Services allein entwickeln und verbinden

MEA, Workshop 4, 05.11.2026

Der Tag

10:00	Ankommen und Überblick
10:15	Teamrunde: einen Service allein entwickeln
11:00	Inhaltsimpuls: ein Service ohne Infrastruktur
12:00	Mittagspause
12:45	Teamarbeit mit dem Ziel: Ihr Service läuft lokal, die Tests sind grün
13:45	Inhaltsimpuls: die Variante mit Spring Events
14:00	Teamarbeit mit dem Ziel: Ihr Service ist lokal grün und vollständig
14:45	Inhaltsimpuls: die echte Kafka-Anbindung
15:30	Architekturdiskussion: Kafka-Anbindung ohne Java
15:45	Teamarbeit: die Grundlage für die Integration
16:15	Abschlussrunde und Hausaufgabe

Rückblick auf den 29.10. mit dem Tutor

- Tutor-Prompt für den Rückblick: archi-lab.io/tutor/mea_ws26/2026-10-29-rueckblick.txt
- auch auf der Kursseite, Workshop 05.11., Agenda 10:00
- alles kopieren, in die eigene KI einfügen
- der Tutor fragt nacheinander:
  - Was haben wir letztes Mal gemacht?
  - Was hat dir besonders gut gefallen?
  - Was hat dir nicht gefallen?
  - Worüber sollten wir noch einmal weiter reden?

> Ca. 10 Minuten - und am Ende Learnlog-Eintrag generieren lassen!

Teamrunde: einen Service allein entwickeln

- je Team ein Beispiel, ca. 10 Minuten
- gelöst oder gescheitert: beides zählt
- am Bildschirm, im Code

> - Wie testen Sie, ohne Kafka?
> - Wie kommen die Events der Nachbarn in Ihren Service?
> - Wie prüfen Sie, was Ihr Service veröffentlicht?
> - kurz: Wie weit sind Contract und Client?

Inhaltsimpuls: der Fahrradverleih

1 Core

zwei Aggregates, keine Ein- und Ausgänge, keine Tests

2 Interfaces und Tests

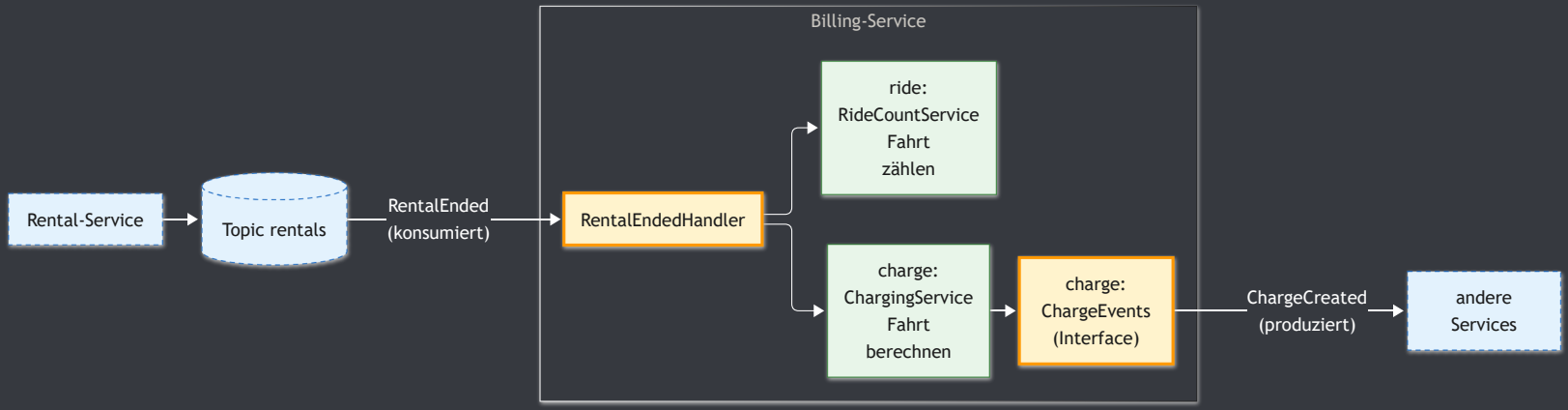
eingehende und ausgehende Events, Tests auf drei Ebenen, ohne Broker

3 Kafka

nur Adapter kommen dazu, heute Nachmittag

gitlab.com/archi-lab/public/learning/learn-event-driven-architecture

Ein Event hinein, ein Event hinaus



grün: aus Modul 1 · orange: neu in Modul 2 · gestrichelt: außerhalb, im Test gemockt

Schritt 1: der Core

```
thkoeln.archilab.servicedevelopment
├── charge
│   ├── domain        Charge, ChargeRepository
│   └── application   ChargingService
└── ride
    ├── domain        RideCount, RideCountRepository
    └── application   RideCountService
```

- je Aggregate `domain` und `application`, wie in ST2
- Preisregel in `Charge`: 50 Cent je angefangene Viertelstunde
- keine Events, keine Tests

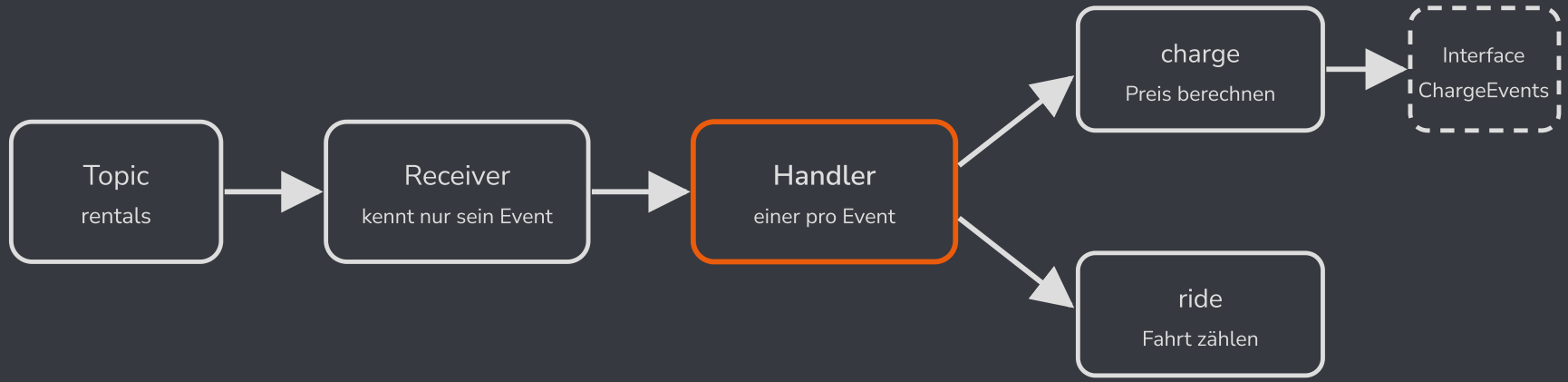
Ausgehend: ein Interface im Aggregate

```
public interface ChargeEvents {
    void chargeCreated( Charge charge );
}
```

```
@Transactional
public Charge chargeForRide( UUID rentalId, UUID customerId, Duration rideDuration ) {
    Charge charge = Charge.forRide( rentalId, customerId, rideDuration );
    chargeRepository.save( charge );
    chargeEvents.chargeCreated( charge );
    return charge;
}
```

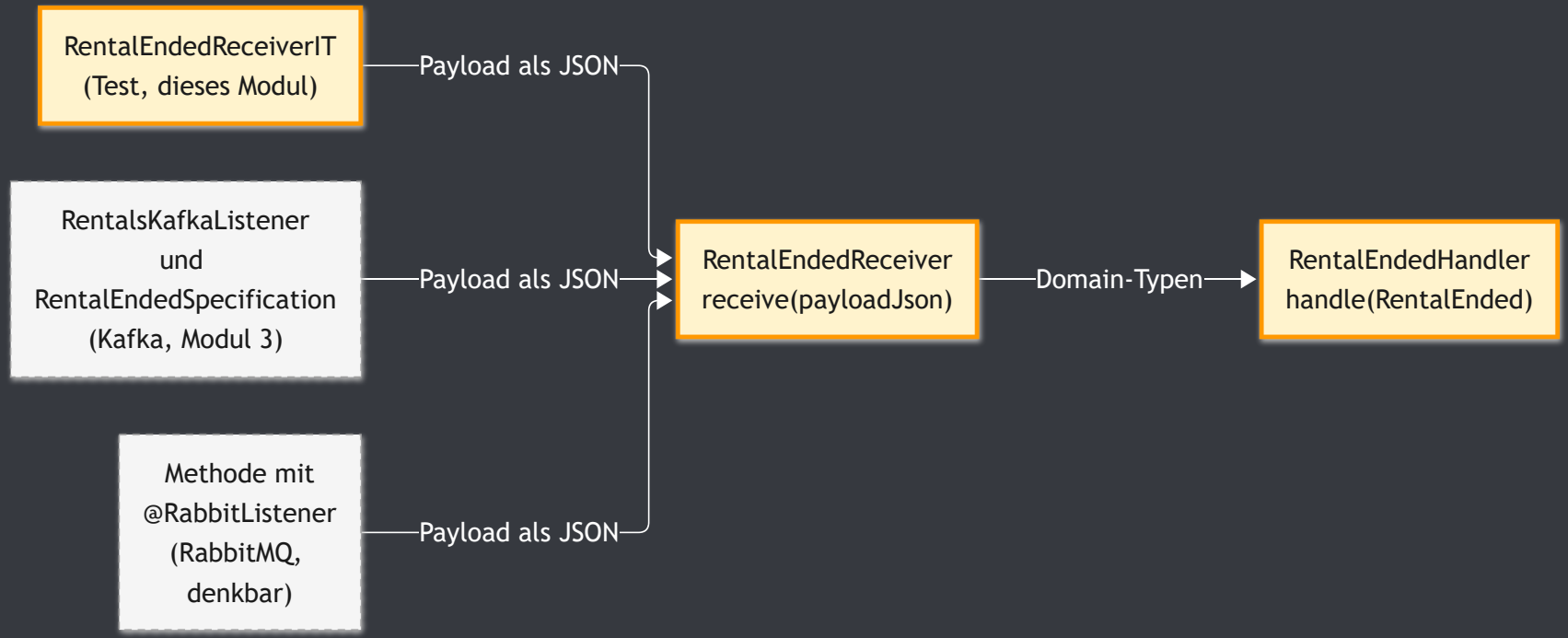
- Interface in `charge.application`
- im Test: `InMemoryChargeEvents`, sammelt statt zu senden
- in Modul 3: `KafkaChargeEvents`

Der Weg einer Nachricht



Receiver	liest den Payload, macht daraus Domain-Typen
Handler	ruft die Aggregates auf, keine Geschäftsregel
Aggregate	veröffentlicht sein eigenes Event über ein Interface

Wer den Receiver aufruft



- keine Annotation, kein Broker
- kennt das JSON-Format, deshalb in `infrastructure`

Vom Payload zu Domain-Typen

```
private record RentalEndedPayload( UUID rentalId, UUID customerId,
                                   Instant startedAt, Instant endedAt ) {}

public RentalEnded toRentalEnded( String payloadJson ) {
    RentalEndedPayload payload = json.readValue( payloadJson, RentalEndedPayload.class );
    Duration rideDuration = Duration.between( payload.startedAt(), payload.endedAt() );
    return new RentalEnded( payload.rentalId(), payload.customerId(), rideDuration );
}
```

- nur die Felder, die Billing braucht
- neue oder geänderte Felder: kein Fehler
- `RentalEnded`: ein Record für den Handler, kein Event

Der Handler

```
public void handle( RentalEnded rental ) {
    chargingService.chargeForRide( rental.rentalId(), rental.customerId(), rental.rideDuration() );
    rideCountService.countRideOf( rental.customerId() );
}
```

- ein Handler pro eingehendem Event
- Reihenfolge an einer Stelle, keine Geschäftsregel
- eine Transaktion pro Aggregate

Tests auf drei Ebenen

1. `ChargeTest`: die Preisregel, ohne Spring
2. `RentalEndedHandlerIT`: der Handler mit Domain-Typen, gegen H2
3. `RentalEndedReceiverIT`: vom JSON bis zum veröffentlichten Event

> kein Kafka, kein Docker: `mvn test` für die Unit-Tests, `mvn verify` für alle

Gestern: ein Beispiel mit KI gebaut

- Abrechnung im Fahrradverleih: Events empfangen, ohne Kafka testen
- mit Claude Code, im Dialog
- Code lief, Tests grün

> und trotzdem: fünfmal verworfen

Was die KI geschrieben hat

Die KI	Warum nicht	Was jetzt gilt
"Port" statt "Interface"	Vokabular einer anderen Architekturschule (Hexagonal), nicht unseres	Interface
Fahrten zählen mit einer Map im Service	Zustand im Service: weg beim Neustart, nicht geteilt zwischen zwei Instanzen, keine Transaktion	Entity + Repository, H2 reicht
Pakete `inbound`, `billing`, `events`	nicht die Aggregate-Struktur aus ST2	je Aggregate `domain` + `application`
zwei Varianten in einem Beispiel	Spring Events und Orchestrator nebeneinander: verwirrend	ein Beispiel, ein Weg
Workaround in der `pom.xml`	repariert mein IDE-Setup an der falschen Stelle	passendes JDK installiert, Workaround raus

Was ich daraus mitnehme

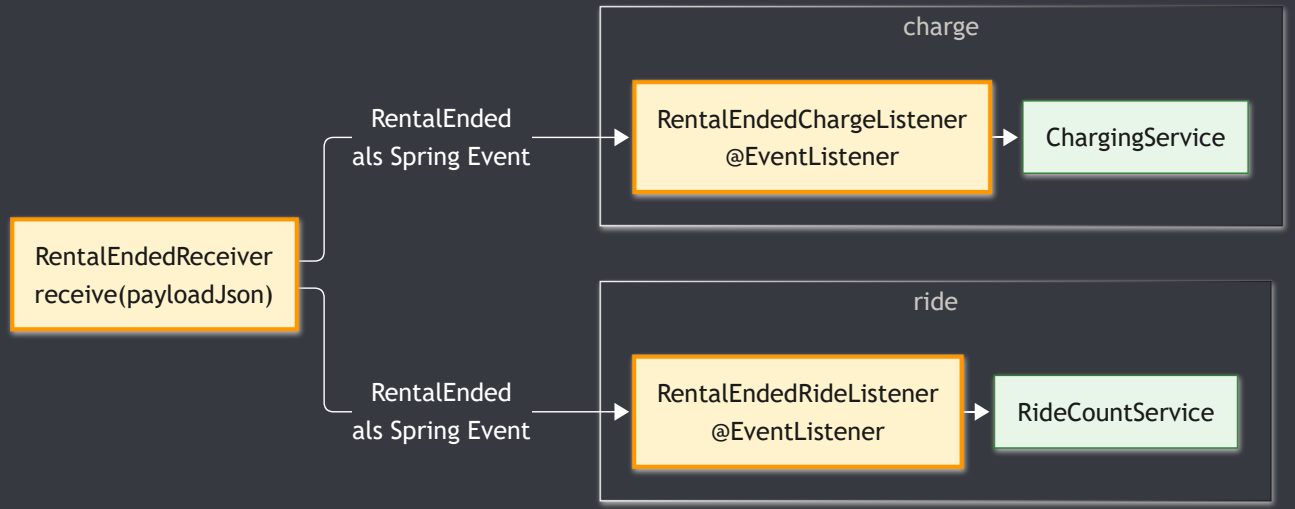
- grüne Tests sind keine gute Architektur
- erst entscheiden, dann prompten
- jeden Diff lesen
- Regeln dort aufschreiben, wo die KI sie liest: `CLAUDE.md`
- zweimal dieselbe Korrektur: eine neue Regel
- Gegenargumente einfordern: "Was spricht dagegen?"

Teamarbeit mit dem Ziel: Ihr Service läuft lokal, die Tests sind grün

1. REST-Endpoints: mit MockMvc testen, ohne Client
2. eingehende Events: Receiver ruft Handler, Nachbar-Events von Hand einspeisen
3. ausgehende Events: Interface im Aggregate, Test-Double im Test
4. Contract in `docs/asyncapi.yaml`: passt zum Code und zum Event-Katalog

> bis 13:45: eine erste Funktion, die Sie lokal vorführen können

Inhaltsimpuls: die Variante mit Spring Events



Vorteil	neue Reaktion = neue Klasse, keine bestehende Klasse ändert sich
Preis	keine Stelle, an der alle Reaktionen und ihre Reihenfolge stehen

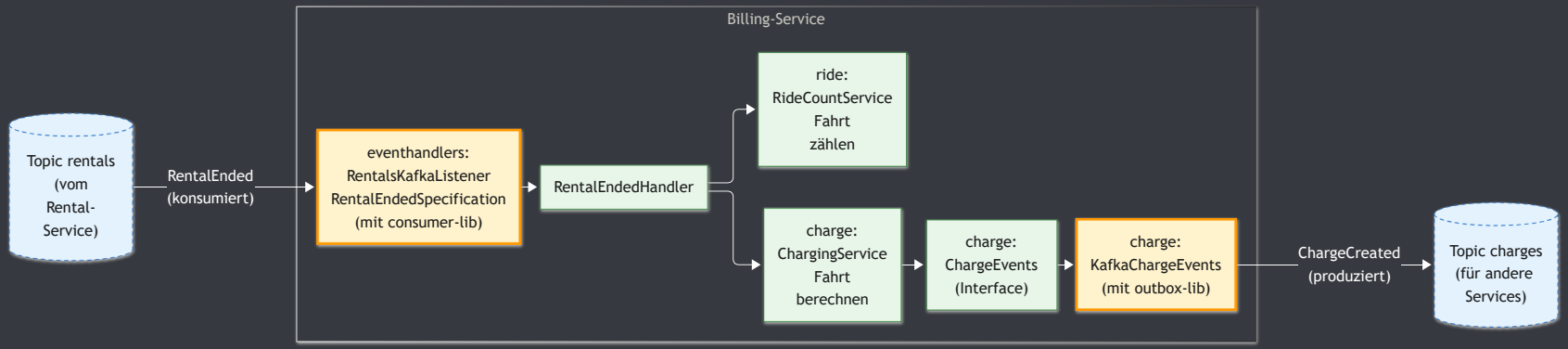
Teamarbeit mit dem Ziel: Ihr Service ist lokal grün und vollständig

- alle drei Nahtstellen
- Handler oder Spring Events: Ihre Entscheidung, begründet
- Teams mit Client: Entwurf der Oberfläche, simuliertes Backend, ADR zum Client
- mit jedem Team: die Issues durchgehen
- bis 14:45

Inhaltsimpuls: die echte Kafka-Anbindung

- Outbox-Bibliothek, drei Module: Core, JPA, direkt Kafka
- MEA: direkt Kafka, ohne Outbox, ohne Datenbank
- Consumer-Bibliothek: liest die Headers, findet das Event
- Producer implementiert Ihr Interface, z. B. `ChargeEvents`
- Kafka-Listener ruft Ihren Receiver auf
- nur neue Klassen, keine Änderung am Rest

Modul 3 im Code



grün: aus Modul 2 · orange: neu in Modul 3 · gestrichelt: außerhalb des Service

Docker und die lokale Entwicklungsumgebung

- Docker installieren
- lokale Entwicklungsumgebung: `students/mea/mea26/local-dev-env`
- Topics in `config/kafka/init.sh` eintragen, Umgebung neu starten
- dann den Service mit dem Profil `kafka` starten

Architekturdiskussion: Kafka-Anbindung ohne Java

> Kotlin, TypeScript, C#: Wie kommen diese Services an Kafka, ohne die Kafka-Hilfsbibliotheken?

Contract	das Format auf dem Topic: Topic, Key, Headers mit ihrer Kodierung, Payload. Nicht die Bibliothek.
Code	Adapter im eigenen Service-Repo, hinter denselben Interfaces wie in Modul 2
Outbox	später: eine Zeile in die Outbox-Tabelle schreiben, der Outbox-Service sendet

> Das Ergebnis halten wir gemeinsam als ADR in `documentation` fest.

Teamarbeit: die Grundlage für die Integration

- Docker installieren
- Kafka-Anbindung mit der lokalen Entwicklungsumgebung ausprobieren
- Ihre Tests bleiben grün, ohne Broker

Abschlussrunde: Wo steht jedes Team?

- die Clients: Entwurf der Oberfläche zeigen und besprechen
- Was läuft lokal, welche Tests sind grün?
- Steht der Contract im Event-Katalog?
- Ist Kafka schon angebunden, wo hängt es?

Ausblick

12.11.	Statusrunde per Video, 18:00 bis 19:00
19.11.	Statusrunde per Video, 18:00 bis 19:00
26.11.	Walking Skeleton vorführen, Abschluss des Grundlagenteils

Hausaufgabe bis 26.11.

- folgt

Fragen?

archi-lab.io