Statusrunde

MEA, 12.11.2026, 18:00

Stand der Repos: 12.11.2026 17:00

Ablauf (30 Minuten)

- Überblick: wo steht jedes Team?
- Übergreifende Themen
- Teilziele vom 05.11., Team für Team
  - A: Event rein, Event raus
  - B: Beispiel-Payloads im Contract
  - C: Client gegen das echte Backend
- Bis zum 19.11.

> Zur Einordnung: Der Stand der Teams beruht auf einer KI-gestützten Durchsicht der Repositories, ohne gründliche manuelle Gegenprüfung. Er zeigt den Stand vom 12.11.2026, 17:00. Was danach gemergt wurde, fehlt hier.

Überblick

Team	A Tests	B Contract	C Client	auf main?	Pipeline main
kitchen-storage	erledigt	weitgehend	weitgehend	ja gemergt 14:57	Tests grün rot nur durch Katalog-Trigger
menu-ordering	weitgehend auf development	erledigt schon vor dem 05.11.	weitgehend	nein development	grün
pickup	teilweise	offen	offen	ja	kein Test-Job rot nur durch Katalog-Trigger
reservation	offen	teilweise	teilweise Speicher-Bug behoben, Start ohne Kafka?	nein walking-skeleton	kein Test-Job rot nur durch Katalog-Trigger

> In keinem Hausaufgaben-Issue ist eine Checkbox abgehakt

Übergreifend: die Arbeit liegt auf Branches

Was wir sehen

- kitchen-storage: heute 14:57 alles nach main gemergt
- menu-ordering: 11 Commits auf development, main vom 05.11.
- reservation: 99 Commits auf walking-skeleton, main vom 03.11.

Warum das ein Problem ist

- versteckte technische Schuld: Merge-Aufwand wächst jeden Tag, keiner sieht ihn
- Nachbarn und Event-Katalog lesen den Contract von main
- menu-ordering musste Beispiele von einem Feature-Branch kopieren

Wie man es fixt

- kleine Merge Requests, die in ein bis zwei Tagen nach main gehen
- Merge Request früh öffnen, auch als Draft
- ADR 0006: Code, Contract, Tests in einem Merge Request
- erledigt ist, was auf main ist

Übergreifend: der Katalog-Trigger zeigt ins Leere

Was wir sehen

- main rot bei pickup, reservation, kitchen-storage
- einziger roter Job: update-event-catalogue
- Fehler: downstream_bridge_project_not_found
- Ursache: alter Projektpfad der Dokumentation, kam mit meiner Vorlage
- menu-ordering hat den Pfad am 03.11. korrigiert

Warum das ein Problem ist

- der Event-Katalog bekommt Ihren neuen Contract nicht
- main dauerhaft rot: ein neuer, echter Fehler fällt nicht mehr auf

Wie man es fixt: eine Zeile in `.gitlab-ci.yml`

```
update-event-catalogue:
  stage: deploy
  trigger:
    project: students/mea/mea26/documentation   # vorher: archi-lab/th-koeln/lehre/mea26/documentation
    branch: main
```

Übergreifend: kein Test-Job in der Pipeline

Was wir sehen

- pickup, reservation: .gitlab-ci.yml prüft nur den Contract
- die Tests laufen nur, wenn jemand sie auf dem eigenen Rechner startet
- kitchen-storage, menu-ordering: Test-Job da, grün

Warum das ein Problem ist

- bricht ein Push einen Test, merkt es niemand, bis jemand zufällig lokal testet
- im Merge Request sieht der Reviewer nicht, ob noch alles läuft
- ab 19.11. Kafka und Nachbarn: dann braucht es die grüne Basis

Wie man es fixt: ein Job `test` in `.gitlab-ci.yml`

pickup

```
test:
  stage: test
  image: node:24-alpine
  script:
    - npm install
    - npm test
```

reservation

```
test:
  stage: test
  image: maven:3.9-eclipse-temurin-21
  script:
    - mvn -B test
```

Übergreifend: Contract und Code laufen auseinander

	Contract	Code
OrderFulfilled (kitchen-storage → pickup)	Topic `fulfillment`	pickup hört auf `bl.fulfillment.events.v1`
CheckIn (reservation), Topic	`reservations`	`reservation`
CheckIn, Feld	`reservationEndsAt`	`sessionEndsAt`
CheckIn, Key	Guest-ID	Tisch-ID
CheckIn, Zeitstempel	`date-time` mit Offset	`LocalDateTime` ohne Offset

> Mit Kafka am 19.11. käme hier nichts an. Fix: der Contract des Producers ist die Referenz (ADR 0002), Code und Contract ändern sich im selben Merge Request (ADR 0006). Issues: pickup #8, reservation #10

Übergreifend: Testen mit Nachbar-Beispielen

Was wir sehen

- kitchen-storage: OrderPlaced, MenuChanged mit den Beispielen von menu-ordering, unverändert kopiert
- menu-ordering: OrderFulfilled, StationManningStatusChanged mit den Beispielen von kitchen-storage
- menu-ordering: CheckIn mit eigenem JSON, reservation hat kein Beispiel
- pickup: OrderFulfilled mit eigenem JSON, orderId "order-42" ist keine UUID

Warum das ein Problem ist

- Test mit eigenem JSON zeigt die Domain-Logik
- aber nicht: versteht der Service die echte Nachricht des Nachbarn?
- genau da bricht die Integration

Wie man es fixt

- Producer: ein Beispiel pro Event im Contract, Payload und Header
- Producer: Test prüft das publizierte Event gegen das eigene Beispiel
- Consumer: Beispiel unverändert in die Test-Ressourcen kopieren, Quelle notieren
- Consumer: Test gibt das Beispiel als JSON-String an den Receiver

A: Event rein, Event raus

kitchen-storage	erledigt	rein MenuChanged, OrderPlaced (menu-ordering) → raus CookingStepChanged, OrderFulfilled rein BoxEmptied (pickup) → raus OrderFulfilled für das wartende Gericht Entscheidungen: doppelte Bestellung, gelöschtes Gericht, keine Box frei; auf main
menu-ordering	weitgehend	rein CheckIn (reservation), StationManningStatusChanged (kitchen-storage), Gast bestellt → raus OrderPlaced rein OrderFulfilled (kitchen-storage) → Bestellung erledigt, kein Event Entscheidung: Station bemannt oder nicht. Fehlt: StockChanged, CookingStepChanged; CheckIn nur Platzhalter; noch nicht auf main
pickup	teilweise	rein OrderFulfilled (kitchen-storage) → Box gefüllt, Test endet hier raus wäre BoxTimeLimitReached, Test dafür startet erst beim Service Fehlt: Beispiel von kitchen-storage statt eigenem JSON; Test bis zum ausgehenden Event; Test-Job
reservation	offen	rein OrderPlaced (menu-ordering, Ihr #8) → noch kein Receiver Fehlt: Receiver, Event-Test, Test-Job

B: Beispiel-Payloads im Contract

kitchen-storage	weitgehend	Beispiel für jede Nachricht, auch IngredientListChanged; Test prüft Payloads, nicht Header; StockChanged erst geplant; auf main
pickup	offen	keine Beispiele für BoxEmptied, BoxTimeLimitReached; Kafka-Tests prüfen das JSON schon, lassen sich anbinden
reservation	teilweise	CheckIn ohne Beispiel, Header auskommentiert; Code publiziert CheckIn jetzt, aber anders als der Contract
menu-ordering	erledigt	Beispiele für MenuChanged, OrderPlaced schon vor dem 05.11., daher kein Issue B; seit heute Tests, die das Publizierte gegen die Beispiele prüfen

C: Client gegen das echte Backend

kitchen-storage	weitgehend	Guest Room und Restaurant Manager vom echten Backend ausgeliefert; Events der Nachbarn per Testseite /dev, über dieselben Receiver wie später Kafka
menu-ordering	weitgehend	Angular-Client über Proxy am echten Backend; Events der Nachbarn per Endpoints /stub/*, ebenfalls über die Receiver
reservation	teilweise	HTML-Seiten, die der Service selbst rendert (JTE, eine Template-Engine wie Thymeleaf); Reservierung wird seit heute gespeichert, Check-in umgesetzt; offen: startet der Service ohne Kafka?
pickup	offen	noch kein Client; neuer Endpoint nur per curl

> Bis 19.11. sind Stubs für die Nachbarn richtig so. Danach kommen die Events per Kafka von den echten Nachbarn.

Dokumentation

- #1 Foto und Transkription der Context Map: offen
- #2 Context Map: Zeile pickup leer
- #3 Clients und ihre Teams: keine Seite
- #4 Glossar: leer
- offene Merge Requests, Review bei mir:
  - documentation !2: ADR 0007 (Agent-Rechte nur für main)
  - local-dev-env !1: Datenbank für kitchen-storage
- offener Merge Request ohne Reviewer: documentation !1, Case-Study-Änderungen von reservation

Bis zum 19.11.: was noch offen ist

- alles auf main, Pipeline grün, mit Test-Job
- Katalog-Trigger: Projektpfad korrigieren (pickup, reservation, kitchen-storage)
- offene Teilziele A, B, C bleiben offen, fällig 19.11.
- Checkboxen in den Issues abhaken: so sehen alle den Stand
- Topics und Felder mit dem Producer-Contract abgleichen (pickup #8, reservation #10)

Bis zum 19.11.: neue Hausaufgabe

- Service per Kafka an die Nachbarn anbinden, zuerst lokal mit local-dev-env
- Happy Path mit den Nachbarteams durchspielen, so weit er schon geht
- Integrationstest mit echtem Kafka: Event des Nachbarn rein, Wirkung prüfen
- REST-Endpoints, die Ihr Client aufruft, ohne laufenden Server testen
  - Spring: MockMvc
  - andere Frameworks: das entsprechende Werkzeug
- Service-Doku im Dokumentations-Repo, Agent-Konventionen im eigenen Repo
- Entwurfsentscheidungen als ADRs

Wo steht, wie es geht?

Kafka an den Service anschließen

- Infopage Einen Service an Kafka anschließen: ein Listener pro Topic, consumer-lib, outbox-lib, Profil `kafka`
- README von local-dev-env, Abschnitt Kafka: Ports, Spring-Konfiguration, Topics anlegen

Integrationstest mit echten Events

- ohne Broker: in derselben Infopage, Abschnitt Ohne Broker testen (Record mit Headern an den Listener)
- mit echtem Kafka: Schritt 4 der Reihe, "Einen Service mit echtem Kafka testen", in Vorbereitung (das Wichtigste siehe nächste Folie)

Welche Event-Header wir verwenden

- Infopage, Abschnitt Die Headers: so setzt die outbox-lib sie, so liest die consumer-lib sie
- README von local-dev-env, Abschnitt Event Headers

REST-Endpoints testen

- Infopage Testing REST Controllers: MockMvc, voller Durchstich oder nur die Web-Schicht

Integrationstest mit echtem Kafka

Kafka im Container, Topics vor dem Service-Start

```
@SpringBootTest
@ActiveProfiles( "kafka" )
@Testcontainers
@Tag( "environment" )
class BillingServiceWithKafkaIT {

  @Container
  static final KafkaContainer KAFKA =
      new KafkaContainer( "apache/kafka:4.1.2" );

  @DynamicPropertySource
  static void kafkaAddress( DynamicPropertyRegistry r ) {
    r.add( "spring.kafka.bootstrap-servers",
           KAFKA::getBootstrapServers );
  }

  @BeforeAll
  static void createTopics() throws Exception {
    try ( Admin admin = Admin.create( Map.of(
        "bootstrap.servers", KAFKA.getBootstrapServers() ) ) ) {
      admin.createTopics( List.of(
          new NewTopic( "rentals", 1, (short) 1 ),
          new NewTopic( "charges", 1, (short) 1 ) ) ).all().get();
    }
  }
```

Event rein, Event raus

```
  @Test
  void aRentalEndedLeadsToAChargeCreated() throws Exception {
    try ( var consumer = consumer() ) {
      consumer.subscribe( List.of( "charges" ) );

      try ( var producer = producer() ) {
        producer.send( rentalEnded() ).get();  // Beispiel
      }                                        // + Header

      var chargeCreated =
          firstRecord( consumer, Duration.ofSeconds( 30 ) );

      assertThat( header( chargeCreated, "event_type" ) )
          .isEqualTo( "ChargeCreated" );
      assertThat( chargeCreated.key() )
          .isEqualTo( CUSTOMER_ID.toString() );
    }
  }
```

- Profil `kafka`: die echten Adapter, Test-Double mit `@Profile("!kafka")`
- `firstRecord`: `poll()` in einer Schleife, der Service arbeitet asynchron
- `@Tag("environment")`: nur mit `mvn verify -Penvironment`, braucht Docker
- Schritt 4 der Reihe, Infopage in Vorbereitung

REST-Endpoints testen mit MockMvc

```java
@SpringBootTest
@AutoConfigureMockMvc
@Transactional
class MessageApiIT {

    @Autowired MockMvc mockMvc;

    @Test
    void aPostedMessageCanBeReadBack() throws Exception {
        mockMvc.perform( post( "/messages" )
                    .contentType( APPLICATION_JSON )
                    .content( "{ \"text\": \"hello over HTTP\" }" ) )
                .andExpect( status().isCreated() )
                .andExpect( header().exists( "Location" ) );

        mockMvc.perform( get( "/messages" ) )
                .andExpect( status().isOk() )
                .andExpect( jsonPath( "$", hasSize( 1 ) ) )
                .andExpect( jsonPath( "$[0].text", is( "hello over HTTP" ) ) );
    }
}
```

Was der Test zeigt

- der echte Spring-MVC-Stack, aber ohne Server und ohne Netzwerk
- Statuscode, Header und JSON so, wie der Client sie bekommt
- prüfen Sie die Felder, die Ihr Client wirklich liest
- läuft in der Pipeline, braucht keine Umgebung

In Ihren Teams

- reservation: erster MockMvc-Test da (GuestTest)
- kitchen-storage: `WebApplicationFactory`, schon da
- pickup: HTTP-Test auf freiem Port, schon da
- menu-ordering: noch keiner