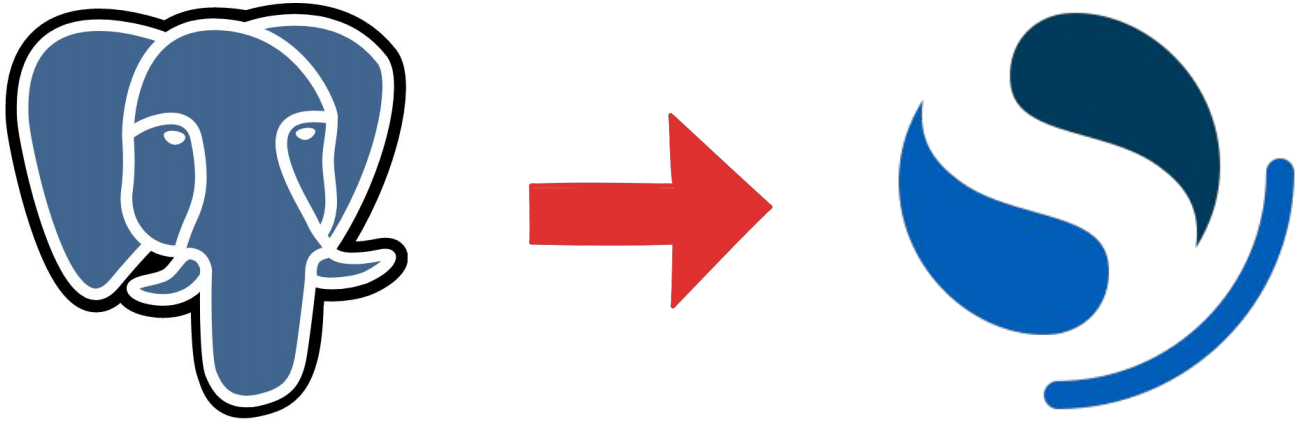
From Postgres To OpenSearch In No Time

Gunnar Morling
Software Engineer, Decodable
@gunnarmorling

decodable



Today's Mission

Learn About...

Change Data Capture

Stream Processing

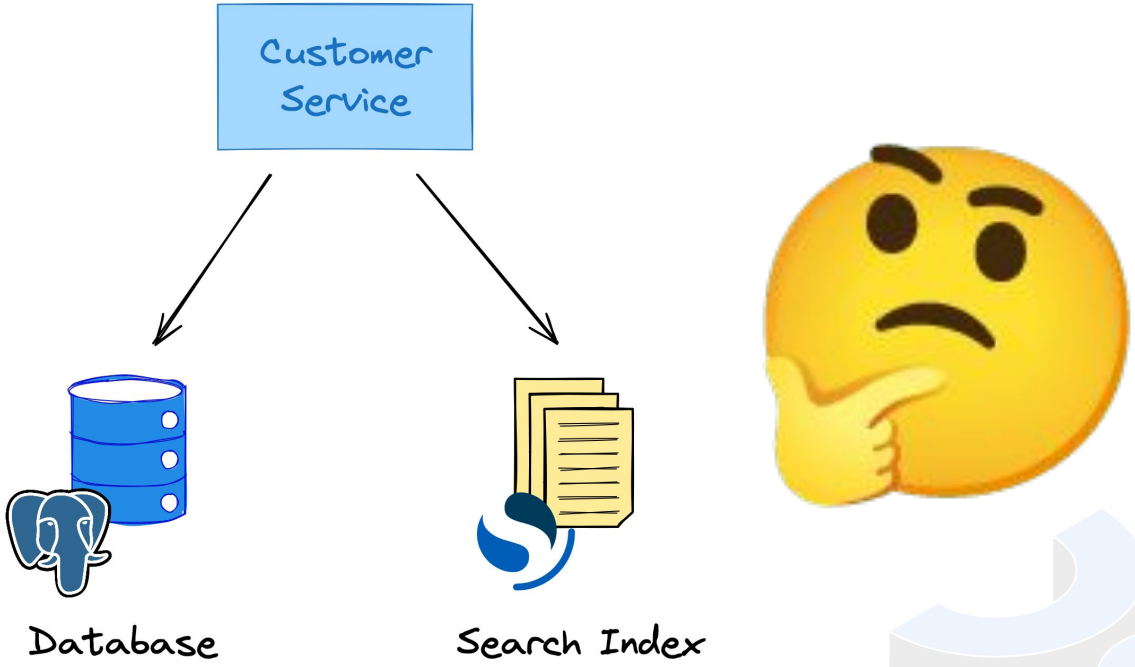
Demo

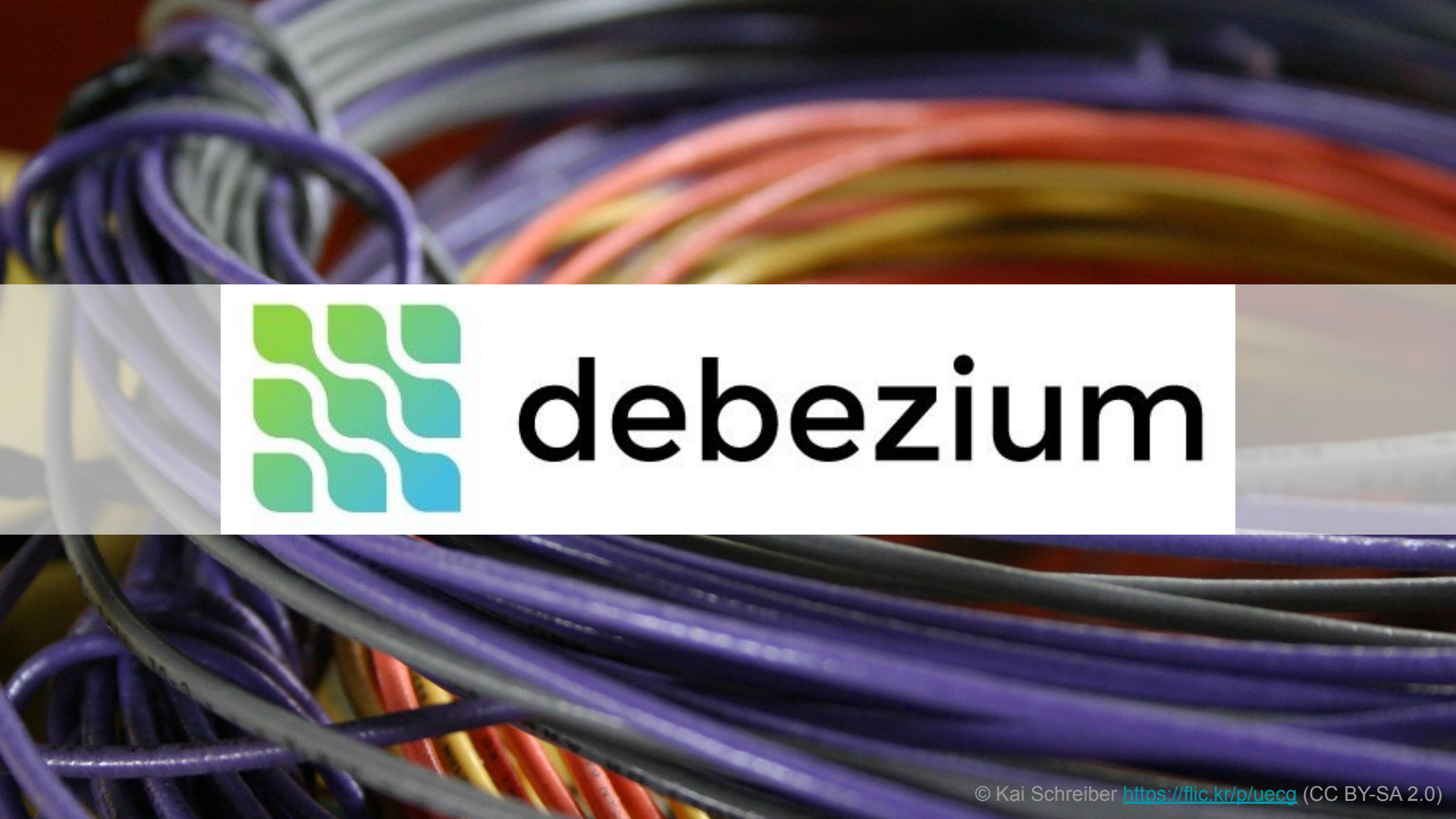
Gunnar Morling

- Software engineer at **Decodable**
- Former project lead of **Debezium**
- **kcctl** 🧸, JfrUnit, ModiTect, MapStruct
- Spec Lead for Bean Validation 2.0
- Java Champion

Updating the Search Index

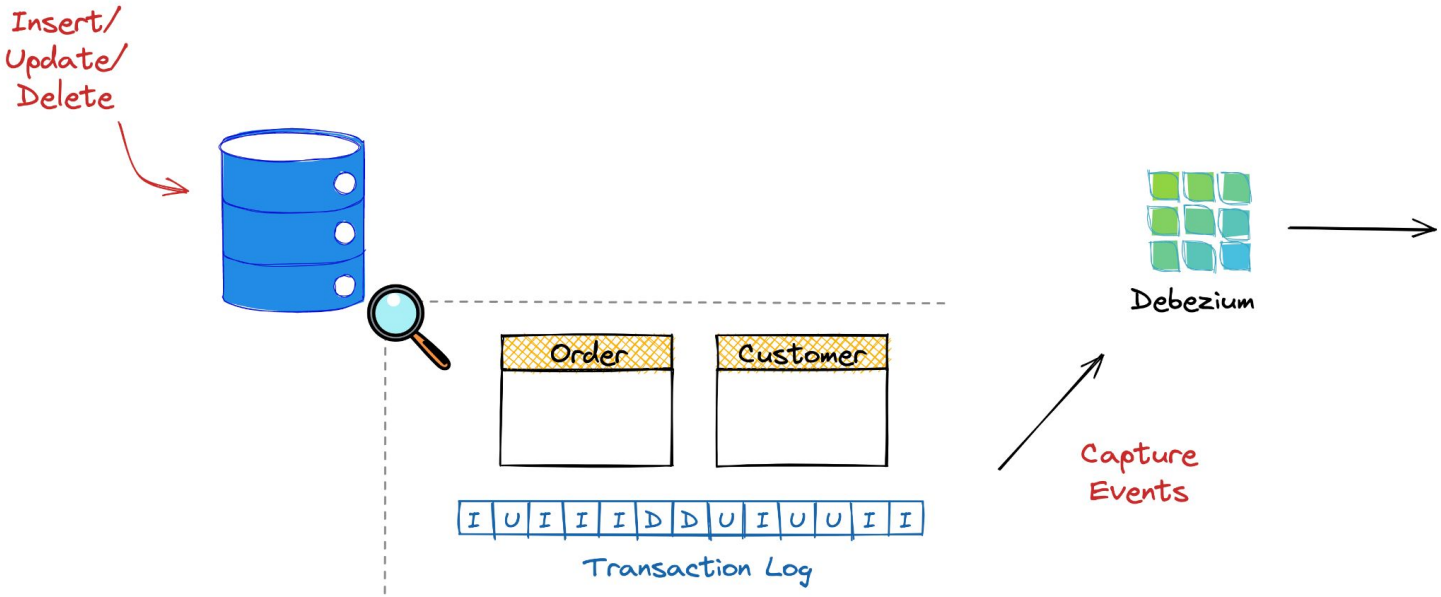
One Idea?





Debezium

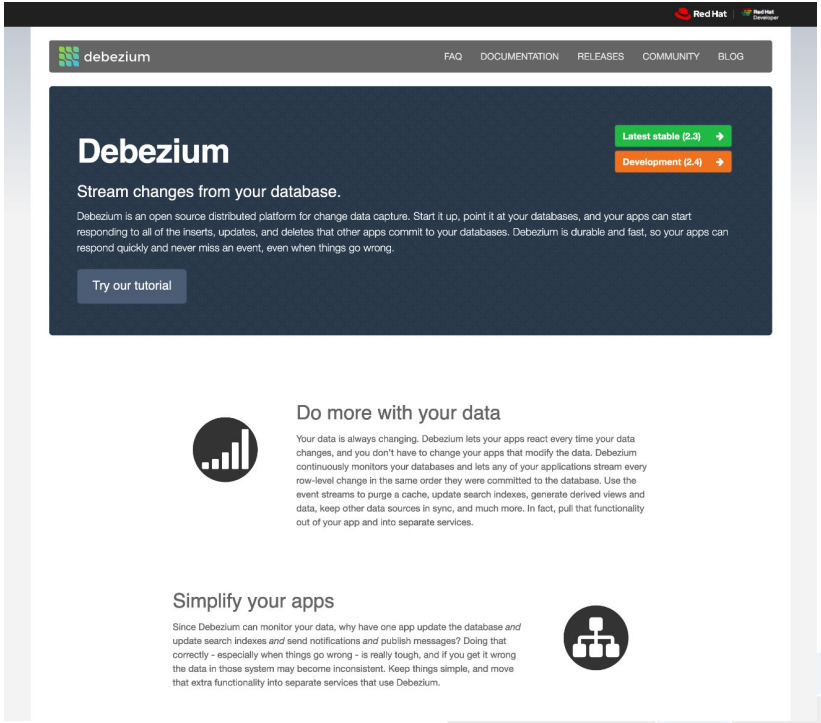
Log-Based Change Data Capture



Debezium in a Nutshell

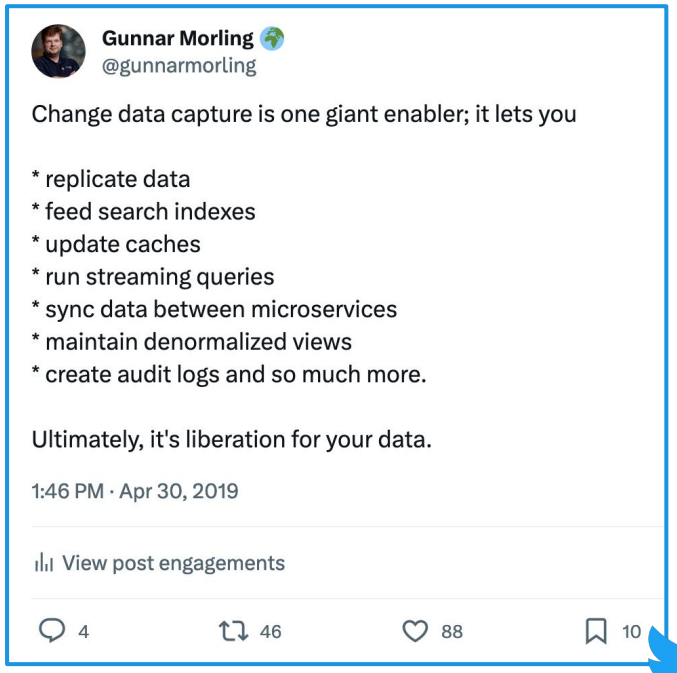
Open-Source Change Data Capture

- A CDC Platform
  - Based on **transaction logs**
  - Snapshotting, filtering, etc.
  - Outbox support
  - Web-based **UI**
- Fully **open-source**, very active community
- Large production deployments



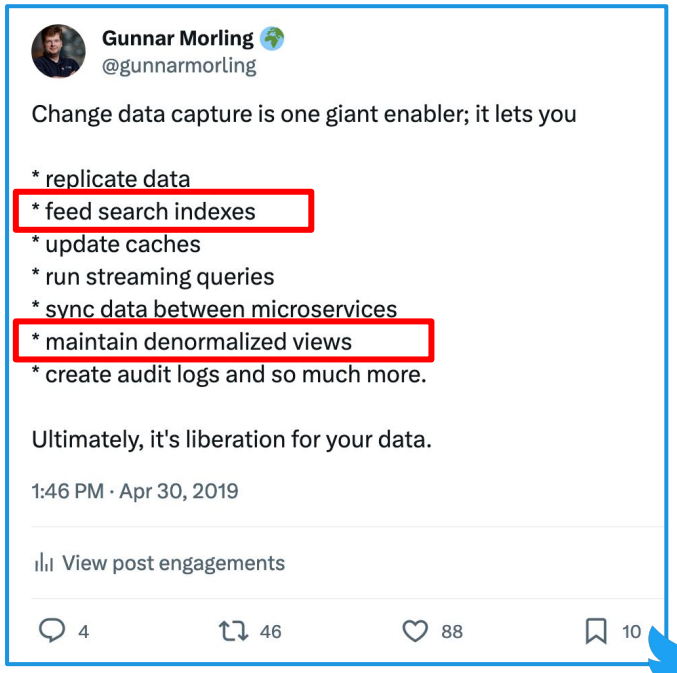
Change Data Capture

Liberation for Your Data



Change Data Capture

Liberation for Your Data



Debezium

Supported Databases

- **Core**
  - MySQL, MariaDB
  - Postgres
  - SQL Server
  - MongoDB
  - Db2, Informix
  - Oracle
- **Community-led**:
  - Vitess, Cassandra, Spanner
- **External**: ScyllaDB, Yugabyte

Debezium: Data Change Events

- **Old** and **new row state**
- **Metadata** on table, TX id, etc.
- **Operation type**, timestamp

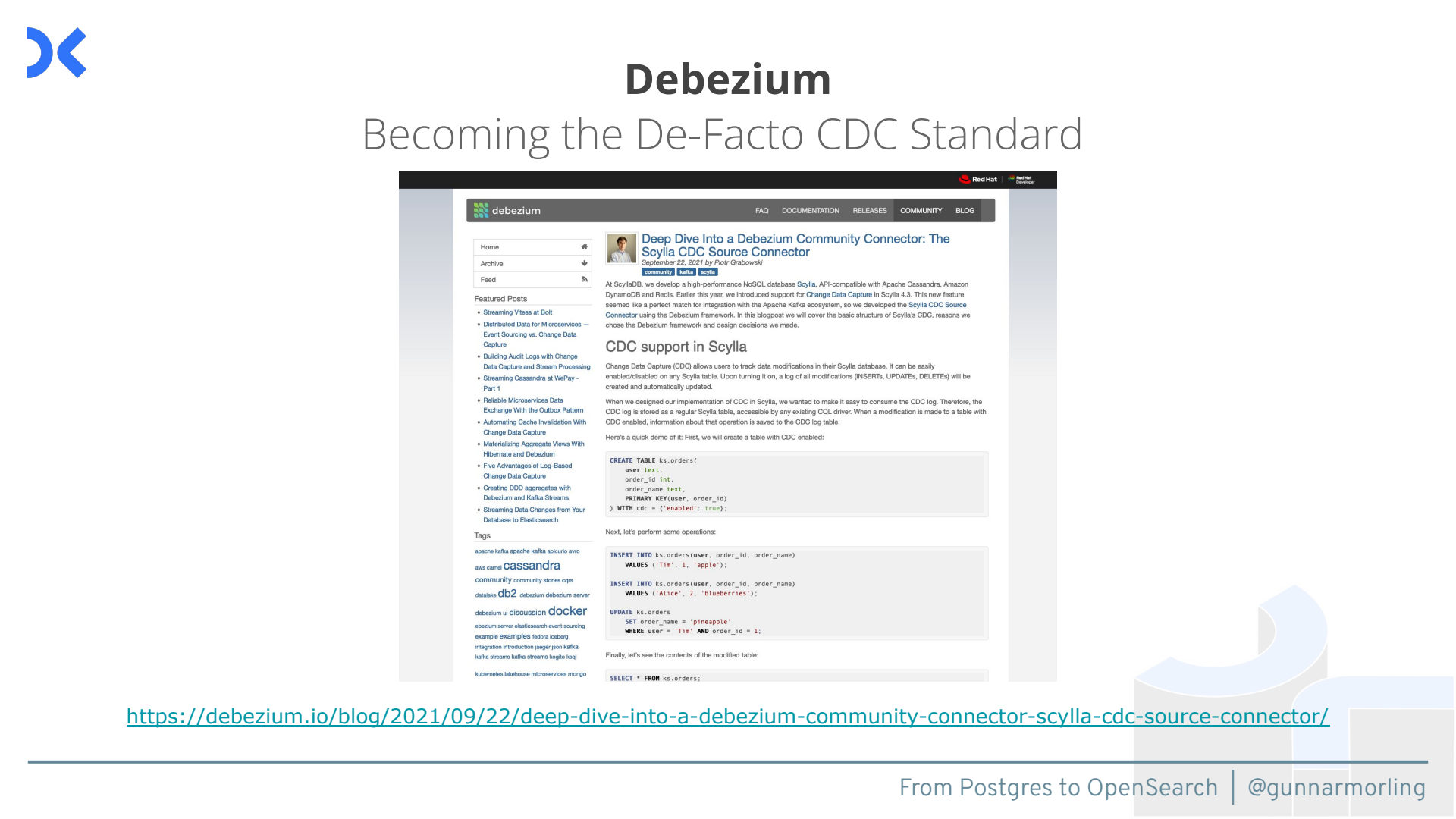
```
{
    "before": {
        "id": 1,
        "first_name": "Anne",
        "last_name": "Kretchmar",
        "email": "annek@noanswer.org"
    },
    "after": {
        "id": 1,
        "first_name": "Anne",
        "last_name": "Grant",
        "email": "annek@noanswer.org"
    },
    ...
}
```

Debezium: Data Change Events

- **Old** and **new row state**
- **Metadata** on table, TX id, etc.
- **Operation type**, timestamp

Debezium: Data Change Events

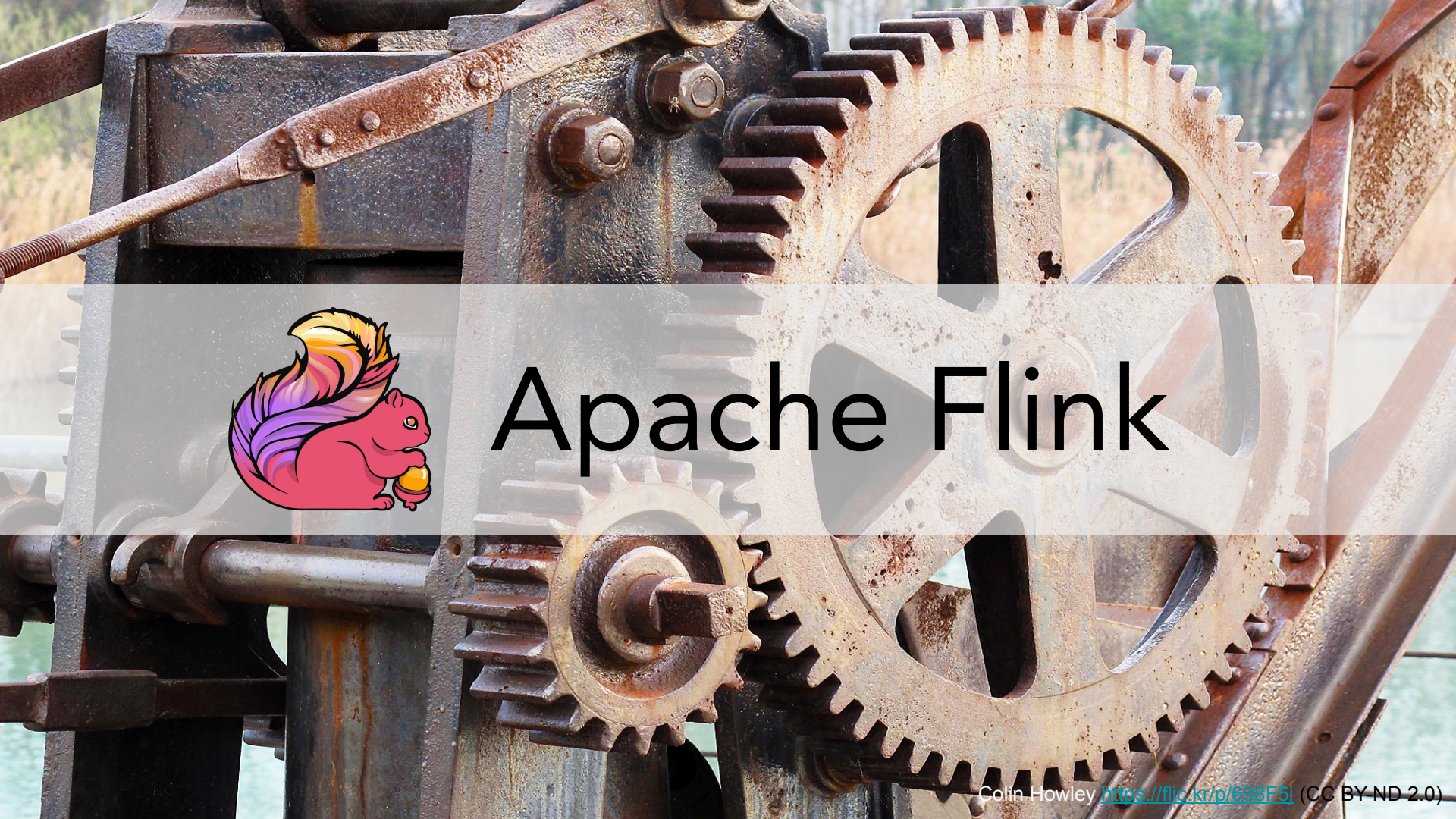
- **Old** and **new row state**
- **Metadata** on table, TX id, etc.
- **Operation type**, timestamp



Debezium

Becoming the De-Facto CDC Standard

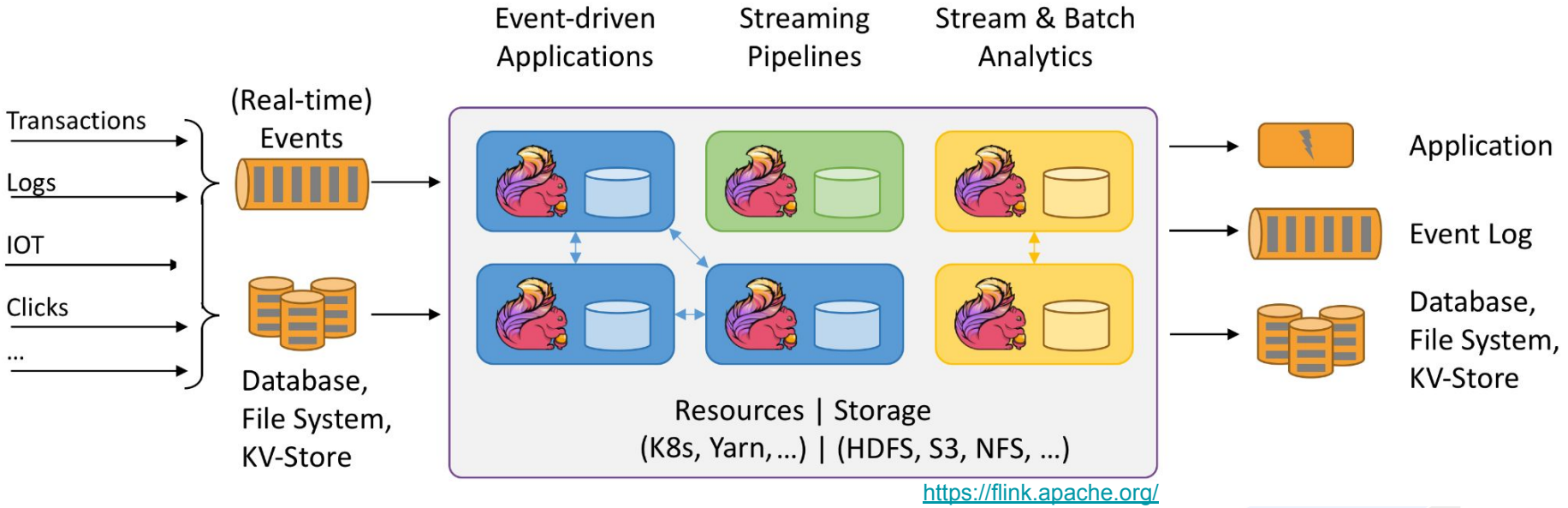
https://debezium.io/blog/2021/09/22/deep-dive-into-a-debezium-community-connector-scylla-cdc-source-connector/



Apache Flink

Apache Flink

Stateful Computations over Data Streams



Apache Flink

Common Use Cases

- Real-time **reporting/dashboards**
- Low-latency **alerting**, notifications
- **Materialized view** maintenance, caches
- Real-time **cross-database sync**, lookup joins, windowed joins, aggregations
- Machine learning: **model serving**, feature engineering
- **Change data capture**, data integration

https://flink.apache.org/poweredby.html

Apache Flink

APIs for Application Development

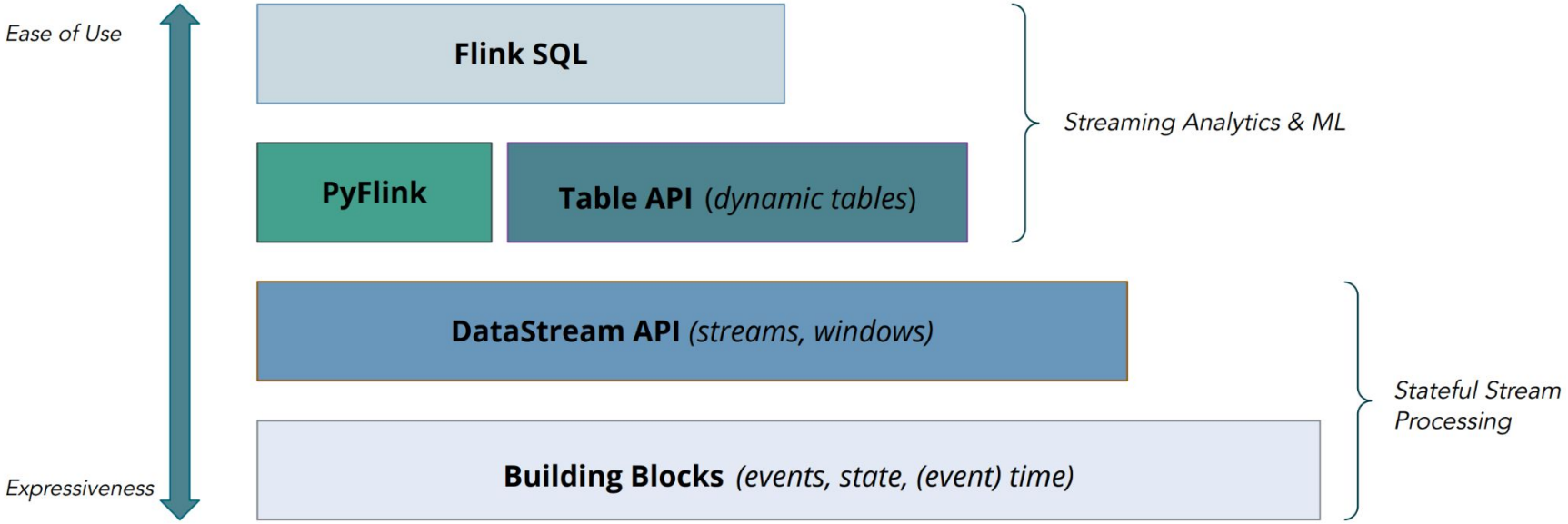


Image source: **"Change Data Capture with Flink SQL and Debezium"** by Marta Paes at DataEngBytes (https://noti.st/morsapaes/liQzgs/change-data-capture-with-flink-sql-and-debezium)

Apache Flink

APIs for Application Development

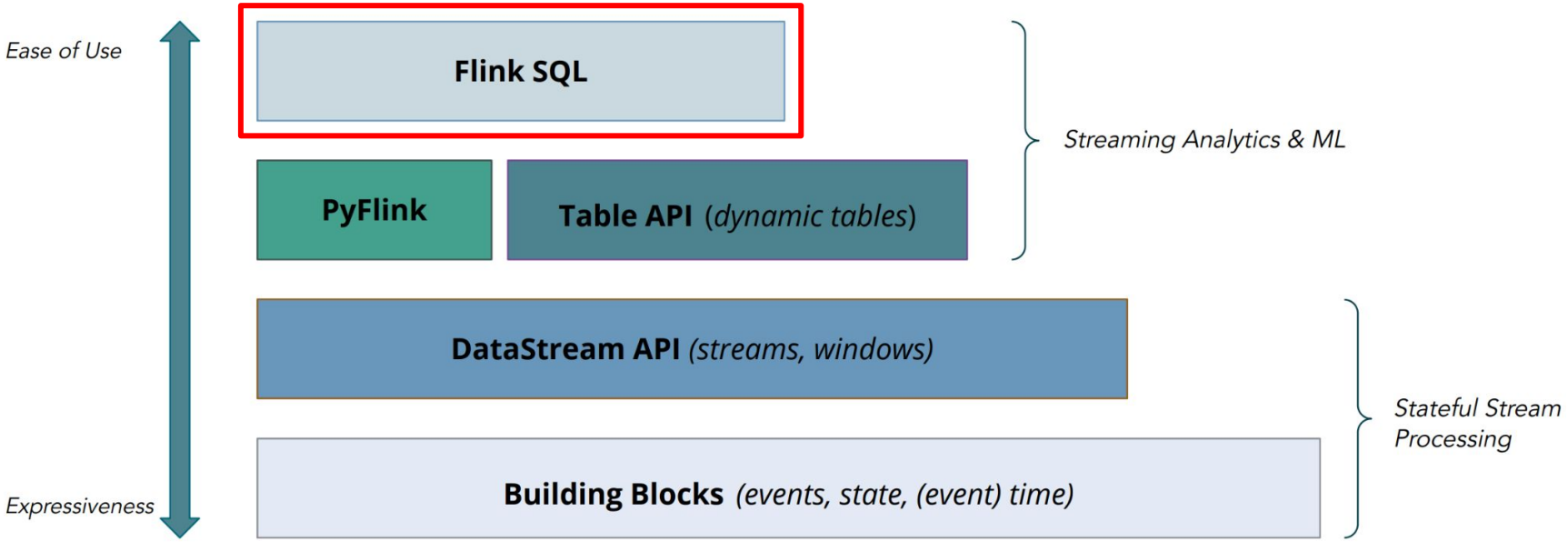
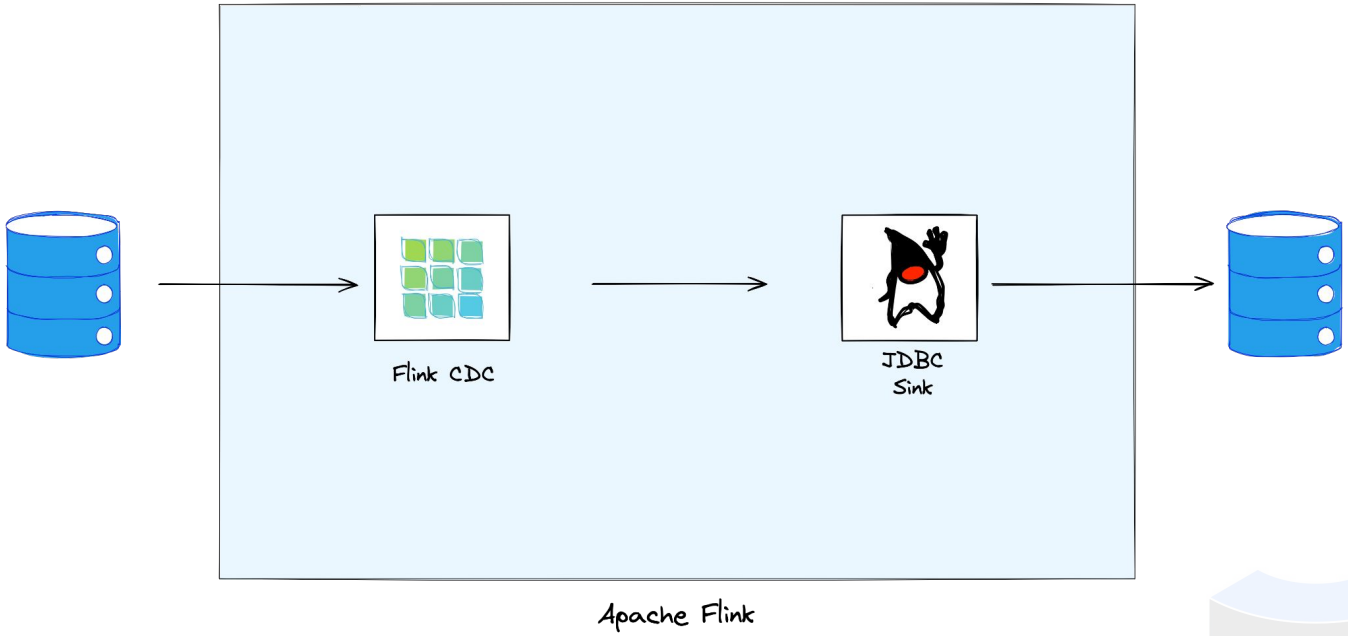


Image source: **"Change Data Capture with Flink SQL and Debezium"** by Marta Paes at DataEngBytes (https://noti.st/morsapaes/liQzgs/change-data-capture-with-flink-sql-and-debezium)

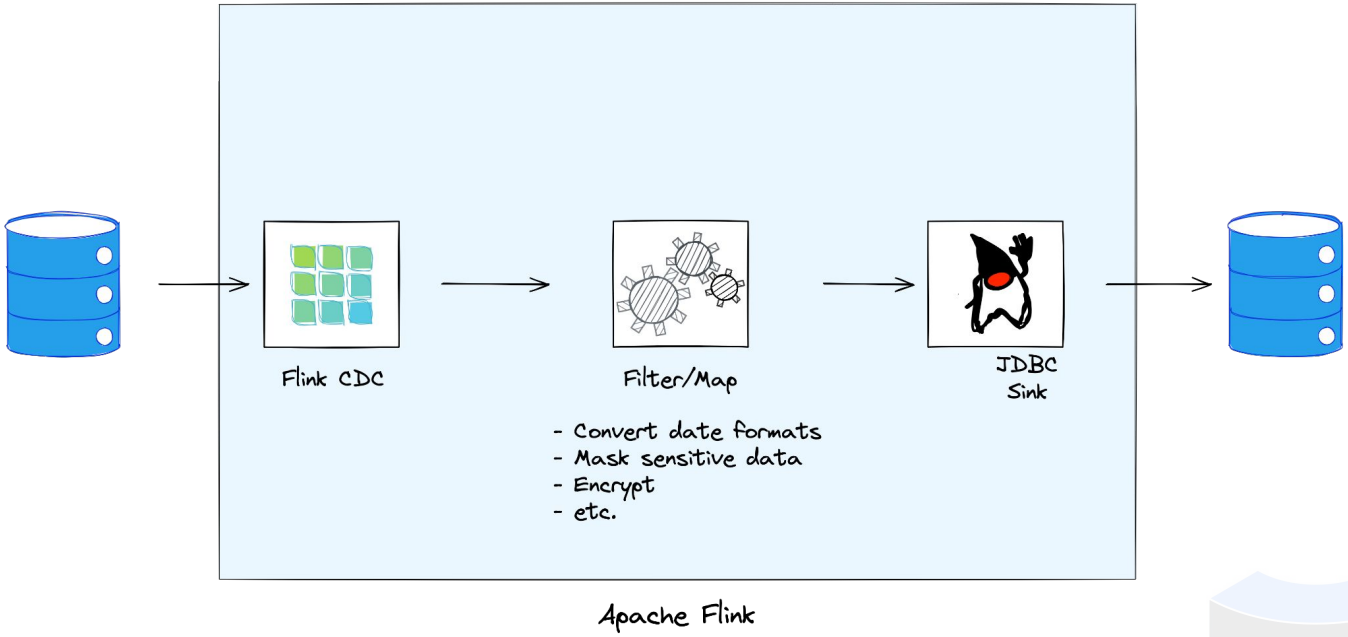
Apache Flink

Stream Processing of Change Data Events



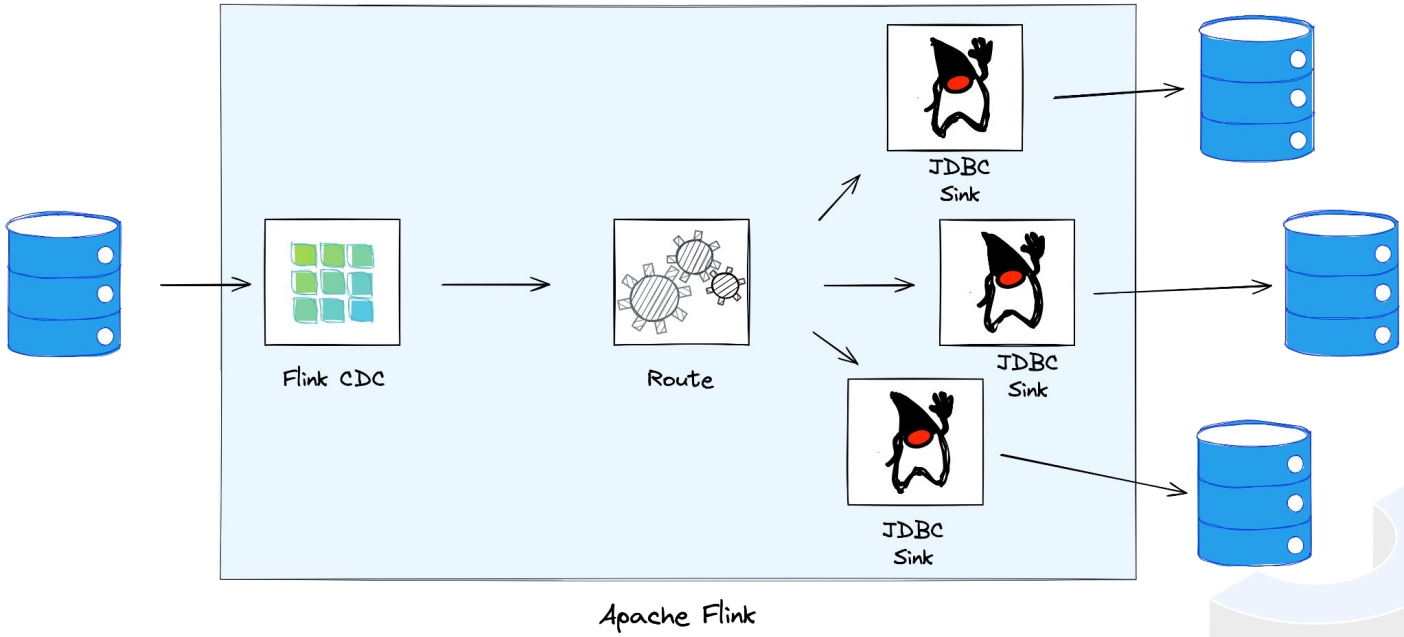
Apache Flink

Stream Processing of Change Data Events



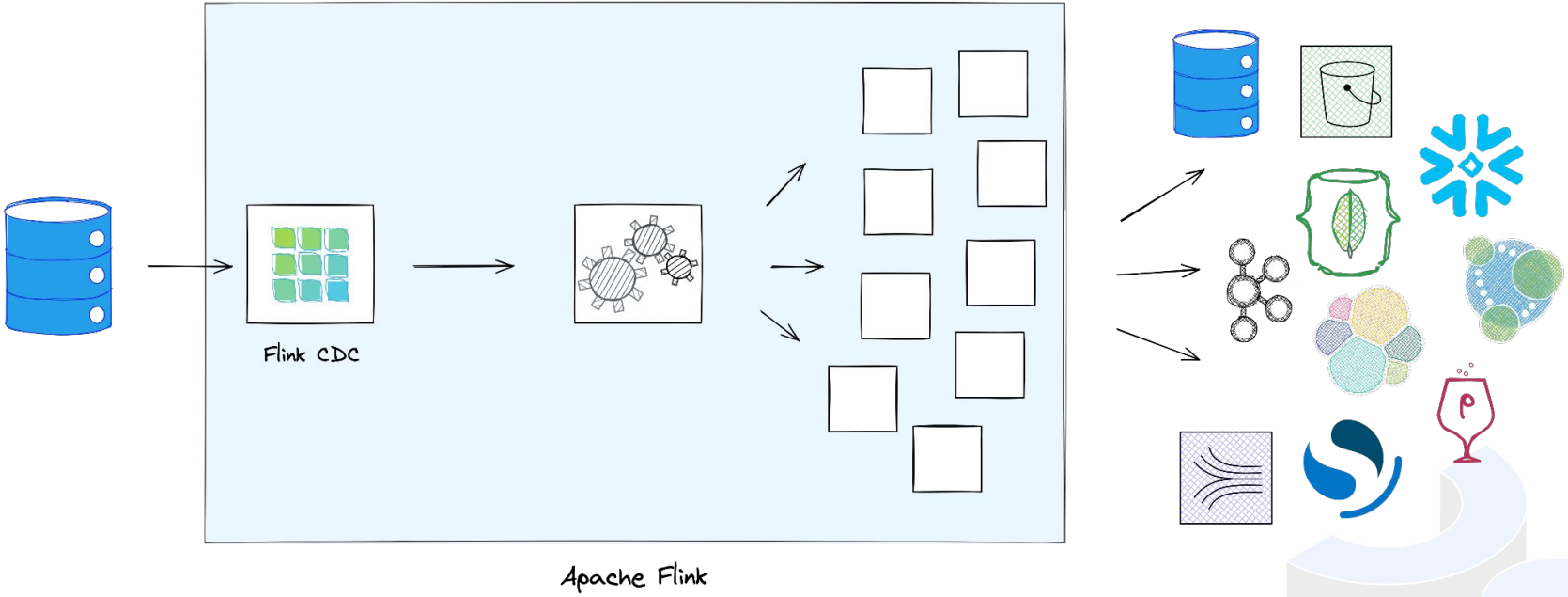
Apache Flink

Stream Processing of Change Data Events



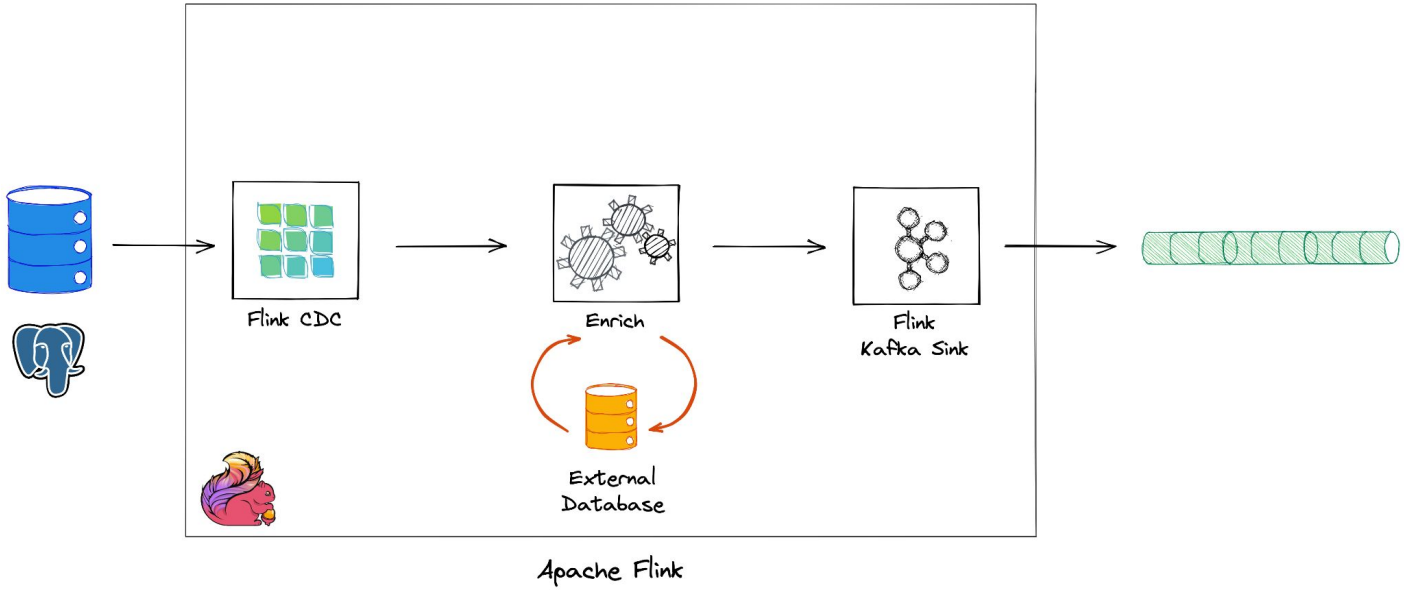
Apache Flink

Stream Processing of Change Data Events



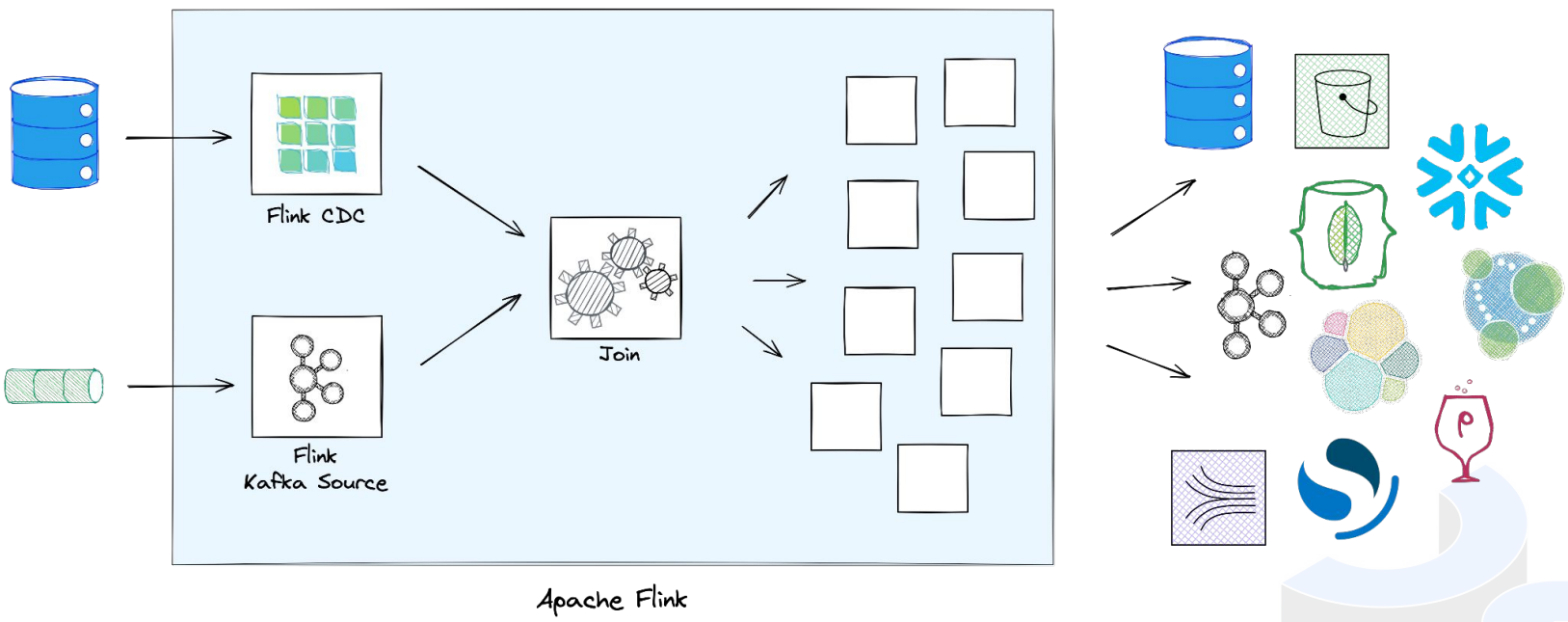
Apache Flink

Stream Processing of Change Data Events



Apache Flink

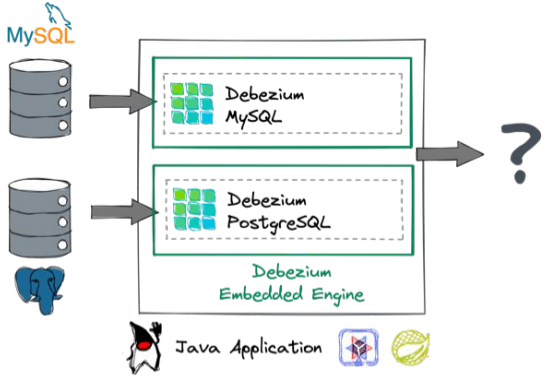
Stream Processing of Change Data Events



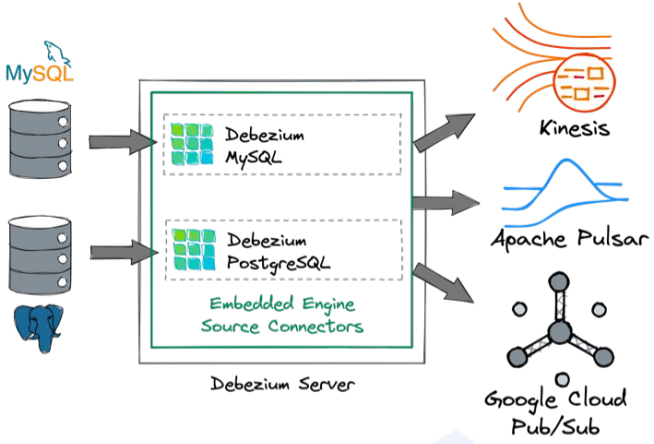
Debezium and Apache Flink

Integration Options

Kafka Connect



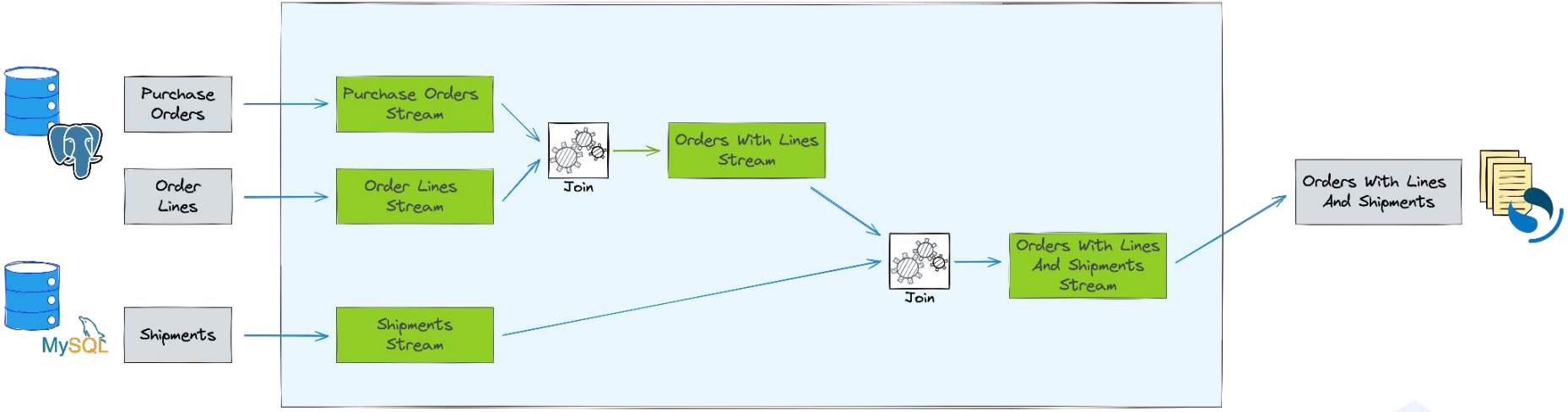
Debezium Embedded Engine



Debezium Server

Driving Full-Text Search

Propagating Joined Data to OpenSearch



Nested Data Structures

UDFs to the Rescue

Purchase_Orders

id	order_date	purchaser_id
10009	2023-02-19	1001
10010	2023-01-17	1003
...	...	...

Orders_Lines

id	order_id	product_id	quantity	price
100040	10009	102	1	39.99
100041	10009	105	2	129.99
100042	10009	107	3	49.49
...	...	...	...	...

```
INSERT INTO orders_with_lines
  SELECT
    po.id,
    po.order_date,
    po.purchaser_id,
    ( SELECT ARRAY_AGG(R (
          ROW(ol.id, ol.product_id, ol.quantity, ol.price))
      FROM order_lines ol
      WHERE ol.order_id = po.id )
  FROM
    purchase_orders po;
```

```
{
  "order_id": 10009,
  "order_date": "2023-01-16",
  "purchaser_id": 1001,
  "lines": [
    {
      "id": 100040,
      "product_id": 102,
      "quantity": 1,
      "price": 39.99
    },
    {
      "id": 100041,
      "product_id": 105,
      "quantity": 2,
      "price": 129.99
    },
    {
      "id": 100042,
      "product_id": 107,
      "quantity": 3,
      "price": 49.49
    }
  ]
}
```

Postgres

Flink

OpenSearch

Nested Data Structures

UDFs to the Rescue

```
public class ArrayAggr <T> extends AggregateFunction<T[], ArrayAccumulator<T>> {

  @Override
  public ArrayAccumulator<T> createAccumulator() { return new ArrayAccumulator<T>(); }

  @Override
  public T[] getValue(ArrayAccumulator<T> acc) { return acc.toArray(); }

  public void accumulate(ArrayAccumulator<T> acc, T o) throws Exception {
    if (o != null) { acc.values.add(o); }
  }

  public void retract(ArrayAccumulator<T> acc, T o) throws Exception {
    if (o != null) { acc.values.remove(o); }
  }

  @Override
  public TypeInference getTypeInference(DataTypeFactory typeFactory) { ... }
}
```

Nested Data Structures

UDFs to the Rescue

```
CREATE TABLE orders_with_lines (
  order_id INT,
  order_date DATE,
  purchaser_id INT,
  lines ARRAY<ROW<id INT, product_id INT, quantity INT, price DOUBLE>>,
  PRIMARY KEY (order_id) NOT ENFORCED
)
WITH (
    'connector' = 'upsert-kafka',
    'topic' = 'orders_with_lines',
    'properties.bootstrap.servers' = 'mykafka:29092',
    'key.format' = 'json', 'value.format' = 'json'
);
```

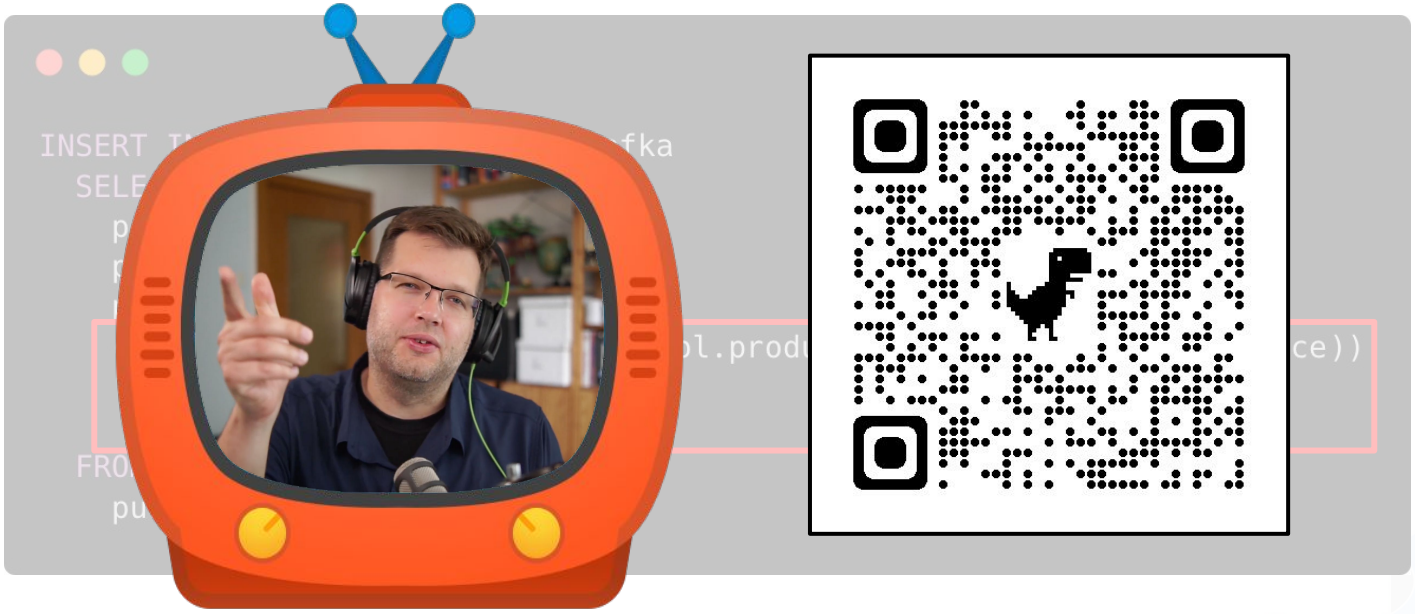
Nested Data Structures

UDFs to the Rescue

```
INSERT INTO orders_with_lines_kafka
  SELECT
    po.id,
    po.order_date,
    po.purchaser_id,
    ( SELECT ARRAY_AGGR(ROW(ol.id, ol.product_id, ol.quantity, ol.price))
      FROM order_lines ol
      WHERE ol.order_id = po.id )
  FROM
    purchase_orders po;
```

Nested Data Structures

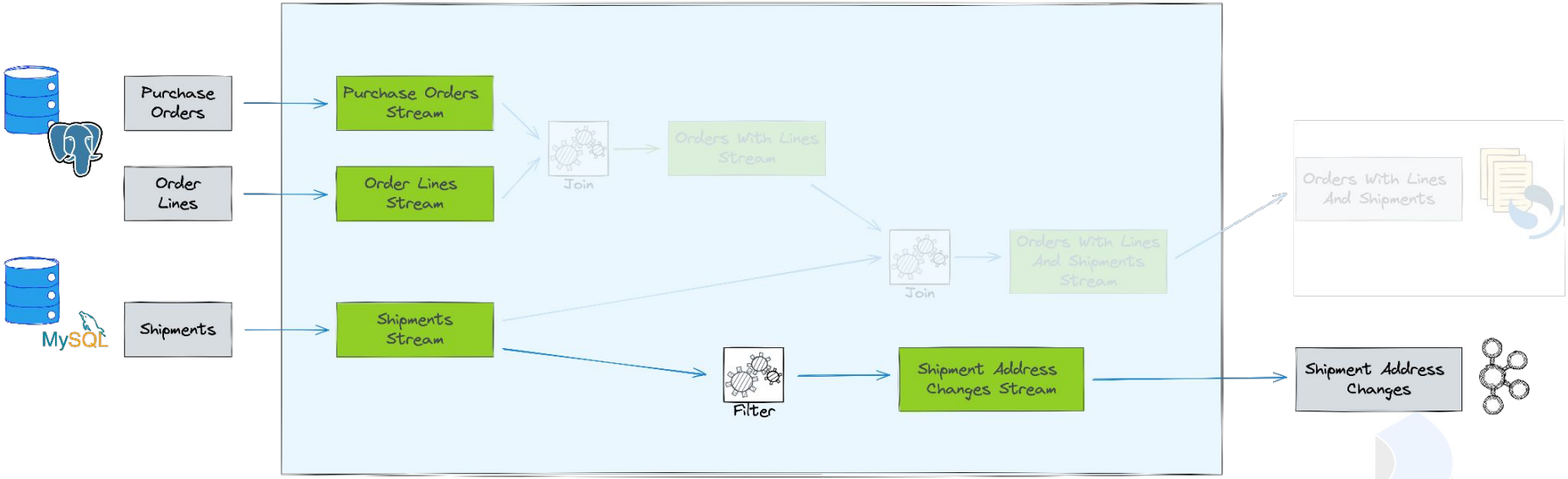
UDFs to the Rescue



https://www.youtube.com/@decodable

Ingest Once…

…Process Multiple Times



Transactional Aggregation

Correlating Events From Same Transaction

```
{
  "before": null,
  "after": {
    "pk": "2",
    "aa": "1"
  },
  "source": {
   ...
  },
  "op": "c",
  "ts_ms": "1580390884335",
  "transaction": {
    "id": "571:53195832",
    "total_order": "1",
    "data_collection_order": "1"
  }
}
```

```
{
  "status": "BEGIN",
  "id": "571:53195829",
  "ts_ms": 1486500577125
}

{
  "status": "END",
  "id": "571:53195832",
  "ts_ms": 1486500577691,
  "event_count": 2,
  "data_collections": [
    {
      "data_collection": "s1.a",
      "event_count": 1
    },
    {
      "data_collection": "s2.a",
      "event_count": 1
    }
  ]
}
```


Wrap-Up

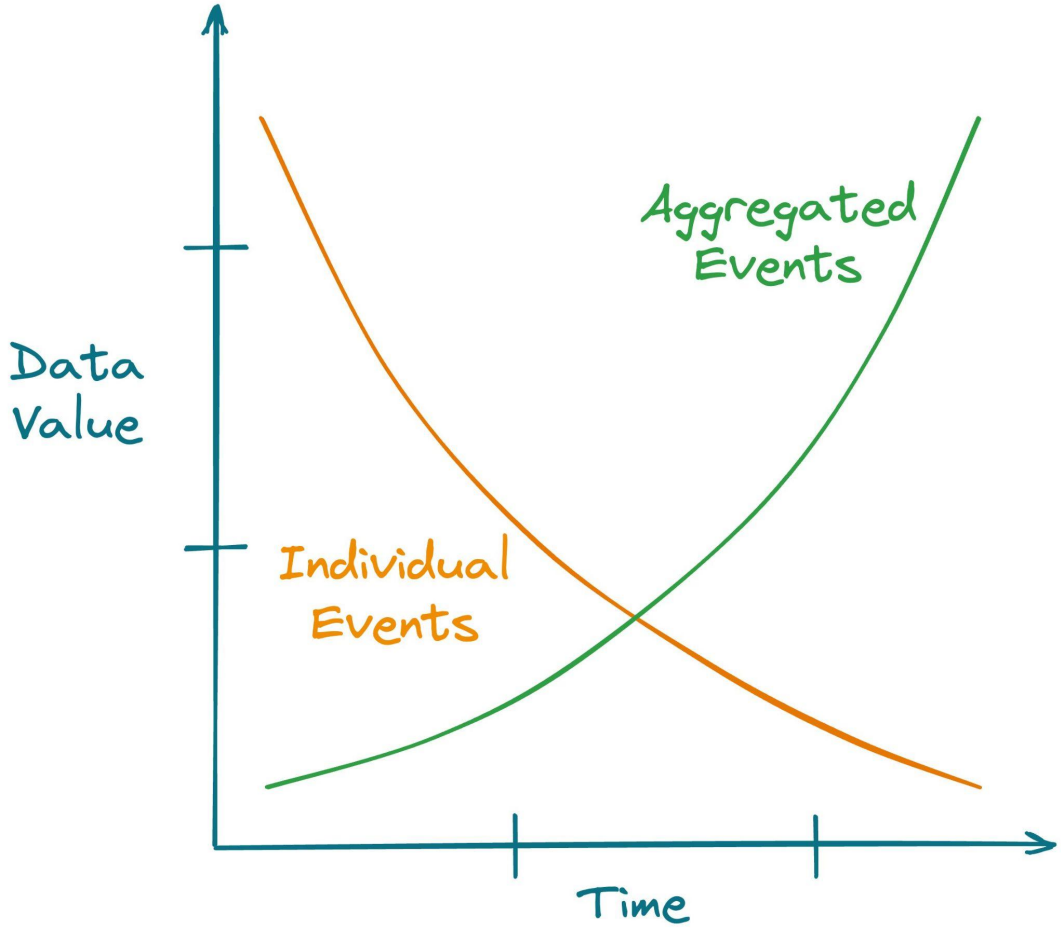
Take Aways

- **Debezium:** Real-time change event streams for your data
- **Debezium and Apache Flink**: Power house of change stream processing
  - Data Integration
  - Data Cleansing
  - Denormalization
  - Aggregations
  - Pattern Matching

Towards Production

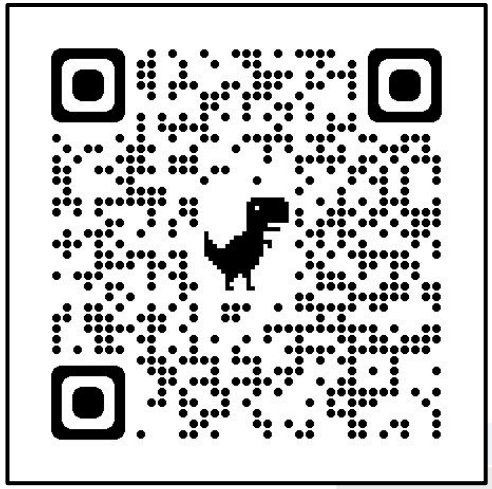
What To Consider

- Provisioning and updating **infrastructure**
- **Deployment** and (auto-)scaling
- **Observability**
- **State management**
- **Schema management** and inference
- **Developer experience**
- **CI/CD**
- **Security** and access control


Data Value
Aggregated Events
Individual Events
Time

Learn More

- **Debezium:** @debezium | https://debezium.io/
- **Apache Flink:** @ApacheFlink | https://flink.apache.org/
- **Getting started** with Flink**:**
  github.com/decodableco/examples
  → flink-learn



Thank You!

Q & A

gunnar@decodable.co

@gunnarmorling


decodable