

Event-Contracts mit AsyncAPI

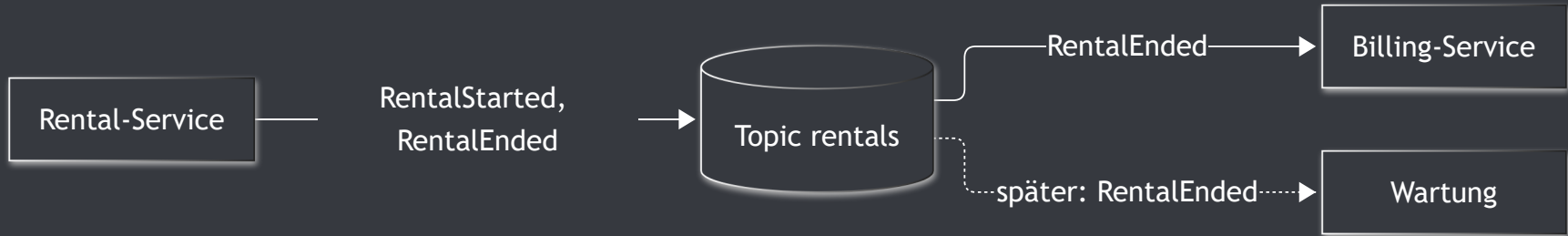
Was ein Service der Welt mitteilt

Events sind die Schnittstelle eines Service

- Wer ein Event konsumiert, verlässt sich auf seinen Namen, seinen Inhalt und sein Topic.
- Ein Event-Contract schreibt das auf, bevor der erste Producer oder Consumer gebaut wird.
- AsyncAPI ist für Events, was OpenAPI für REST-Endpoints ist: ein maschinenlesbares Format, aus dem Dokumentation, Prüfungen und Code entstehen können.

Ein Contract ist billig zu ändern, solange noch niemand ihn implementiert hat.

Das Beispiel: ein Fahrradverleih



- Der Rental-Service veröffentlicht, was mit einer Ausleihe passiert ist.
- Der Billing-Service kennt die Tarife und rechnet ab, sobald eine Ausleihe endet.
- Jeder Service hat seinen eigenen Contract, aus seiner eigenen Sicht.

Die vier Bausteine

- `info` und `servers`: welcher Service, welche Version, welcher Broker
- `channels`: wohin Events fließen; bei Kafka ist die `address` der Topic-Name
- `operations`: was der Service mit einem Channel tut, `send` oder `receive`
- `components`: die Messages mit Headers, Payload und Kafka-Key, und ihre Schemas

Die Message

```
RentalEnded:
  name: RentalEnded
  title: Rental ended
  summary: A customer has returned a bike to a station.
  headers:
    $ref: '#/components/schemas/EventHeaders'
  payload:
    $ref: '#/components/schemas/RentalEndedPayload'
  bindings:
    kafka:
      key:
        type: string
        format: uuid
        description: The rental id.
      bindingVersion: 0.5.0
```

```
RentalEndedPayload:
  type: object
  required:
    - rentalId
    - bikeId
    - customerId
    - endStationId
    - startedAt
    - endedAt
  properties:
    rentalId:
      type: string
      format: uuid
    bikeId:
      type: string
    customerId:
      type: string
      format: uuid
    endStationId:
      type: string
    startedAt:
      type: string
      format: date-time
    endedAt:
      type: string
      format: date-time
```

Tatsache oder Aufruf in Verkleidung?

ChargeCustomer

- Ein Befehl in der Form eines Events
- Der Rental-Service muss wissen, dass es eine Abrechnung gibt und was sie tun soll
- So eng gekoppelt wie ein synchroner Aufruf

RentalEnded

- Eine Tatsache, in der Vergangenheitsform
- Der Rental-Service meldet nur, was passiert ist
- Was daraus folgt, entscheidet jeder Consumer selbst

Die Daten, die die Consumer brauchen, und nicht mehr

- Die Abrechnung braucht, wer gefahren ist und wie lange: `customerId`, `startedAt`, `endedAt`.
- Kein Preis: Den Tarif kennt die Abrechnung. Ändert er sich, merkt der Rental-Service nichts.
- Kein Name, keine Adresse: Die braucht hier niemand.
- Eine Wartung, die später Fahrten zählt, konsumiert dasselbe Event, ohne dass der Producer etwas ändert.

Jedes Feld im Payload ist ein Versprechen, das der Producer nicht ohne seine Consumer zurücknimmt.

Eine Definition pro Event

```
channels:
  rentals:
    address: rentals
    description: Owned by the rental service; billing only reads from it.
    messages:
      RentalEnded:
        $ref: './rental-service.asyncapi.yaml#/components/messages/RentalEnded'

operations:
  receiveRentalEnded:
    action: receive
    channel:
      $ref: '#/channels/rentals'
    messages:
      - $ref: '#/channels/rentals/messages/RentalEnded'
```

Der Consumer verweist auf die Message im Contract des Producers. Legt man alle Contracts nebeneinander, entsteht der Event-Katalog des Systems.

Einen Contract prüfen

- Im Browser: studio.asyncapi.com zeigt Fehler sofort und rendert den Contract als Dokumentation.
- Auf der Kommandozeile:

```
npx @asyncapi/cli validate contracts/rental-service.asyncapi.yaml
```

- Gültig heißt nur: formal richtig. Ob die Events taugen, entscheiden die drei Fragen.

Drei Prüffragen für Ihren Contract

1. Meldet jedes Event eine Tatsache, oder ist es ein Aufruf in Verkleidung?
2. Trägt es die Daten, die die Consumer brauchen, und nicht mehr?
3. Fehlt ein Consumer, oder steht einer drin, der das Event gar nicht braucht?

