C:\Users\haller\Desktop\Logo_HFU.tif

Bachelorarbeit

in

Wirtschaftsinformatik

Softwaremigration in die Cloud

Entwicklung und Evaluation einer Event-Sourcing-Strategie im Bereich der Echtzeitdatenverarbeitung

|  |  |
| --- | --- |
| Referent: | Prof. Dr. Peter Mattheis |
| Korreferent: | Michael Kost |
| Vorgelegt am: | 28.02.2023 |
| Vorgelegt von: | Maximilian Brauchle |
|  | 262300 |
|  | Stuttgarter Straße 37  71735 Eberdingen |

# Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Eberdingen, 25.02.22 Maximilian Brauchle

# Abstract

Zunehmend möchten Unternehmen ihre Echtzeitdatenverarbeitungssysteme in die Cloud migrieren, wissen aber weder, welche Komponenten sie dafür brauchen, noch wie sie ein solches System in die Cloud migrieren können. Das Ziel der Arbeit ist es einen Lösungsansatz, für die Migration einer Plattform zur Echtzeitdatenverarbeitung, zu bieten, basierend auf einer Event-Sourcing Strategie. Daraus ergibt sich folgende Forschungsfrage: Wie kann ein System zur Echtzeitdatenverarbeitung, anhand einer Event-Sourcing Architektur, in die Cloud zu migriert werden.

# Inhaltsverzeichnis

Inhalt

[Eidesstattliche Erklärung III](#_Toc120551919)

[Abstract V](#_Toc120551920)

[Inhaltsverzeichnis VII](#_Toc120551921)

[Abbildungsverzeichnis XI](#_Toc120551922)

[Tabellenverzeichnis XII](#_Toc120551923)

[1 Einleitung 13](#_Toc120551924)

[1.1 Motivation 13](#_Toc120551925)

[1.2 Problemstellung 13](#_Toc120551926)

[1.3 Zielsetzung 13](#_Toc120551927)

[1.4 Abgrenzung 14](#_Toc120551928)

[1.5 Überblick 14](#_Toc120551929)

[2 Literaturrecherche 14](#_Toc120551930)

[2.1 Einführung Cloud Computing 14](#_Toc120551931)

[2.1.1 Vorteile Cloud Computing 15](#_Toc120551932)

[2.1.2 Probleme und Limitierungen 15](#_Toc120551933)

[2.2 Kernelemente der Echtzeitdatenverarbeitung 16](#_Toc120551934)

[2.2.1 Definition Event Sourcing 16](#_Toc120551935)

[2.2.2 Anwendungsfälle Event Sourcing 16](#_Toc120551936)

[2.2.3 Limitierungen/Nachteile Event Sourcing 17](#_Toc120551937)

[2.2.4 CQRS-Pattern 17](#_Toc120551938)

[2.2.5 Domain Driven Design 18](#_Toc120551939)

[2.2.6 Das Problem mit CRUD 18](#_Toc120551940)

[2.2.7 Abgrenzung der vorgestellten Patterns 19](#_Toc120551941)

[3 Anforderungen an das System 19](#_Toc120551942)

[3.1 Anforderungen an Echtzeitdatenverarbeitungssysteme 20](#_Toc120551943)

[3.2 Stakeholder 21](#_Toc120551944)

[3.3 Projektziele 21](#_Toc120551945)

[3.4 Funktionale Anforderungen 21](#_Toc120551946)

[3.5 Nicht-funktionale Anforderungen 23](#_Toc120551947)

[3.5.1 Qualitätsanforderungen 24](#_Toc120551948)

[3.5.2 Technische Anforderungen 24](#_Toc120551949)

[3.5.3 Sonstige Lieferbestandteile 25](#_Toc120551950)

[3.5.4 Vertragliche Anforderungen 25](#_Toc120551951)

[3.6 Anforderungsprüfung 25](#_Toc120551952)

[3.6.1 Anforderungsprüfung nach INVEST-Kriterien 25](#_Toc120551953)

[3.6.2 Auswertung und Korrektur 26](#_Toc120551954)

[4 Echtzeitdatenverarbeitung in der Cloud 30](#_Toc120551955)

[4.1 Komponenten einer Echtzeitdatenverarbeitungsplattform 30](#_Toc120551956)

[4.1.1 Datenintegration und Analyse 30](#_Toc120551957)

[4.1.2 Lambda Architektur 31](#_Toc120551958)

[4.1.3 Event-Driven Architektur 32](#_Toc120551959)

[4.2 Event-Sourcing Architektur für Echtzeitdatenverarbeitung 33](#_Toc120551960)

[5 Implementation einer Event-Sourcing Strategie in Azure 36](#_Toc120551961)

[5.1 Referenzarchitektur in Azure 36](#_Toc120551962)

[5.2 Datengrundlage 36](#_Toc120551963)

[5.3 Programmierung 36](#_Toc120551964)

[5.4 Verwendete Dienste 36](#_Toc120551965)

[5.5 Handhabung des Programms 36](#_Toc120551966)

[5.6 Alternative Lösungsansätze 36](#_Toc120551967)

[5.7 Evaluation 36](#_Toc120551968)

[5.8 Ergebnisse 36](#_Toc120551969)

[6 Schlussfolgerung 36](#_Toc120551970)

[7 Zusammenfassung 36](#_Toc120551971)

[8 Ausblick 36](#_Toc120551972)

[9 Literaturverzeichnis 37](#_Toc120551973)

[10 Abkürzungsverzeichnis 41](#_Toc120551974)

[A. [Anhang] 43](#_Toc120551975)

# Abbildungsverzeichnis

[Abbildung 1: Use Case Diagramm zur Echtzeitdatenverarbeitung 23](#_Toc121390265)

[Abbildung 2: korrigiertes Use Case Diagramm 29](#_Toc121390266)

[Abbildung 3: Lambda-Architektur 32](#_Toc121390267)

[Abbildung 4: Event-Driven-Architektur 33](#_Toc121390268)

[Abbildung 5: Event-Sourcing Architektur 35](#_Toc121390269)

[Abbildung 6: Echtzeitdatenverarbeitung in Azure 38](#_Toc121390270)

# 

# Tabellenverzeichnis

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EinleitungMotivation Technologischer Fortschritt ist fortlaufend im Wandel so sind Cloud basierte Systeme kaum mehr aufzuhalten und Unternehmen verlagern zunehmend ihre Anwendungen in die Cloud [1] S.10. Auch MHP als IT-Management und Beratungsunternehmen stellt ein System zur Echtzeitdatenverarbeitung im Motorsport bereit. Diese Applikation, die stark auf eine Event-Sourcing Architektur setzt, soll in die Cloud migriert werden. Abgesehen von dem Verwaltungsaufwand und den hohen Anschaffungskosten für die Infrastruktur, beanspruchen Systeme wie diese viele Ressourcen und müssen bei hohen Auslastungen reibungslos arbeiten. Die Clouddienste wirken derartigen Problemen entgegen jedoch fehlt es einigen Unternehmen an Wissen und Erfahrung ein solches System basierend auf einer Event-Sourcing Architektur in der Cloud zu entwickeln [2] S.643. Problemstellung In der vorliegenden Arbeit wird darauf eingegangen wie sich ein System zur Echtzeitdatenverarbeitung basierend auf einer Event-Sourcing Architektur in der Cloud implementieren lässt. Zielsetzung Basierend auf definierten Anforderungen an Echtzeitdatenverarbeitungssysteme werden erforderliche Komponenten des Systems abgebildet. Diese werden als Grundlage für die Erstellung der Event-Sourcing Architektur herangezogen. Als nächstes wird eine Referenzarchitektur für die Cloud erstellt. Diese gilt als Basis für die spätere Implementierung der Softwarelösung. Darüber hinaus werden auch alternative Lösungsmöglichkeiten integriert und evaluiert. Das Ziel dahinter ist es Unternehmen einen roten Faden zu bieten, wie sich eine Event-Sourcing Architektur basierend auf solchen Systemen erstellen lässt. Denn so können Kosten und Zeit für das eigenständige Erarbeiten von Implementierungsmöglichkeiten gespart werden. Abgrenzung Die Literatur beschränkt sich in Bezug auf Event-Sourcing oftmalig auf theoretische Ansätze oder auf Lösungen, die nicht für die Cloud ausgelegt sind. Derartige Architekturen werden auch für die Cloud bereitgestellt, diese begrenzen sich jedoch oft auf nur einen spezifischen Anwendungsfall. Die vorliegende Arbeit dagegen, bietet dem Leser einen ganzheitlichen praktischen Ansatz zur Implementierung einer Event-Sourcing Strategie für Echtzeitdatenverarbeitungssysteme basierend auf einer Vielzahl von vergleichbaren Anwendungsfällen. Überblick In der vorliegenden Arbeit wird im folgenden Kapitel zur Literaturrecherche zunächst auf grundlegende Begriffe und Methoden eingegangen, die für das weitere Verständnis der Arbeit wichtig sind und darüber hinaus der aktuelle Stand der Literatur erläutert. Darauffolgend werden im dritten Kapitel bereits definierte funktionale sowie nicht-funktionale Anforderungen beschrieben. Diese werden schließlich auf ihre Stimmigkeit überprüft und korrigiert. Im anschließenden Kapitel 4 werden Komponenten einer Echtzeitdatenverarbeitungsplattform aufgezeigt und anhand dessen eine Event-Sourcing Architektur entworfen. So wird anschließend im 5ten Kapitel die Implementierung einer solchen Strategie beschrieben und andere Lösungsmöglichkeiten aufgezeigt. Zu guter Letzt werden die Ergebnisse zusammengetragen, bewertet und die Arbeit daraufhin im letzten Abschnitt zusammengefasst LiteraturrechercheEinführung Cloud Computing Cloud Computing beschreibt einen technologischen Ansatz, der dem Nutzer eine Sammlung von Diensten und Ressourcen flexibel und skalierbar bereitgestellt wird. Der komplette Betrieb und Wartungsaufwand dieser Dienste, wird dem Anbieter überlassen. Der Nutzer kann diese Leistungen in Anspruch nehmen und zahlt nur für die Nutzungsdauer der gemieteten Dienste. Der Anwender kann sich nach Bedarf, bereits fertige Software als Dienst mieten, kann aber auch nur die benötigte IT-Infrastruktur beziehen [1],S.6. Im Bereich Cloud Computing wird zwischen drei Cloud Modellen unterschieden. So gibt es in der Implementierung aus Anwendersicht Unterschiede in der Sicherheit aber auch hinsichtlich der Flexibilität. Zum einen gibt es die privatisierte Cloud, diese kann nur eingeschränkt genutzt werden und ist vergleichbar mit dem Intranet. Applikationen, Daten und andere Ressourcen befinden sich dabei im firmeneigenen Rechenzentrum. Bei einem Public Cloud Modell unterliegen die Daten und Dienste der Verantwortung des Cloud Anbieters. Das Dritte und Letzte Modell ist das Hybrid Cloud Modell und ist als Mischform der beiden anderen Ansätze zu verstehen. Das wird oft verwendet, wenn unkritische Applikationen, Daten und Ressourcen ausgelagert werden sollen und gleichzeitig geschäftskritische Anwendungen mit sensiblen Daten weiterhin intern betreut werden müssen [1],S.8. Darüber hinaus muss bei der Auswahl der passenden Dienste zwischen verschiedenen Ansätzen unterschieden werden. So wird dem Anwender beim Infrastructure-as-a-Service-Modell lediglich ein Teil der Infrastruktur zur Verfügung gestellt. Das kann zum Beispiel eine virtuelle Maschine oder Speicherplatz sein. Die Konfiguration und den Betrieb übernimmt der Cloud Anbieter, während sich der Anwender um das Betriebssystem und die Installation benötigter Dienste selbst kümmern muss. Beim Konzept Platform-as-a-Service wird dem Anwender darüber hinaus noch eine integrierte Entwicklungsumgebung mitgeliefert. So verfügen Entwickler über ein einheitliches Werkzeug, um zu programmieren, kompilieren und den Code anschließend in einer einheitlichen Umgebung auszuführen. Die Standardisierung der Entwicklungsumgebung ist besonders hilfreich, wenn es sich um große Entwicklungsteams handelt. Zu guter Letzt gibt es noch das Software-as-a-Service-Modell. Der Anwender kann zudem ganze Anwendungen in Anspruch nehmen, die er zur Verarbeitung seiner Daten nutzen kann. Der Anwender kann hierbei aber wenig bis gar keine Konfigurationen vornehmen eine typische Software-as-a-Service Anwendung ist beispielsweise GoogleMail oder Twitter. Zusammenfassend hat der Anwender beim Infrastructure-as-a-Service-Modell die größte Flexibilität aber auch den größten Aufwand. Das Software-as-a-Service-Konzept offeriert dagegen die geringste Mühe ist jedoch auch die unflexibelste Variante [2] S.642ff. Im Rahmen dieser Arbeit wird die Cloud als Infrastructure as a Service eingesetzt. Es werden bestimmte Dienste für die Umsetzung des Echtzeitdatenverarbeitungssystem beansprucht, diese werden aber selbstständig gemäß den Anforderungen konfiguriert und angepasst. Vorteile Cloud Computing Zunehmend wechseln Unternehmen von einer firmeninternen IT-Infrastruktur zu einem Cloud Computing basierten Ansatz. Zum einen, weil dem Nutzer hierdurch Zugriff auf Daten in Höchstgeschwindigkeit geboten werden und diese auch zu beinahe 100% verfügbar sind [2],S.26. Kunden profitieren außerdem von einer geringen Kapitalbindung. Es entstehen bei der Nutzung von Cloud Computing keinerlei Anschaffungskosten von IT-Infrastruktur wie Server, Lizenzen oder von Stellflächen. Des Weiteren wird die Komplexität des IT-Betriebs auf ein Minimum reduziert. Schließlich kann auch auf wechselnde Kapazitätsbedarfe schnell und flexibel reagiert werden. Ferner können sich Unternehmen durch Cloud-Dienste besser auf ihre Kernaufgaben wie der Optimierung von Geschäftsprozessen konzentrieren. Darüber hinaus ändert sich Technologien schneller und häufiger als früher. Die Cloud ermöglicht es schnell auf Technologieveränderungen zu reagieren und so einen potenziellen Marktvorteil zu erlangen [5] S.15ff. Das und noch weitere Vorteile wie Pay-Per-Use Preismodelle, skalierbare Ressourcennutzung und kurze Vertragslaufzeiten machen Cloud Computing zu einem attraktiven Angebot [1],S.14ff. Probleme und Limitierungen In der Cloud zu arbeiten, bringt jedoch auch Probleme und Restriktionen mit sich. Die Transparenz der verwendeten Sicherheitsmaßnahmen von Cloud Anbietern ist in den letzten Jahren gestiegen, jedoch fehlen trotzdem noch Informationen zu beispielsweise der Qualität des Programmcodes oder der genutzten Schnittstellen und der damit einhergehenden Verschlüsselung. Es gibt jedoch auch Open-Source Alternativen, die einen vollen Einblick auf die Sicherheit und den Programmcode bieten. Der vorherrschende Marktanteil obliegt jedoch den großen Unternehmen. So haben Microsoft und Amazon mit ihren Cloud Diensten einen Marktanteil von mehr als 50% [Statista Quelle einfügen]. Als Sicherheitsnachweise werden oftmals nur Zertifizierungen der Datenzentren genutzt ohne andere Kriterien. Bei Anwendungen, die mit personenbezogenen Daten arbeiten, muss der Cloud Anbieter in Bezug auf bestimmte Datenschutzvorschriften, dem Kunden rechtverbindlich und nachweisbar zusichern können, dass die verwendeten Daten auch innerhalb des vereinbarten Ortes bleiben. Public Cloud Infrastrukturen sind jedoch technologisch auf eine länderübergreifende Datenverarbeitung ausgelegt. Durch die mangelnde Transparenz und fehlende Überprüfbarkeit dieser Anforderungen führen dazu, dass Public Cloud Modelle in dieser Hinsicht nicht geeignet sind [3],S.392 ff. Dazu kommt das bei einem Schadenfall oft keine forensische Untersuchung unternommen wird, die Schadenfolgekosten reduzieren könnte. Das liegt daran, dass die Untersuchung dem Cloud Anbieter als Inhaber des Rechenzentrums der betroffenen Daten unterliegt. Diese führen eine solche Analyse aufgrund von haftungsrechtlicher Relevanz und fehlender rechtlicher Verpflichtung aber nicht durch. Problematisch sind auch die von der Cloud verwendeten Speicherstrukturen, so können bei einer Prüfung auch Daten von nichtbetroffenen Firmen offengelegt werden, die auf demselben Datenträger liegen [3],S.394 ff. Aufgrund der zentralen Datenhaltung erhöht sich auch die Angriffsfläche gegenüber potenziellen Angreifern. Des Weiteren können fehlende Standards hinsichtlich der Integration und der Migration schnell die Vorteile einer Cloud zu Nichte machen [5] S.17. Weiterführend gilt es abzuwägen ab wann sich eine Cloud lohnt und wann die hauseigene Lösung doch die bessere Wahl ist. Weiterführend beklagen sich immer mehr Unternehmen über signifikant ansteigende IT-Kosten. So wurden zwar anfänglich IT-Budgets festgelegt und eingehalten, diese aber schnell im Laufe der Zeit überschritten. Deswegen sollte das Kosten-Nutzen-Verhältnis stimmen und muss genau geplant werden [7] S.5ff. Kernelemente der EchtzeitdatenverarbeitungDefinition Event Sourcing Herkömmliche Systeme lesen Daten aus dem Speicher, nehmen Modifikationen vor und aktualisieren den Speicher mit den neuen Werten. Dabei werden Transaktionen verwendet, die die Daten in der Zeit sperren, um die Konsistenz zu wahren. Bei der Aktualisierung von Objekten können Konflikte auftreten und ohne separate Kontrollinstanz, die die Veränderungen der Daten aufzeichnet, ist der Verlauf und ein älterer Stand der Daten nicht mehr nachvollziehbar und reproduzierbar. Event Sourcing ist ein Ansatz, bei dem Ereignisse gespeichert werden, die eine Statusänderung eines Objekts beschreiben. Die Ereignisse werden, hingegen des traditionellen Ansatzes, nicht überschrieben, sondern einer Historie von bereits bestehenden Ereignissen hinzugefügt. Die Datenbank besteht dementsprechend aus einer Abfolge von unveränderlichen Ereignissen, die zu jeder Zeit den aktuellen Status des dazugehörigen Objekts reproduzieren können [4],S.52. Anwendungsfälle Event Sourcing Mit Event Sourcing ist eine umfassende Analyse von Daten möglich und wird daher in Systemen eingesetzt, die den Zweck, Grund oder die Absicht von Datenveränderungen erfassen möchten. Darüber hinaus kann das Entwurfsmuster dazu beitragen Aktualisierungskonflikte auf ein Minimum zu reduzieren. Event Sourcing liefert außerdem eine Historie von allen Ereignissen eines Objekts. So können Anwendungen die Daten durchgehend kontrollieren und auf einen beliebigen Zustand in der Vergangenheit zurückspielen [4],S.52. Limitierungen/Nachteile Event Sourcing Um mit Event Sourcing effizient zu arbeiten, müssen Snapshots von Ereignissen erstellt werden. Damit müssen nur nachfolgende Ereignisse, beginnend ab dem letzten Snapshot, verarbeitet werden. Werden keine Snapshots erstellt, müssen bei jeder Datenbankabfrage alle Events von Beginn an verarbeitet werden, dass folglich zu Performanceproblemen führt [4],S.54. Obendrein wird in Verbindung mit Event Sourcing in der Regel ein Dienst zur Vermittlung und Sortierung der Ereignisse genutzt. So können Ereignisse zwischen der Datenbank und anderen Applikationen, die diese weiterverarbeiten, ausgetauscht werden. Je nach Qualität des Nachrichtendienstes können Ereignisse auf den Weg verloren gehen, oder werden in falscher Reihenfolge empfangen. Infolgedessen kann der aktuelle Zustand eines Objekts nicht mehr richtig reproduziert werden [5],S.12. CQRS-Pattern CQRS steht für Command Query Responsibility Segregation und entwickelte sich aus dem Command Query Separation Pattern. Die Idee hinter diesem Entwurfsmuster war Abfragen und Kommandos bei Anwendungen zu trennen. Unter Abfragen versteht man Aktionen, die bestimmte Informationen von gespeicherten Objekten bereitstellen. Kommandos dagegen sind Aktionen, die gespeicherte Objekte verändern, erstellen oder löschen. Das Ziel dahinter ist es, die Zuverlässigkeit, Konsistenz des Systems sowie die Wiederverwendbarkeit der einzelnen Komponenten zu steigern. Während CQS als Entwurfsmuster auf Mikroebene zu verstehen ist, beschreibt das CQRS-Pattern die Trennung der Verantwortlichkeiten zwischen Abfragen und Verwaltung der Daten auf der Makroebene. Die Trennung von Schreib- und Lese-APIs, Modellen und Datenbanken werden als die wichtigsten Bausteine des CQRS-Entwurfsmusters zusammengefasst [6],S.451. Domain Driven Design Domain-Driven Design beschreibt einen Ansatz zur Erstellung von Software aus der Sicht des Unternehmens und stellt dabei die Fachlichkeit in den Vordergrund. Dabei wird berücksichtigt, dass Softwareentwickler nicht die einzigen sind, die beim Erstellen der Anwendung involviert sind. Auch die Fachleute für jene die Software entwickelt wird, bringen ein Verständnis für die zu lösenden Probleme mit. Um in solch einem interdisziplinären Team zu kommunizieren, wird sich auf eine gemeinsame Sprache der Ubiquitous Language geeinigt. (Auf die Bestandteile der Ubiquitous Language eingehen). Im Wesentlichen wird Domain-Driven Design in zwei Bestandteile unterteilt. Zum einem dem strategischen Design und zum anderen dem taktischen Design. Beim strategischen Design wird herausgearbeitet, was für Software gebaut wird und warum. Das taktische Design beschäftigt sich mit der Frage wie die Anwendung umgesetzt wird zum Beispiel wie können bestimmte Komponenten implementiert werden [7], S.15. Das Problem mit CRUD Für den Aufbau einer datenorientierten Anwendung wird in der Regel eine Schichtenarchitektur verwendet. Die Idee dahinter ist es die Verantwortlichkeiten zwischen der Präsentation der Daten, der Geschäftslogik und der Datenhaltung zu trennen. So sollte beispielsweise die Persistenzschicht strikt von den Mechanismen, die zum Speichern oder Abrufen von Daten verwendet werden, getrennt werden. Sie ist führt lediglich Operationen auf die Datenbank aus. Diese Datenbankoperationen werden als C, R, U, D (Create, Read, Update, Delete) ausgedrückt [8], S.16.  Die Verwendung von CRUD ist der meistverbreitete Ansatz im Bereich der Softwareentwicklung und funktioniert gut für kleine und einfache Anwendungen. Jedoch wirft das Modell auch Probleme auf. Zum einen werden alle Datenmodelle und Sichten oftmals aus einer einzigen Datenquelle bezogen. Somit muss der Datenspeicher sowohl über schnelle Lese- als auch schnelle Schreib-Operationen verfügen. Die Benutzung von Caches mindert das Problem erhöht jedoch die Komplexität. Darüber hinaus werden mit Update-Anweisungen mehrere Verantwortlichkeiten auf einmal beansprucht, da mit diesen Operationen sowohl Daten gelesen als auch geändert wieder in die Datenbank geschrieben werden. Für das Verändern von Daten werden beispielsweise Benutzerdaten an ein Benutzerobjekt übergeben anhand dessen kann jedoch nicht direkt das passende Objekt in der Datenbank aktualisiert werden. Das Benutzerobjekt muss erst in der Geschäftslogik aufbereitet und and die Persistenzschicht übergeben werden. Diese nimmt dann die eigentliche Datenbankoperation vor. Aufgrund dessen nimmt die Komplexität bei wachsenden Anwendungen stark zu und der Code wird schnell unübersichtlich [8], S.17. Abgrenzung der vorgestellten Patterns Bei allen vorgestellten Ansätzen handelt es sich um Entwurfsmuster für die Umsetzung von Softwaresystemen. Domain Driven Design dient dabei als Leitfaden für die Modellierung von modernen Anwendungen, um die Entwicklung komplexer Systeme zu reduzieren. Hierzu werden verbundene Kontexte verwendet, um zusammenhängende Teile als Komponenten, des zu implementierenden Systems zusammen zu fassen. Damit die Komponenten miteinander kommunizieren können wird Messaging eingesetzt. Darunter wird verstanden, dass Daten ausgetauscht werden jedes Mal, wenn ein neues Ereignis am System ankommt. Event Sourcing und CQRS beschreiben beide Möglichkeiten zur konkreten Umsetzung einer Messaging Strategie [9], S.19. Das Event Sourcing Entwurfsmuster ist aus der Domain Driven Design Gemeinschaft entsprungen und wird zusammen mit dem CQRS-Pattern eingesetzt. Es ist sehr stark an CQRS gebunden. So kann Event Sourcing nur mit CQRS umgesetzt werden. Es braucht jedoch nicht zwingend ein Event Sourcing Muster zur Umsetzung von CQRS [6], S.452. Anforderungen an das System In diesem Kapitel werden zunächst die Projektziele und daraufhin die fachlichen Anforderungen beschrieben. Aus diesen Informationen werden dann die funktionalen Anforderungen abgeleitet und im Anschluss auch die nicht-funktionalen Anforderungen erläutert. Abschließend werden diese auf ihre Stimmigkeit und Qualität geprüft. Anforderungen an Echtzeitdatenverarbeitungssysteme Big Data wurde in den letzten Jahren in fast allen denkbaren Bereichen wie der Informationstechnologie, dem Gesundheitssektor, der Industrie 4.0, der Bildung immer präsenter. Der Begriff beschreibt jedoch nicht nur die Menge, sondern auch die Vielfalt an unterschiedlichen Daten. Dazu kommt aber auch die Schnelligkeit der Datenverarbeitung. Millionen von neuen Informationen müssen in Millisekunden erfasst und verarbeitet werden. Systeme die nicht nur enorme Datenmengen bewältigen, sondern auch in Echtzeit verarbeiten können, gewinnen dadurch zunehmend an Bedeutung [10] S.2. Darauffolgend werden die Anforderungen an ein solches System im Motorsportumfeld beschrieben. Diese finden sich jedoch auch in vielen anderen Bereichen wieder. Dazu gehört der Gesundheitssektor. Daten müssen hier in besonders kritischen aber auch im Alltag erfasst und ausgewertet werden. Dabei werden Biosensoren an die verletzte oder erkrankte Person angebracht und diese in Echtzeit analysiert. So kann in lebensbedrohlichen Situationen rechtzeitig reagiert werden [11] S.65.  Aber auch ganze Städte nutzen Informationstechnologie, um Bereiche wie Nachhaltigkeit zu adressieren und Dienste als auch die Infrastruktur von Städten zu verbessern. So können freie Parkplätze, Buswartezeiten oder auch Wetterwarnungen unmittelbar bereitgestellt werden. Das bedeutet jedoch auch, dass Städte zu großen technologischen Ökosystemen wachsen. Diese Datenmengen gilt es fortlaufend zu bearbeiten, um kontinuierlich aktuelle Informationen zu gewährleisten [12] S.2.  Echtzeitdatenverarbeitung ist außerdem in der Industrie 4.0 unabdingbar. Notwendige Daten müssen in Sekunden erfasst, verarbeitet und analysiert werden. So kann ein System anhand der Sensordaten auf einen Fehler einer Maschine rechtzeitig reagieren und Produkte an eine andere Maschine weiterleiten [13] S.12.  Im Motorsportumfeld sind strategische Entscheidungen aufgrund von eingehenden Sensordaten entscheidend für den weiteren Verlauf des Rennens. Deswegen ist es notwendig, dass Auswertungen dem Rennteam so zeitnah wie möglich zur Verfügung stehen. Dieser Anwendungsfall wird nun im Folgenden genauer untersucht. Stakeholder Für das Echtzeitdatenverarbeitungssystem im Motorsport Umfeld gibt es drei verschiedene Interessensgruppen. Das ist neben dem Kunden zum einem das Entwicklungsteam der MHP und zum anderen externe Personen. Dazu gehört beispielsweise das Entwicklungsteam des Kunden und des Managements. Die Ziele und Anforderungen wurden von MHP bereits definiert und sind für diese Arbeit vorgegeben. Projektziele Im Folgenden werden die Projektziele für das Datenverarbeitungssystem für die Formel-E tabellarisch aufgezeigt.   |  |  | | --- | --- | | **ID** | **Ziel** | | Z1 | Die Daten sollen dem Rennteam in Echtzeit bereitgestellt werden. | | Z2 | Das IT-Personal und die damit verbundene Wartung soll auf ein Minimum reduziert werden. | | Z3 | Die Hardware an der Rennstrecke soll auf das Minimum reduziert werden. | | Z4 | Die time-to-market für neue Funktionen soll reduziert werden. | | Z5 | Die Auswertungen der Daten sollen dem Rennteam übersichtlich zur Verfügung stehen |   Tabelle 1: Ziele des Echtzeitdatenverarbeitungssystems Funktionale Anforderungen Dieses Kapitel beschreibt die funktionalen Anforderungen an das Echtzeitdatenverarbeitungssystem und dient als Grundlage für die zu entwickelnde Software. Funktionale Anforderungen beschreiben die Eigenschaften eines Systems, die zu erfüllen sind, um mit dieser Software ein fachliches Problem lösen zu können. Diese lassen sich als Anwendungsfälle beschreiben [14] S.5. Im Folgenden werden die Use Cases schriftlich in Form von User Storys mit Hilfe der Schablone nach [15] S.156, beschrieben.  A1: Als Entwicklungsteam, möchte ich ältere Zustände in der Datenbank reproduzieren können, damit ich mögliche Abweichungen oder Fehler einsehen und beheben kann. Das System soll dementsprechend fähig sein, Datenänderungen aufzuzeichnen.  A2: Als Entwicklungsteam, möchte ich, dass eingehende Ereignisse dauerhaft und in der richtigen Reihenfolge gespeichert werden, damit ich zu jedem beliebigen Zeitpunkt auf die Daten zugreifen kann  A3: Als Rennteam möchte ich Daten in Echtzeit bereitgestellt bekommen, damit ich umgehend über wichtige Ereignisse informiert werde wie zum Beispiel, wenn der Fahrer die Ziellinie überquert hat. Das System soll daher fähig sein, materialisierte Sichten zu erstellen und Snapshots von Ereignissen zu erstellen.  A4: Als Rennteam möchte ich Auswertungen der Ereignisse übersichtlich dargestellt bekommen, damit ich auf einen Blick Informationen wie die Rundenzeit, verbleibende Zeit oder die aktuelle Runde einsehen kann. Das System soll deswegen fähig sein, JSON-Daten auszuwerten und über ein User-Interface klar darzustellen.  A5: Als Rennteam möchte ich mich auf die Richtigkeit der Auswertungen verlassen können, damit ich während des Rennens die richtigen Entscheidungen treffe. Das System muss daher fähig sein, Ereignisse in der richtigen Reihenfolge zu verarbeiten, Schreib- und Leseoperationen zu trennen und Aktualisierungskonflikte zu vermeiden.  A6: Als Rennteam möchte ich wichtige Informationen des Timing Servers an der Rennstrecke erhalten, damit ich während des Rennens richtige strategische Entscheidungen treffe. Das System muss dementsprechend fähig sein, IoT-Geräte mit der Cloud zu verbinden. Es soll also Cloud-nativ umgesetzt werden, aber auch externe Komponenten integrieren können.    Abbildung 1: Use Case Diagramm zur Echtzeitdatenverarbeitung Nicht-funktionale Anforderungen Die funktionalen Anforderungen sind bereits definiert und so kann sichergestellt werden das eine Plattform entsteht, die der Endnutzer erfolgreich einsetzten, kann. Nun gibt es aber noch weitere Eigenschaften, die die Nutzbarkeit des Systems wesentlich beeinflussen. Dabei handelt es sich um Anforderungen, die rund um das Softwaresystem zu berücksichtigen sind. Diese werden in vier unterschiedlichen Gruppen klassifiziert. Das sind Qualitätsanforderungen, technischen Anforderungen, sonstige Lieferbestandteile und vertragliche Anforderungen, welche im Folgenden in Bezug auf das System genauer betrachtet werden, [16] S.84. Qualitätsanforderungen Die Qualitätsanforderungen werden in Tabelle 2 zusammengefasst.   |  |  | | --- | --- | | **Qualitätsmerkmal** | **Beschreibung** | | Performance | Das System soll fähig sein, Daten in Echtzeit zu verarbeiten. | | Zuverlässigkeit | Das System soll fähig sein, sich gegenüber Ausfällen abzusichern. | | Benutzbarkeit | Das System soll benutzerfreundlich sein und leicht in Betrieb genommen werden können. | | Sicherheit | Das System soll fähig sein, Daten vor Dritten unzugänglich zu machen und Datenschutzkonform zu arbeiten | | Korrektheit | Das System soll mit der vorgegebenen Spezifikation übereinstimmen | | Änderbarkeit | Das System soll lose gekoppelt sein, um sich bei ändernden Anforderungen anpassen zu können. |  Technische Anforderungen Unter technischen Anforderungen werden alle Eigenschaften beschrieben, die den technischen Rahmen für das Softwareprojekt bilden. Das sind in der Regel Hardwareanforderungen aber auch Forderungen, die für die Entwicklung notwendig sind. So wird beispielsweise definiert mit welcher Software das System zusammenarbeiten soll oder in welcher Programmiersprache das System zu entwickeln ist [16] S.87.  Für die zu entwickelnde Plattform sind nun einige technische Anforderungen vorgegeben. Die Plattform soll Betriebssystemunabhängig laufen und mit Python und NodeJs als Programmiersprachen entwickelt werden. Darüber hinaus sollen die Server des Systems aus Datenschutzgründen innerhalb der EU liegen. Zu guter Letzt soll Microsoft Azure oder AWS von Amazon als Cloud Anbieter eingesetzt werden. Sonstige Lieferbestandteile Normalerweise wird nicht nur die eigentliche Software an den Kunden ausgeliefert, sondern auch die dazugehörigen Produkte wie zum Beispiel Hardware oder Projektdokumentationen [16] S.87. Das zu entwickelnde System soll daher mit allen Entwicklungsdokumenten, Projektdokumentationen und einer Installationsanleitung, ausgeliefert werden. Vertragliche Anforderungen Vertragliche Anforderungen spielen im Rahmen dieser Arbeit keine Rolle, die zu fertige Lösung soll jedoch bis zum Ende der vereinbarten Vertragslaufzeit entwickelt werden. Anforderungsprüfung Anforderungen sind nicht nach der ersten Erfassung endgültig. Im Gegenteil weisen sie oftmals einige Mängel auf, die es zu prüfen und korrigieren gilt. So widersprechen sich einige Anwendungsfälle, sind unvollständig oder sind finanziell oder technisch gar nicht machbar S.177. In der Regel reicht jedoch eine einmalige Anforderungsprüfung nicht, da es bei komplexen Systemen schwer wird, alle Fehler auf einen Blick zu erkennen. Darüber hinaus entwickeln sich Anforderungen im Laufe des Projektes weiter und die Fehlerbehebung kann zu neuen Fehlern führen S.178. Deshalb wird die Anforderungsprüfung in einem iterativen Prozess durchgeführt. Da es sich in dieser Arbeit um eine überschaubare Anzahl an Anforderungen handelt, wird die Anforderungsprüfung einmalig durchgeführt. Hinsichtlich der Prüfung von Anforderungen wird zwischen einer Vielzahl von unterschiedlichen Qualitätskriterien differenziert. Im vorherigen Kapitel wurden die funktionalen Anforderungen in Form von User Stories erfasst. Daher bietet es sich an, diese nach den INVEST-Qualitätskriterien zu untersuchen. Anforderungsprüfung nach INVEST-Kriterien Um bewerten zu können, ob es sich um qualitativ hochwertige Anforderungen handelt, bedarf es konkrete und objektive Kriterien. Dazu gehören die INVEST-Kriterien für User Stories:  **Independent**: Die Stories sollten unabhängig voneinander sein und können eigenständig geplant und umgesetzt werden.  **Negotiable**: Die Stories sind verhandelbar und sind nicht an einen bestimmten Vertrag gebunden  **Valuable**: Stories müssen dem Kunden einen Mehrwert bieten.  **Estimable**: Der Aufwand der Stories ist schätzbar.  **Small**: Jede Story sollte innerhalb einer Iteration umsetzbar sein. Eine Iteration entspricht maximal eine Woche.  **Testable**: Für jede Story sind Kriterien definiert wurden, die festlegen, ob sie fertig ist S.182.  Im Folgenden werden die zuvor erfassten User Stories nach den oben genannten Kriterien geprüft. Als Antwortmöglichkeiten stehen ja, nein und nicht ganz zur Verfügung. Sobald eine User Story ein nein oder ein nicht ganz in einer der Kriterien aufweist, wird dieser genauer betrachtet und Antworten mit nein korrigiert.   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **ID** | **Independent** | **Negotiable** | **Valuable** | **Estimable** | **Small** | **Testable** | | A1 | Nicht ganz | Ja | Ja | nein | Nein | Ja | | A2 | Nicht ganz | Ja | Ja | Ja | Ja | Ja | | A3 | Nicht ganz | Ja | Ja | Ja | Ja | Ja | | A4 | Ja | Ja | Ja | Ja | Nein | Nicht ganz | | A5 | Ja | Ja | Ja | Nein | Nein | Ja | | A6 | Ja | Ja | Ja | Ja | Ja | Ja |   Tabelle 2: Auswertung der Anforderungen nach den INVEST-Kriterien Auswertung und Korrektur Tabelle 3 zeigt, dass außer der Anforderung A6, alle anderen User Stories Mängel aufweisen. Die Anforderungen A2 und A3 weisen kein nein in einer der Kriterien auf und werden daher als erstes betrachtet. A2 lässt Mängel im Kriterium Independent erkennen. Das liegt daran, dass diese Anforderung sich mit A1 überschneidet. Bei beiden geht es um die Datenspeicherung. Anforderung 1 beschäftigt sich jedoch eher mit der Frage wie die Daten und in welcher Form die Daten in die Datenbank abgelegt werden und bei A2 geht es darum auf welchem Medium die Daten abgelegt werden. A3 ist ebenfalls nur bedingt unabhängig. Die User Story lässt sich isoliert bearbeiten, dafür müssen jedoch erst Ereignisse in der Datenbank vorliegen.  Anforderungen A1, A4 und A5 müssen nun nochmal genauer betrachtet werden, da diese ein nein, in mindestens einem Kriterium aufweisen:  **A1**: Als Entwicklungsteam, möchte ich ältere Zustände in der Datenbank reproduzieren können, damit ich mögliche Abweichungen oder Fehler einsehen und beheben kann. Das System soll dementsprechend fähig sein, Datenänderungen aufzuzeichnen.  Die Anforderung ist wertschöpfend für den Kunden, da hierdurch Fehler und Abweichungen nochmal untersucht und frühzeitig behoben werden können. Testbar ist die User Story auch, da es sich in der Datenbank ablesen lässt, ob einkommende Ereignisse aufgezeichnet werden oder nicht. Das Problem ist, dass die Anforderung nicht genau spezifiziert ist und daher der Umfang und die Größe der User Story nicht geschätzt werden kann. Dementsprechend macht es Sinn die Anforderung zu unterteilen:  **A1.1**: Als Entwicklungsteam möchte ich Zugriff auf eine versionierte Änderungshistorie aller Ereignisse haben, damit ich mögliche Abweichungen oder Fehler leichter erkennen und beheben kann.  **A1.2**: Als Entwicklungsteam möchte ich anhand der Änderungshistorie, Zustände von älteren Versionen reproduzieren können, damit ich gemachte Fehler oder Änderungen wieder rückgängig machen kann.  **A4**: Als Rennteam möchte ich Auswertungen der Ereignisse übersichtlich dargestellt bekommen, damit ich auf einen Blick Informationen wie die Rundenzeit, verbleibende Zeit oder die aktuelle Runde einsehen kann. Das System soll deswegen fähig sein, JSON-Daten auszuwerten und über ein User-Interface klar darzustellen.  Anforderung A4 ist nicht genau testbar, da es bis zu einem gewissen Grad subjektiv ist, ob eine Darstellung übersichtlich ist oder nicht. Aber das Problem ist, dass die User Story einen zu großen Umfang hat. Die Auswertung und die Darstellung sollten in zwei separaten Schritten passieren:  **A4.1**: Als Rennteam möchte ich Ergebnisse der Datenanalyse übersichtlich dargestellt bekommen, damit ich auf einen Blick Informationen wie die Rundenzeit, verbleibende Zeit oder die aktuelle Runde einsehen kann.  **A4.2**: Als Rennteam möchte ich Ereignisse aus externen Geräten wie dem Timing Server und anderen Ressourcen ausgewertet bekommen, damit ich basierend darauf strategisch die richtigen Entscheidungen treffen kann.  **A5:** Als Rennteam möchte ich mich auf die Richtigkeit der Auswertungen verlassen können, damit ich während des Rennens die richtigen Entscheidungen treffe. Das System muss daher fähig sein, Ereignisse in der richtigen Reihenfolge zu verarbeiten, Schreib- und Leseoperationen zu trennen und Aktualisierungskonflikte zu vermeiden.  Die User Story A5 ist zu umfangreich und kann daher weder genau abgeschätzt noch innerhalb einer Woche umgesetzt werden. Auch in diesem Fall macht es Sinn die Anforderung spezifischer zu definieren:  **A5.1**: Als Rennteam möchte ich mich auf die Richtigkeit und die Aktualität der Auswertungen verlassen können, damit ich während des Rennens die richtigen Entscheidungen treffe. Das System muss daher fähig sein, Ereignisse in der richtigen Reihenfolge zu verarbeiten.  **A5.2**: Als Rennteam möchte ich nicht, dass wichtige Daten aufgrund von Aktualisierungskonflikten oder ähnlichem, verloren gehen, damit ich während des Rennens alle erforderlichen Informationen bereitgestellt bekomme, die ich brauche. Das System muss daher fähig sein Schreib- und Leseoperationen zu trennen, um Aktualisierungskonflikte zu minimieren.  Abbildung zwei stellt das veränderte Use Case Diagramm dar, dass alle Anwendungsfälle abbildet. Dabei sind die korrigierten Anforderungen hellblau markiert.    Abbildung 2: korrigiertes Use Case Diagramm Echtzeitdatenverarbeitung in der CloudKomponenten einer Echtzeitdatenverarbeitungsplattform Heutzutage steigt die Nachfrage nach Plattformen, die enorme Mengen an Daten verarbeiten und analysieren können. Immer mehr Systeme haben Probleme damit, die riesigen Datenströme und die damit verbundene Diversität zu bewältigen. Bezugnehmend zu [17] S.356 besteht ein klarer Trend in Richtung Echtzeitdatenverarbeitungssystemen, jedoch schafft es nur ein kleiner Teil diesen Anforderungen gerecht zu werden. Datenintegration und Analyse Die Echtzeitdatenverarbeitung lässt sich in zwei wesentliche Schritte unterteilen. Zum einen der Datenintegration, diese erfolgt gewöhnlich nach dem extract-transform-load-prozess, auch ETL-Pipeline genannt. Dabei werden die Daten aus den Datenquellen extrahiert, verarbeitet und in die gewünschte Zieldatenbank geladen. Die Datenanalyse erfolgt nach der Datenintegration und dient dazu, dem Nutzer wichtige Informationen mithilfe von Analysetools zu liefern. Herkömmliche Werkzeuge reichen jedoch oftmals nicht aus, um die Daten auch in Echtzeit liefern zu können. In dem letzten Jahrzehnt ist jedoch eine Vielzahl von Analysesoftware aufgetretenen, die diesem Problem entgegenwirken. Diese Werkzeuge sind dafür gemacht, Daten in Echtzeit zu erfassen, abzurufen und zu präsentieren. Darüber hinaus werden intensiviert NoSQL Datenbanken in leistungskritischen Szenarios eingesetzt. Diese zeichnen sich mit ihrer Key-Value Speichern vor allem durch Schemafreiheit, Replikationsunterstützung, einer hohen Konsistenz, Unterstützung großer Datenmengen und einer hohen Effizienz aus. Neben der Vielzahl an Tools gilt es aber auch ein Bewusstsein darüber zu entwickeln in welchem Zusammenhang die gelieferten Informationen stehen. Sowohl das Entwicklungsteam als auch die Ingenieure und Analysten an der Rennstrecke sollten ein klares Verständnis darüber haben, wie die Auswertungen zu verstehen sind. So sollten Kausalitäten und Korrelationen frühzeitig erkannt und Koinzidenzen aufgedeckt werden. Angenommen die Verlaufskurven der kürzer werdenden Rundenzeiten und der absteigenden Fahrernummern stimmen praktisch denkungsgleich überein. So sollte sich das Team im Klaren sein, dass es sich dabei weder um eine Kausalität noch um eine Korrelation, sondern vielmehr um eine Koinzidenz handelt. Es gibt also keinen direkten Zusammenhang der zwei Variablen noch sind sie voneinander abhängig. Es ist stattdessen eine rein zufällige Übereinstimmung. Lambda Architektur In der Vergangenheit wurden Daten hauptsächlich chargenbasiert verarbeitet und ausgeliefert, das hat sich im Laufe der Zeit nun geändert und Systeme müssen fähig sein Daten in Echtzeit zu verarbeiten und auszuliefern. Das erfordert eine Reihe von Tools die es zunächst mal auszuwählen und dann zu koordinieren gilt. Eine generische Lösung bietet die Lambda Architektur. Die Idee dahinter ist es Ergebnisse als Sichten im Voraus zu verarbeiten, um diese Sichten dann adhoc abrufen zu können. Folgende Abbildung beschreibt die Lambda Architektur. Diese besteht aus drei Schichten. Die Batch-Schicht sammelt eingehende Daten und vorverarbeitet diese zu Sichten. Die Daten sind jedoch nicht aktuell, gewöhnlich sind in der Zwischenzeit schon neue Daten eingetroffen, die verarbeitet werden müssen. Dafür ist die Speed-Schicht verantwortlich. Diese schließt die Lücke, indem sie die neuesten Daten in Echtzeit verarbeitet. Jede Abfrage des Nutzers wird nun durch die Serving-Schicht beantwortet, indem die verarbeiteten Sichten sowohl von der Batch- als auch von der Speed-Schicht abgefragt werden. Die Ergebnisse beider Schichten werden zusammengeführt und ausgeliefert. Eine weitere Besonderheit der Architektur ist, dass die Daten erstmal unverarbeitet gespeichert werden, so können zukünftige Algorithmen leicht hinzugefügt werden, indem eine neue Sicht auf die Daten erstellt wird. Des Weiteren können Fehler in der Verarbeitung leichter korrigiert werden, da nach der Fehlerbehebung die Daten nochmal neu berechnet werden können [17] S.357ff.    Abbildung 3: Lambda-Architektur  Dieser Ansatz hat einige Vorteile ist in der Umsetzung jedoch sehr komplex und weist Redundanzen auf. Darüber hinaus werden hierbei zahlreiche Technologien eingesetzt, die sowohl die Anforderungen für die Batch-Verarbeitung als auch für die Echtzeitverarbeitung, abdecken müssen. Diese gilt es obendrein untereinander zu koordinieren und zu synchronisieren. Aufgrund der unterschiedlichen Codebasen wird dadurch auch der Wartungs- und Testaufwand der Anwendungen erhöht. Die Lambda Architektur ist also ein guter generischer Ansatz, in der Praxis jedoch schwer umzusetzen, außerdem stellt sich die Frage, ob das zu implementierende System sowohl eine Batch-Schicht als auch eine Speed-Schicht braucht [10] S.5. Event-Driven Architektur Dagegen eignet sich Ereignisgesteuerte Architektur für verteilte Systeme. Diese zeichnet sich nämlich durch ihren nicht-monolithischen Ansatz, der losen Kopplung und ihren asynchronen Stil aus [18] S.52. Hierbei werden Funktionen zur Bearbeitung nicht in Intervallen ausgelöst, sondern immer genau dann, wenn neue Ereignisse auftreten wie beispielsweise die Statusänderung eines Objektes oder eine neue Bestellung. Dieser Ansatz eignet sich zum einen besser für die Echtzeitverarbeitung und zum anderen werden Ressourcen wie Rechnerleistung effizienter eingesetzt [19] S.1. Die Architektur richtet sich nach dem publish-subscribe-Prinzip. Das heißt Ereignisse werden aus verschiedenen Ereignisquellen heraus generiert und an ein System veröffentlicht, dass einen Geschäftsprozess abbildet. Nach Bearbeitung der neu eingetroffenen Ereignisse werden die Interessenshalter, die diesen Geschäftsprozess abonniert haben, über die neuen Daten informiert. Daraus ergeben sich folgende Komponenten, die für den Aufbau einer Ereignisgesteuerten Architektur benötigt werden. Die erste Komponente besteht aus den Ereignisgeneratoren. Jedes Ereignis wird aus einer bestimmten Quelle generiert, das kann ein externes System, Sensoren oder auch eine Datenbank sein. Die zweite Komponente beschäftigt sich mit der Übermittlung der Ereignisse an Bearbeitungssysteme oder Zieldatenbanken. Die Dritte Komponente stellt das Bearbeitungssystem selbst dar. Dort werden die Ereignisse evaluiert, bearbeitet und nach den Anforderungen der Interessenshaltern in die richtige Form gebracht. Die eigentlichen Konsumenten der Ereignisse stellen den letzten Baustein dar. Dazu gehören Analyseplattformen, weitere externe Systeme oder Datenbanken aber auch Personen, die daraufhin ggf. weitere Aktionen auf den Ereignissen vornehmen [20] S.4ff. Event-Sourcing beschreibt in dem Kontext eine Methode, wie die Daten persistiert werden. Darauf wird im Folgenden genauer eingegangen [21] S.353.    Abbildung : Event-Driven-Architektur Event-Sourcing Architektur für Echtzeitdatenverarbeitung Wie bereits erwähnt folgt das Event-Sourcing Muster nicht dem traditionellen Konzept, Zustände von Objekten zu ändern, indem bestimmte Attribute ersetzt werden. So wird nämlich nur der jeweils letzte Status eines Objektes erfasst. Vielmehr wird jedes Ereignis gespeichert, dass zu einer Statusänderung des Objektes führt. Somit ist dem System nicht nur der aktuelle Zustand einer Entität bekannt, sondern verfügt auch über eine komplette Historie aller Zustandsänderungen der Objekte. Diese Ereignisse werden typischerweise in einer Datenbank gespeichert. Das Besondere hierbei ist, dass Ereignisse weder geändert noch gelöscht, sondern nur hinzugefügt werden können. Diese Datenbank ist daher unveränderbar und wird damit als only-source-of-truth bezeichnet. Durch diese Art der Speicherung kann sowohl der aktuelle als auch ältere Zustände jedes Objektes reproduziert werden. Aus Performancegründen macht es Sinn eine weitere Datenbank einzusetzen, welche die Ereignisse aus der ersten Speicherquelle nutzt, um materialisierte Sichten zu speichern. Diese Sichten dienen zur Unterstützung der Performance von Nutzeranfragen. Darüber hinaus können Snapshots abgespeichert werden, damit nicht jedes Mal der Zustand eines Objektes von Grund auf neu berechnet werden muss. Die Datenbank kann dabei auch flüchtig sein, wie beispielsweise eine in-memory-Datenbank, weil sich der Zustand der Entitäten zu jeder Zeit neu berechnen lässt. Event-Sourcing wird üblicherweise in Verbindung mit dem CQRS-Muster eingesetzt. Zusammenfassend werden bei diesem Ansatz Schreib- und Leseoperationen voneinander getrennt. So werden zustandsändernde Ereignisse an die erste Datenbank geschrieben und Leseoperationen werden von der zweiten Datenbank verarbeitet [21] S.353. Die folgende Abbildung beschreibt zusammenfassend eine Architekturlösung für Echtzeitdatenverarbeitungssysteme unter Berücksichtigung der Ereignisgesteuerten Architektur in Verbindung mit dem Event-Sourcing- und dem CQRS-Muster. Zu sehen sind dabei die Datenquellen, aus denen die Ereignisse generiert werden. Diese werden über einen Nachrichtenübermittlungsdienst weitergeleitet an eine Datenbank. Diese Datenbank dient als Ereignisprotokoll, ist unveränderbar und gilt als only-source-of-truth. Sobald neue Ereignisse auftreten, wird dadurch in der Regel das Bearbeitungssystem ausgelöst, dass diese die neuen Ereignisse aus der Datenbank ausliest und weiterverarbeitet. Darüber hinaus werden materialisierte Sichten erstellt und in einem flüchtigen Speicher abgelegt. Nutzer und andere Systeme wie Analyseplattformen nutzen nun die auf Performance optimierten Daten für weitere Zwecke. Die Verantwortlichkeiten werden hier klar getrennt. So werden einkommende Ereignisse in die erste Datenbank geschrieben und die Interessenshalter oder auch Abonnenten erhalten lesenden Zugriff auf die zweite Datenbank.    Abbildung : Event-Sourcing Architektur Implementation einer Event-Sourcing Strategie in AzureKomponenten in Azure Die zuvor definierte Event-Sourcing Architektur soll nun in die Cloud projiziert werden. Als Cloudanbieter wird Microsoft Azure eingesetzt. Für jede Komponente werden passende von Azure bereitgestellte Dienste gesucht. Anschließend werden diese Dienste in Bezug auf die bereits definierten Anforderungen evaluiert und daraufhin eine Referenzarchitektur in Azure aufgestellt.  *Ereignisgeneratoren:* Bei den Datenquellen im Motorsportumfeld handelt es sich um Informationen von außen. Das sind Sensor- und Telemetriedaten, Wetterdaten aber auch Daten von anderen externen Quellen wie dem Timing Server. Diese müssen nun aus externen Geräten an die Cloud übermittelt werden, damit diese zur weiteren Verarbeitung zur Verfügung stehen. Die Informationen werden dazu an ein IoT-Gateway geschickt und können von dort dann an Ressourcen der Cloud gesendet werden. Das IoT-Gateway stellt dabei den Vermittler zwischen den IoT-Geräten und der Cloud dar. Als direkte Schnittstelle zur Cloud werden die Dienste IoT-Hub oder Event-Hub eingesetzt. Diese sind dafür konzipiert hohe Mengen an einkommenden Daten von externen Geräten zu verwalten. Durch diesen Weg können die Daten in der Cloud weiterverarbeitet werden [22] S.20.  *Nachrichtenkanal*: Bei hohen Mengen an einkommenden Datenströmen empfiehlt es sich Ereignisse über einen Nachrichtenkanal zu vermitteln, der mit diesen hohen Datenzufluss umgehen kann und einkommende Ereignisse ggf. sortiert. Darüber hinaus kann gesteuert werden, wohin die Daten von dort gesendet werden. Azure stellt dafür drei Dienste bereit die Abhilfe schaffen. Das sind der Azure Service Bus, Azure Event Hub und der Azure Event Grid. Der Event Grid tauscht nicht die eigentlichen Daten, sondern nur Statusänderungen der angebundenen Dienste aus. Beispielsweise kann der Event Grid ausgelöst werden, wenn sich ein neues Gerät mit dem IoT Hub verbunden hat. Der Azure Service Bus stellt eine alternative Lösung dar, weil hierdurch die eigentlichen Nachrichten zwischen Diensten ausgetauscht werden. Der Datentransfer erfolgt nach dem first-in-first-out-Prinzip. Es wird garantiert, dass Informationen, die als erstes den Dienst erreichen diesen auch als Erstes wieder verlassen. Darüber hinaus unterstützt der Dienst Transaktionen, Filterungen und es wird garantiert, dass keine Daten verloren gehen. Zu guter Letzt gibt es noch den Event Hub. Dieser wurde für die Erfassung und Verarbeitung von Datenströmen konzipiert. Es handelt sich hierbei um einen Nachrichtendienst, der es erlaubt hohe Mengen an Daten wie zum Beispiel Telemetriedaten aus verschiedenen Datenquellen zu verarbeiten [23] S.317.  *Ereignisspeicher*: Ein Ereignisspeicher ist im Grunde eine Datenbank, die Ereignisse persistiert. Dadurch kommen erstmal viele Dienste von Azure als Speichermedium infrage. Je nach Anforderungen und Kompatibilität mit anderen Diensten, wird die Auswahl jedoch eingeschränkt. So eignet sich die Azure SQL-Server Lösung als relationale Datenbank. Azure Storage untergliedert sich in Azure Table Storage als NoSQL-Datenbank und Azure Blob Storage für das Ablegen jeglicher Art von Dateien wie JSON-Dokumente, Text- oder auch Audiodateien. Azure Table Storage gilt als die kostengünstigste Variante für NoSQL-Datenbanken kann aber nur nach Partitions- und Zeilenschlüssel indexiert werden. Sobald ein zweiter Index erstellt werden soll, muss dieser eigenständig programmiert werden. Als Alternative für Azure Table Storage gibt es außerdem Azure CosmosDB. Hier werden die Daten in Form von JSON-Dokumenten abgespeichert. Azure Event Hub kann ebenfalls als Speichermedium gesehen werden, da es die einkommenden Ereignisse bis zu 30 Tagen speichern kann. Das hat den Vorteil, dass kein weiterer Service in Anspruch genommen werden muss, sofern 30 Tage für den entsprechenden Anwendungsfall ausreichend sind [24] S.68-73.  *Bearbeitungssystem*: Wenn es zur weiteren Bearbeitung der Daten kommt, ist es in der Regel sehr schwer einen geeigneten Dienst dafür zu finden. Denn hierbei fließt die Geschäftslogik mit ein, somit kann das System je nach Anforderungen stark variieren. Eine Möglichkeit ist es, ein eigenständiges System zu entwickeln das Schnittstellen zur Cloud bietet, sodass Ereignisse über den Datenkanal an die Bearbeitungsplattform geschickt werden. Eine andere Option bietet Azure Functions. Das ist ein serverloser Dienst. Das bedeutet der Entwickler kann sich vollkommen um die Entwicklung der bestimmten Anwendung kümmern. Die Architektur und Infrastruktur übernimmt dabei Azure Functions [25] S.3. So kann reiner Code in einen Azure Dienst gegossen und so zum Laufen gebracht werden.  *Flüchtiger Speicher*: Die verarbeiteten Daten gilt es nun in einem weiteren Speicher abzulegen. Der Speicher muss nicht fähig sein, die Daten dauerhaft persistieren zu können. Das bedeutet, wenn die Stromversorgung unterbrochen wird, gehen die Daten verloren. Die Auswertungen können jedoch zu jederzeit reproduziert werden, da alle benötigten Informationen im Ereignisspeicher abrufbar sind. Aufgrund dessen, können Informationen so schnell und effizient wie möglich abgerufen werden, da die abgelegten Daten im Arbeitsspeicher vorliegen und nicht erst von der Festplatte gelesen werden müssen. Das ist besonders performant, wenn in Situationen, in denen fortlaufend dieselben Daten abgefragt werden. Um einen flüchtigen Speicher in die eigene Anwendung zu implementieren, muss zunächst identifiziert werden, welche Daten wiederholt und oft abgefragt werden. Denn nicht jede Information ist gleich relevant und aufgrund der Tatsache, dass der Arbeitsspeicher begrenzt ist, können nicht alle Daten hier gehalten werden. Azure bietet neben den bereits oben beschriebenen Speichermöglichkeiten darüber hinaus noch den Azure Redis Cache Dienst. Dieser ist eine NoSQL Key-Value-Datenbank und ist die Azure eigene Lösung für die Open-Source Variante von Redis [24] S.260.  *Abonnenten*: Die vorverarbeiteten Daten können nun für die Abonnenten zugänglich gemacht werden. Das können externe Systeme, Datenbanken oder Analyseplattformen sein. Als Speichermedium bietet Azure wie bereits erwähnt einige Dienste an. Auch für Analyseplattformen gibt es Dienste wie Azure Stream Analytics, die zur Analyse von großen Mengen an Ereignisströmen dienen [22] S.33. Außerdem können auch hauseigene Systeme und externe Ressourcen, mit passenden Schnittstellen zur Cloud, über die optimierten Daten informiert werden.    Abbildung 6: Echtzeitdatenverarbeitung in Azure Auswahl der Komponenten Um Ereignisse von externen Quellen und IoT-Geräten entgegennehmen zu können eignen sich sowohl der IoT Hub als auch der Event Hub. IoT Hubs können darüber hinaus externe IoT Geräte verwalten und haben weitere Fähigkeiten, so unterstützen sie beispielsweise unterschiedliche Transportprotokolle im Gegensatz zum Event Hub [26] S.20. Da in vielen Anwendungsfällen aber besonders auch im Motorsportumfeld auf viele unterschiedliche IoT Geräte zugegriffen wird, ist es wichtig diese verwalten und kontrollieren zu können. Auf diese Weise kann beispielsweise geprüft werden, welche Geräte derzeit verfügbar sind und mit welchen Geräten sich der IoT Hub nun verbinden soll. Sind die externen Geräte erst einmal mit der Cloud verbunden, können nun die eigentlichen Ereignisse ausgetauscht werden. Im Motorsportumfeld spielen Wetterdaten, Sensor- und Telemetriedaten des Rennautos, sowie andere Daten von Geräten an der Rennstrecke wie dem Timing Server eine wichtige Rolle. Somit besteht eine Flut aus Informationen, die so schnell wie möglich verarbeitet und den Renningenieuren zugänglich gemacht werden müssen. Für diese Anforderung eignet sich der Azure Event Hub Dienst. Dieser gilt als Big Data Pipeline und kann Datenströme in Echtzeit entgegennehmen und verarbeiten. Darüber hinaus kann es Sinn machen zunächst Azure Event Grid mit dem IoT Hub zu verbinden, um die Konnektivität der externen Geräte zu prüfen. So wird kontrolliert welche Geräte verfügbar und gerade verbunden sind. Darauf erfolgt dann erst der eigentliche Datentransfer an den Event Hub. DatengrundlageProgrammierungVerwendete DiensteHandhabung des ProgrammsAlternative LösungsansätzeEvaluationErgebnisseSchlussfolgerungZusammenfassungAusblick |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | J. Repschläger, D. Pannicke und R. Zarnekow, „Cloud Computing: Definitionen, Geschäftsmodelle und Entwicklungspotenziale,“ in *HMD Praxis der Wirtschaftsinformatik*, Berlin, DE, Springer Nature, 2010, pp. 6-15. |
| [2] | D. Reilly, C. Wren und T. Berry, „Cloud Computing: Pros and Cons for Computer Forensic Investigations,“ International Journal Multimedia and Image Processing, Liverpool, UK, 2011. |
| [3] | A. Haas und A. Hofmann, „Risiken aus der Nutzung von Cloud-Computing-Diensten: Fragen des Risikomanagements und Aspekte der Versicherbarkeit,“ in *Zeitschrift für die gesamte Versicherungswissenschaft*, Berlin, DE, Springer-Verlag, 2014, pp. 377-407. |
| [4] | B. Erb und F. Kargl, „Combining Discrete Event Simulations and Event Sourcing,“ ICST, Ulm, DE, 2014. |
| [5] | S. Copei und A. Zündorf, „Commutative Event Sourcing vs. Triple Graph Grammars,“ Kassel, DE, 2021. |
| [6] | S. Chiusano, T. Cerquitelli, R. Wrembel, K. Norvag, B. Catania, G. Vargas-Solar und E. Zumpano, Communications in Computer and Information Science. New Trends in Database and Information Systems, Cham, CH: Springer Nature, 2022. |
| [7] | V. Khononov, Learning Domain-Driven Design. Aligning Software Architecture and Business Strategy, Kalinfornien, US: O'Reilly Media, 2021. |
| [8] | S. Diakov, T. Zubrei und A. Samoidiuk, „Application of Event Sourcing and CQRS Patterns in distributed Systems,“ 2019. |
| [9] | M. Pantelelis und C. Kalloniatis, „Object Relational Mapping Vs. Event Sourcing: Systematic Review,“ in *Electronic Government and the Information Systems Perspective*, Vienna, Austria, 2022. |
| [10] | M. Feick, N. Kleer und M. Kohn, „Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa,“ Gesellschaft für Informatik, Bonn, DE, 2018. |
| [11] | S. Nastic, T. Rausch, O. Scekic, S. Dustar, M. Gusev, M. Koteska, B. Jakimovski, S. Ristov und R. Prodan, „A Serverless Real-Time Data Analytics Platform for Edge Computing,“ *IEEE Internet Computing,* Nr. 4, pp. 64-71, 2017. |
| [12] | R. Tönjes und e. al., „Real Time IoT Stream Processing and Large-scale Data Analytics for Smart City Applications,“ in *Proc. Eur. Conf. Netw. Commun.*, Bologna, IT, 2014. |
| [13] | M. Hermann, T. Pentek und B. Otto, „Design Principles for Industrie 4.0 Scenarios: A Literature Review,“ Business Engineering Institute, St. Gallen, CH, 2015. |
| [14] | M. Brenner und M. Garschhammer, „Requirements Engineering und IT Service Management. Ansatzpunkte einer integrierten Sichtweise,“ Munich Network Management Team, Munich, DE, 2006. |
| [15] | U. Kusay-Merkle, Agiles Projektmanagement im Berufsalltag. Für mittlere und kleinere Projekte, Berlin, DE: Springer-Verlag, 2018. |
| [16] | S. Kleuker, Grundkurs Softwareengineering mit UML. Der pragmatische Weg zu erfolgreichen Softwareprojekten. 3.Auflage, Weisbaden, DE: Springer-Verlag, 2013. |
| [17] | X. Liu, N. Iftkhar und X. Xie, „Survey of Real-time Processing Systems for Big Data,“ in *Proceedings of the 18th International Database Engineering & Applications Symposium*, Porto, Portugal, 2014. |
| [18] | B. Erb und F. Kargl, „Combining Discrete Event Simulations and Event Sourcing,“ in *Seventh International Conference on Simulation Tools and Techniques*, 2014. |
| [19] | A. Pogiatzis und G. Samakovitis, „An event-driven serverless ETL pipeline on AWS,“ *Applied Sciences,* Bd. 11, Nr. 1, p. 191, 2020. |
| [20] | B. M. Michelson, „Event-driven architecture overview,“ *Patricia Seybold Group,* Bd. 2, Nr. 12, pp. 10-1571, 2006. |
| [21] | B. Erb und F. Kargl, „A conceptual model for event-sourced graph computing,“ in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, 2015, pp. 352-355. |
| [22] | R. Stackowiak, Azure Internet of Things Revealed. Architecture and Fundamentals, Apress, 2019. |
| [23] | J. Bruce, „Azure Tooling,“ in *Essential Visual Studio 2019*, Berkeley, CA, Apress, 2020, pp. 297-328. |
| [24] | R. Reagon, „Azure Data Storage Overview,“ in *Web Applications on Azure*, Berkeley, CA, Apress, 2018, pp. 61-76. |
| [25] | R. Sawhney, Beginning Azure Functions. Building Scalable and Serverless Apps, Berkeley, CA: Apress, 2019. |
| [26] | B. Erb und F. Kargl, „A conceptual model for event-sourced graph computing,“ in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, 2015, pp. 352-355. |

# Abkürzungsverzeichnis

# [Anhang]