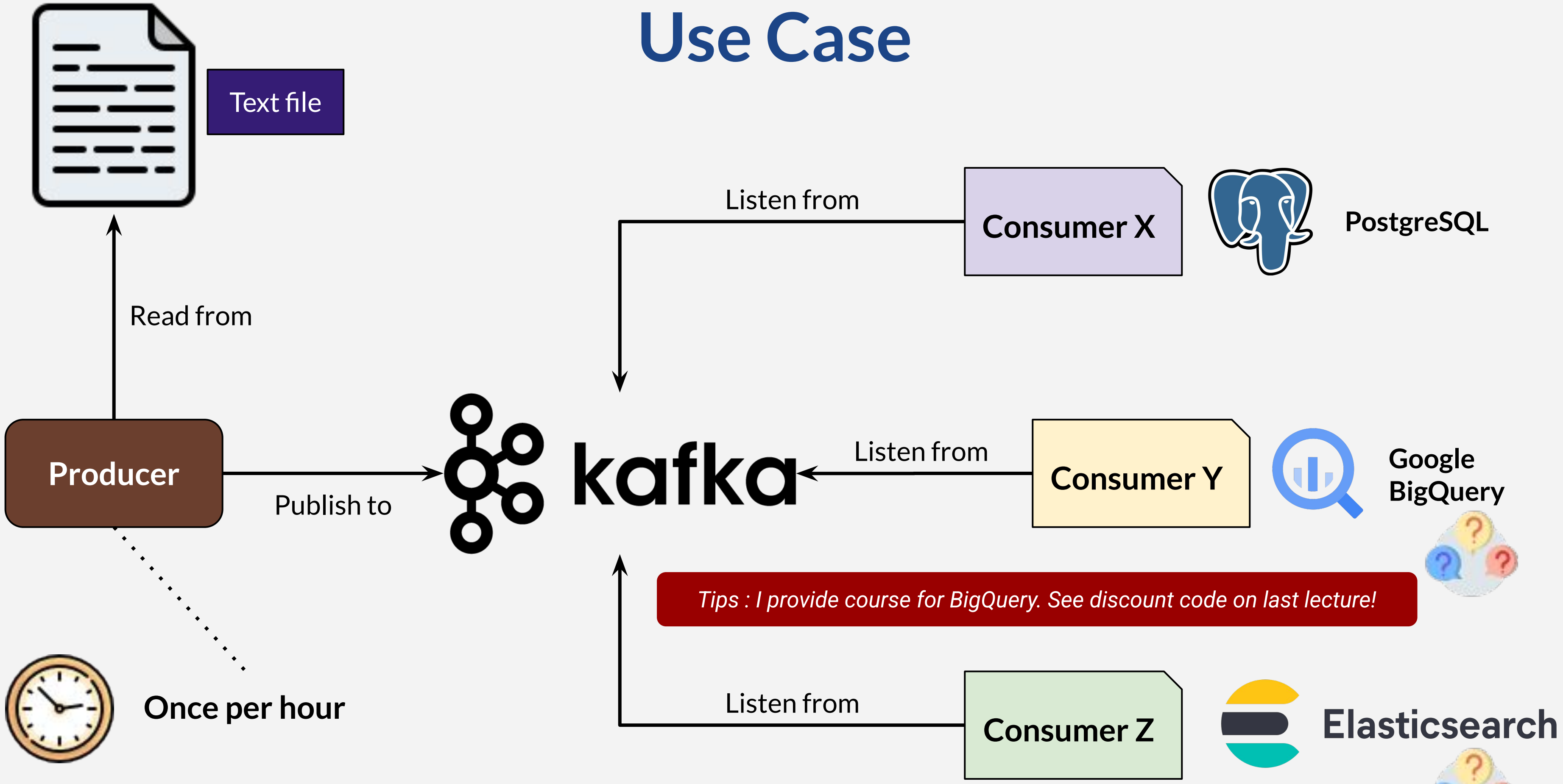
Message In, Message Out

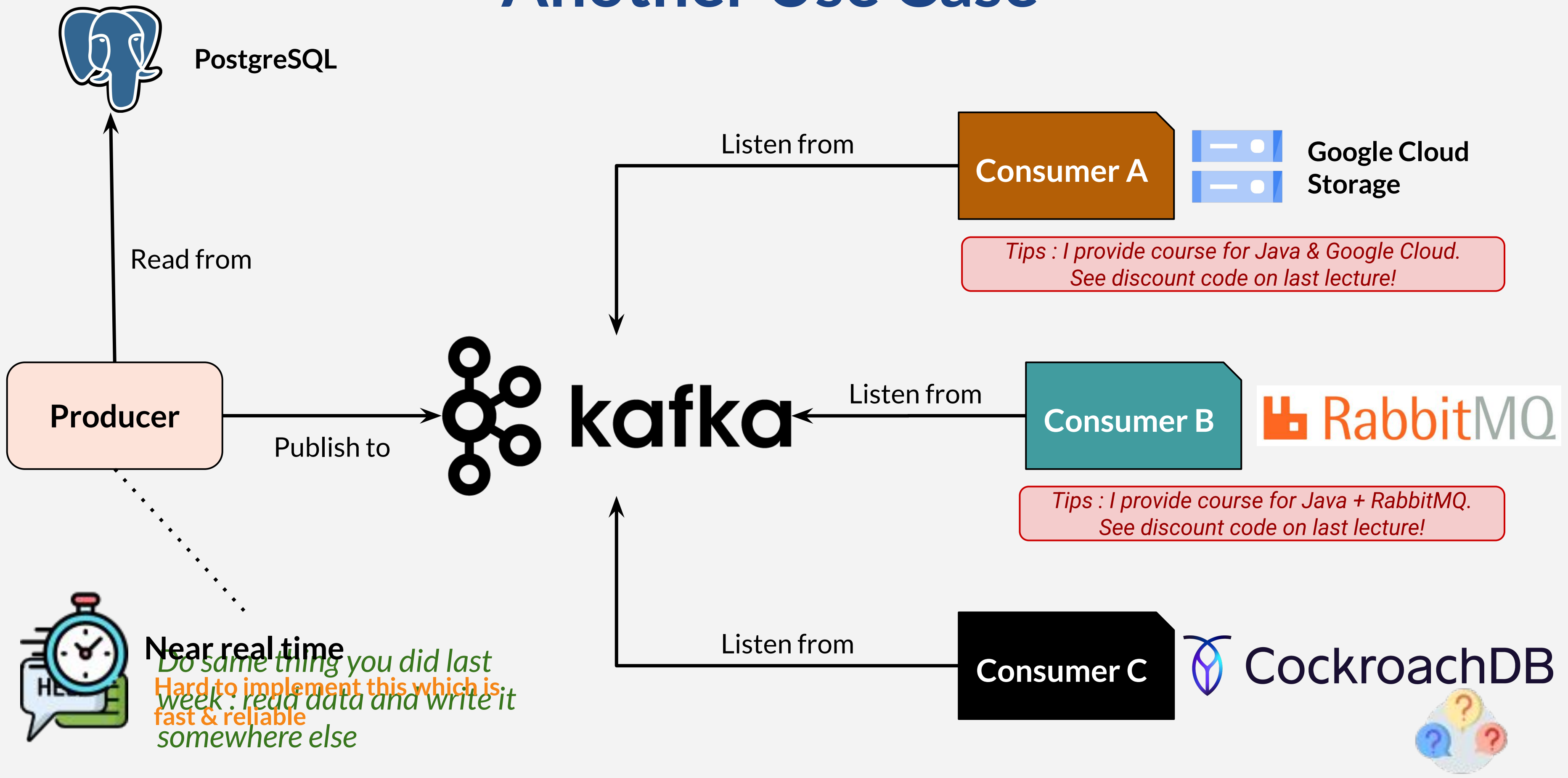
Use Case



Tips : I provide course for BigQuery. See discount code on last lecture!

Tips : I provide course for Java + Elasticsearch. See discount code on last lecture!

Another Use Case



Tips : I provide course for Java & Google Cloud. See discount code on last lecture!

Tips : I provide course for Java + RabbitMQ. See discount code on last lecture!

Do same thing you did last week : read data and write it somewhere else

Hard to implement this which is fast & reliable

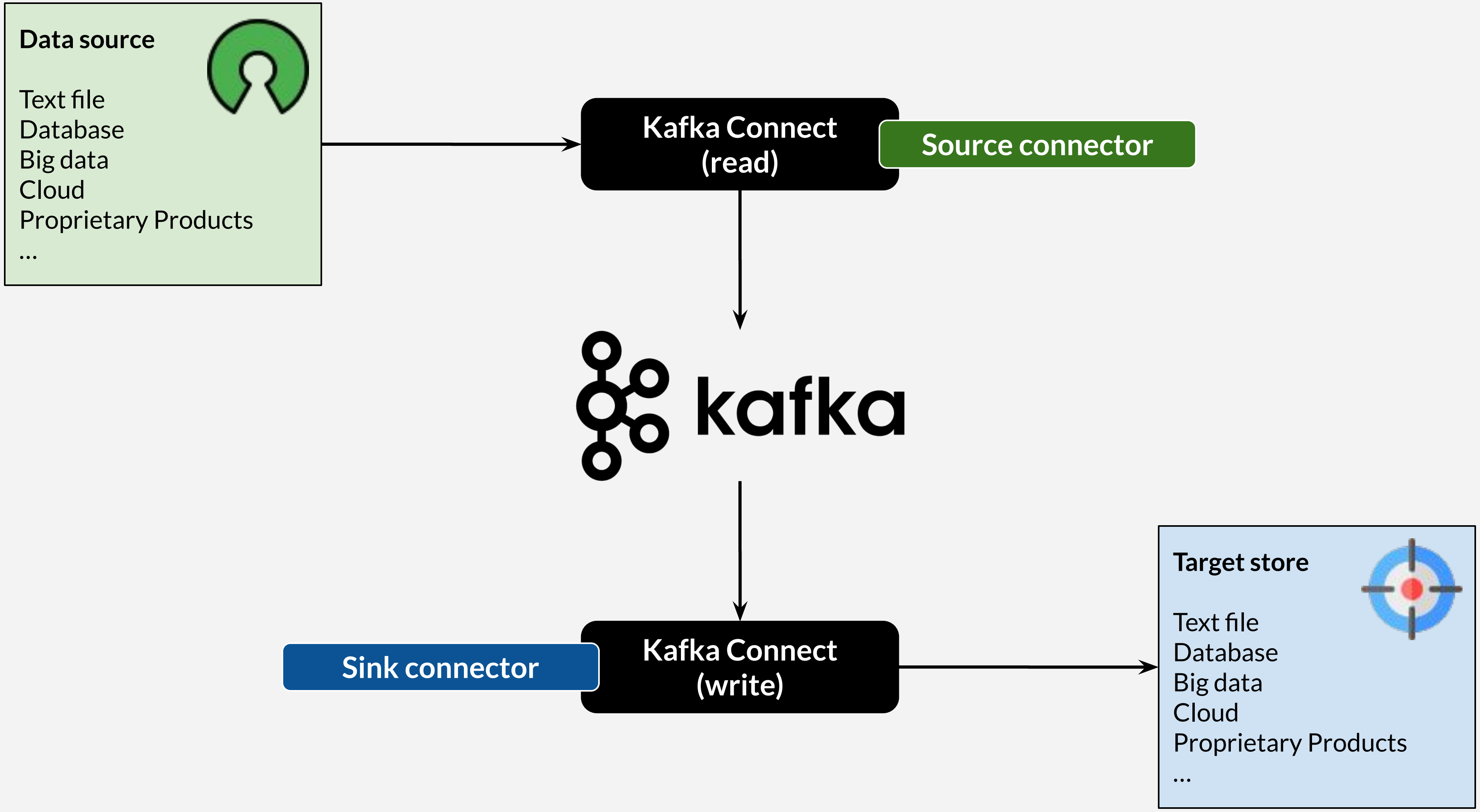
Message In, Message Out

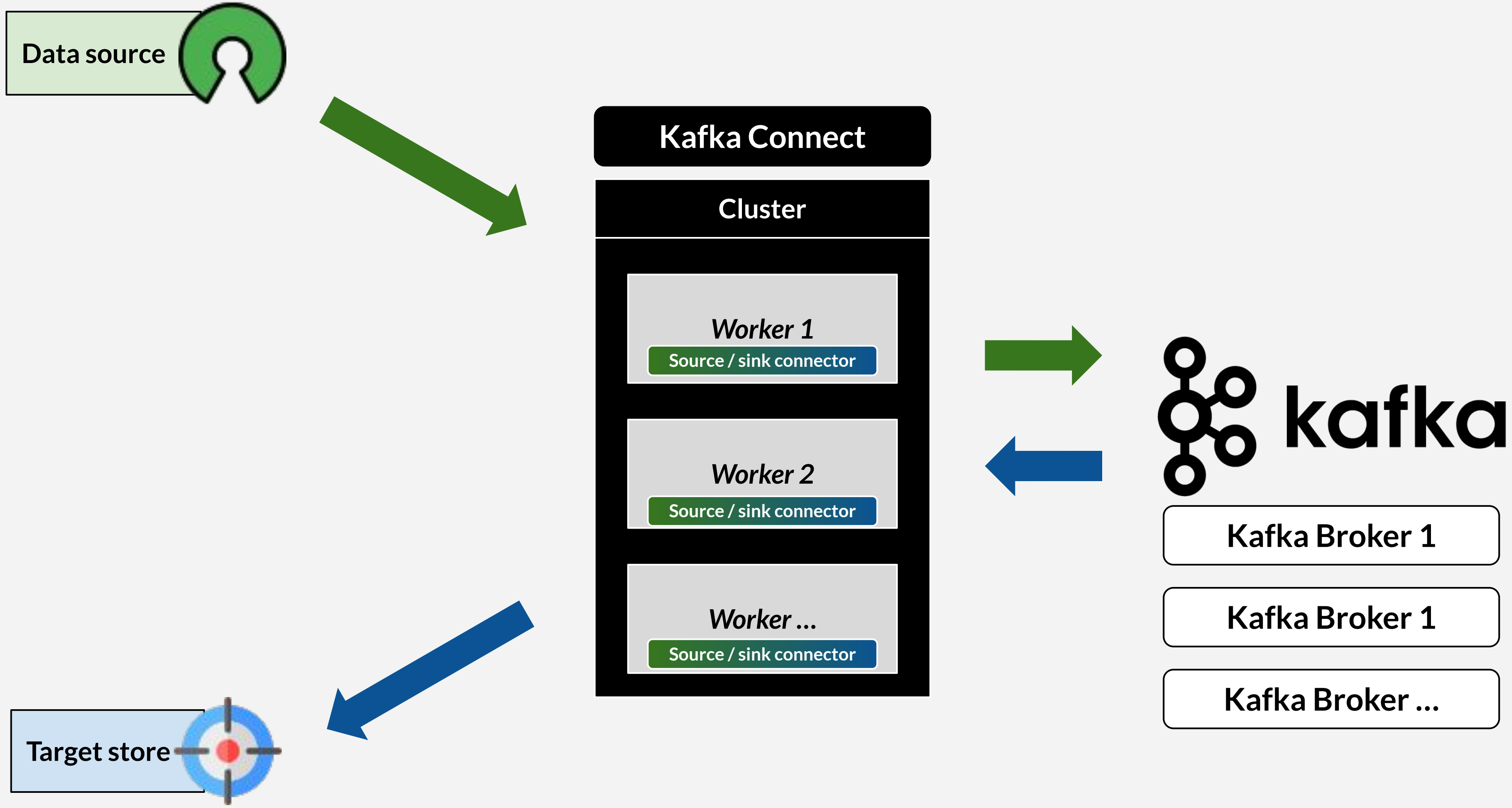
- Happens in real life
- Today we have multiple data sources (**in**)
  - text file, relational databases
  - Non relational database, email, cloud, social media, specific software, etc
  - Can be more than 100
- Multiple target data store (**out**)
  - local text file, FTP, relational / non-relational databases, big data, cloud, etc

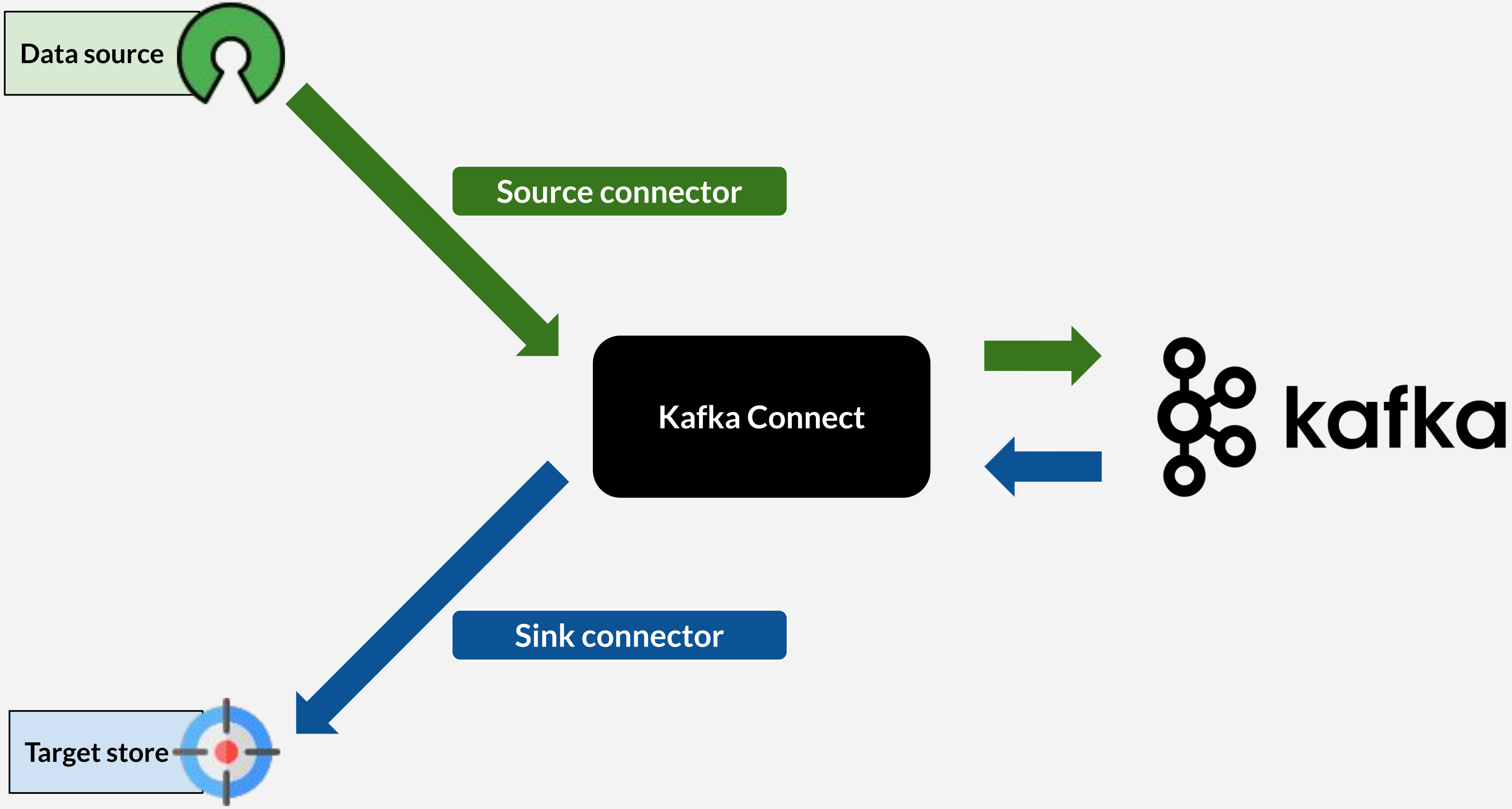
On Kafka

- Write a lot of producers and consumers
- Extra time & effort for performance & reliability
- Good news : ***you don't have to write your own!***
- People / companies already wrote them
- Read plugin : from non-kafka into kafka
- Write plugin : from kafka into non-kafka
- **Kafka Connect**

Introducing Kafka Connect


Data source
Text file
Database
Big data
Cloud
Proprietary Products
...
Kafka Connect (read)
Source connector
kafka
Sink connector
Kafka Connect (write)
Target store
Text file
Database
Big data
Cloud
Proprietary Products
...


Data source
Kafka Connect
Cluster
Worker 1
Source / sink connector
Worker 2
Source / sink connector
Worker …
Source / sink connector
kafka
Kafka Broker 1
Kafka Broker 1
Kafka Broker …
Target store


Data source
Source connector
Kafka Connect
kafka
Sink connector
Target store

Kafka Connect

- Additional platform for kafka
- Data integration
- Transfer data between Kafka - non kafka
- Horizontally scalable & fault tolerant
- Uses connectors for interact with kafka server

Connectors

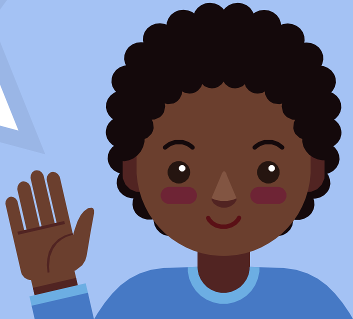
- Java jar file
- Plugin for kafka connect
- Interface between kafka and non-kafka
- **Source** connector : read (ingest) into kafka (**producer**)
- **Sink** connector : write from kafka to non-kafka (**consumer**)
- Install connector for specific need
- Write configuration (json)
- Declarative configuration
- Fast & reliable

Connectors

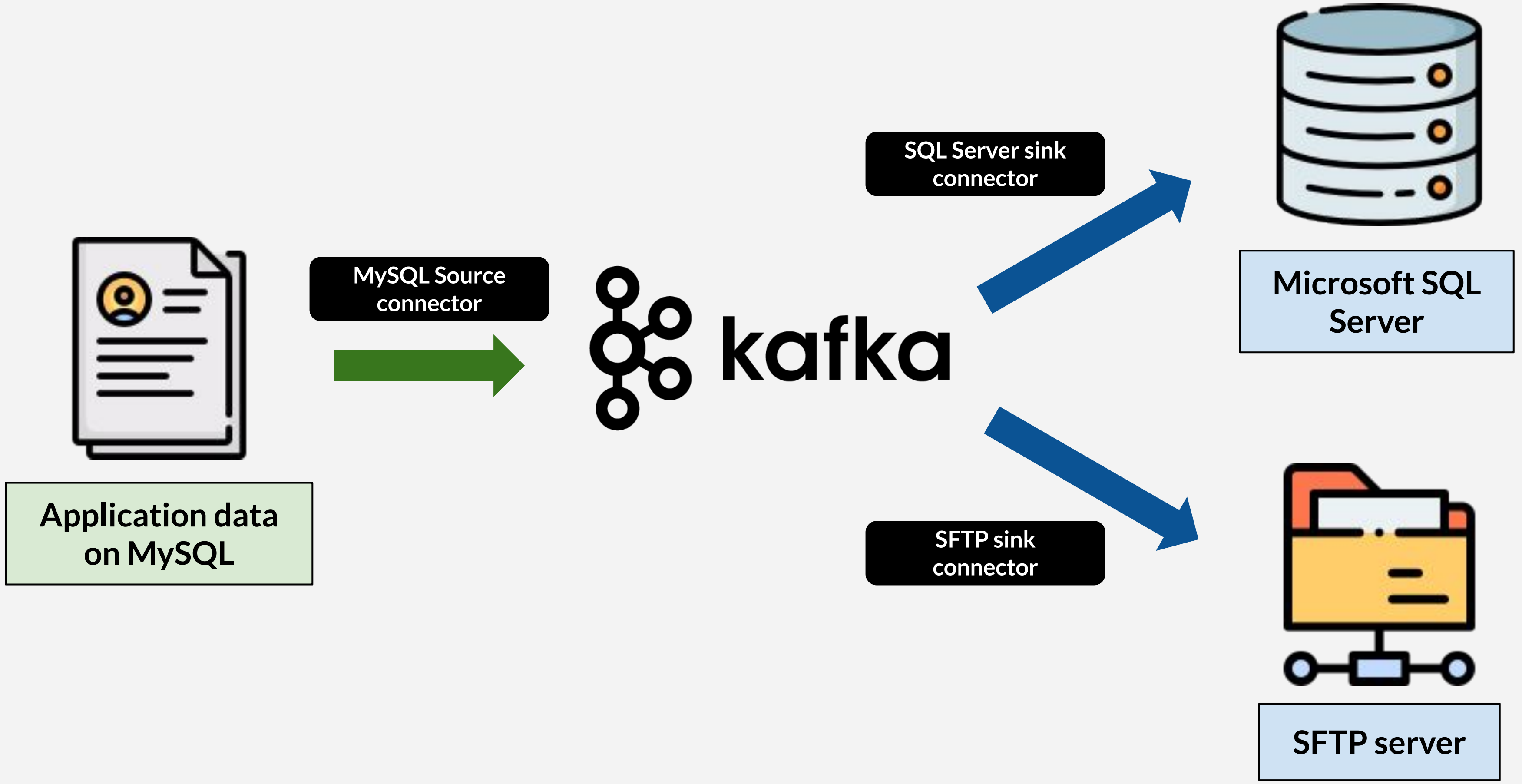
- A lot of source / sink connectors
- Shorten time and effort

How to Get Connectors?

- Curated list on ***confluent.io/hub***
- Google : *kafka source / sink connector for xxx*
- Build your own



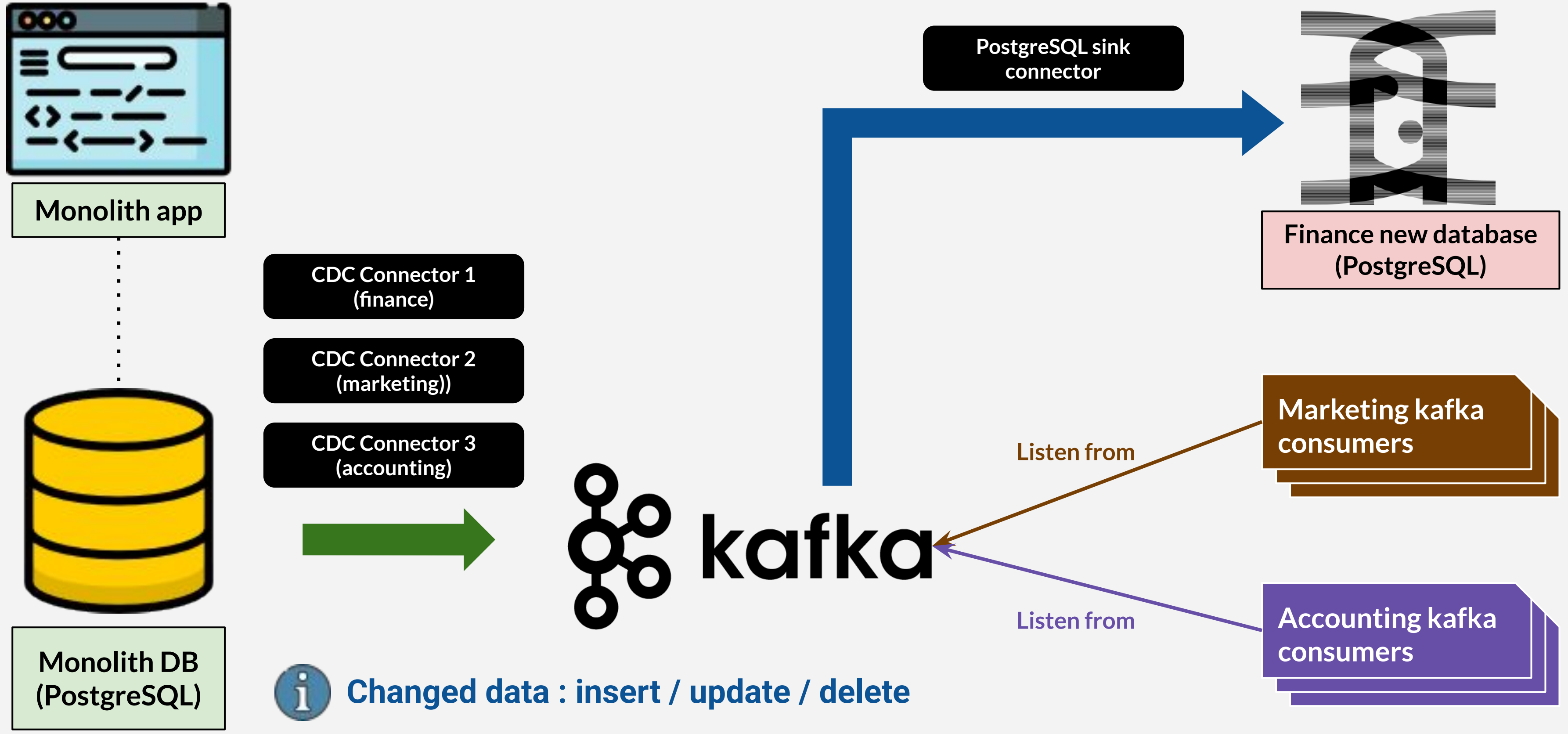
Use Case : Write to Data Stores



Use Case : Modernize Legacy System

- Modernize legacy monolith into microservices
- Modernization is hard and long
- Legacy system still needs to run during modernization
- Modernize functionalities part by part
- Use kafka connect CDC (Change Data Capture) connectors
- *Microservice Architecture & Pattern course (and discount) available on last lecture of this course*

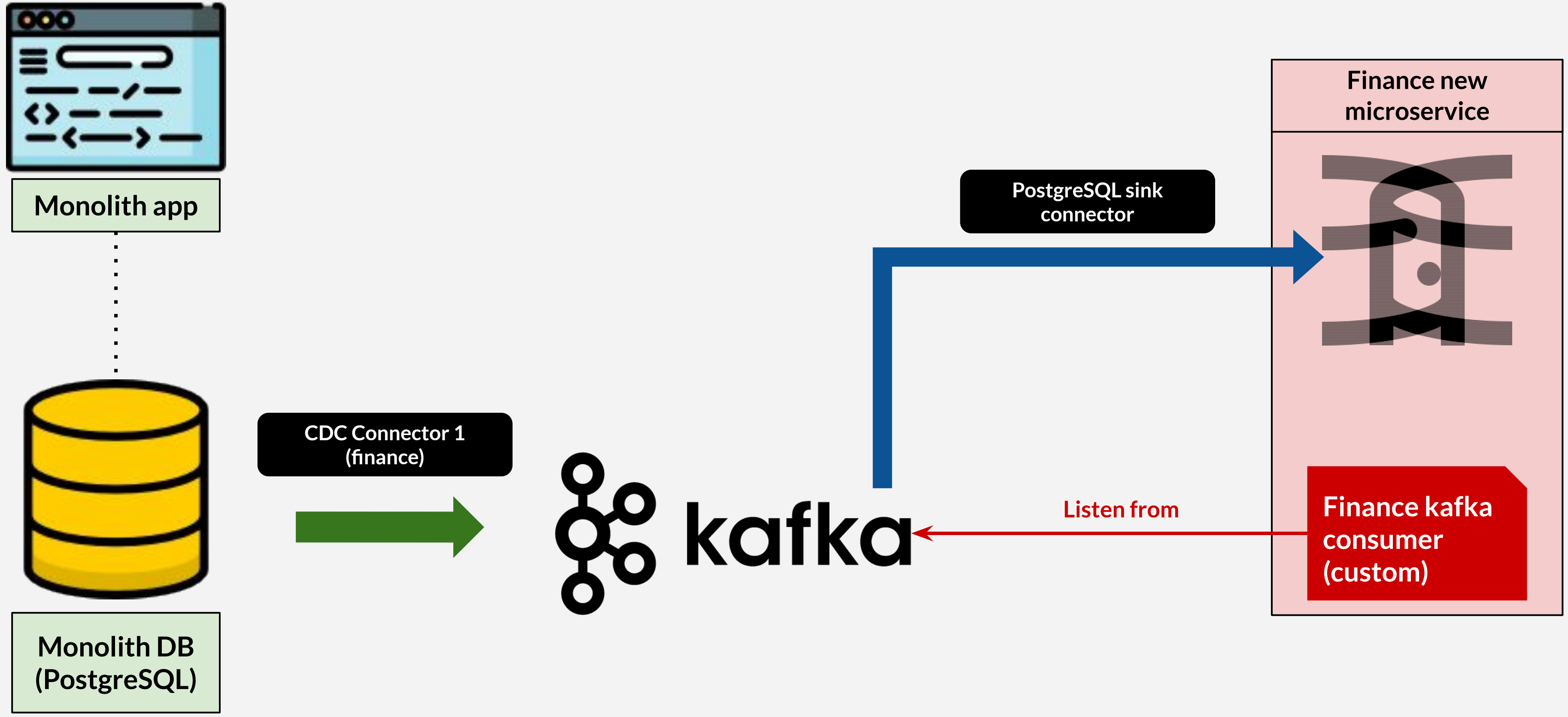
Use Case : Modernize Legacy System



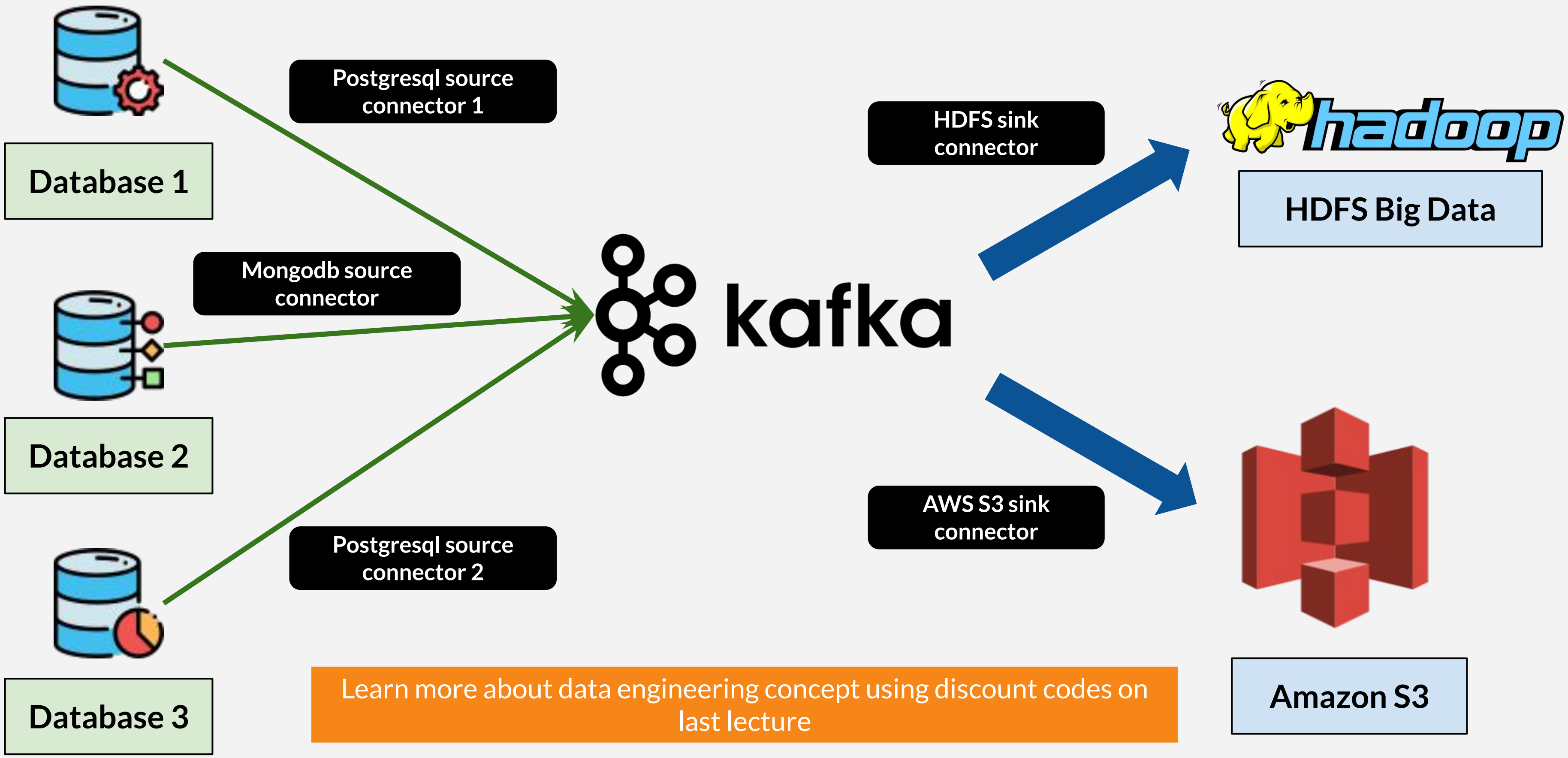
Monolith app
Monolith DB (PostgreSQL)
CDC Connector 1 (finance)
CDC Connector 2 (marketing))
CDC Connector 3 (accounting)
kafka
PostgreSQL sink connector
Finance new database (PostgreSQL)
Listen from
Marketing kafka consumers
Listen from
Accounting kafka consumers

Changed data : insert / update / delete

Use Case : Modernize Legacy System


Monolith app
Monolith DB (PostgreSQL)
CDC Connector 1 (finance)
kafka
PostgreSQL sink connector
Finance new microservice
Listen from
Finance kafka consumer (custom)

Use Case : Data Engineering ETL Pipeline



Learn more about data engineering concept using discount codes on last lecture

Kafka Connect

- *Cluster* can has one or more *workers (servers)*
- *Connector* + **user configuration** = ***task***
- A *user configuration* might spawn one or more *task*
- *Worker* will execute *task*

Kafka Connect on Docker

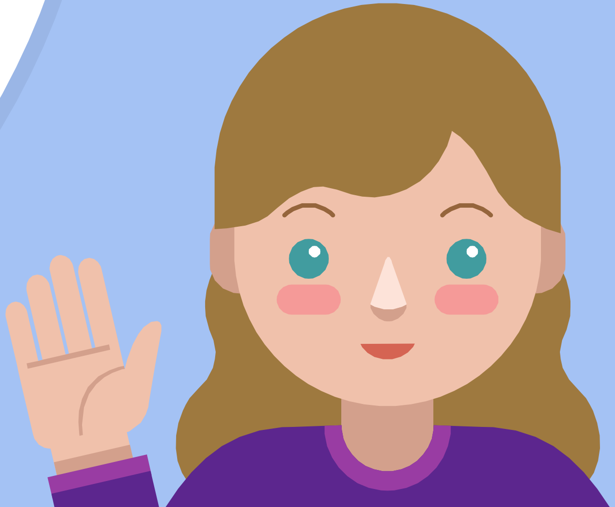
```
#> docker-compose -f [script-file] -p [project] down
```

Part	What to run (in sequence)
1 - core kafka	`#> docker-compose -f docker-compose-core.yml -p core up -d`

2 - kafka connect	`#> docker-compose -f docker-compose-core.yml -p core down`
	`#> docker-compose -f docker-compose-connect.yml -p connect up -d`
	`#> docker-compose -f docker-compose-connect-sample.yml -p connect-sample up -d`

3 - kafka full	`#> docker-compose -f docker-compose-connect.yml -p connect down`
	`#> docker-compose -f docker-compose-connect-sample.yml -p connect-sample down`
	`#> docker-compose -f docker-compose-full.yml -p full up -d`
	`#> docker-compose -f docker-compose-full-sample.yml -p full-sample up -d`

Sample Use Cases



Use Cases

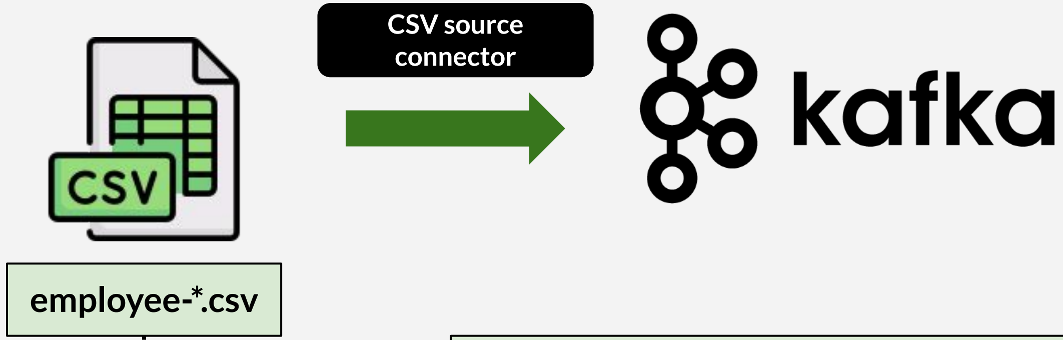
- Sample use cases
  - Basic connector usage
  - Legacy modernization with CDC
  - Data engineering
- Note
  - Need at least 4 GB free memory for docker
  - Simple use cases only
  - Data validity is not a concern
  - Temporary data (stored on docker containers)

Sample Data

- SFTP tool in this course : *Filezilla*
- PostgreSQL tool in this course : *Dbeaver*
- Feel free to use your own tools
- Available on Resource & Reference
- Sample data generated using mockaroo.com
- Mockaroo.com schema available on Resource & Reference

Use Case : Basic Connector

File Source



```
employee_id,first_name,last_name,email,gender,birth_date,salary
79-8035819,Hanny,Stubs,hstubs0@github.com,F,1996-02-02,7061
24-9676514,Val,Bonnick,vbonnick1@icq.com,M,1991-09-11,2657
61-3900911,Juanita,Lanfranconi,jlanfranconi2@rediff.com,F,1992-07-08,7241
28-0495338,Elsie,Brownlea,ebrownlea3@theglobeandmail.com,F,1985-11-03,8026
13-9980006,Chris,Kettlestringes,ckettlestringes4@indiegogo.com,M,1994-12-14,1015
```

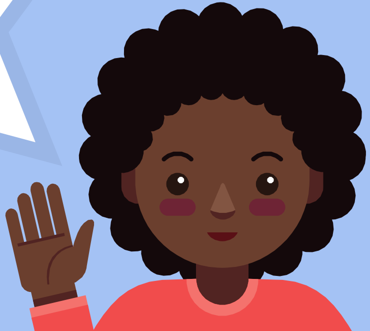
```
{
    "schema": {
        "type": "data type",
        "fields": {


        }
    },
    "payload": {
        "field1": "value 1",
        "field2": "value 2"
    }
}
```

Embedded schema
Automatically generated by source connector
Required by most sink connector

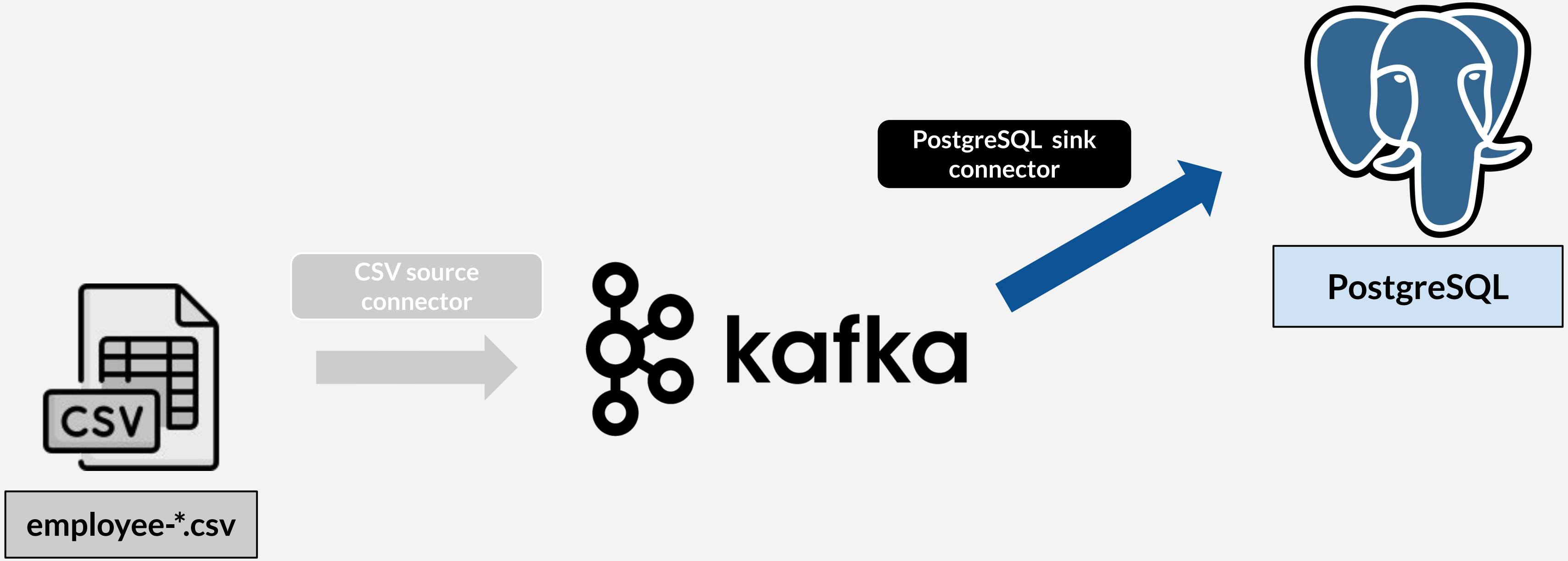
Schema

- Contract
- Set of rules for message, to be obeyed
- Field name, data type, mandatory / optional
- Kafka connect able to work with dynamic data structure
- Need to inform data structure (schema)
- Kafka source connector auto generate schema



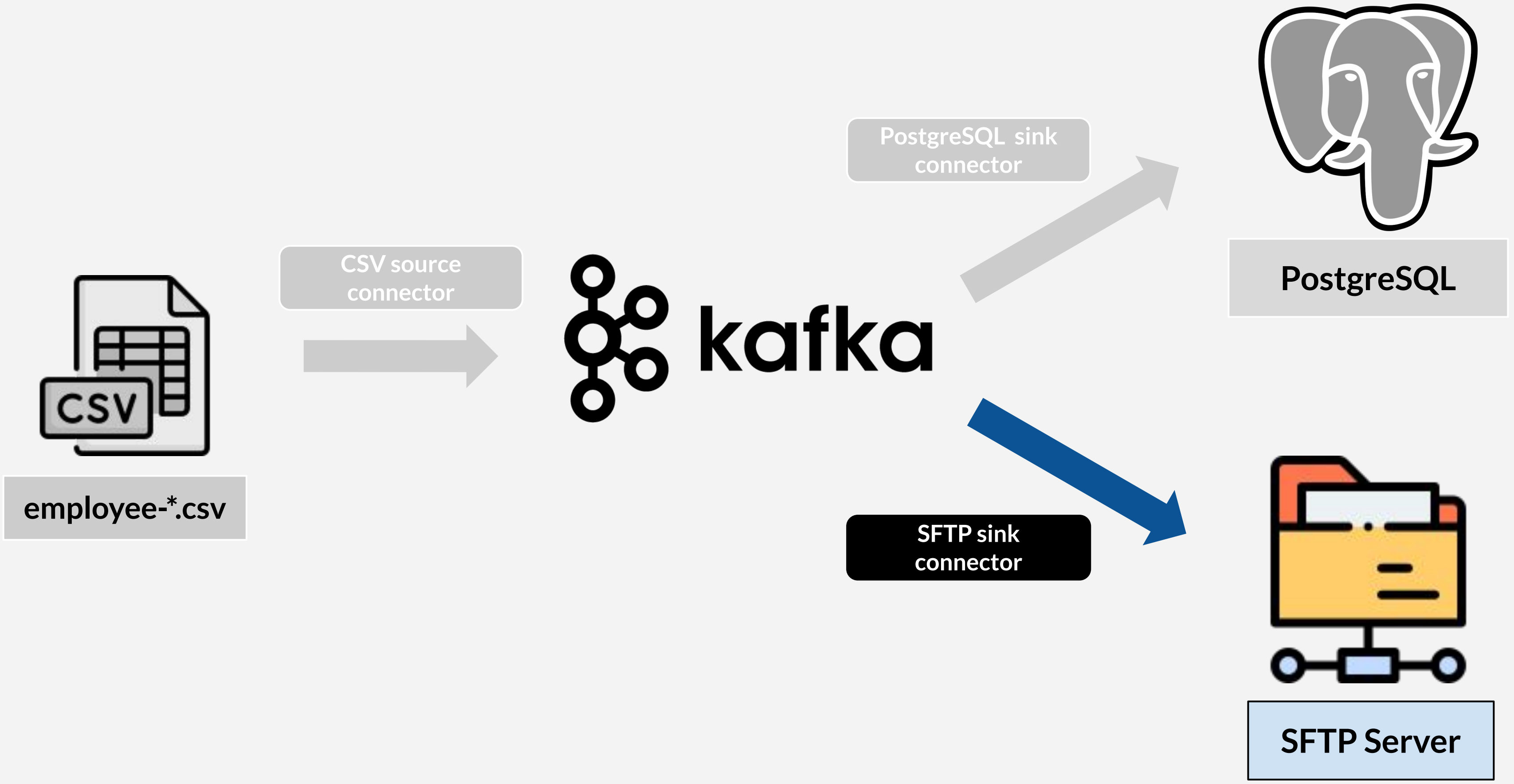
Use Case : Basic Connector

Database Sink


CSV source connector
PostgreSQL sink connector
kafka
employee-*.csv
PostgreSQL

Use Case : Basic Connector

SFTP Sink


employee-*.csv
CSV source connector
kafka
PostgreSQL sink connector
PostgreSQL
SFTP sink connector
SFTP Server

Use Case : Legacy Modernization

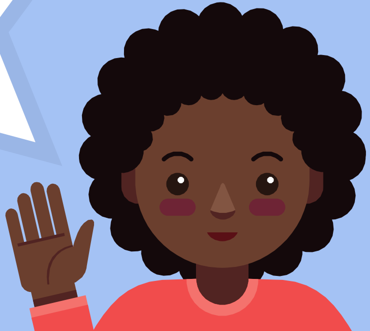
Change Data Capture

Legacy Modernization

- Existing (legacy) application without kafka
- Rebuild into microservice with kafka
- Two systems (legacy & new) running together
- Synchronize data change (new, update, delete)
- Data source is from legacy application
- How?

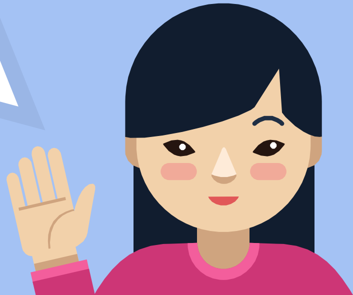
Change Data Capture - 1st Way

- Scheduler(hourly) that query data from legacy database based on `last_updated_date` field
- But
  - Not every table has `last_updated_date` field
  - One hour is too long, need low latency (e.g. 60 seconds)
  - Scheduler with short delay might problematic

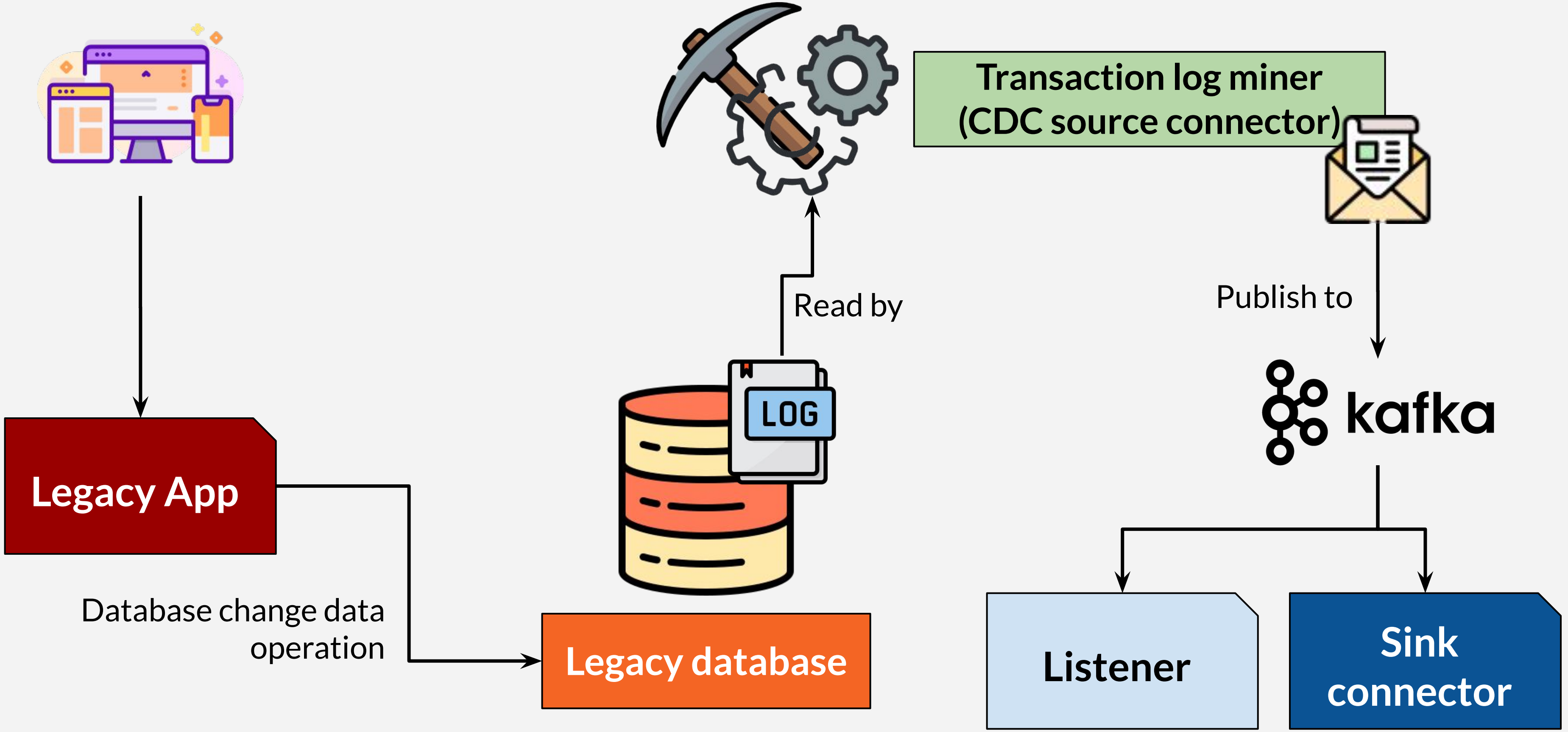


Change Data Capture - 2nd way

- Each data change directly sent to kafka
- Coding kafka integration points in legacy application is painful & not worth the effort
- Good if we can implement this, but with minimal effort
- Possible without writing code
- Database transaction log tailing
- Kafka connect with CDC (change data capture) source connector

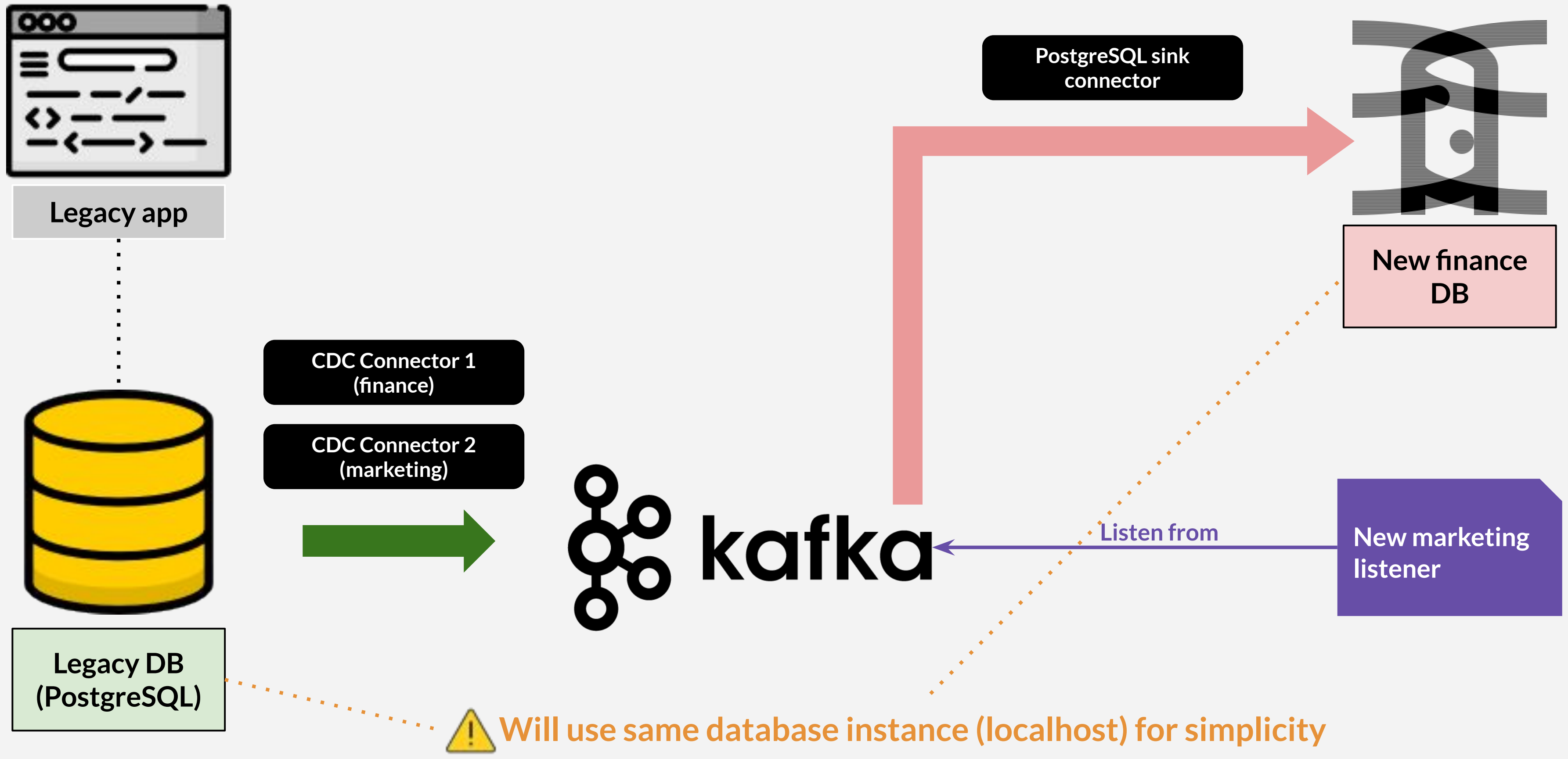


Transaction Log Tailing



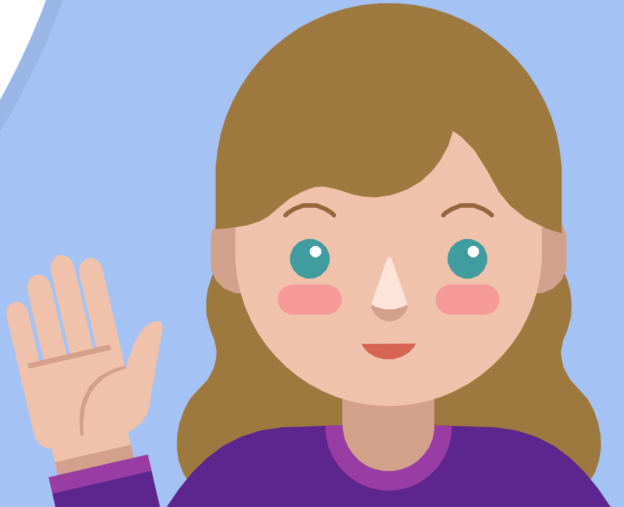
Microservice Architecture & Pattern

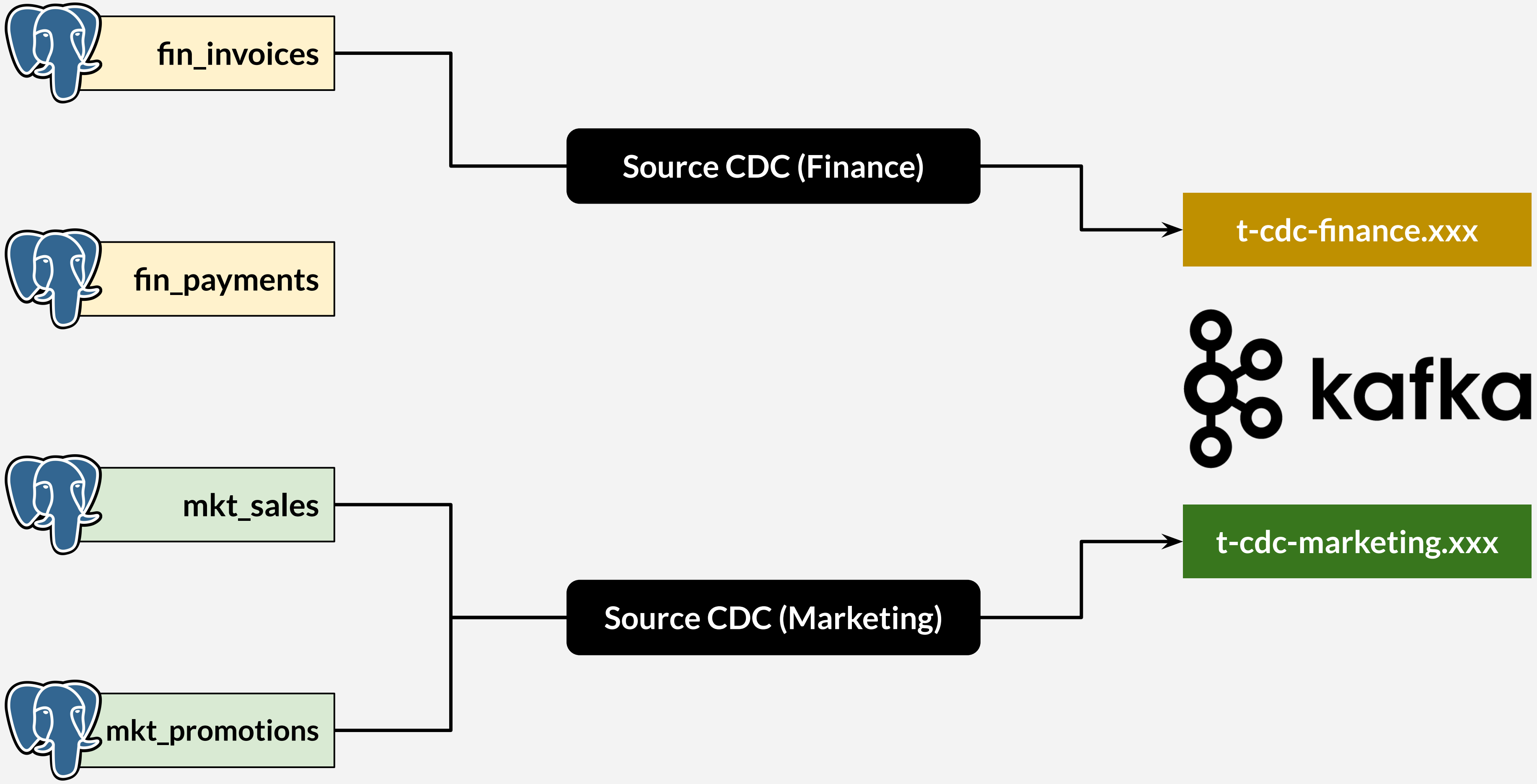
- Transaction log tailing is just one pattern to ease your life
- Available in my course **Microservice Architecture & Pattern with Java & Kafka**
- Grab discount code on last section of this course


Legacy app
PostgreSQL sink connector
New finance DB
CDC Connector 1 (finance)
CDC Connector 2 (marketing)
kafka
Listen from
New marketing listener
Legacy DB (PostgreSQL)
Will use same database instance (localhost) for simplicity

Use Case : Legacy Modernization

CDC PostgreSQL Connector




fin_invoices
fin_payments
mkt_sales
mkt_promotions
Source CDC (Finance)
Source CDC (Marketing)
t-cdc-finance.xxx
kafka
t-cdc-marketing.xxx

Use Case : Legacy Modernization

PostgreSQL Sink Connector

Use Case : Legacy Modernization

Marketing Consumer

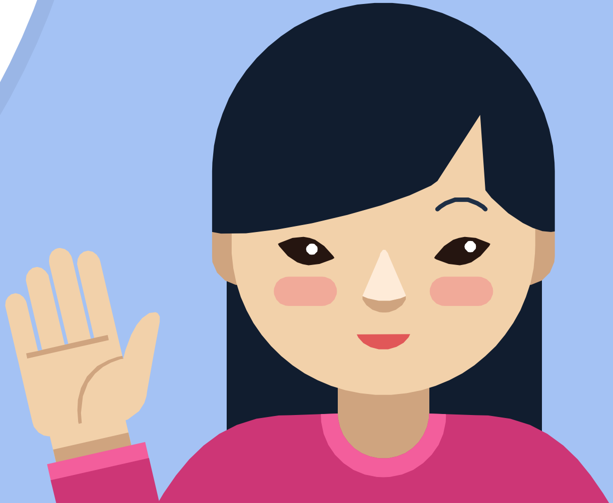
Spring Initializr

- start.spring.io
- Java project with gradle
- Group : **`com.course.kafka`**
- Artifact : **`kafka-connect-sample`**
- Package name : **`com.course.kafka`**
- Dependency : **Spring Kafka, Spring Kafka Stream**

Json Deserializer?

- Why use default (String deserializer)
- JSON (De)serializer is part of Spring
- Spring specific mechanism to (de)serialize
- Works out-of-the-box if publisher uses Spring JSON serializer
- CDC connector is **not** part of spring framework
- Process JSON string manually

It's not A One-Stop-Solution



Kafka Connect

- Basic use case works
- Complex use case needs more effort
- In real life:
  - Kafka connect can helps
  - But it’s not a magical problem solver
- Need proper Kafka Connect configuration

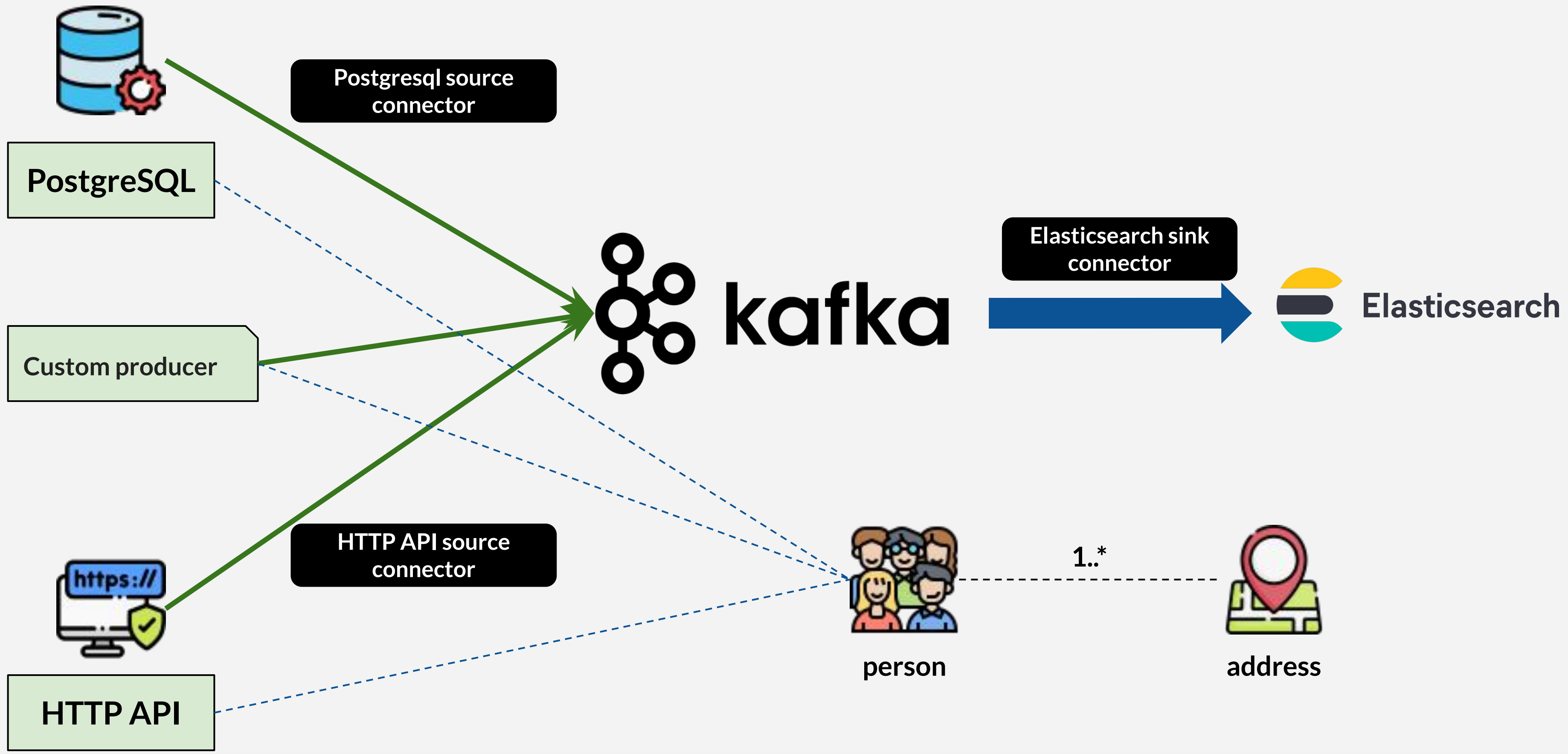
Kafka Connect

- Not entirely without cost
- Connector might need license
- Example : cloud storage
- Amazon / Google / Azure

Use Case : Data Engineering

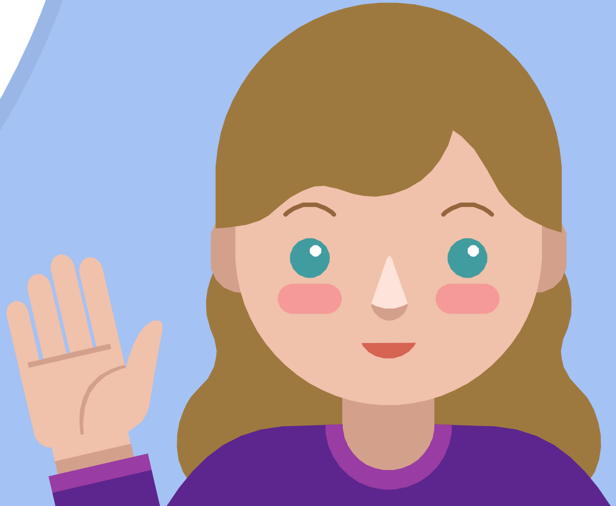
Database Source

Use Case : Data Engineering ETL Pipeline


PostgreSQL
Postgresql source connector
Custom producer
HTTP API source connector
HTTP API
https://
kafka
Elasticsearch sink connector
Elasticsearch
person
1..*
address

Use Case : Data Engineering

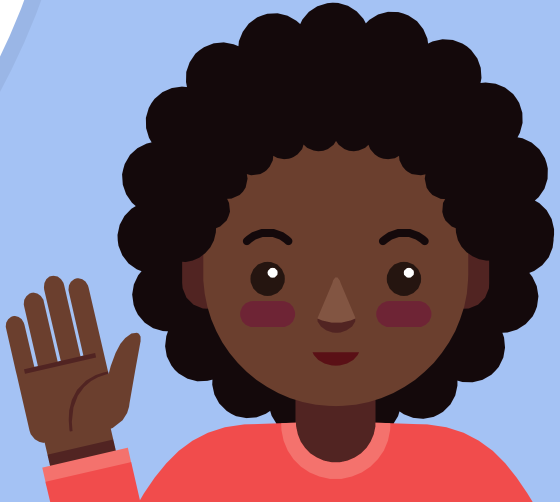
HTTP Source



Use Case : Data Engineering

Custom Source

Data Format Differences



Database

```
{
  "schema":{
    ...
  },
  "payload":{
    "person_id":2100,
    "id_card_number":"401-99-8581",
    "full_name":"Hamlin Kayes",
    "email":"hkayes2r@bigcartel.com",
    "address":"0 Westerfield Place",
    "city":"Sājūr",
    "postal_code":null
  }
}
```

Schema + payload
Actual data in payload
Using snake_case
Single address

HTTP

```
{
  "schema":{
    ...
  },
  "payload":{
    "value":"{\"id_card_number\":\"444-25-9069\",\"full_name\":\"Rebekkah
Toller\",\"email\":\"rtoller8@ezinearticles.com\",\"addresses\":[{\"address_id\":6710,
\"address\":\"5 Lillian
Parkway\",\"city\":\"Lyubim\",\"postal_code\":\"152470\"},{\"address_id\":7860,\"addre
ss\":\"86278 Calypso Trail\",\"city\":\"Gaotieling\",\"postal_code\":null}]}",
    "key":"9d38e4bf-3b1d-32d9-b78b-7a8135233597",
    "timestamp":1644280143619
  }
}
```

Schema + payload
Actual data in payload.value
Using snake_case
Array address

Custom

```
{
  "personId":84527,
  "fullName":"Joe Auer",
  "email":"joeauer99@hotmail.com",
  "addresses":[
    {
      "addressId":64288,
      "address":"4639 Everett Walks",
      "city":"Stacyfort",
      "postalCode":null
    }
  ]
}
```

No schema, no payload
Represents actual data
Using camelCase
Array address

Database

```
{
  "schema":{
    ...
  },
  "payload":3102
}
```

Schema + payload
Actual data in payload
Address ID

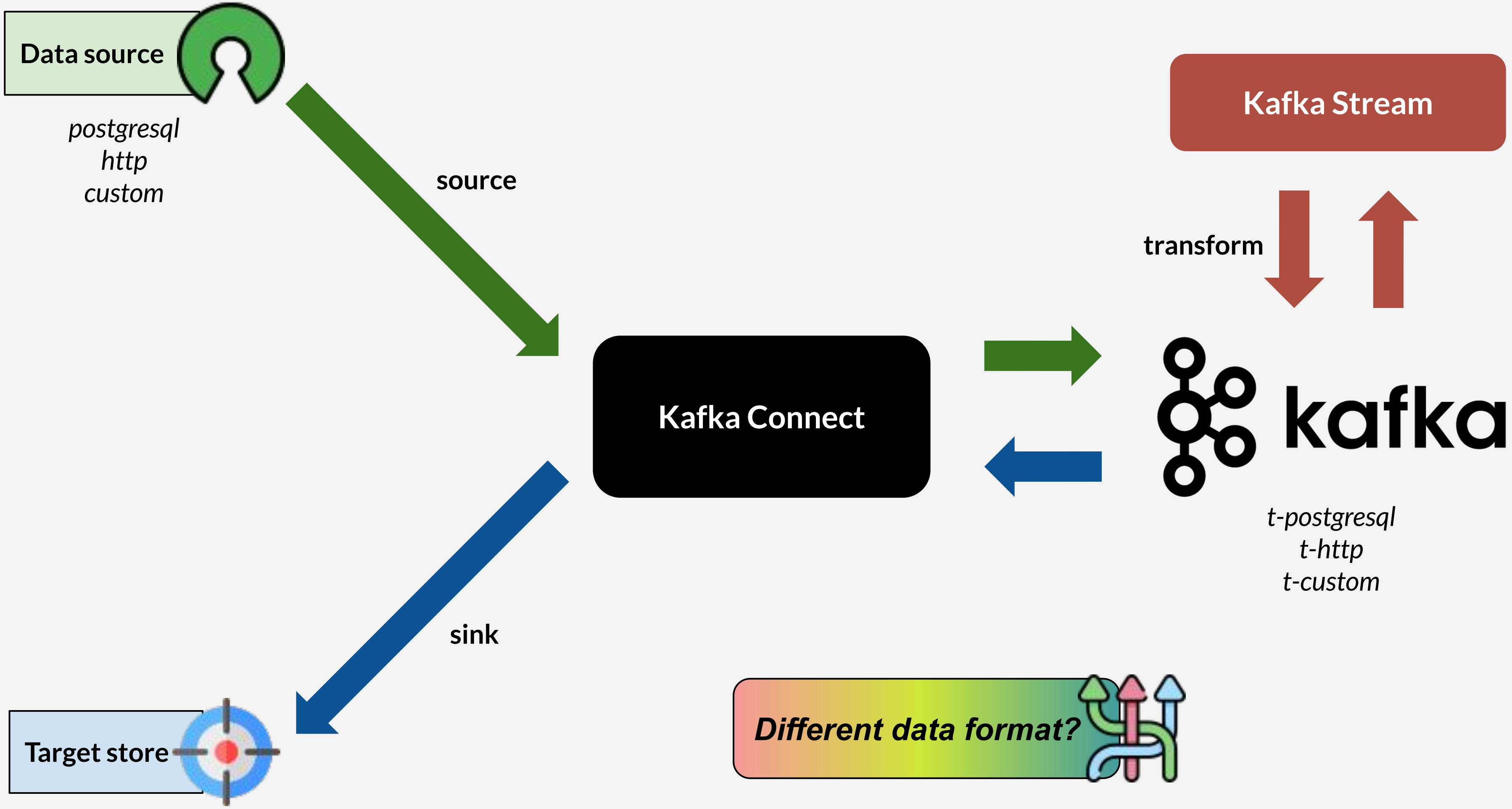
HTTP

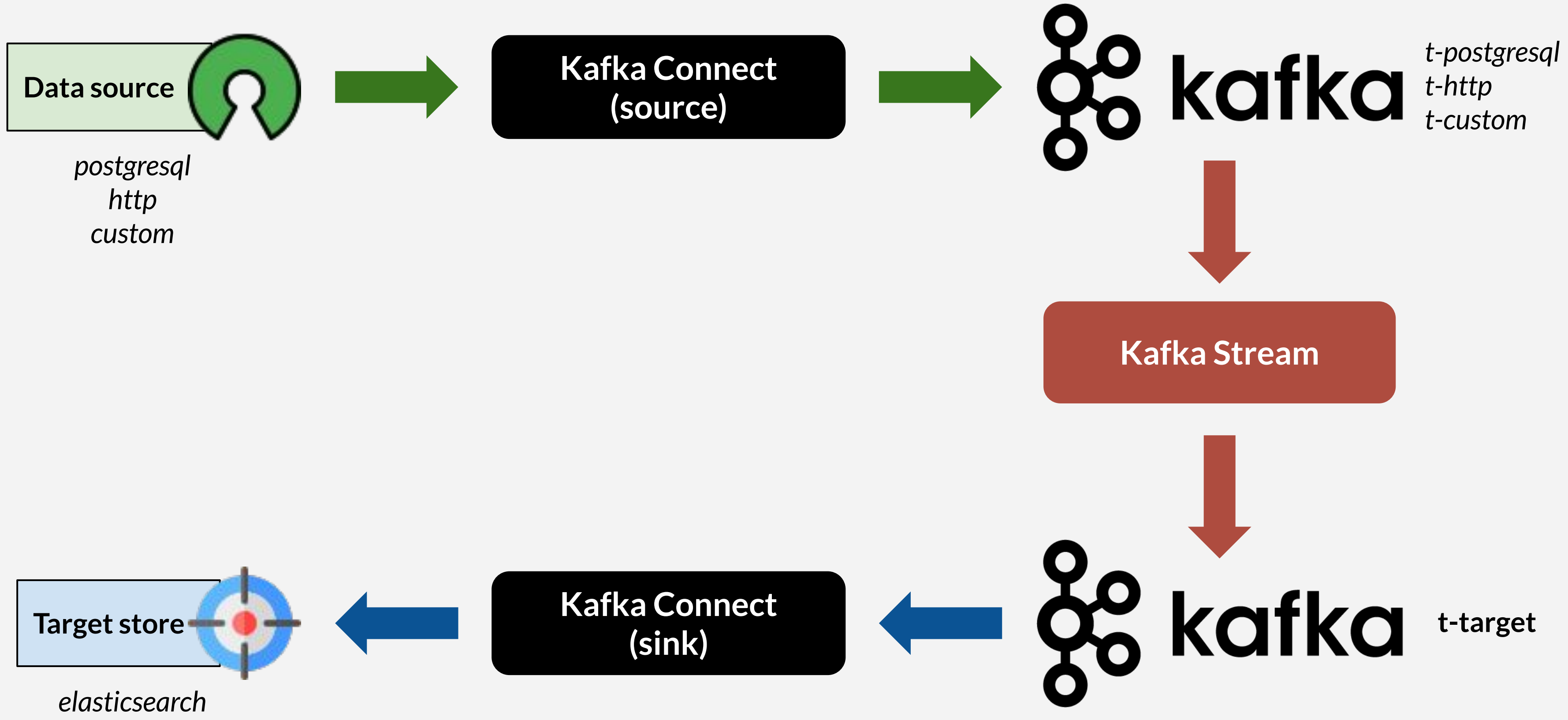
```
{
  "schema":{
    ...
  },
  "payload":{
    "key":"62aeb383-539a-3ea5-99cb-a16fb4b3681b"
  }
}
```

Schema + payload
Actual data in payload.key
Random UUID

Custom

No key


Data source
postgresql
http
custom
source
Kafka Connect
sink
Target store
Kafka Stream
transform
kafka
t-postgresql
t-http
t-custom
Different data format?


Data source
postgresql
http
custom
Kafka Connect (source)
kafka
t-postgresql
t-http
t-custom
Kafka Stream
kafka
t-target
Kafka Connect (sink)
Target store
elasticsearch

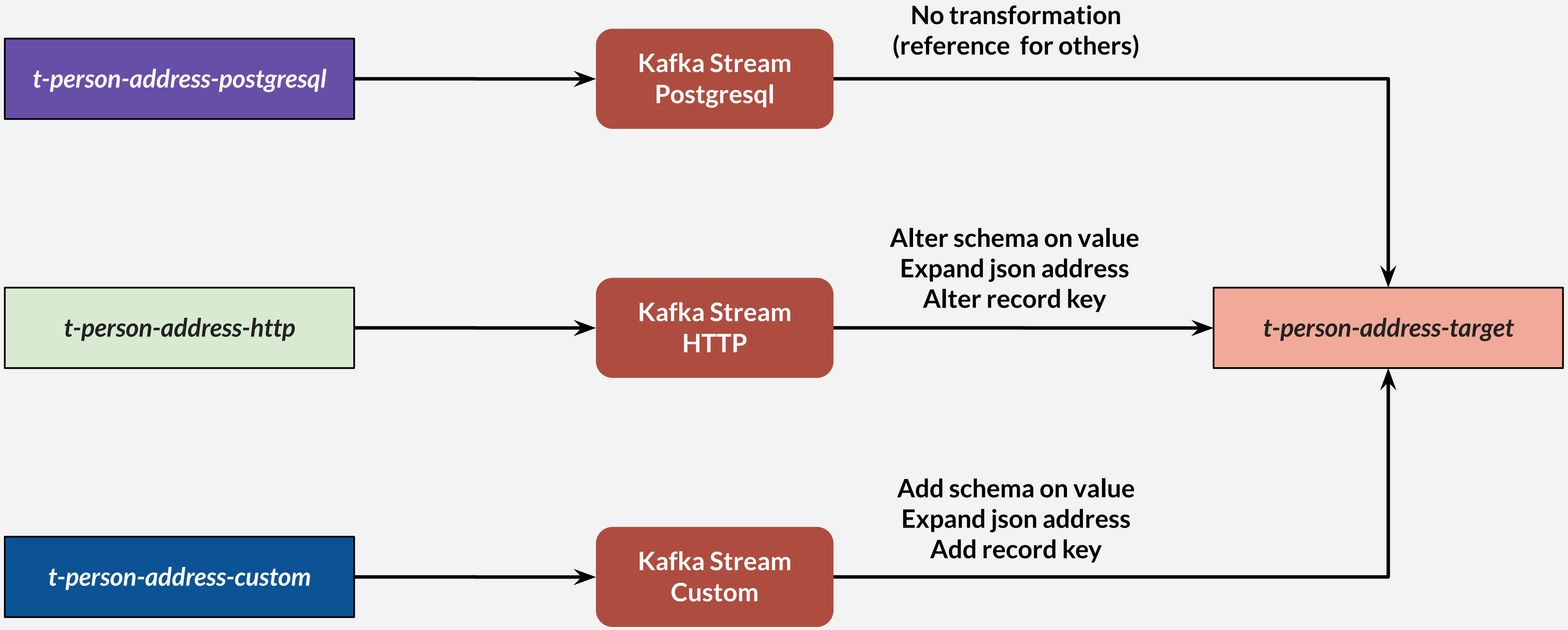
Kafka Stream & Kafka Connect

Code Overview

Kafka Stream & Kafka Connect

- Source topics (from source connectors)
  - *t-person-address-postgresql*
  - *t-person-address-http*
  - *t-person-address-custom*
- Target topic (for sink connector)
  - *t-person-address-target*
- Use java generic coding practice
- String serde for key and value

Kafka Stream Transformation



All

Topic	*Class*	*Description*
all	`message.KafkaConnectMessage`	Wrapper class for kafka connect message that contains `schema` and `payload`. Payload uses java generic.
	`schema.KafkaConnectSchema`	Represents `schema` on `KafkaConnectMessage`

PostgreSQL Source

Topic	*Class*	*Description*
t-person-address-postgresql	-	No need to define class representing message. This is the message format we will use on target topic.
	`stream.PersonAddressFromPostgresqlStream`	Kafka stream: 1. Take data from *t-person-address-postgresql* 2. Send as is, to *t-person-address-target* No transformation needed, since the record format (key & value) is what we expect.

HTTP Source

Topic	*Class*	*Description*
t-person-address-http	`message.KafkaConnectPersonAddressFromHttpMessage`	Represents message retrieved by HTTP source connector (contains full json string on `payload.value`)
	`message.KafkaConnectPersonMessageSnakeCase` `message.KafkaConnectAddressMessageSnakeCase`	Holder for converting json string in `payload.value` into java object (the actual person & address data). Fields are snake_case.
	`stream.PersonAddressFromHttpStream`	Kafka stream: 1. Take data from *t-person-address-http* 2. Transform (expand json `addresses` and convert each address into one record). Alter schema on record key & value 3. Send to *t-person-address-target*

Custom Source

Topic	*Class*	*Description*
t-person-address-custom	`message.PersonMessage` `message.PersonAddress`	Represents message published by dummy scheduler. Fields are camelCase.
	`stream.PersonAddressFromCustomStream`	Kafka stream: 1. Take data from *t-person-address-custom* 2. Transform (expand json **`addresses`** and convert each address into one record). Add schema on record key & value 3. Send to *t-person-address-target*

Target Sink (From All Sources)

Topic	*Class*	*Description*
t-person-address-target	**`message.KafkaConnectPersonTargetMessage`**	Represents **`payload`** on record value (class **`KafkaConnectMessage`**) to be published to *t-person-address-target*. This actually the same format as data published by JDBC source connector at *t-person-address-postgresql*
	`schema.KafkaConnectPersonAddressTargetKeySchema`	Singleton for creating **`schema`** on record key to be published to *t-person-address-target*.
	`schema.KafkaConnectPersonAddressTargetValueSchema`	Singleton for creating **`schema`** on record value to be published to *t-person-address-target*.

Kafka Stream & Kafka Connect

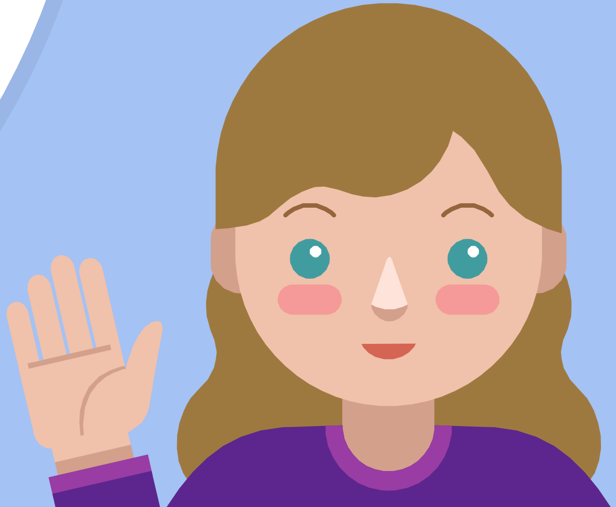
Code Hands on

Tip : Override Converter

Use Case : Data Engineering

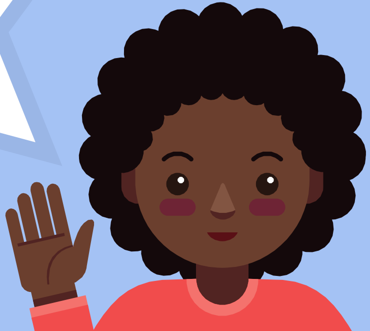
Elasticsearch Sink

AI Assistant and Kafka Connect



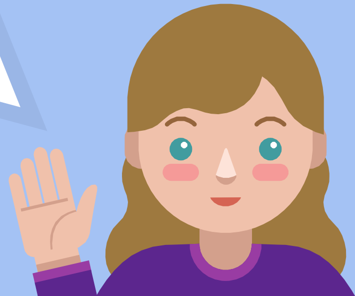
AI Assistant & Kafka Connect

- Can use AI assistant up to a certain point
- Generate basic Kafka Connect JSON configuration
- Sometimes fine-tuning by human is better
  - Limited / outdated AI assistant knowledge base
  - Might need long & detailed instruction
  - Might be faster to read & configure the connector based on recent documentation



AI Assistant & Kafka Connect

- Not much AI assistant usage in the Kafka Connect lessons
- Some prompts example provided on ***Resources & References***
- Generate JSON body that resembles connectors in this course
- Only some examples
- Difference between AI assistant vs manual
- AI assistant output is non-deterministic
- AI assistant generated JSON body will not be executed in this course



AI Assistant & Kafka Connect

- Explain existing configuration
- Ask AI assistant to explain existing JSON configuration

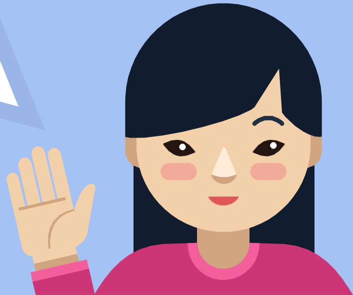
Kafka User Interface

Kafka GUI

- Use GUI instead console
- Available on market
- Might need license
- Check tool’s website for pricing & license

Kafka GUI

- Confluent Control Center (confluent.io)
- Kafdrop (github.com/obsidiandynamics/kafdrop)
- Kafka-ui (github.com/provectus/kafka-ui)
- Lenses (lenses.io)
- Conduktor (conduktor.io)
  - We will use this



Conduktor

Conduktor

- Using new topics
- Dummy producer, consumer
- Dummy kafka stream
- Source code available on **Resource & Reference**
- *Note : Conduktor features might different from the course*