Prof. Dr. Bernhard Hollunder
Fakultät Informatik
bernhard.hollunder@hs-furtwangen.de

Hochschule Furtwangen
Robert-Gerwig-Platz 1
D-78120 Furtwangen

Wintersemester 2024/25

Streaming Systems

Praktikum

Bearbeitungszeitraum: 10. Oktober 2024 – 23. Januar 2025

Version vom 24. September 2024

Rahmenbedingungen

In dem Praktikum "Streaming Systems" geht es vor allem um das praktische Arbeiten mit den in der Vorlesung vorgestellten Konzepten, Technologien und Frameworks. Neben der praktischen Umsetzung der gestellten Aufgaben sollen Sie Ihre Lösungen auch in schriftlicher Form dokumentieren. Erstellen Sie hierzu ein Dokument, in dem Sie die gewählten Lösungsansätze und erzielten Ergebnisse aufgabenweise darstellen und bewerten. Sie können sich an folgenden Fragen orientieren:

- Welche Lösungs- und Umsetzungsstrategie wurde gewählt?
- Wie schätzen Sie die Leistungsfähigkeit der Lösung ein? Wurden neben der Basisfunktionalität zusätzliche Funktionen realisiert?
- Welche nicht-funktionalen Anforderungen (z.B. Skalierbarkeit und Durchsatz) wurden untersucht? Mit welchem Ergebnis?
- Wie haben Sie sich von der Korrektheit und Vollständigkeit der Lösung überzeugt? Gibt es eine systematische Teststrategie?
- Wie können die berechneten Ergebnisse auf geeignete Weise dargestellt werden? Gibt es naheliegende Visualisierungsmöglichkeiten?
- Welche zusätzlichen Frameworks und Bibliotheken wurden eingesetzt? Warum?

Beachten Sie, dass einige Aufgaben bewußt nicht vollständig spezifiziert wurden. Seien Sie kreativ und schließen Sie mögliche Lücken auf sinnvolle Weise.

Die Aufgaben sollten in Zweierteams bearbeitet werden.

Die Abschlusspräsentationen aller Lösungen finden gruppenweise nach individueller Absprache ab dem 27. Januar statt. Präsentationen von Zwischenergebnissen erfolgen in den Praktikumsstunden.

Die Gesamtnote für das Praktikum setzt sich folgendermaßen zusammen: 2/3 lauffähige Lösungen der Aufgaben und 1/3 Dokumentation. Achten Sie auf "Lesbarkeit" der Dokumentation sowie auf Berücksichtigung der oben formulierten Fragen.

Aufgabenübersicht

Die angegebenen Bearbeitungstermine dienen zur zeitlichen Orientierung.

1. Installation von Apache ActiveMQ und Apache Kafka sowie *Getting Started*, 10. Oktober
2. Event Sourcing, Kernfunktionalität, 17. und 24. Oktober

3. Event Sourcing, erweiterte Funktionalität, 31. Oktober
4. Event Sourcing, Laufzeitmessungen, 7. November
5. Event Sourcing, Apache Kafka als *Event Store*, 14. und 21. November
6. Datengenerator für Datenanalyse "Verkehrsüberwachung", 28. November
7. Datenanalyse "Verkehrsüberwachung" mit Apache Beam, 5. und 12. Dezember
8. Complex Event Processing "Verkehrsüberwachung" mit EPL und Esper, 19. Dezember und 9. Januar
9. Read-Process-Write-Pattern (optional)
10. Vergleichende Analyse - Lessons Learned, 16. und 23. Januar

Aufgabe 1 [10. Oktober]

In den kommenden Praktikumsaufgaben werden u.a. die Message Broker *Apache ActiveMQ* und *Apache Kafka* eingesetzt. In dieser Aufgabe sollen Sie diese Systeme installieren und sich mit der Installation bzw. Konfiguration vertraut machen. Nutzen Sie hierzu einschlägige Beschreibungen[1].

Nach der Installation der Systeme überprüfen Sie die Korrektheit der Installation, in dem Sie jeweils eine einfache Anwendung "zum Laufen bringen". Ein tieferes Verständnis der Systeme ist hierzu nicht erforderlich. Dieses wird in den nachfolgenden Vorlesungs- und Praktikumsstunden aufgebaut.

Aufgabe 2 [17. und 24. Oktober]

Ein elektrisch angetriebenes Fahrzeug kennt seinen Startpunkt und die jeweils relative Bewegung zum letzten Standort. Dies kann beispielsweise über die Rotation der Räder abgeleitet werden. Einfachheitshalber befindet sich das Fahrzeug auf einem Punkt in einer Ebene (d.h. in einem 2D-Koordinatensystem). Der Weg von dem aktuellen Punkt des Fahrzeuges zum nächsten wird als Bewegungsvektor $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$ dargestellt. Durch diesen Vektor kann ausgehend von der aktuellen Position $A = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ die nächste Position B durch Vektoraddition $B = \begin{pmatrix} x_1 + x \\ y_1 + y \end{pmatrix}$ bestimmt werden.

Das Fahrzeug kennt selbst nicht seine absolute Position, meldet kontinuierlich die Bewegungsvektoren seiner Fahrt an ein weiteres System zur Auswertung. In dem folgenden Szenario bewegen sich mehrere Fahrzeuge, die jeweils einen eindeutigen Namen haben.

Zu den Schnittstellen:

(1) Die Position eines Fahrzeuges bzw. ein Bewegungsvektor wird durch folgenden Typ repräsentiert:

```
public record Position(int x, int y) implements Serializable {
}
```

(2) Die folgende Schnittstelle dient dazu, Fahrzeuge zu erstellen und diese wieder zu löschen. Auch wird die Schnittstelle von Fahrzeugen regelmäßig genutzt, um ihre relativen Bewegungsdaten zu melden:

[1] Vgl. `https://www.mastertheboss.com/jboss-frameworks/activemq/getting-started-with-activemq-artemis/`, `https://kafka.apache.org/quickstart`.

```
public interface VehicleCommands {
   void createVehicle(String name, Position startPosition) throws Exception;
   void moveVehicle(String name, Position moveVector) throws Exception;
   void removeVehicle(String name) throws Exception;
}
```

Verändert ein Fahrzeug seine Position, so wird dies über die `moveVehicle(...)`-Methode unter Angabe des Fahrzeugnamens und der relativen Position gemeldet.

(3) Über eine Query-Schnittstelle können Informationen zu den Fahrzeugen abgefragt werden. Es ist auch möglich, die Fahrzeuge zu bestimmen, die sich aktuell auf einer bestimmten Position befinden.

```
public interface Query {
   public VehicleDTO getVehicleByName(String name);
   public Enumeration<VehicleDTO> getVehicles();
   public Enumeration<VehicleDTO> getVehiclesAtPosition(Position position);
}
```

(4) Beachten Sie, dass für die Rückgabe von Daten auf der Query-Seite der Typ `VehicleDTO` verwendet wird. Dieser Typ wird ausschließlich auf der *Read Side* verwendet, um Anfragen an Fahrzeuge und deren Positionen zu beantworten. `VehicleDTO` ist ein "reines" Datenobjekt und hat abgesehen von *Gettern* keine weiteren Funktionalitäten. `getPosition()` liefert die aktuelle Position des Fahrzeugs. Wie oft sich ein Fahrzeug bewegt hat, kann über `getNumberOfMoves()` abgefragt werden.

```
public interface VehicleDTO {
   public String getName();
   public Position getPosition();
   public int getNumberOfMoves();
}
```

Erstellen Sie – basierend auf diesen Schnittstellen-Vorgaben – eine *Event Sourcing* Anwendung gemäß der Komponenten der Grobarchitektur von Folie "CQRS with Event Sourcing". Hierzu können Sie wie folgt vorgehen:

- Erstellen Sie eine oder mehrere Klassen für die erforderlichen *Commands* sowie den *Command Handler*. Sie benötigen ein Domänenmodell, um Validierungen durchführen zu können. Soll beispielsweise ein weiteres Objekt mit einem bereits vergebenen Namen angelegt werden, so wird dieser Befehl nicht ausgeführt. Wird ein Befehl nicht ausgeführt bzw. abgelehnt, so soll dies über eine Exception dem Aufrufer mitgeteilt werden. Ein `moveVehicle(...)`-Befehl soll abgelehnt werden, wenn der Bewegungsvektor der Null-Vektor ist.

- Einfachheitshalber soll in dieser Aufgabe das Domänenmodell nicht auf Basis der im *Event Store* abgelegten Ereignisse aufgebaut werden. Stattdessen kann eine Map zur Verwaltung der aktuell vergebenen Fahrzeugnamen verwendet werden, weitere Informationen werden im Domänenmodell (zunächst) nicht benötigt. Wird ein Kommando akzeptiert, werden die erforderlichen Ereignisse erzeugt. Ein Beispiel für ein Ereignistyp ist etwa `VehicleCreatedEvent`. Im einfachsten Fall wird ein Kommando auf ein entsprechendes Ereignis abgebildet. Je nach Kommando muss auch das Domänenmodell aktualisiert werden.

- Die erzeugten Ereignisse werden im *Event Store* abgelegt. Nutzen Sie ein JMS Messaging System wie etwa *Apache ActiveMQ* zur Verwaltung der Ereignisse. Der *Command*

Handler nimmt hierbei die Rolle des Nachrichtenproduzenten ein. Wird ein Befehl angenommen und durchgeführt, werden hieraus Ereignisse abgeleitet, die dem *Event Store* übergeben werden. Die Projektion konsumiert die erzeugten JMS-Ereignisse und aktualisiert das *Query Model*. Beachten Sie, dass in einer zukünftigen Aufgabe *Apache ActiveMQ* durch *Apache Kafka* ersetzt wird. Beim Schnittstellenentwurf sollten Sie hierauf bereits Rücksicht nehmen.

- Implementieren Sie die `Query` Schnittstelle. Hierzu benötigen Sie einen *Query Handler* als eigenständige Komponente. Überlegen Sie sich eine geeignete Datenstruktur für das zugrundeliegende *Query Model*. Das *Read Repository* kann beispielsweise Maps verwalten, um Anfragen auf einfache Weise beantworten zu können.
- Nun benötigen Sie noch eine Projektion, die die im *Event Store* eingegangenen Ereignisse auswertet und das *Query Model* aktualisiert.

Erstellen Sie eine Client-Anwendung, die über die Schnittstelle `VehicleCommands` Fahrzeug-Objekte erzeugt und deren Positionen über `moveVehicle(...)` ändert. Auch sollten Fahrzeuge entfernt werden.

Achten Sie auf eine strikte Trennung zwischen dem Domänenmodell und dem *Query Model*, sodass beide Komponenten unabhängig voneinander weiterentwickelt werden können. Überlegen Sie sich, wie Sie auf sinnvolle Weise die Korrektheit und Vollständigkeit Ihrer Lösung prüfen können. Setzen Sie nachfolgend das Testkonzept um.

Aufgabe 3 [31. Oktober]

Erweitern Sie Ihre Lösung von Aufgabe 2 dahingehend, dass folgendes Verhalten umgesetzt wird.

1. Nach einer bestimmten Anzahl von Bewegungen (z.B. 20) eines Fahrzeuges soll dieses entfernt werden. Wird also das zwanzigste Mal der `moveVehicle(...)` Befehl für ein Objekt angewendet, wird diese Bewegung nicht ausgeführt. Stattdessen wird das Fahrzeug gelöscht, der Name wird freigegeben und kann nachfolgend wieder vergeben werden. Überlegen Sie sich, wie das Domänenmodell auf einfache Weise erweitert werden kann, um diese Funktionalität umzusetzen.
2. Bewegt sich ein Fahrzeug auf eine Position, auf die es bereits gewesen ist, soll das Fahrzeug entfernt werden. Der Name des Fahrzeugs wird freigegeben und kann nachfolgend wieder vergeben werden. Erweitern Sie das Domänenmodell, so dass diese Funktionalität umgesetzt werden kann.
3. Betrachten Sie nun folgende Erweiterung. Bei der Durchführung von `moveVehicle(...)` bei einem Fahrzeug soll überprüft werden, ob sich auf der neuen Position bereits ein Fahrzeug befindet. In diesem Fall soll letzteres entfernt werden. Wie kann diese Funktionalität umgesetzt werden? Ist es sinnvoller das Domänenmodell geeignet anzupassen oder sollte eher die Query-Schnittstelle genutzt werden? Prüfen Sie die Möglichkeiten und setzen Sie eine Variante um.

Aufgabe 4 [7. November]

Führen Sie Laufzeitmessungen durch. Wie lange dauert es, bis das *Query Model* aktualisiert ist, nachdem ein `moveVehicle(...)` Befehl aufgerufen wurde? Ermitteln Sie hierbei auch die durchschnittliche Zeit für den Transport eines Ereignisses vom Produzenten zum Konsumenten, d.h. der Projektion.

Optional: Setzen Sie sowohl das *Polling*- als auch *Push*-Modell auf der Konsumentenseite um. Können Sie Laufzeitunterschiede feststellen?

Prof. Dr. Bernhard Hollunder
Fakultät Informatik
bernhard.hollunder@hs-furtwangen.de

Hochschule Furtwangen
Robert-Gerwig-Platz 1
D-78120 Furtwangen

Aufgabe 5 [14. und 21. November]

Setzen Sie nun *Apache Kafka* zur (persistenten) Speicherung der Ereignisse im *Event Store* ein.

Des Weiteren soll nun das Domänenmodell auf Basis der im *Event Store* gespeicherten Ereignisse dynamisch bei der Abarbeitung eines Befehls aufgebaut werden. Realisieren Sie entsprechende *Event Store*-Abfragen wie etwa `loadNamesOfMovingVehicles()` als Apache Kafka Konsumenten. Dies bedeutet auch, dass die in den Aufgaben 2 - 4 verwendete Map zur Speicherung der Objektnamen auf der *Write Side* nicht mehr benötigt wird.

Aufgabe 6 [28. November]

In den folgenden Aufgaben soll mit Hilfe unterschiedlicher Programmiermodelle die Verarbeitung kontinuierlicher Datenströme untersucht werden. Hierzu wird ein Szenario aus der Domäne "Verkehrsüberwachung" betrachtet. Zum einen sind die zu lösenden Aufgaben in diesem Kontext schnell zu verstehen, ohne dass tiefgehendes Domänenwissen aufgebaut werden muss. Zum anderen sind die durchzuführenden Datenanalysen und -aufbereitungen repräsentativ für eine Vielzahl anderer Fachdomänen.

Entwickeln Sie zunächst einen Testdatengenerator, der Datensätze mit folgender Struktur erzeugt: `timestamp id val-0, val-1, val-2, ..., val-n`.

Beispiel:

```
2024-10-23T10:01:29.551Z 2 20.4,33.5,40.0
2024-10-23T10:01:40.895Z 1 35.6,-17.4,28.6
2024-10-23T10:01:54.759Z 2 42.1,34.7
2024-10-23T10:02:10.452Z 1
2024-10-23T10:02:20.596Z 3 28.4,21.1
...
```

Eine Zeile soll folgendermaßen interpretiert werden: Der führende Zeitstempel gibt den Zeitpunkt der Datenerzeugung an. Die folgende Zahl legt den eindeutigen Namen eines Sensors fest. Die nachfolgenden Gleitkommazahlen sind die zu einem Zeitpunkt gemessenen Geschwindigkeitswerte des Sensors in der Einheit Meter pro Sekunde (m/s). Die Anzahl der Sensoren auf einem Streckenabschnitt sowie die Anzahl der zu einem Zeitpunkt zu erzeugenden Geschwindigkeitswerte sowie die minimale und maximale Geschwindigkeit soll konfigurierbar sein. Ebenso kann die Taktung, also der zeitliche Abstand zwischen zwei erzeugten Datensätzen, durch zwei `int` Zahlen `m1` und `m2` eingestellt werden. Dies bedeutet, dass der nächste Datensatz mindestens `m1` und höchstens `m2` Millisekunden nach dem vorhergehenden erzeugt wird. Es sollen gelegentlich auch negative Geschwindigkeitswerte produziert werden.

Die Datensätze sollen in einen *Apache Kafka* Topic eingestellt werden.

Aufgabe 7 [5. und 12. Dezember]

In dieser Aufgabe soll die Durchschnittsgeschwindigkeit in der Einheit km/h für jeden Sensor in Zeitfenstern vorgegebener Länge (z.B. im 30 Sekundentakt) ermittelt werden.

Die berechneten Werte können nun genutzt werden, um

1. den zeitlichen Verlauf der Durchschnittsgeschwindigkeit an einer Messstelle (d.h. eines Sensors) sowie
2. die Durchschnittsgeschwindigkeiten auf einem Streckenabschnitt (definiert durch eine Folge von Sensoren) zu einem bestimmten Zeitpunkt

zu ermitteln.

Ein Datensatz ohne Geschwindigkeitswert (vgl. Zeile 4 im obigen Beispiel) soll nicht berücksichtigt werden. Eine negative Geschwindigkeit (Messfehler) soll ebenso von der weiteren Verarbeitung ausgeschlossen werden.

Erstellen Sie auf Basis des Pipeline-/Transforms-Programmiermodells eine Anwendung, die die Datensätze aus dem *Apache Kafka* Topic konsumiert und die geforderten Berechnungen durchführt. Nutzen Sie hierzu *Apache Beam*. Eine *Apache Beam* Pipeline soll die Datensätze über den KafkaIO-Adapter einlesen und nachfolgend aufbereiten, bereinigen sowie aggregieren.

Hinweise zum Vorgehen:

- Definieren Sie geeignete `PCollection`s und `Transform`s zur Berechnung der geforderten Ausgabe.
- Es bietet sich an, mit `Transform`s wie etwa `GroupByKey`, `Combine` und `Window` zu arbeiten.
- Der Typ `IntervalWindow` hat die Methoden `start()` und `end()` zur Abfrage des Start- und Endzeitpunkts.

Überlegen Sie sich eine geeignete intuitive textbasierte Ausgabe der berechneten Ergebnisse und implementieren Sie diese. Ergänzend können Sie eine graphische Darstellung (beispielsweise mit Grafana) realisieren.

Wie können Sie die Korrektheit Ihrer Lösung nachweisen?

Aufgabe 8 [19. Dezember und 9. Januar]

Die Kernfunktionalität von Aufgabe 7 soll nun mittels CEP umgesetzt werden. Ergänzend sollen Sie "komplexe" Ereignisse identifizieren, definieren und umsetzen. Verwenden Sie hierzu das System Esper und EPL.

Erstellen Sie entsprechende EPL-Anfragen zur Datenbereinigung und Berechnung der Durchschnittsgeschwindigkeit.

Nachdem die Durchschnittsgeschwindigkeit für die Sensoren pro Zeitfenster berechnet wurden, sollen diese jeweils als (aggregiertes) Ereignis ist das System eingestellt werden. Definieren Sie hierzu einen entsprechenden Ereignistyp und verwenden Sie die `insert into` Klausel, um Instanzen von diesem Typ zu erzeugen.

Diese erzeugten Ereignisse sollen nun wie folgt weiter verarbeitet werden: Kommt es innerhalb einer bestimmten Zeit (z.B. 30 Sekunden) zu einem starken Abfall der Durchschnittsgeschwindigkeit, soll ein weiteres Ereignis erzeugt werden, welches auf eine mögliche Stauentwicklung hinweist. Definieren Sie hierzu einen weiteren Ereignistyp.

Überlegen Sie sich geeignete Testfälle, wie Sie die Korrektheit Ihrer EPL-Anfragen überprüfen können.

Aufgabe 9 (optional)

Implementieren Sie das Read-Process-Write-Pattern mit *Apache Kafka*. Betrachten Sie insbesondere Fehlersituationen, die zeigen, dass alle Datensätze tatsächlich genau einmal verarbeitet werden.

Aufgabe 10 [16. und 23. Januar]

In den bisherigen Praktikumsaufgaben wurden vorgegebene Aufgabenstellungen mit unterschiedlichen Technologien und Frameworks bearbeitet. Das Ergebnis war jeweils eine lauffähige Implementierung. Hierbei haben Sie das jeweilige Programmiermodell angewendet und die Arbeitsweise bzw. Fähigkeiten der Technologien praktisch erproben können.

In dieser abschließenden Aufgabe sollen Sie ein noch tieferes Verständnis bezüglich der in der Vorlesung eingeführten Konzepte, Technologien, Frameworks und Programmiermodelle erwerben.

Konkret sollen Sie folgende Systeme

- Java Messaging System (JMS)
- Apache Kafka
- Apache Beam
- Complex Event Processing (CEP) mit EPL / Esper

miteinander vergleichen. Damit verbunden sollen Sie eine Bewertung der aus Ihrer Sicht jeweiligen Stärken und Schwächen vornehmen bzw. den jeweiligen Erfüllungsgrad von Kriterien einschätzen.

Bei der Bewertung können Sie sich anhand der folgenden Kriterien orientieren:

1. Repräsentation von Ereignissen: Wie und mit welchen Datenstrukturen werden Ereignisse repräsentiert? Wie flexibel schätzen Sie die Datenstrukturen ein? Werden Vererbungsstrukturen unterstützt? Welche Metainformationen werden mitgeführt?
2. Wie werden Ereignisse / Nachrichten erzeugt und wie können diese dem Broker übermittelt werden? Welche Schritte sind hierzu erforderlich? Können Ereignisse gebündelt übertragen werden?
3. Wie kann eine Konsumentenanwendung Ereignisse / Nachrichten von dem Broker entgegennehmen? Welche Stategien werden hierzu angeboten?
4. Werden Ereignisse / Nachrichten dauerhaft gespeichert?
5. Welche Auslieferungs- und Verarbeitungsgarantien werden unterstützt?
6. Welche Maßnahmen müssen ergriffen werden, damit Ereignisse / Nachrichten gegen einen unberechtigten Zugriff gesichert werden können?
7. Welche Programmiermodelle werden angeboten, um eintreffende Nachrichten / Ereignisse zu verarbeiten? Werden zustandsbehaftete Transformationen wie etwa Aggregationsfunktionen unterstützt und können Nachrichten / Ereignisse zeitlich gruppiert werden? Ist eine Auswertung nach Ereigniszeit möglich?
8. Welche Skalierungsmöglichkeiten werden vom System unterstützt, um mitwachsen zu können hinsichtlich Datenvolumen, Datenfrequenz sowie Anzahl von Produzenten und Konsumenten?
9. Gibt es Konzepte hinsichtlich einer Ausfallsicherheit des Systems? Wenn ja, wie ist die grundsätzliche Arbeitsweise.
10. Typsicherheit: Wie typsicher ist das Programmiermodell? Wie können Fehler bei der Interpretation einer Nachricht erkannt werden? Z.B. erwartet ein Konsument einen Ereignistyp A, erhält jedoch eine Instanz vom Typ B.
11. In welchen Programmiersprachen können die Produzenten- und Konsumentenanwendungen erstellt werden?

Geben Sie auch eine persönliche Einschätzung zu den Stärken, Schwächen und dem zukünftigen Potenzial der genannten Technologien.

Ihre Darstellungen zu dieser Aufgabe 10 können Sie in einem separaten Pdf-Dokument hinterlegen oder in das Lösungsdokument der Aufgaben 1 - 9 aufnehmen.