das verwendete Schema, welches zuvor über die Avro-Datei eingelesen und in einen String konvertiert wurde. Es folgt der Aufruf der Methode `produceData()`, welche den eigentlichen Nachrichtenversand an das Topic anstößt. Dies erfolgt unter Verwendung der Parameter `topic`, `key`, `value` und `callback_function` (optional). Sobald die Daten versendet wurden, wird der Producer mittels des Methodenaufrufs `flush()` geleert.

Hinweis

Die Besonderheit dieser Klasse ist demnach die Ablage von Daten in einem Topic. Durch den `partitionKey` lässt sich eine konkrete Partition des Topics ansteuern. Diese Partition dient dem Datenkonsumenten als Bezugsquelle, von der er erwarten kann, die für ihn und seine Bedürfnisse bereitgestellten Daten dort abzurufen.

So richtet sich der `key` mit dem Wert `text,` an den Konsumenten `ConsumerDiagramm`, der Wert `ds` an den Konsumenten `ConsumerDS`[24] und der Wert `rdb` an den Konsumenten `ConsumerRDB`[25].

Anmerkung: In dieser Arbeit wird die Variante `key="text"` ausgearbeitet und dargestellt.

Abruf der Daten

Die Klasse `ConsumerDiagramm` stellt den Datennutzer dar. Datennutzer können verschiedene Ausprägungen und spezielle Anforderungen an die Daten haben. So können die Daten z. B. unverändert als Dokument in einem Document Store oder auch in Form von Datensätzen in einer relationalen Datenbank abgelegt werden. In dieser Arbeit verfolgt der Datennutzer das Ziel, die Daten bei Bedarf abzurufen und visuell als Kurvendiagramm darzustellen.

Wie schon die Klasse `ProducerAvroMessage`, wird auch die Klasse `ConsumerDiagramm` von `DataProductService` instanziiert. `ConsumerDiagramm` wird mit den Parametern Bootstrap-Server-URL, der Schema-Registry-URL sowie dem Avro-Schema-Dateipfad konfiguriert. Die Übergabe des Avro-Schemas ist notwendig, um die empfangenen Daten mit dem entsprechenden Avro-Schema zu deserialisieren.

Durch den Aufruf der Methode `consumeData()`, erfolgt eine Rückführung der zuvor durchgeführten Serialisierung. Auch hier wird das Avro-Schema geladen, ein Schema-Registry-Client konfiguriert und ein Avro-Deserializer mit den entsprechenden Werten instanziiert. Im Anschluss kann der Consumer seinen Datenabruf durchführen.

[24] *DS* steht für *Document Store*, einem Datenbanktyp für Dokumente mit unterschiedlichen Datenschemata.
[25] *RDB* steht für *Relationale Datenbank*, welche Tabellen-organisiert mit festem Datenschema arbeitet.

Der Consumer wird einer *Consumer-Group*[26] zugeordnet und mit einer Angabe versehen, welches Daten-Offset, d. h. welche Position er in der Nachrichten-Queue, er ansprechen soll. Der Wert `earliest` veranlasst ihn, die Daten-Queue von Anfang an zu lesen, während über den Wert `latest` der zuletzt eingetragen Wert angesprochen wird.

Der Consumer kann sich nun für ein Topic registrieren. Dazu ruft er die Methode `subscribe()` auf und belegt sie mit dem Topic-Namen als Parameter. Im Anschluss liest er die Daten in eine While-Schleife aus. Dabei wird der Nachrichteninhalt durch den Avro-Deserializer in das Zielformat übertragen. Da die Daten als Dictionary entgegen genommen wurden, erzeugt der Avro-Deserializer durch den Aufruf von `dict_to_WaterMeasurement()` ein neues Objekt `WaterMeasurement`, welches mittels der Methode `visualization()` ein entsprechendes Kurvendiagramm ausgibt. Nachdem alle Daten eingelesen wurden, wird der Lesevorgang des Consumers mittels der Methode `close()` beendet.

Die folgende Grafik stellt das erstelle Kurvendiagramm dar.

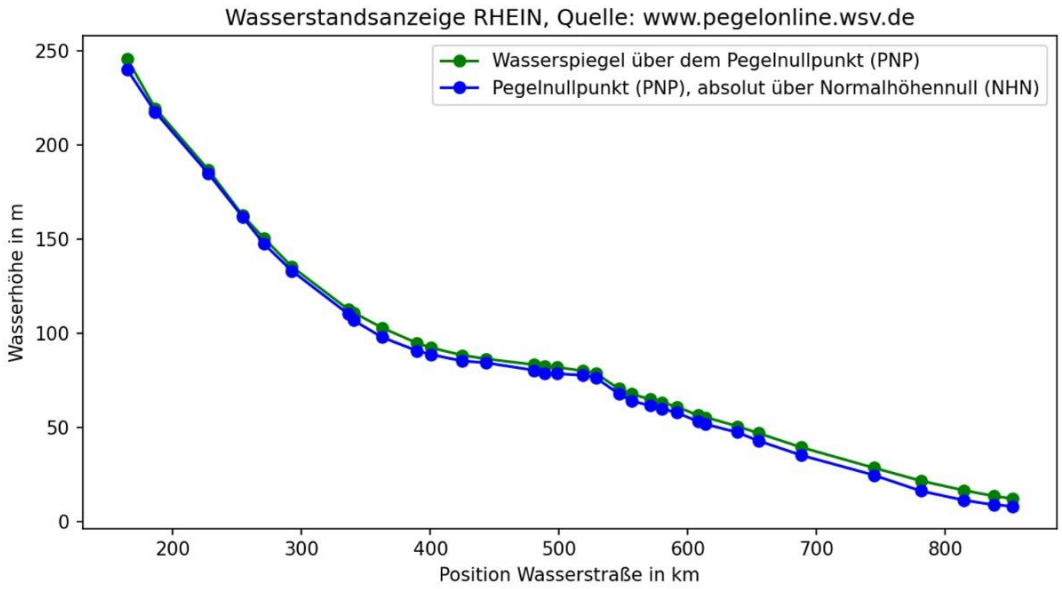

Abbildung 15: Ausgabe der Klasse ConsumerDiagramm.

[26] Die *ConsumerGroup* erhöht den Rang der Parallelisierung, da die Datensätze in den Partitionen eines Topics auf die Consumer einer Gruppe verteilt werden. Bei nur einem Consumer, darf dieser die Nachrichten aller Partitionen eines Topics nacheinander verarbeiten. Im Rahmen dieser Arbeit spricht ein Consumer explizit eine Partition mit entsprechendem *partitionKey* an, da er dort die für ihn aufbereiteten Daten erwartet.

Der Ablauf sowie die Aufruf-Reihenfolge werden im folgenden Sequenzdiagramm dargestellt.

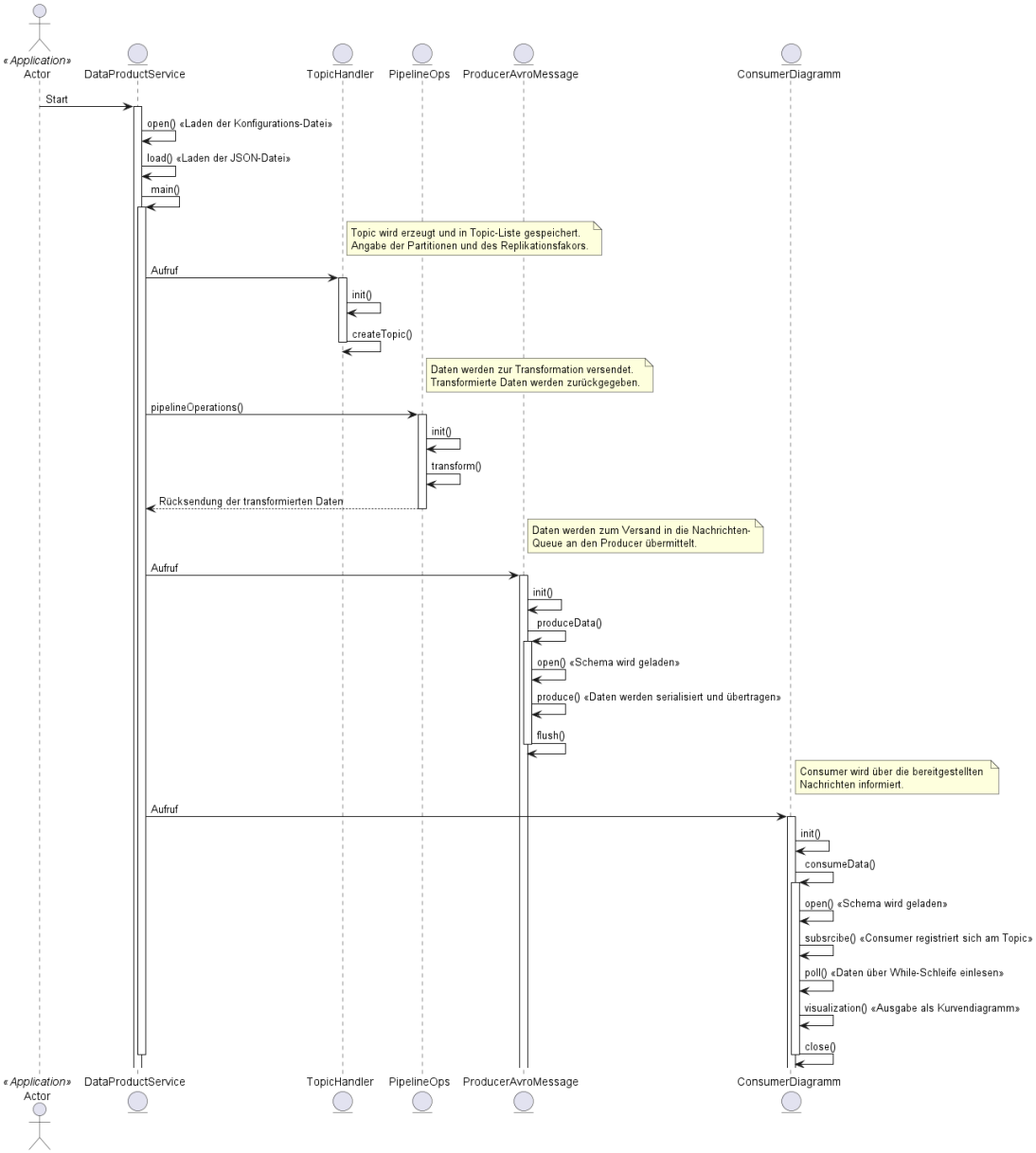

Abbildung 16: Sequenzdiagramm des Lösungsansatzes.

Wie bereits erwähnt, können alternative Datennutzer zum Einsatz kommen. Neben verschiedenen Datenbanktypen, können die Daten auch in andere Systeme einfließen. Solche Systeme können bspw. für Monitoring[27] und Observability[28] genutzt werden. Auch die Weitergabe an andere Datenprodukte ist möglich, wird in dieser Arbeit jedoch nicht weiter vertieft.

[27] *Monitoring* (dt. Überwachung) bezeichnet das Beobachten der Leistung eines Systems über einen Zeitraum.
[28] *Observability* ist ein Prozess in modernen Anwendungen, der Monitoring-Tools zum Erkennung von Störungen einsetzt. Ziel ist die frühzeitige Identifikation potentieller Probleme. Hierzu beobachtet der Prozess die Inputs und Outputs der Anwendungen.

6 Bewertung des Lösungsansatzes

Die in Kapitel 4.4.3 zusammengefassten Charakteristiken dienen als Grundlage zur Bewertung des in Kapitel 5.3 vorgestellten Lösungsansatzes. Im Rahmen des Lösungsansatzes wurde eine Data Engineering Pipeline nach dem Data Mesh-Prinzip implementiert.

Für jede der fünf Charakteristik-Gruppen, sollen folgende Fragen beantwortet wurden:

1. Lassen sich die Charakteristiken umsetzen?
2. Welche Herausforderungen bestehen?
3. Welche Empfehlungen lassen sich ableiten?

Die Antworten sollen dazu beitragen, ein Gesamtergebnis zu erarbeiten.

C1: Einordnung der Pipeline

Es ist ohne weiteres möglich, die Data Engineering Pipeline im **Datenprodukt** (C1-2) zu platzieren. Durch die Definition des Architekturquantums ist dafür explizit ein Platz ausgewiesen worden. Durch die Anforderung, einer möglichst wenig Programm-Code umfassenden Pipeline, wurde in der Implementierung eine simple Form der Pipeline-Umsetzung gewählt.

Herausforderung

Eine Herausforderung kann die Kapselung der Pipeline im Datenprodukt darstellen. Durch diese Isolierung ist es schwer möglich, Programmcode nachhaltig zu implementieren. Durch den starken Kontextbezug, ist die Pipeline ausschließlich dem Datenprodukt zugeordnet, womit sie auch schwer zu prüfen und zu kontrollieren ist. Die Beantwortung der folgenden Fragen bleibt offen:

- Ist die Pipeline zeitgemäß implementiert?
- Gibt es Schwachstellen (z. B. Sicherheitslücken, Performance-Engpässe), die in anderen Pipelines bereits identifiziert wurden?
- Gibt es aufgrund der starken Isolation dennoch einen stetigen Wissenstransfer?
- Wie wird nachgehalten, dass manche Arbeiten an mehreren Stellen neu implementiert wurden?
- Gibt es einen Hinweis darüber, ob alle Möglichkeiten ausgeschöpft wurden?
- Werden Pipeline-Tools in mehreren Datenprodukten erneut betrieben (Vermeidung von vielfachen Lizenzkosten)?
- Wie lässt sich eine Mandantentrennung umsetzen?

Empfehlung

Trotz der vielen Herausforderungen, gibt es auch Lösungsansätze, die Hilfestellungen bieten können, um die Nachteile der Isolation in den Griff zu bekommen. So hat sich z. B. aus dem DevOps[29]-Kontext heraus mit dem *Site Reliability Engineer* eine Rolle etabliert, die ein Bindeglied zwischen Entwicklung, Betrieb und Fachlichkeit darstellt. Demzufolge könnte der SRE die fehlenden Anregungen, Kontrollen und den Wissenstransfer hinsichtlich der Pipeline, über die Datenproduktgrenzen hinweg, bereitstellen und somit das Datenprodukt-Team um domänenübergreifende Informationen und Kenntnisse bereichern.

Diese Rolle könnte auch Ansatz, nach „mehr Generalisten, weniger Spezialisten“ und dem damit befürchteten Fach-Wissensverlust, entgegenwirken.

Die Gefahr des autonomen Betriebs von Pipeline-Tools in jedem Datenprodukt und den damit verbundenen Aufwänden (Lizenzkosten, Patch-Management etc.), lässt sich mittels der **Self-Serve Data Platform** (C1-4) lösen. In diesem Fall wird der Betrieb einer Anwendung zentral vorgehalten und jedes Datenprodukt bekommt über entsprechende Schnittstellen Zugriff auf die zugeteilten Instanzen.

C2: Datenprodukt

Das Datenprodukt enthält sowohl **Programm-Code**, als auch **-Logik** (C2-1). Die erarbeitete Lösung wird mittels Python implementiert und stellt eine Anwendung dar, die Daten abruft, analysiert, transformiert und bereitstellt. Eine programmatische Umsetzung ist aufgrund zahlreicher Programmiersprachen aus dem Data Science Umfeld, wie bspw. Python, R, Java, SQL und C++ gut umsetzbar. Insbesondere Python bietet sich durch seine zahlreichen Bibliotheken und Frameworks zum Aufbau von Echtzeit-Pipelines an. Python besitzt eine einfache und sehr zugängliche Syntax.

Java kommt eher in Bereichen mit komplexen, mathematischen Aufgaben zum Einsatz, während R häufig bei statistischen Berechnungen und Analysen verwendet wird. SQL wiederum findet Anwendung bei Abfragen und Manipulation von Daten in Datenbanken und nimmt eine wesentliche Bedeutung bei der Daten-Extraktion und -Transformation ein. Bei Fragen mit Bezug zum Speicher-Management, bietet C++ gute Kontrollmöglichkeiten und effiziente Lösungen in der Verwaltung und Skalierbarkeit von Data Science Anwendungen.

Die Auswahl auf Python zur Umsetzung der Pipeline-Anwendung liegt demnach im Rahmen der empfohlenen Implementierungs-Lösungen.

[29] Unter *DevOps* werden Praktiken und Prinzipien zusammengefasst, die den Entwicklungs- und Bereitstellungsprozess von Software automatisieren. Ziel ist eine optimale Unterstützung agiler Entwicklungsmethoden sowie eine Verbesserung der teamübergreifenden Kommunikation und Zusammenarbeit (vgl. Farooqui & Chikoti, 2021).

Die Umsetzung von **Schnittstellen auf Code-Basis** lässt sich ebenfalls mit einfachen Python-Mitteln umsetzen. So wird bspw. der Datensatz im JSON-Format über eine JSON-Befehl eingelesen und steht zur weiteren Verarbeitung zur Verfügung.

Das Implementierungsbeispiel stellt eine recht simple Form dar. Das Einlesen aus andere Quellen, wie z. B. einer Datenbank oder eines Daten-Streams, erfordert eine wenig mehr Logik, welche mittels geeigneter Bibliotheken unterstützt wird.

Eine gute Unterstützung lässt sich zudem durch geeignete Konnektoren erzielen. Konnektoren sind Bestandteil des Kafka Ökosystems und ermöglichen den Datenaustausch zwischen Kafka und Fremdsystemen. Dabei wird zwischen *Source-* und *Sink-Konnektoren* unterschieden, die zum einen die Datenquellen oder das Ziel definieren und zum anderen, die eigentliche Datenübertragung durchführen.

Konnektoren kommen in dieser Arbeit nicht zum Einsatz. Sie sind jedoch für die Anbindung von externen Quellen und Zielen, wie z. B. MongoDB oder PostgreSQL zu empfehlen.

Die Forderung von **Policy als Code** ist an dieser Stelle schwer umzusetzen. Es ist anzunehmen, dass von Data Mesh erwartet wird, während der Pipeline-Arbeit entsprechend Regularien stetig zu prüfen sind. An dieser Stelle stellt sich die Frage, ob die Pipeline der richtige Ort für die Prüfungen ist.

Es wird darauf verwiesen, dass globale Governance-Regeln über den Sidecar-Prozess des Datenprodukts verwaltet werden, daher findet dieser Punkt in der vorliegenden Arbeit keine Anwendung im Pipeline-Umfeld.

Da **Daten und Metadaten** (C2-2) mittels externer Systeme eingesammelt werden, ist es fraglich, ob Metadaten in der Pipeline explizit erhoben werden müssen? Es empfiehlt sich die Nutzung externer Systeme, die einmalig auf der Self-Serve Data Platform allen Datenprodukten zentral zur Verfügung gestellt werden. Über entsprechende „Agenten“ oder Open-Source-Werkzeuge, werden Daten eingesammelt und an externe Systeme weitergeleitet, die anhand entsprechender Metriken und KPIs[30], die Metadaten auswerten.

Wie bereits erwähnt, ist die **Plattformabhängigkeit** (C2-3) bei Datenprodukten im Data Mesh zu tolerieren, ohne jedoch genaue Vorgaben zu machen, welche Form von Plattform zugrunde gelegt wird. Aufgrund der Vielseitigkeit der Unternehmens- und Organisationsformen, ist hier ein Plattform- und Architekturmodell zu wählen, welches den Anforderungen und Gegebenheiten eines Unternehmens am meisten entsprechen.

[30] Beim Begriff *Key Performance Indikator (KPI)* (dt. Schlüsselkennzahl) handelt es sich um ein Leistungsmaß, welches in einem betriebswirtschaftlichen Umfeld erhoben und zur Bewertung genutzt wird.

Die Plattform stellt die relevantesten Services dar, die von den Datenprodukten genutzt werden können und vermeidet somit mehrfache Installationen gleicher Pipeline-Tools etc. Zudem hält sie Speichermöglichkeiten vor, die von allen Datenprodukten genutzt werden können. Die Ablage der Datenprodukte selbst sich bspw. mittels eines Daten-Katalogs realisieren.

In dieser Arbeit wird der Pipeline-Prozess näher beleuchtet, somit stehen die Kommunikationsflüsse und -verbindungen im Vordergrund. Die Entscheidung für diese Aufgabe fiel auf Kafka, welche eine gute Basis zur Realisierung dieser Anforderungen bietet.

Ansätze von Kafka im Data Mesh findet man in den Werken von H. Dulay und S. Mooney (Dulay & Mooney, 2023) sowie A. Bellemare (Bellemare, 2023) thematisiert. Aufgrund der Nähe der Komponenten aus Kafkas Ökosystem zu den Data Mesh-Prinzipien, eignet sich Kafka gut um den Kommunikationsfluss einer Pipeline abzubilden.

Herausforderung

Für die Realisierung der Pipeline im Datenprodukt bieten sich auch Herausforderungen. So sind nicht alle Strukturkomponenten des Architekturquantums relevant. Dennoch nehmen sie Einfluss auf die Gestaltung der Pipeline. Im Bereich von Code und Logik ist es z. B. herausfordernd, wie sich die Schnittstellengestaltung als Code gestaltet. Dazu gehört die Bekanntmachung, die Umsetzung er Kompatibilität sowie die konkrete Ausgestaltung der API.

Bzgl. der Plattformabhängigkeit zieht die Zentralisierung in gewisser Weise wieder „durch die Hintertür ein“. Das mag um einen daran liegen, dass Datenbankhersteller und Data Warehouse- und Data Lake-Anbieter daran interessiert sind, ihre Position im Data Analytics Bereich zu festigen. Zum anderen, weil es aus Kosten- und Lizenzgründen, keinen eigenen Speicher pro Datenprodukt geben darf. Die Zentralität ist bei Data Mesh demnach nicht ganz abgeschafft, sie wird lediglich auf die Self-Serve Data Plattform verlagert. Einem „Selbstbedienungs-Portal“ oder auch „Werkzeugschrank“ für benötigte Komponenten (Speicher, Workflow-Tools, Bibliotheken etc.). Eine Herausforderung dabei, ist die Ausgestaltung der Schnittstellen, die Standardisierung der Tools und die richtige Balance zwischen Autonomie und gemeinsamer Basis.

Empfehlung

Der Bereich Policy als Code und eine damit einhergehende Automatisierung, ist nur schwer umzusetzen. In den Arbeiten von (Wider et al., 2023) gibt es einen Ansatz, die Konzepte der *SLAs* und *SLOs* aus dem Site Reliability Engineering aufzugreifen und Schnittstellen mit geeigneten Regeln zu versehen und somit auch einer Prüfung zu unterziehen. Hierzu werden lediglich die Input- und Ouptut-Ports mit geeigneten SLOs versehen, da diese eine hohe technische Ausprägung haben und den Weg beschreiben, wie Daten angeboten werden.

C3: Transformationsprozesse

Im Rahmen dieser Arbeit wurde für die Implementierung der Pipeline **ein programmatischer Ansatz** (C3-1) auf Basis von Python gewählt. Anhand geeigneter Bibliotheken zur Datenanalyse (z. B. pandas), kann mit einfachen Mitteln eine Datenanalyse und -transformation umgesetzt werden. Die Transformation erfolgt **datenflussbasiert** (C3-2), indem die Schritte Ingestion, Extract, Transform und Load auf einfachem Wege angewandt werden. Somit lässt sich auch dieser Punkt ohne große Herausforderungen umsetzen.

Es wäre ohne weiteres möglich, eine **ML-Transformation** (C3-3) anzuwenden. Da der gewählte Anwendungsfall keine geeignete Aufgabenstellung bietet, wurde dieser Ansatz nicht weiter verfolgt.

Zeitbasierte Transformationen (C3-4) sind in erster Linie programmatische Umsetzungen mit Fokus auf die Transformation fest definierter Zeitspannen. Es ist zu erwarten, dass diese Transformationsschritte anhand spezialisierter Bibliotheken implementiert werden, daher sollte die Umsetzung keine größere Herausforderung an die Pipeline im Data Mesh stellen. Auch ein solcher Anwendungsfall wird in dieser Arbeit nicht näher beleuchtet, daher wird diesem Ansatz nicht nachgegangen.

Herausforderung und Empfehlung

Größere Herausforderungen lassen sich für die Transformationsprozesse nicht feststellen. Es sind keine Schritte zur Verbesserung erforderlich.

C4: Datenquellen und Datennutzer

In Kap. 4.4.2 sind die verschiedenen Archetypen von Datenprodukten vorgestellt worden. Die Ausprägung, des in dieser Arbeit verwendeten Datenproduktes, orientiert sich an den quellenorientierten Datenprodukten (C4-1), deren Input sich über die Pipeline-Schritte transformiert und im Anschluss mehreren Datennutzern zur Verfügung gestellt werden.

Durch die Nutzung von Kafka als zugrundeliegende technologische Plattform, ist diese Konstellation sehr effizient zu bedienen. Das Konzept der Topics ermöglicht die Breitstellung der Daten für eine Vielzahl von Nutzern. Die Daten werden nach der Transformation in einem für sie vorgesehenen Topic abgelegt. Interessierte Nutzer können sich beim *Bootstrap-Server* registrieren, und das gewünschte Topic abonnieren. Bei Bedarf können die Daten zu gegebenem Zeitpunkt abgerufen werden. Dabei verschwinden die Daten nach Abruf nicht aus der Nachrichten-Queue, sondern erlauben einen mehrmaligen Bezug. Die Nutzer können die Daten mittels sequentiellem Zugriff für eine zuvor definierte Zeit konsumieren.

Durch den Einsatz einer Schema-Registry, lassen sich die Daten getrennt von ihrer Darstellung serialisieren. Erst bei Abruf der Daten, werden diese in ein gewünschtes Zielformat transferiert. Der Vorteil dabei ist, dass dem Nutzer die Daten so bereitgestellt werden, wie er sie für seine Zwecke benötigt. Diese Art der Bereitstellung entspricht auch den Usability-Prinzipien des Datenproduktes.

Herausforderung

Als Herausforderung kann genannt werden, dass die Verwaltung der Schemata eine anspruchsvolle Aufgabe sein kann. Bspw. wäre es wünschenswert die Schemata mittels Versionierung zu kennzeichnen, so dass die Auswahl eines Anbieters diese Anforderung berücksichtigen sollte. Die Fa. Confluent Inc. wirbt bei ihrer Schema-Registry mit einer Versionsverwaltung der vorhandenen Schemata (Confluent Inc., 2024).

Zudem gibt es bislang offene Fragen bei der Umsetzung von Multi-Mandantenfähigkeit von Kafka. Entsprechende Konzepte sind erforderlich, wenn die Anwendung mehrere Geschäftsfelder umfasst, die weiterhin voneinander separiert werden sollen.

C5: Anforderung an die Daten

Grundsätzlich lässt sich sagen, dass alle von Data Mesh geforderten Datenformate bedient werden. Hinsichtlich **multimodaler Daten** (C5-1), wird auf die Verwendung der Schema-Registry, die eine Vielzahl von Datenschemata verwalten, verwiesen. Durch Deserialisierung mit einem gewählten Schema, wird das gewünschte Zielformat ausgegeben.

Bitemporale Daten (C5-2) ermöglichen die Betrachtung festdefinierter Zeitspannen. Das ist im Umfang von Kafka soweit möglich, da die Datensätze nach Bezug durch einen Datennutzer nicht gelöscht werden. Demnach verbleiben sie in der Nachrichten-Queue und können mittels Zugriff auf ein Offset, die Betrachtung auf einen zeitlich bestimmten Raum eingrenzen.

Die **Unveränderlichkeit der Daten** (C5-3) ist durch die Nachrichten-Queue gegeben. Diese Datenstruktur bietet lediglich einen Puffer, in dem Nachrichten vorübergehend gespeichert werden. Dabei ist eine bestimmte Reihenfolge vorgegeben. Demnach ist eine Änderung der Daten während des Nachrichtenversands nicht vorgesehen. Werden bestehende Datensätze modifiziert und erneut versandt, werden diese als neuer Datensatz an die Queue übergeben. Der ursprüngliche Zustand der Daten bleibt in der Queue gespeichert.

Dementsprechend trifft auch die Forderung nach einem **Read-Only-Zugriff** (C5-4) zu. Eine Revisionssicherheit, durch den Verbleib aller Daten in der Queue, ist somit gegeben.

Herausforderung

Durch die ständig wachsenden Nachrichten-Queues, wachsen die Dimensionen von Kafka hinsichtlich des Einsatzes von Brokern und Rechenleistung.

Insbesondere beim Zugriff auf historische Daten ist mit einem Effizienzverlust zu rechnen. Hintergrund dafür ist, dass Data Mesh auf Basis einer Kappa-Datenarchitektur empfohlen wird (vgl. Kap. 3.3). Wie bereits geschildert, entfällt bei Kappa der Batch-Layer, der maßgeblich für die Aufbewahrung historischer Daten zum Einsatz kommt. Demnach muss die verwendete Streaming-Plattform dazu in der Lage sein, historische Daten an den Datennutzer zurückzugeben. Dazu muss sie entweder ihr Streaming-Cluster durch Hinzufügen weiterer Broker erweitern oder die Fähigkeit zum Einsatz eines sog. *Tired Storage* aktivieren.

Bei Tired Storage handelt es sich um Ebenen in der Streaming-Plattform, die außerhalb der Broker liegen. Dort können Daten ausgelagert und bei Bedarf wieder geladen werden. In diesem Fall entfällt die Notwendigkeit die Plattform durch Hinzufügen weiterer Broker zu skalieren (vgl. Dulay & Mooney, 2023, S. 22-23).

Im Falle von Kafka verlangt die horizontale Skalierung des Cluster auch Rechenleistung ab, da nun mehr zusätzliche Partitionen angesprochen und verwaltet werden.

Empfehlung

Kafka selbst bietet Tired Storage seit Version 3.6.0 an und ermöglicht somit Rechen- und Speicher-Ressourcen unabhängig voneinander zu skalieren (vgl. Maison, 2023). Demnach ist die Funktionalität, die erstmal 2019 über KIP-405 (Kafka Improvement Proposals) vorgeschlagen wurde, in einem frühen Stadium (*Early Access*) verfügbar. Diese Version weist jedoch noch einige Einschränkungen auf, so dass sie noch keine Eignung für produktive Anwendungen aufweist (vgl. Maison, 2023).

7 Ergebnis und Ausblick

Insgesamt lässt sich sagen, dass die Umsetzung der Pipeline in Data Mesh einige Herausforderungen birgt. Weg von der Sicherheit einer etablierten, zentralen Pipeline-Arbeit, wandert die Verantwortung in die geplant autonomen Datenprodukte. Dabei stellt sich schnell die Frage nach der doppelten Verarbeitung. Durch den engen Bezug zum Datenprodukt-Kontext, ist eine schnelle Reaktion auf Änderungen und Anpassungen möglich. Im Umkehrschluss gibt es kaum Kontrolle darüber, ob das Datenprodukt in seiner Sinnhaftigkeit noch gerechtfertigt bzw. ob die Relevanz der einzelnen Transformationsschritte noch gegeben ist.

Weitere offene Punkte stammen aus dem Bereich

- Sicherheit in der Pipeline,
- Automatisierung und Einfluss von Policies auf die Pipeline,
- Schnittstellenmanagement und Kompatibilität sowie
- Performance-Testing.

Viele dieser Bereiche greifen auf etablierte Tools zurück, die zentral auf der Self-Serve Data Platform bereitgestellt werden müssen. Dieser Punkt wiederspricht zwar dem Gedanken der Dezentralisierung, ist jedoch aus lizenzrechtlichen Gründen ein Argument für ein effizientes Kosten-Nutzen-Verhältnis (vgl. Kap. 2.2).

Rolle von Kafka

Fragen der Kommunikation und fachlichen Datenaufbereitung und Bereitstellung, lassen sich auf Basis des Publish-Subcribe-Prinzips am Beispiel von Kafka, gut lösen. Auf technischer Ebene betrachtet, kann Kafka viele Aspekte nicht aufgreifen. Ursache dafür ist, dass Kafka schlichtweg nicht für alle Facetten der Pipeline-Arbeit ausgelegt ist und einen klaren Fokus auf den einfachen Nachrichtentransfer legt.

Die Vorteile von Kafka liegen insbesondere in der Organisation des Nachrichtenversands sowie der Bereitstellung und Eingruppierung der Nachrichten in einen bestimmten Themenkontext (Topic). Stärken hat Kafka daher in der Ansprache der Quell- und Zielsysteme bzw. der Datenerzeuger und der Datennutzer.

Abgrenzung zur CI/CD-Pipeline

Bei klassischen CI/CD-Pipelines werden eine Vielzahl von Prozessen und Tools in der Pipeline berücksichtigt. Dazu gehören:

- Tests (Statische Codeanalyse, Secrets Detection, Dynamische Analyse, Dependency Scan, Container Scans)
- Eingebundene Repositories
- Einrichten von Workflows für spezielle Pipeline-Szenarien
- Berücksichtigung verschiedener Plattformen/Stages (z. B. Entwicklung, Test, Integration, Produktion)

Aufgrund ihrer Ausrichtung auf den Software-Entwicklungsprozess anstelle des Transformationsprozesses analytischer Daten, sind CI/CD-Pipelines nicht Gegenstand dieser Arbeit. Dennoch besitzt sie Eigenschaften, von denen die Data Engineering Pipeline profitieren könnte (vgl. Cowell et al., 2023, S. 89). Dazu zählen z. B.:

- Functional tests,
- Security scans,
- Code quality scans,
- Performance tests,
- Licence scanning,
- Fuzz testing etc.)

Zudem ist Data Engineering Pipelines nur sehr schwer ein Pipeline-Status, wie z. B. *running, pending, skipped* und *canceled* zu entnehmen. Offen ist ob ein Workflow konfigurierbar ist (vgl. Cowell et al., 2023, S. 102).

Schlussendlich lässt sich behaupten, dass sich der Data Mesh-Ansatz gut auf die Data Engineering Pipeline abbilden lässt. Somit wird die Forschungsfrage nach der Umsetzbarkeit der Pipeline nach den Data Mesh-Prinzipien positiv beantwortet.

Hinsichtlich der dabei entstehenden Herausforderungen, gilt es noch eine Vielzahl von Lösungen zu erarbeiten, die nach Ausrichtung und Gestaltung der Unternehmen und ihrer Geschäftsfelder, sehr individuell ausfallen können. Diese Herausforderungen können als Einstieg in neue Forschungsfragen betrachtet werden.

Ausblick

Im Rahmen der Umsetzung von Data Mesh, gibt es noch viele Bereiche, die neue Forschungsfelder eröffnen. So bildet bspw. der Sidecar-Prozess ein Themenfeld, für den bisher kein Standard existiert. Der **Sidecar-Prozess** ist an das Datenprodukt angeschlossen und realisiert die Konfiguration und Ausführung von Richtlinien im Zusammenhang mit der Self-Serve Data Platform (vgl. Dehghani, 2023, S. 188). Inwieweit hier Vorgaben für den Pipeline-Prozess verortet werden, ist zu hinterfragen, daher gibt es an dieser Stelle Forschungsbedarf.

Ein weiteres Forschungsfeld bildet die Ausgestaltung der analytischen Schnittstellen. Diese Schnittstellen sollen das Auffinden, die Erschließung, die Überwachung der Datenprodukte sowie die Bereitstellung ihrer Daten realisieren. Dehghani bezeichnet diese Schnittstellen als **High-Level-APIs**, die alle diese Funktionen kapseln und betont, dass es für diese noch keine durchgängig akzeptierten Konventionen gibt (vgl. Dehghani, 2023, S. 190).

Wie zuvor schon erwähnt, bildet der Bereich der Umsetzung von Governance durch eine möglichst weite **Automatisierung der Policies** ein weiteres Themenfeld, für das Lösungen zu erarbeiten sind. Ob diese Regeln auch in den Pipeline-Prozess mit einfließen wäre zu prüfen.

Im Hinblick auf die Zunahme von **Künstlicher Intelligenz (KI)** im automatisierten Entwicklungsumfeld, sind bisher noch keine Ansätze im Zusammenspiel mit Data Mesh ersichtlich. Da Data Mesh größtenteils code-basiert arbeitet um eine weitgehende Automatisierung umzusetzen, sind auch hier zukünftig die Einflüsse von KI zu erwarten.

Data Mesh bleibt ein spannendes Thema, mit dem sich die analytische Datenverarbeitung noch lange beschäftigen wird.

A Referenzen / Literaturverzeichnis

Amazon (2024). *Was ist der Unterschied zwischen Kafka und RabbitMQ*. Abgerufen am 07.01.2024 von https://aws.amazon.com/de/compare/the-difference-between-rabbitmq-and-kafka/.

Astorino, S., & Simmonds, M. (2022). *Data Fabric - An Intelligent Data Architecture for AI.* Boise: MC Press, LLC.

Bachmann, R., Kemper, G. & Gerzer, T. (2014). *Big Data – Fluch oder Segen?* Heidelberg, München, Landsberg, Frechen, Hamburg: mitp.

BARC (2023). *Data Mesh: Game Changer or Just Hot Air?* Abgerufen am 02.08.2023 von https://barc.com/de/research/data-mesh/.

Bellemare, A. (2023). *Building an Event-Driven Data Mesh: Patterns for Designing & Building Event-Driven Architectures.* Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly.

Bitkom e. V. (2022). *Data Mesh – Datenpotenziale finden und nutzen.* Abgerufen am 22.07.2023 von https://www.bitkom.org/sites/main/files/2022-06/220531_LF_Data_Mesh.pdf.

Botthof, A., & Hartmann, E. (2015). *Zukunft der Arbeit in Industrie 4.0 – Neue Perspektiven und offene Fragen.* Berlin: Springer.

Brewer, E. (2000). *Towards Robust Distributed Systems*. *Symposium on Principles of Distributed Computing (PODC) der University of California, Berkeley.* Abgerufen am 12.05.2024 von https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf.

Chandler, P., Sweller, J. (1991). *Cognitive load theory and the format of instruction*. In *Cognition and Instruction*, 8, S. 293-332.

Confluent Inc. (2024). *Schema Registry Concepts*. Abgerufen am 21.03.2024 von https://docs.confluent.io/cloud/current/sr/fundamentals/index.html.

Cowell, C., Lotz, N., Timberlake, C. (2023). *Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples.* Packt Publishing: Birmingham.

Databricks (2024). *Data Lakehouse: Was ist ein Data Lakehouse?* Abgerufen am 10.05.2024 von https://www.databricks.com/de/glossary/data-lakehouse.

Dehghani, Z. (2019). *How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh*. Abgerufen am 16.10.2023 von https://martinfowler.com/articles/data-monolith-to-mesh.html.

Dehghani, Z. (2020). *Data Mesh Principles and Logical Architecture.* Abgerufen am 13.11.2023 von https://martinfowler.com/articles/data-mesh-principles.html

Dehghani, Z. (2023). *Data Mesh - Eine dezentrale Datenarchitektur entwerfen* (1. Auflage). Heidelberg: dpunkt.verlag GmbH.

Devlin, B. (1997). *Data Warehouse: From Architecture to Implementation.* Addison-Wesley.

Dixon, J. (2010). *James Dixon´s Blog: Pentaho, Hadoop, and Data Lakes.* Abgerufen am 10.05.2024 von https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/.

Dulay, H., Mooney, S. (2023). *Streaming Data Mesh: A Model for Optimizing Real-Time Data Services.* Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly.

Farooqui, S. M. & Chikoti, V. V. (2021). *Hands-on Site Reliability Engineering: Build Capability to Design, Deploy, Monitor, and Sustain Enterprise Software Systems at Scale.* BPB Publications: Neu-Delhi.

Fowler, M., & Sadalage, P. J. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* New Jersey: Addison-Wesley.

Freiknecht, J., & Papp, S. (2018). *Big Data in der Praxis: Lösungen mit Hadoop, Spark, HBase und Hive. Daten speichern, aufbereiten, visualisieren.* München: Hanser.

Frick, D., Gadatsch, A., Kaufmann, J., Lankes, B., Quix, C., Schmidt, A., & Schmitz, U. (2021). *Data Science: Konzepte, Erfahrungen, Fallstudien und Praxis.* Wiesbaden: Springer.

Gartner (2024). *Big Data*. Abgerufen am 11.04.2024 von https://www.gartner.com/en/information-technology/glossary/big-data.

GDWS - Webservices (2024). *PEGEL ONLINE Webservices*. Abgerufen am 22.03.2024 von https://www.pegelonline.wsv.de/webservice/dokuRestapi.

GDWS (2024). *PEGELONLINE*. Abgerufen am 21.03.2024 von https://www.pegelonline.wsv.de/gast/start.

Goedegebuure, A., Kumara, I., Driessen, S., van den Heuvel, W.-J., Monsieur, G., Tamburri, D. & Di Nucci, D. (2023). *Data Mesh: a Systematic Gray Literature Review.* (1), 29.

GO FAIR Initiative (2016). *FAIR Principles*. Abgerufen am 23.04.2024 von https://www.go-fair.org/fair-principles/.

Gupta, L. (2024). *Apache Kafka Tutorial*. Abgerufen am 17.01.2024 von https://howtodoinjava.com/kafka/apache-kafka-tutorial/.

Hechler, E., Weihrauch, M., & Wu, Y. C. (2023). *Data Fabric and Data Mesh Approaches with AI: A Guide to AI-based Data Cataloging, Governance, Integration, Orchestration, and Consumption.* New York: Apress.

Ishwarappa, & J., A. (2015). *A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology.* Procedia Computer Science, 48, S. 319-324.

Jones, C., Wilkes, J., Murphy, N. & Smith, C. (2024). *Google - Site Reliability Engineering: Service Level Objectives.* Abgerufen am 22.05.2024 von https://sre.google/sre-book/service-level-objectives/.

Laudon, K. C., Laudon, J. P. & Schoder, D. (2006). *Wirtschaftsinformatik - Eine Einführung.* München: Pearson Studium.

Maison, M. (2023). *Getting started with tiered storage in Apache Kafka.* Abgerufen am 24.05.2024 von https://developers.redhat.com/articles/2023/11/22/getting-started-tiered-storage-apache-kafka.

Marz, N., & Warren, J. (2016). *Big Data: Entwicklung und Programmierung von Systemen für große Datenmengen und Einsatz der Lambda-Architektur.* Frechen: mitp.

McGregor, S. E. (2022). *Practical Python Data Wrangling and Data Quality: Getting Started With Reading, Cleaning, and Analyzing Data.* Beijing/Boston/Farnham/Sebastopol/Tokyo: O'Reilly Media.

McKinney, W. (2019). *Datenanalyse mit Python: Auswertung von Daten mit pandas, NumPy und Jupyter.* Heidelberg: dpunkt.

Niehoff, M. (2023). *Data Mesh – ein Gamechanger?* Abgerufen am 02.08.2023 von https://www.informatik-aktuell.de/betrieb/datenbanken/data-mesh-ein-gamechanger.html.

Wider, A., Verma, S., & Atif, A. (2023). *Decentralized Data Governance as Part of Data Mesh Platform: Concepts and Approaches. arXiv preprints*. Abgerufen am 26.04.2024 von https://arxiv.org/abs/2307.02357.

Wikipedia01 (2024). *David_Wheeler.* Abgerufen am 23.05.2024 von https://de.wikipedia.org/wiki/David_Wheeler.

Wikipedia02 (2024). *Warteschlange (Datenstruktur).* Abgerufen am 23.05.2024 von https://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur).

Wikipedia03 (2024). *Logdatei.* Abgerufen am 23.05.2024 von https://de.wikipedia.org/wiki/Logdatei.

Wilkinson, M. D., Dumontier, M., Ijsbrand, J., Appleton, G., & Axton, M. (2016). *The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data, 3*. Abgerufen am 24.04.2024 von https://www.nature.com/articles/s41597-019-0009-6.

WSV (2024). *Generaldirektion Wasserstraßen und Schifffahrt*. Abgerufen am 21.03.2024 von https://www.gdws.wsv.bund.de/DE/gdws/01_ueber-uns/ueber-uns-node.html.

B Anhang

Installationsanleitung

Verwendete Technologien:

- Python 3.12
- Apache Kafka
- Zookeeper
- Confluent Schema-Registry
- Docker, Docker Compose

Empfohlene Werkzeuge zur Entwicklung und Verwaltung:

- Visual Studio Code (DIE)
- Docker Desktop

Laden der Anwendung

Die Anwendung **DataProductService** wird als Image auf einem USB-Stick ausgeliefert.

Dieses Image muss in eine lokale Docker-Laufzeitumgebung des Zielsystems geladen werden. Dazu wird die Docker gestartet (über Kommandozeile oder über **Docker Desktop**).

Mittels des Kommandozeilen-Befehls

```
docker load -i D:/masterarbeit-skrause/dataproduct-service.tar
```

wird das Docker-Image in die lokale Docker-Installation kopiert und sollte in der Anwendung **Docker Desktop** als Image aufgelistet werden.

Starten der Anwendung

Zusätzlich befindet sich auf dem USB-Stick eine **docker-compose.yaml** Datei. Diese Datei muss in ein gewünschtes Verzeichnis auf dem Zielsystem kopiert werden. Anschließend wird die containerisierte Umgebung mittels des folgenden Kommandozeilen-Befehls erzeugt:

```
docker-compose up -d
```

Hinweise

Der Download externer Images wird angestoßen.

- Apache Kafka (confluentinc/cp-kafka:latest)
- Schema-Registry (confluentinc/cp-schema-registry:latest)
- Zookeeper (confluentinc/cp-zookeeper:latest)

Diese können Größen in GB-Umfeld besitzen.

Beenden der Anwendung

Die Container-Umgebung wird mittels des folgenden Befehls heruntergefahren:

```
docker-compose down
```

Anschließend werden alle Container gelöscht, wobei die Images jedoch erhalten bleiben.

Urheber von Software

The Apache Software Foundation
Die Apache Software Foundation ist für die Entwicklung von Apache Kafka zuständig.

Confluent Inc.
Confluent stellt eine Daten-Streaming-Plattform für Apache Kafka bereit bietet eine Vielzahl von Lösungen für Unternehmen, über On-Premises- und Multi-Cloud-Umgebungen hinweg. Beispiele sind Container Images, die Schema-Registry und der Cluster-Manager Zookeeper.