# Stream Processing and Change Data Capture

## A Story with Kafka and Debezium

Alain PHAM
Senior Middleware Specialist
Solution Architect @ Red Hat
apham@redhat.com

Laurent BROUDOUX
Senior Middleware Specialist
Solution Architect @ Red Hat
lbroudou@redhat.com

Red Hat

# The Why of Event Driven Architecture and Change Data Capture ?

# The Rise of Event Driven Patterns in Modern Applications

1

**Agility**

Loose coupling between applications to allow continuous upgrades

2

**Isolation and Availability**

Deployments with isolation to enable high availability

3

**Real Time**

Users expect real time experience in modern apps

4

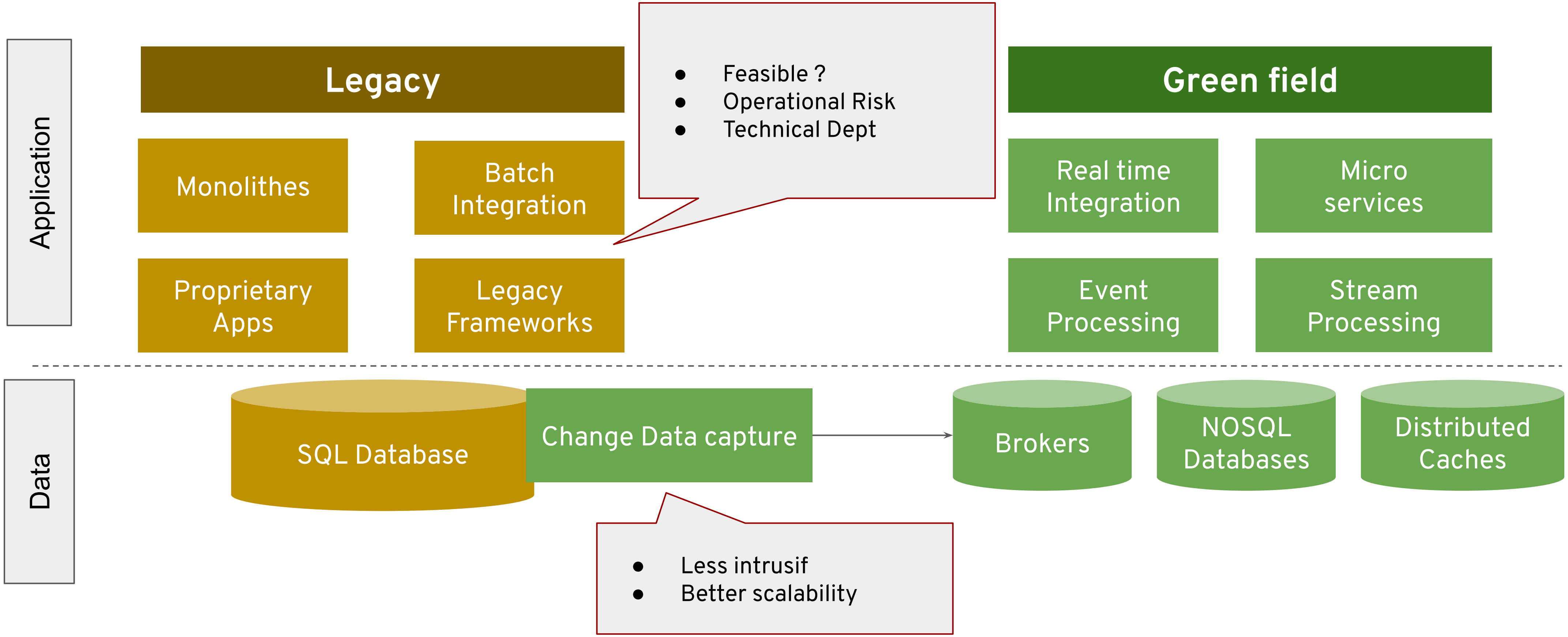
**Multi Clouds**

Multi cloud deployments require a robust communication layer for synchronisations

## Events

# Why Data Change Capture?

## A strategy to modernize Legacy Systems

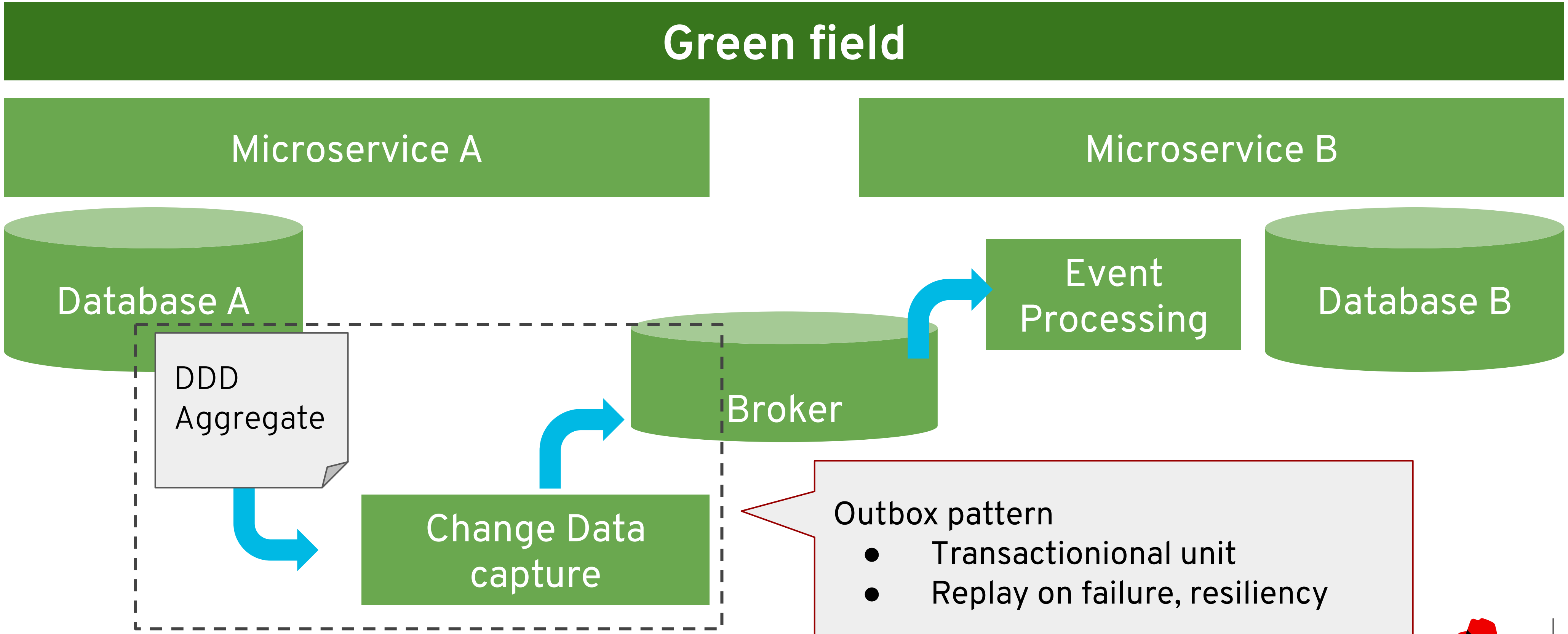


- Feasible ?
- Operational Risk
- Technical Dept

- Less intrusif
- Better scalability

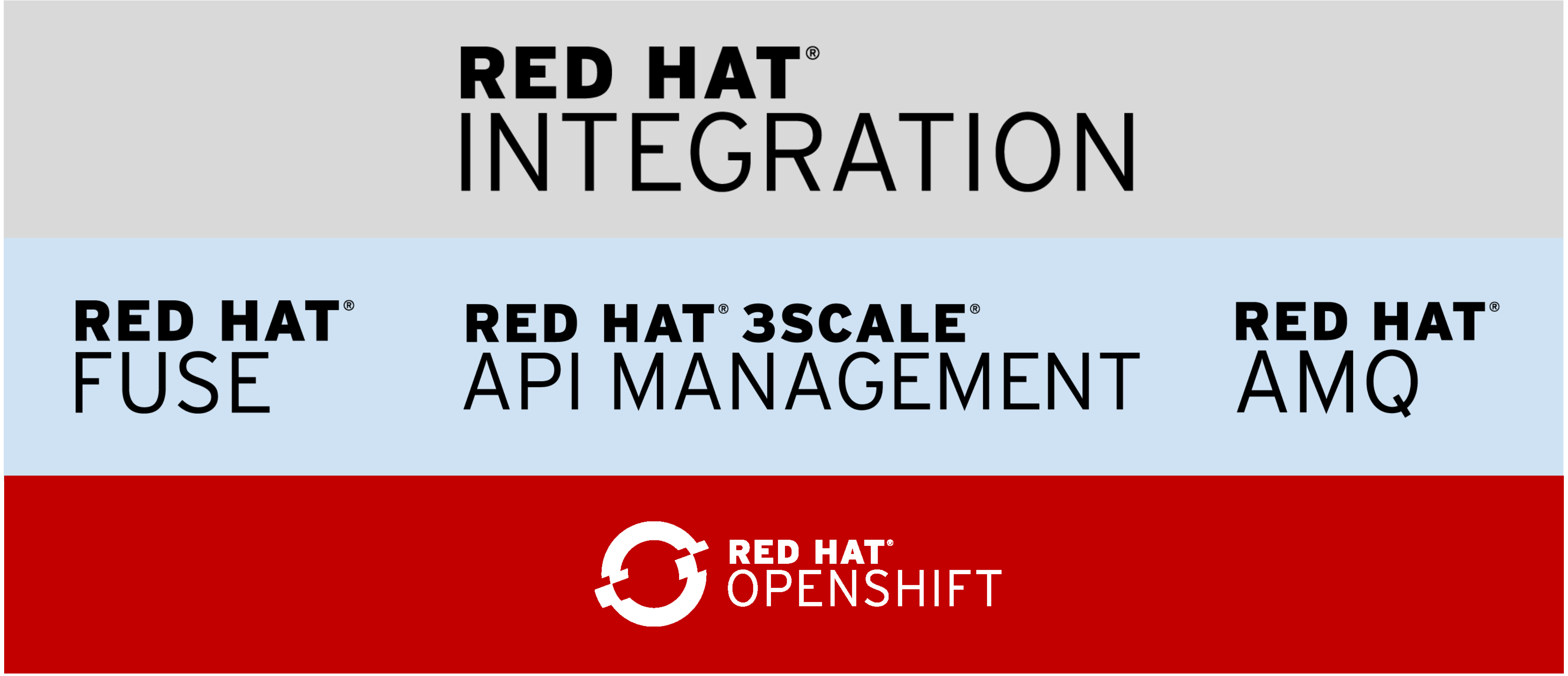
# Why Data Change Capture?

A Strategy to simplify synchronisations between microservices



Outbox pattern

- Transactionional unit
- Replay on failure, resiliency



RED HAT® INTEGRATION

RED HAT® FUSE

RED HAT® 3SCALE® API MANAGEMENT

RED HAT® AMQ

RED HAT® OPENSHIFT

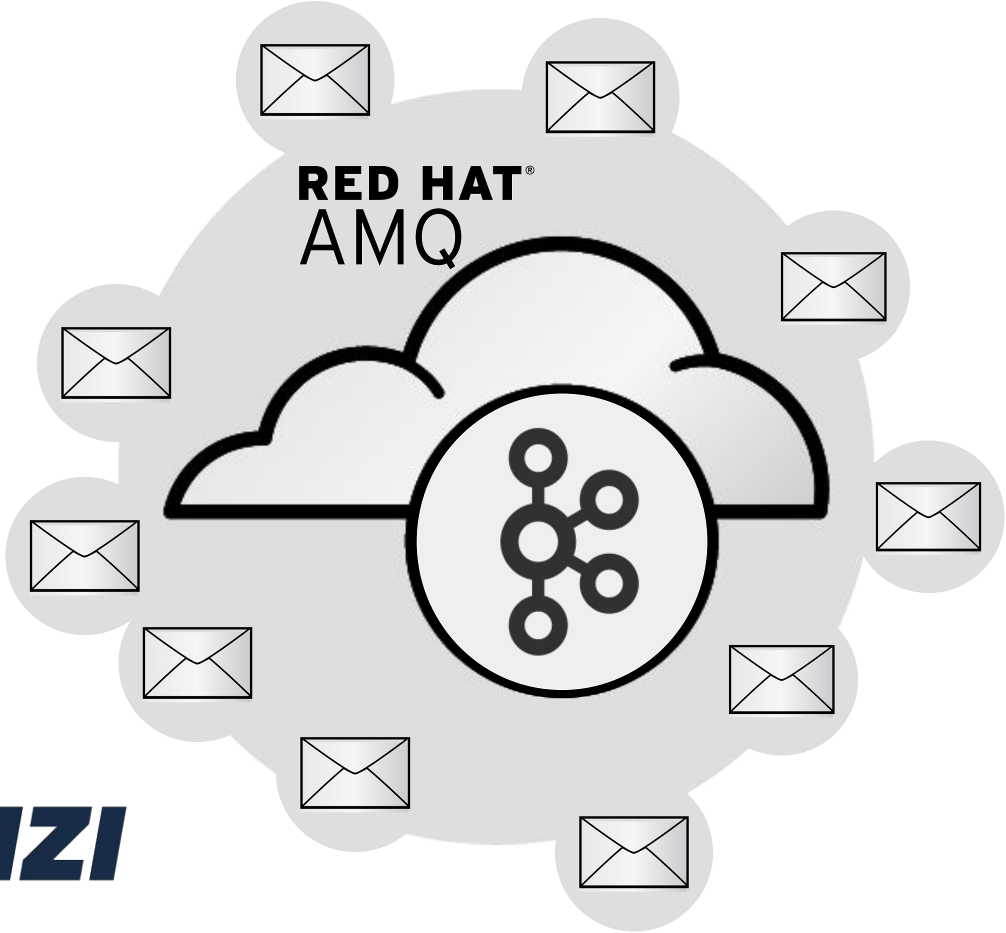
Enterprise data streaming platform distribution based on Apache Kafka.

Available standalone on Red Hat Enterprise Linux VMs/bare metal or on OpenShift (based on Strimzi project).

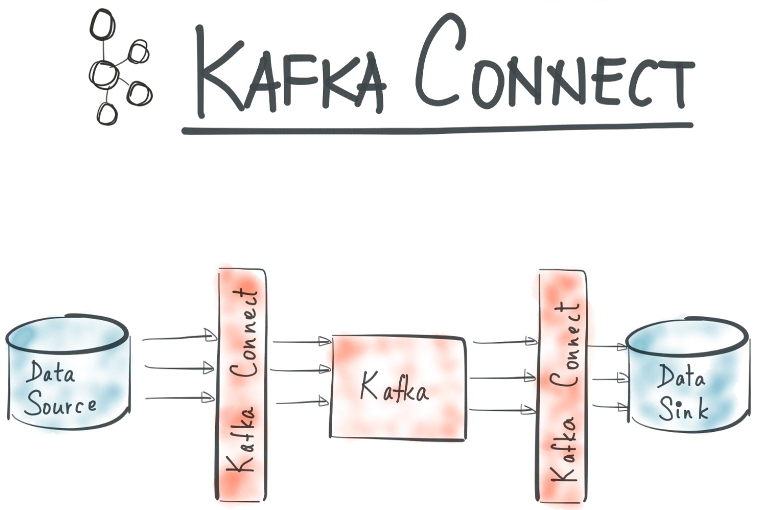
Horizontally-scalable, fault-tolerant commit log with stream processing capabilities.

RED HAT® OPENSHIFT

STRIMZI

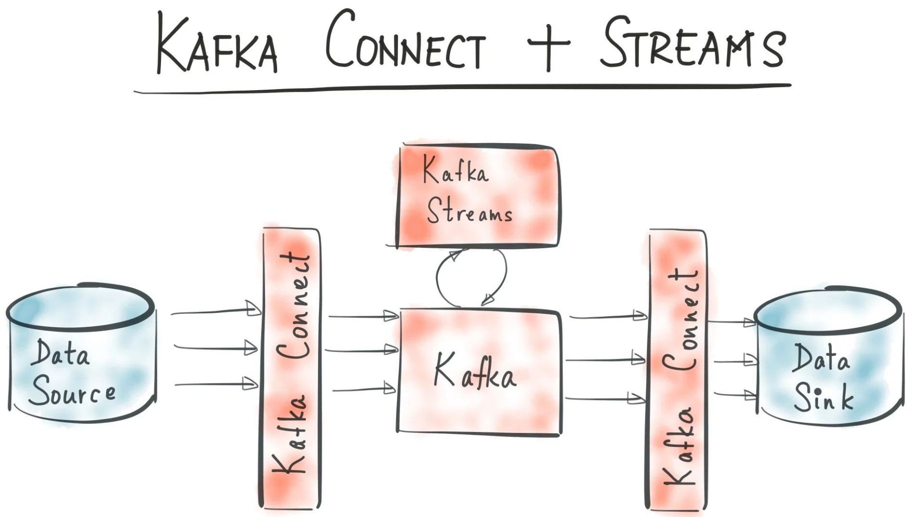


# Integration Patterns for Data Processing



Transfert de masse IN/OUT Kafka

Single Message Transformation

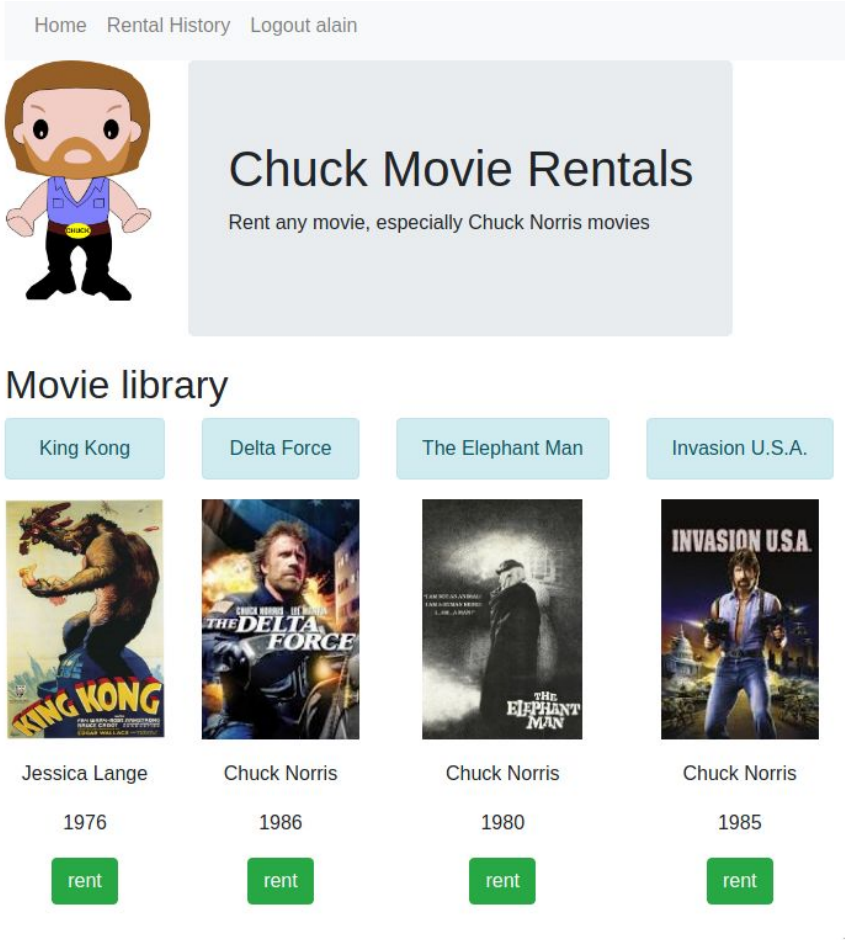


Process de masse IN Kafka

Multiple Messages Transformation

# Use case of the day

# A company to transform



Home Rental History Logout alain

Chuck Movie Rentals

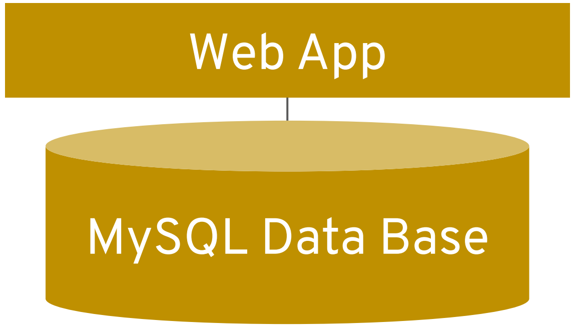
Rent any movie, especially Chuck Norris movies

Movie library

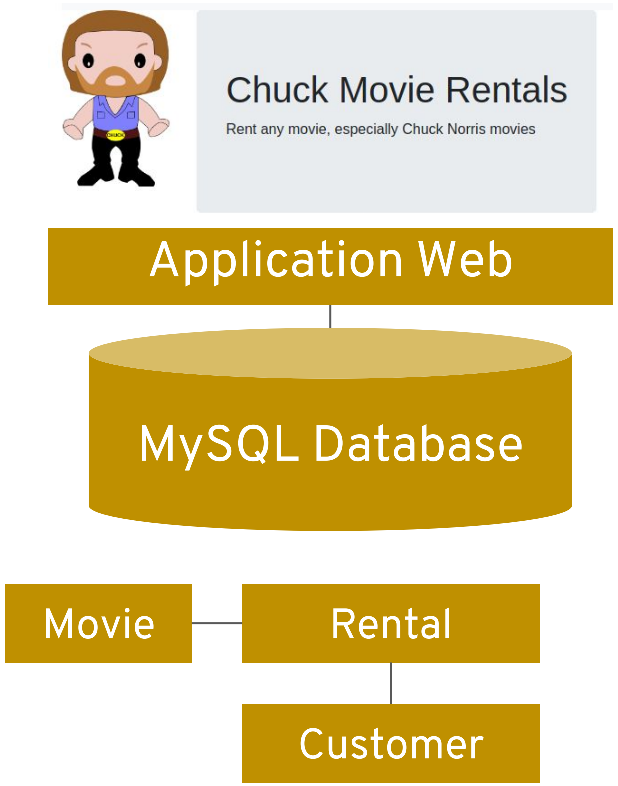
| King Kong | Delta Force | The Elephant Man | Invasion U.S.A. |
|---|---|---|---|
| Jessica Lange | Chuck Norris | Chuck Norris | Chuck Norris |
| 1976 | 1986 | 1980 | 1985 |
| rent | rent | rent | rent |

- Objectives
  - Make customers come back & grow revenue
- How ?
  - Merchandising
  - Make the rental experience more memorable
  - Especially for Chuck Norris fans!

## Existing system



# Our plan

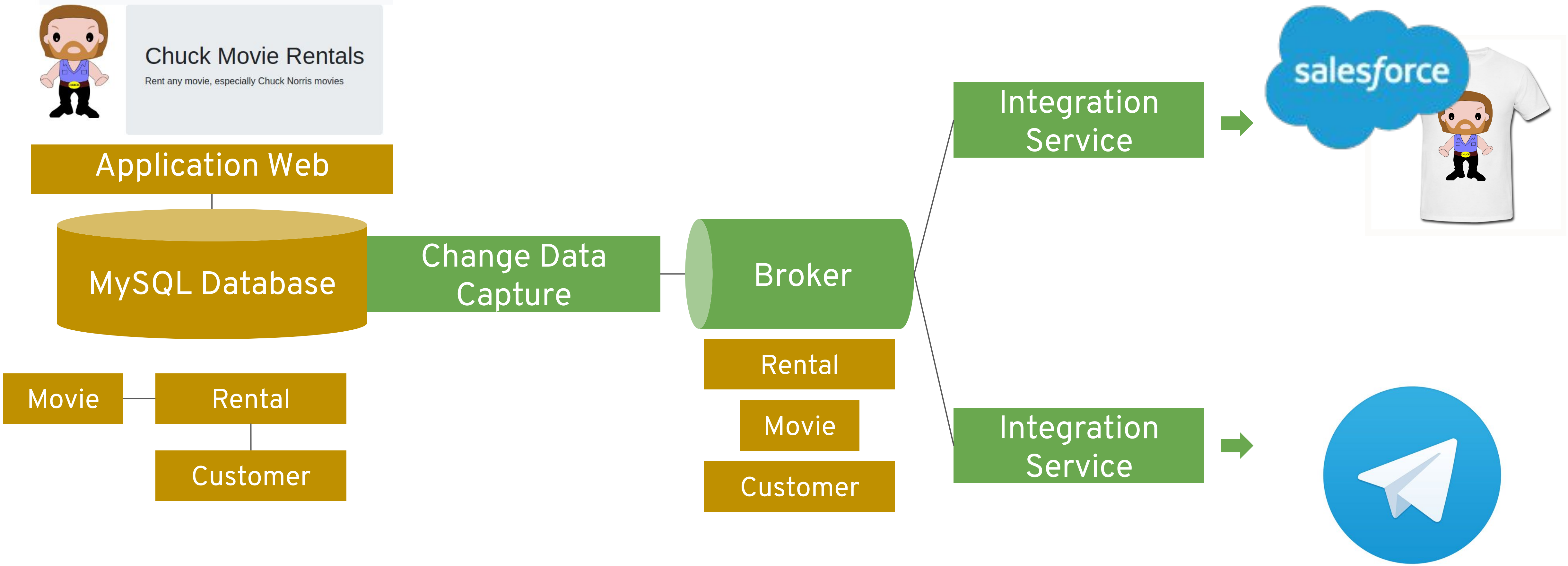


If customer rents a Chuck Norris movie

- Add customer as contact to prospect for merchandising  in Sales Force.

- Initiate Customer/Salesman interaction with a Chuck Norris Fact through Telegram

# Overview Architecture



Chuck Movie Rentals

Rent any movie, especially Chuck Norris movies

Application Web

MySQL Database

Movie

Rental

Customer

Change Data Capture

Broker

Rental

Movie

Customer

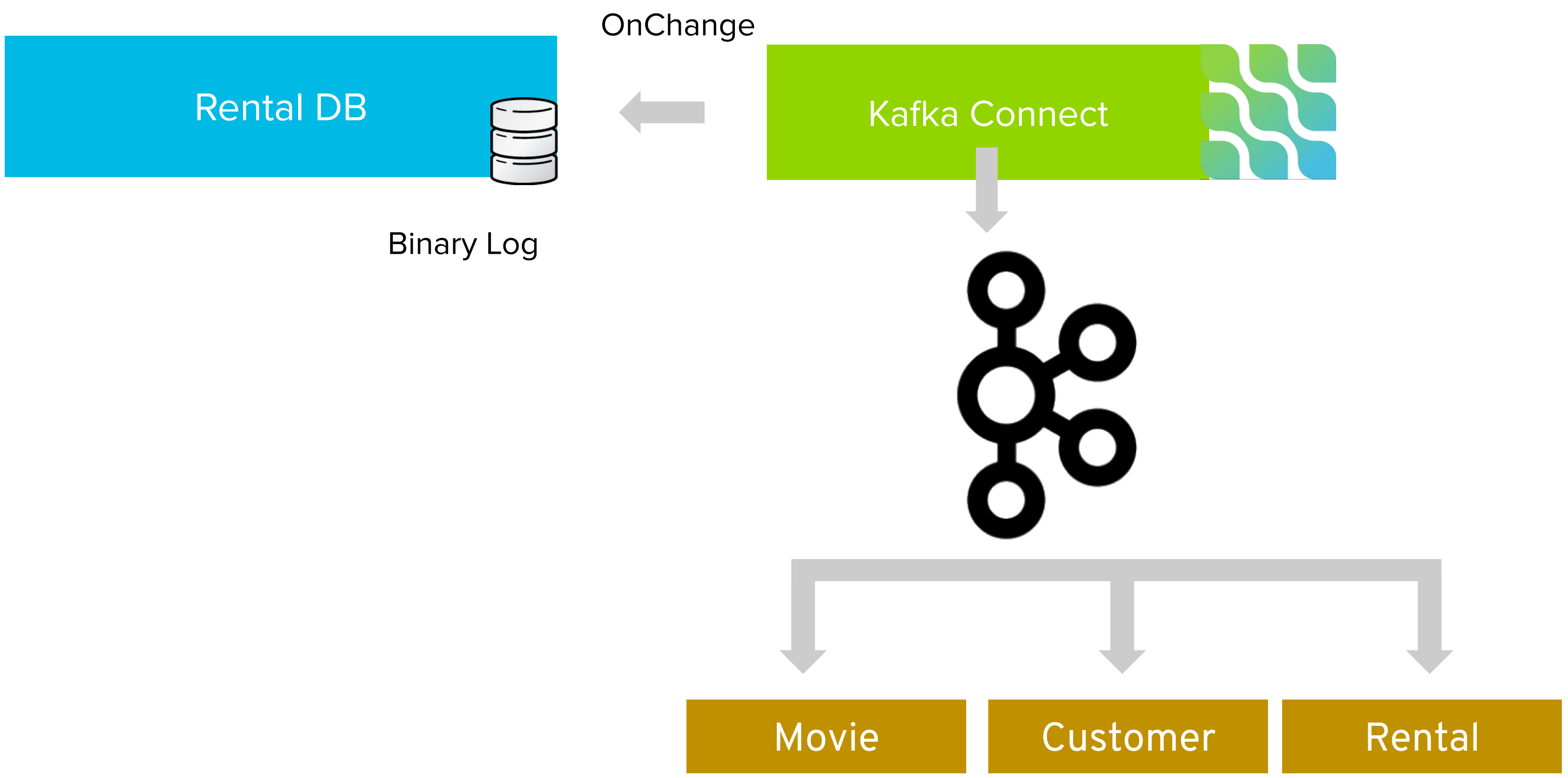
Integration Service

salesforce

Integration Service

# Demo Time!

# Change Data Capture avec debezium



OnChange

Rental DB

Kafka Connect

Binary Log

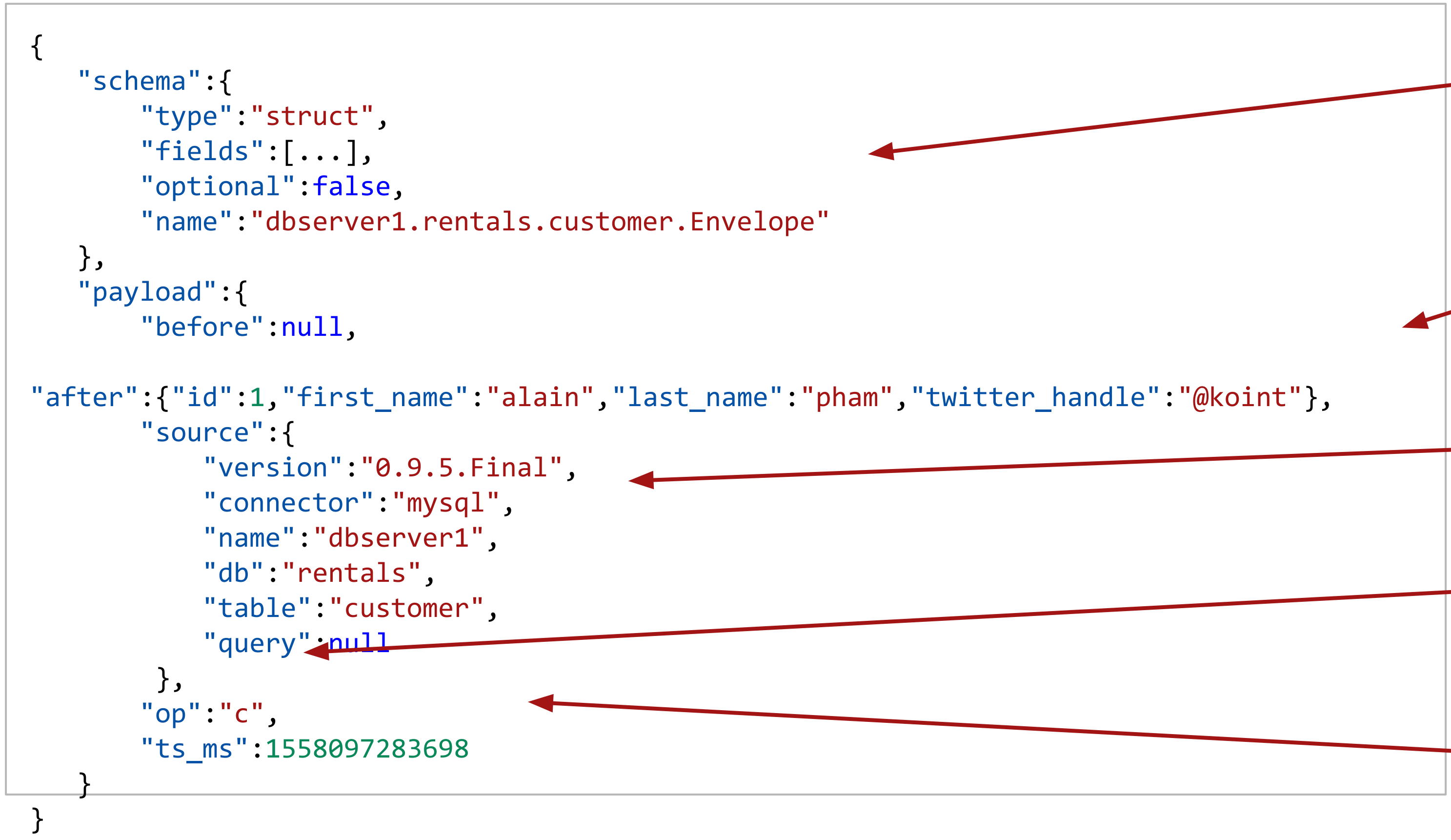
Movie

Customer

Rental

# Structure d’un événement

debezium



```
{
   "schema":{
       "type":"struct",
       "fields":[...],
       "optional":false,
       "name":"dbserver1.rentals.customer.Envelope"
   },
   "payload":{
       "before":null,

"after":{"id":1,"first_name":"alain","last_name":"pham","twitter_handle":"@koint"},
       "source":{
           "version":"0.9.5.Final",
           "connector":"mysql",
           "name":"dbserver1",
           "db":"rentals",
           "table":"customer",
           "query":null
        },
       "op":"c",
       "ts_ms":1558097283698
   }
}
```

Le schéma des éléments de la payload

Les snapshots de la ligne impactée (ici before==null car création)

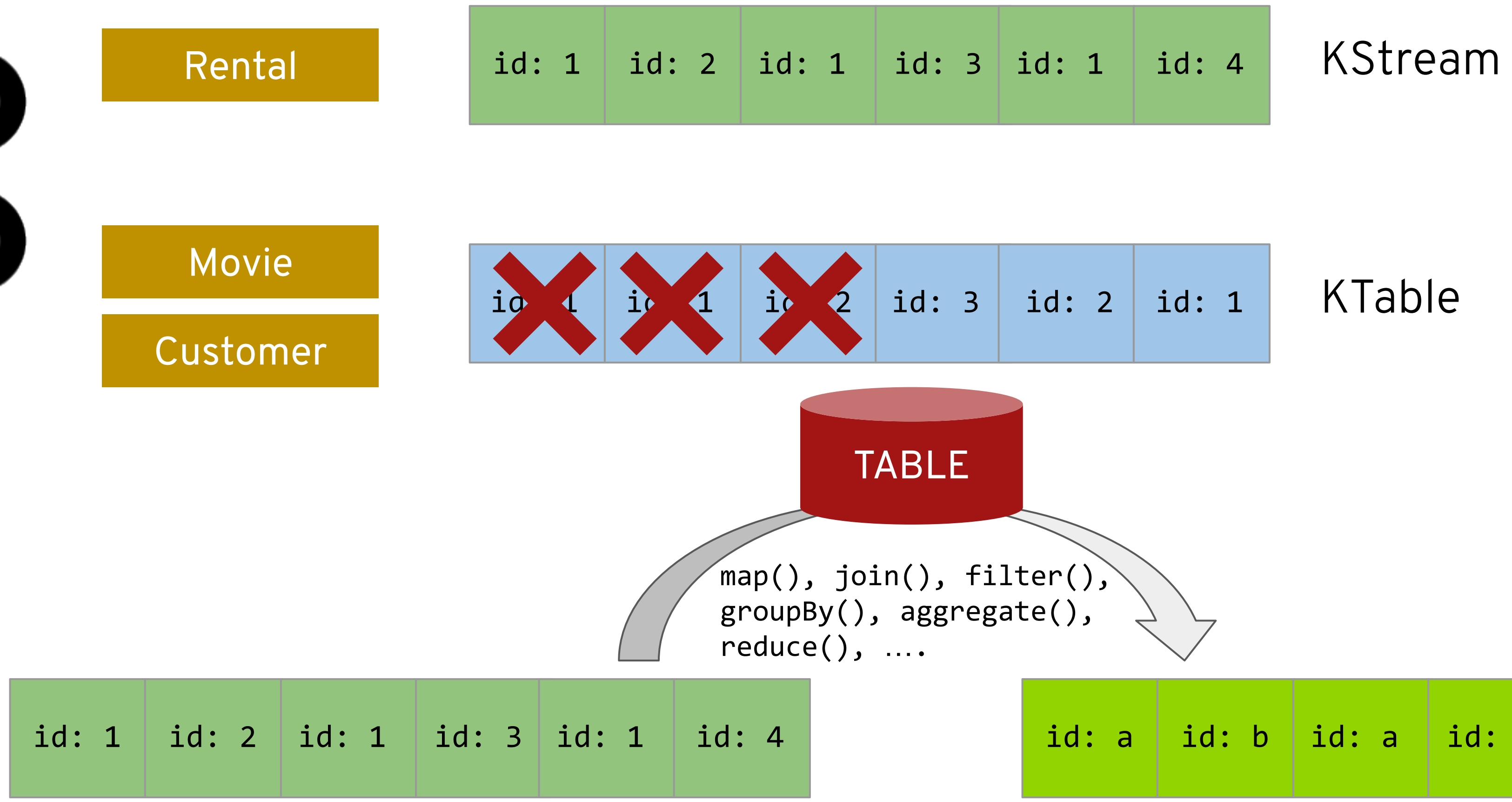
La source de l’événement

Le type d’opération (c, u ou d pour create, update ou delete)

Le timestamp de l’événement

Red Hat

# Structures de données Kafka Streams

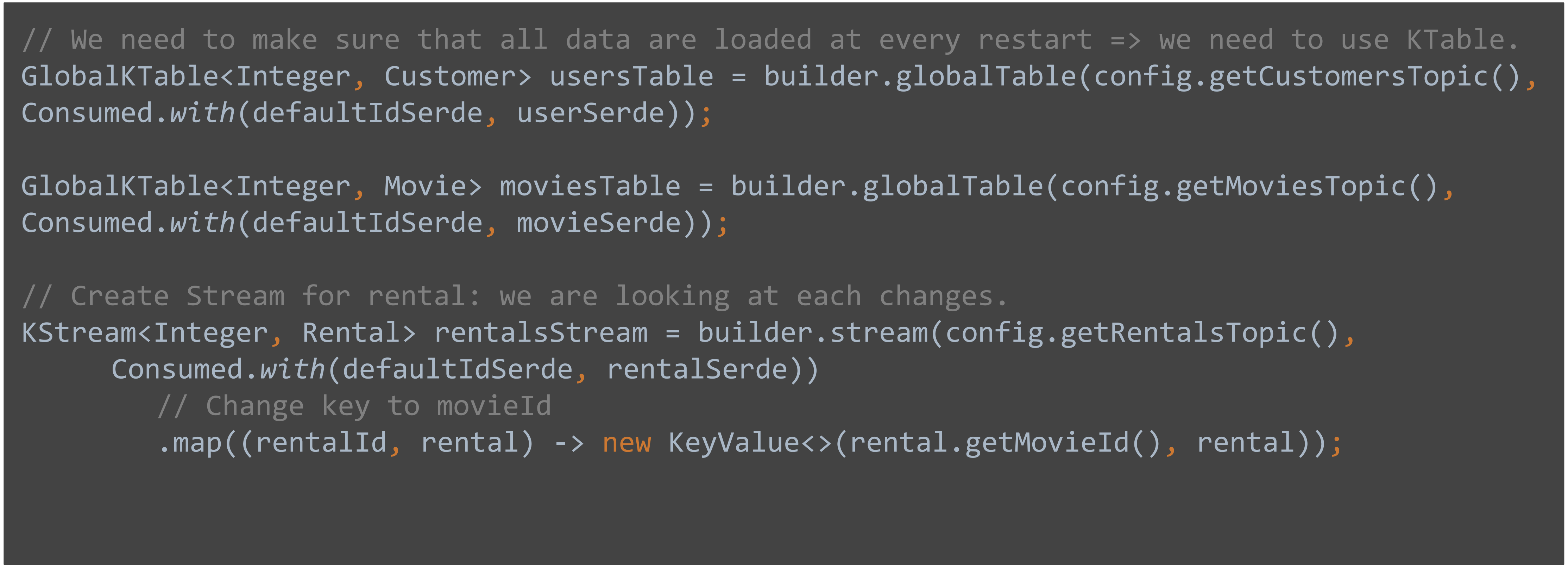


```
map(), join(), filter(),
groupBy(), aggregate(),
reduce(), ….
```

Sequences of un bounded facts

Sequence of limited facts
Keep only last value
Good for reference/master data

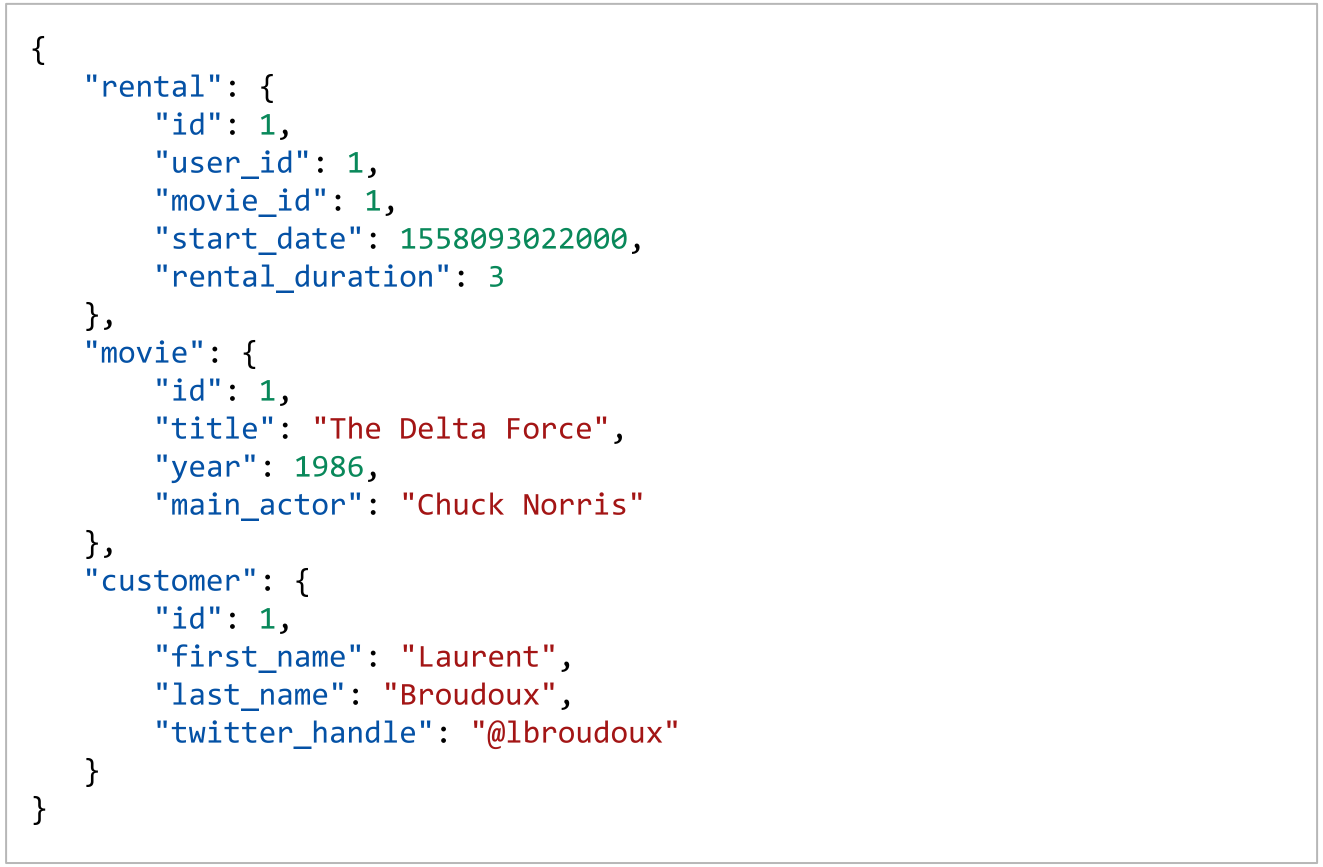
# Show me some code !



```
// We need to make sure that all data are loaded at every restart => we need to use KTable.
GlobalKTable<Integer, Customer> usersTable = builder.globalTable(config.getCustomersTopic(),
Consumed.with(defaultIdSerde, userSerde));

GlobalKTable<Integer, Movie> moviesTable = builder.globalTable(config.getMoviesTopic(),
Consumed.with(defaultIdSerde, movieSerde));

// Create Stream for rental: we are looking at each changes.
KStream<Integer, Rental> rentalsStream = builder.stream(config.getRentalsTopic(),
     Consumed.with(defaultIdSerde, rentalSerde))
        // Change key to movieId
        .map((rentalId, rental) -> new KeyValue<>(rental.getMovieId(), rental));
```

```
// Now, let the magic happens!!
KStream<Integer, CustomerRentalMovieAggregate> chuckNorrisRentalsStream =
     rentalsStream
           // Join with movies table and build an aggregate POJO.
           .join(moviesTable,
                   (leftKey, leftValue) -> leftKey,
                   (rental, movie) -> new CustomerRentalMovieAggregate(rental, movie))
           // Filter only Chuck Norris movies
           .filter((movieId, urmAggregate) ->
urmAggregate.getMovie().getMainActor().equals("Chuck Norris"))
           // Change key to customerId.
           .map((movieId, urmAggregate) -> new
KeyValue<>(urmAggregate.getRental().getCustomerId(), urmAggregate))
           // Join with customers and complete the aggregate.
           .join(usersTableInt,
                   (leftKey, leftValue) -> leftKey,
                   (urmAggregate, customer) -> completeAggregate(urmAggregate, customer));

// Publish to out topic.
chuckNorrisRentalsStream.to(config.getTargetTopic(), Produced.with(Serdes.Integer(),
aggregateSerde));
```

# The result of the aggregation



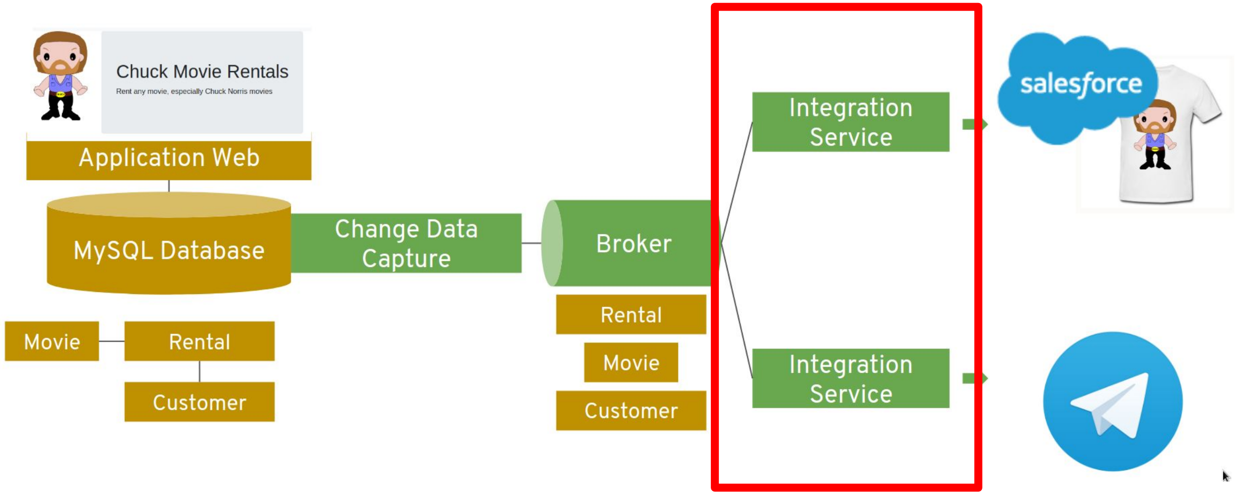
```
{
   "rental": {
       "id": 1,
       "user_id": 1,
       "movie_id": 1,
       "start_date": 1558093022000,
       "rental_duration": 3
   },
   "movie": {
       "id": 1,
       "title": "The Delta Force",
       "year": 1986,
       "main_actor": "Chuck Norris"
   },
   "customer": {
       "id": 1,
       "first_name": "Laurent",
       "last_name": "Broudoux",
       "twitter_handle": "@lbroudoux"
   }
}
```

L’objet Rental

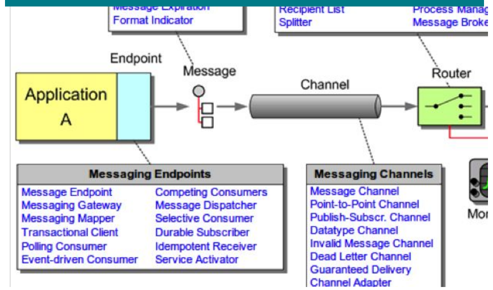
L’objet Movie

L’objet Customer

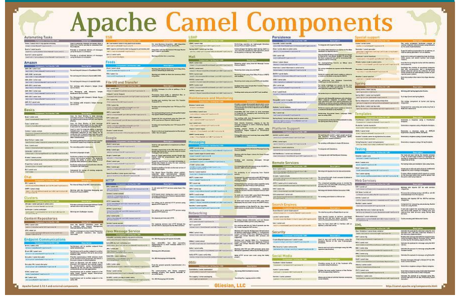
# Services d’Integration avec Red Hat Fuse



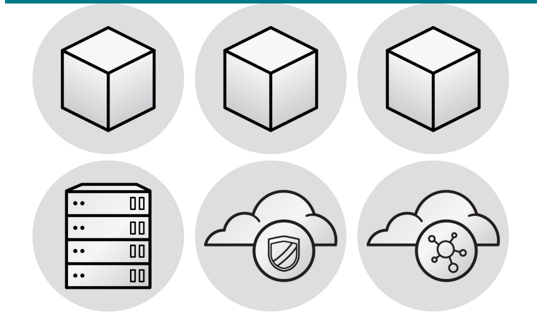
## 65 Integration Patterns



## +180 Connectors and Dataformats

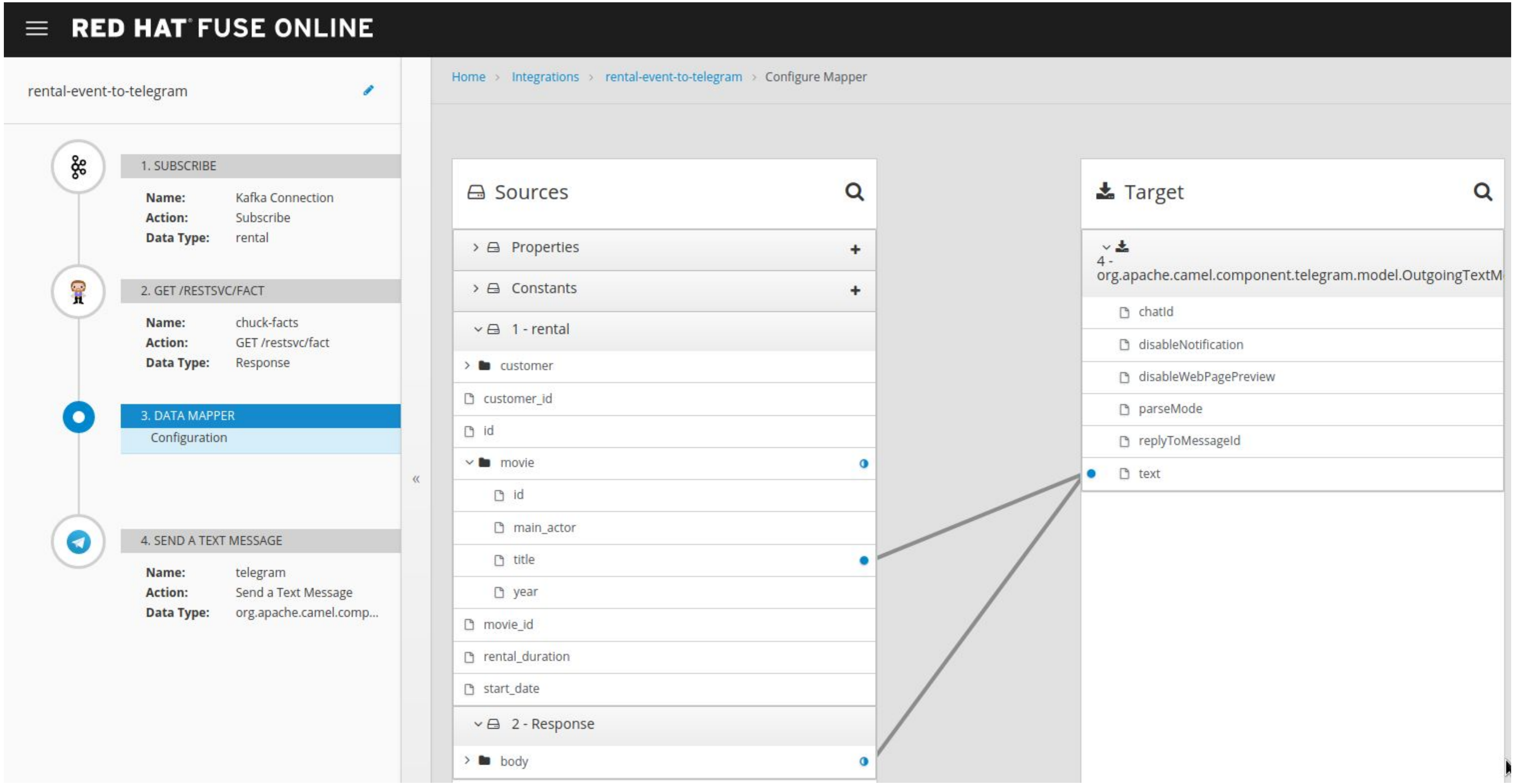


## Run everywhere : Bare Metal, Virtual, Cloud..



## Enable Citizen Developers

# Fuse Online Implementation



RED HAT FUSE ONLINE

rental-event-to-telegram

Home › Integrations › rental-event-to-telegram › Configure Mapper

1. SUBSCRIBE

**Name:** Kafka Connection
**Action:** Subscribe
**Data Type:** rental

2. GET /RESTSVC/FACT

**Name:** chuck-facts
**Action:** GET /restsvc/fact
**Data Type:** Response

3. DATA MAPPER

Configuration

4. SEND A TEXT MESSAGE

**Name:** telegram
**Action:** Send a Text Message
**Data Type:** org.apache.camel.comp...

# Chuck Norris is Stronger than GDPR !!

# Nous avons vu :

**MODERNIZATION**

How to add new functionality without impacting an existing system using Debezium (CDC)

**REACTIVE**

How to perform stream processing in a reactive approach using Kafka Streams, work in memory with little resource usage

**INTEGRATION**

How to innovate easilty and quickly using Fuse Online by leveraging existing connectors and graphical mapping tools.

**COMPLÉMENTARITÉ**

Stream processing offers a complementary development model to correlate events. Camel can then be used as a swiss-knife to bring the various connectors.

# Repo to demo source code

https://github.com/lbroudoux/chuck-norris-streams

# Thank you

in linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

f facebook.com/redhatinc

twitter.com/RedHat