



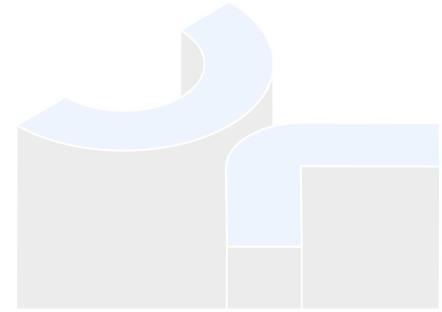
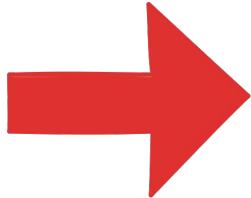
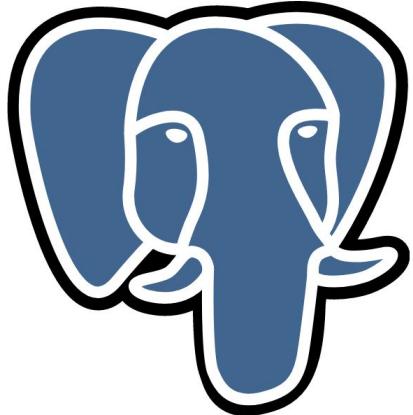
# From Postgres To OpenSearch In No Time

Gunnar Morling

Software Engineer, Decodable

 @gunnarmorling







# Today's Mission

Learn About...

Change Data  
Capture



Stream  
Processing



Demo





# Gunnar Morling

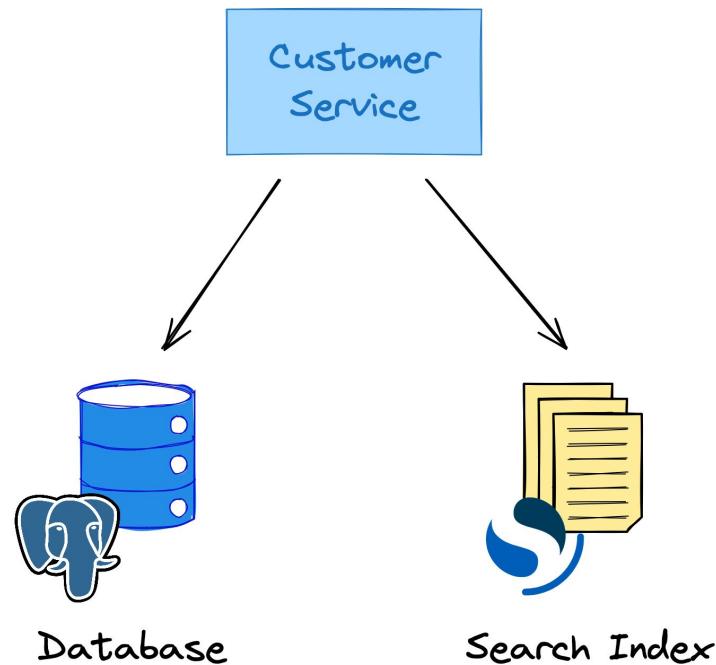


- Software engineer at **Decodable**
- Former project lead of **Debezium**
- **kctl** 🐻, JfrUnit, ModiTect, MapStruct
- Spec Lead for Bean Validation 2.0
- Java Champion



# Updating the Search Index

One Idea?



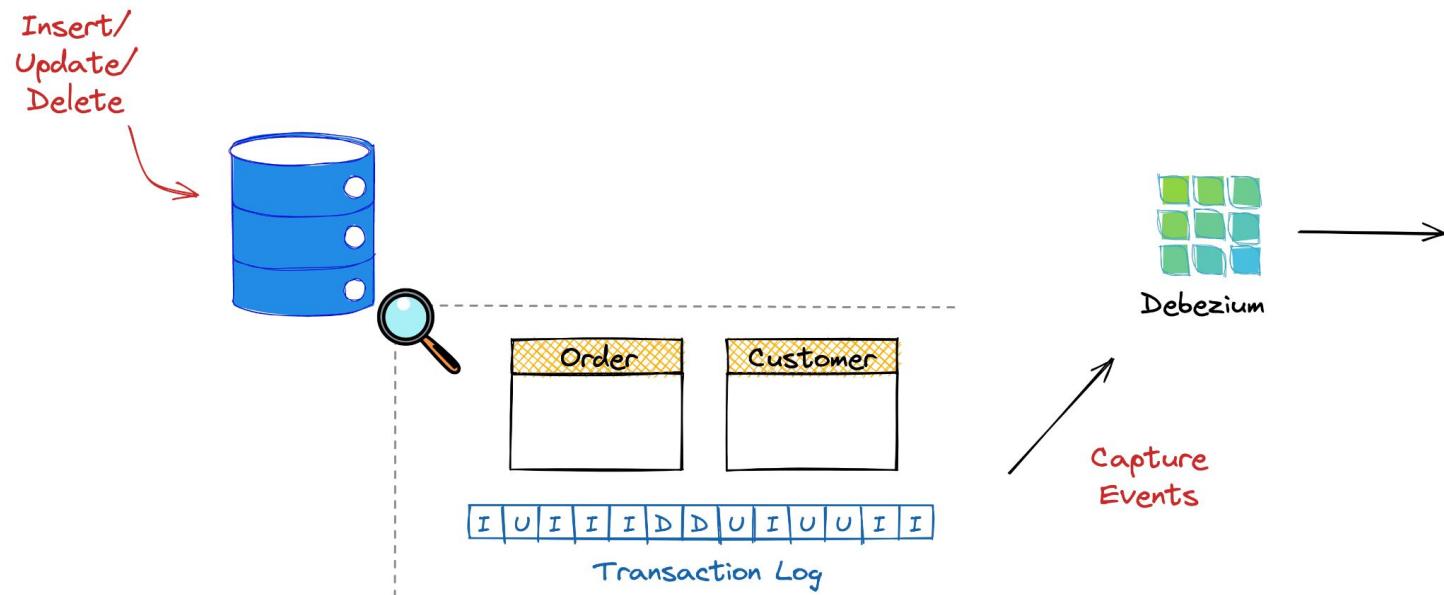


# debezium



# 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

The screenshot shows the official Debezium website. At the top, there's a navigation bar with links to FAQ, DOCUMENTATION, RELEASES, COMMUNITY, and BLOG. The Red Hat and Debezium logos are also present. The main header is "Debezium" with the subtitle "Stream changes from your database." Below this, a brief description explains Debezium as an open-source platform for change data capture. A prominent green button says "Latest stable (2.3) →". To its right, an orange button says "Development (2.4) →". A "Try our tutorial" button is located below the description. The page then splits into two main sections: "Do more with your data" (featuring a signal icon) and "Simplify your apps" (featuring a server icon). Both sections contain detailed descriptions of Debezium's capabilities.



# Change Data Capture

## Liberation for Your Data



Gunnar Morling

@gunnarmorling

Change data capture is one giant enabler; it lets you

- \* replicate data
- \* feed search indexes
- \* update caches
- \* run streaming queries
- \* sync data between microservices
- \* maintain denormalized views
- \* create audit logs and so much more.

Ultimately, it's liberation for your data.

1:46 PM · Apr 30, 2019

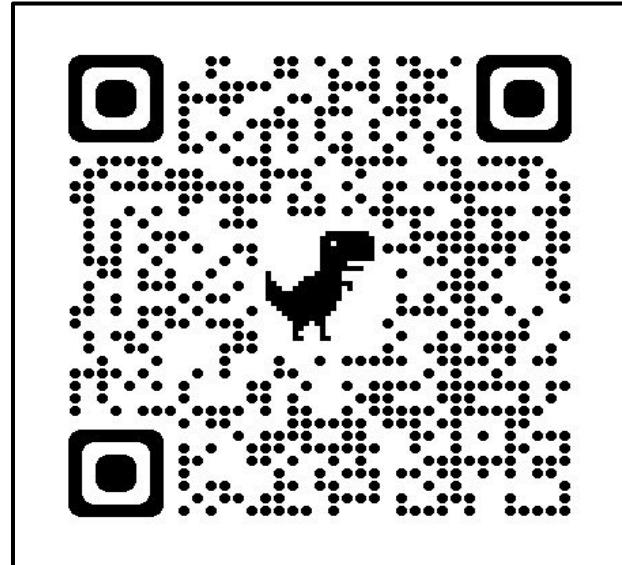
[View post engagements](#)

4

46

88

10





# Change Data Capture

## Liberation for Your Data



Gunnar Morling

@gunnarmorling

Change data capture is one giant enabler; it lets you

- \* replicate data
- \* feed search indexes
- \* update caches
- \* run streaming queries
- \* sync data between microservices
- \* maintain denormalized views
- \* create audit logs and so much more.

Ultimately, it's liberation for your data.

1:46 PM · Apr 30, 2019

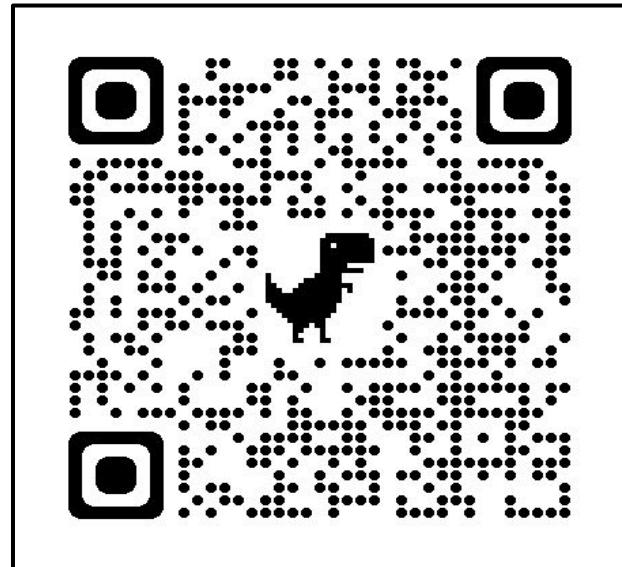
[View post engagements](#)

4

46

88

10





# 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

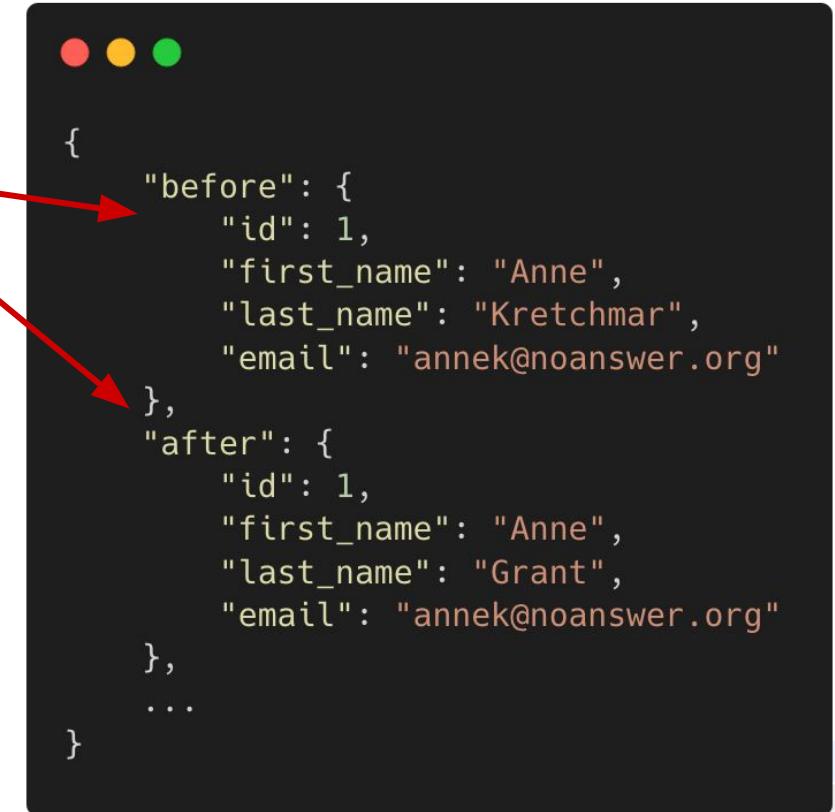
- **Old and new row state**
- **Metadata** on table, TX id, etc.
- **Operation type**, timestamp

```
{  
    "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



```
{  
  "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



```
{  
  ...  
  "source": {  
    "version": "1.6.0.Alpha1",  
    "connector": "postgresql",  
    "name": "PostgreSQL_server",  
    "ts_ms": 1559033904863,  
    "snapshot": true,  
    "db": "postgres",  
    "schema": "public",  
    "table": "customers",  
    "txId": 555,  
    "lsn": 24023128  
  },  
  "op": "u",  
  "ts_ms": 1559033904863  
}
```



# Debezium

## Becoming the De-Facto CDC Standard

The screenshot shows a blog post on the Debezium website. The header includes the Debezium logo, navigation links for FAQ, DOCUMENTATION, RELEASES, COMMUNITY, and BLOG, and a Red Hat logo. The main content is titled "Deep Dive Into a Debezium Community Connector: The Scylla CDC Source Connector" by Piotr Grabowski, posted on September 22, 2021. The post discusses the development of a high-performance NoSQL database Scylla, API-compatible with Apache Cassandra, Amazon DynamoDB and Redis. It highlights the introduction of support for Change Data Capture in Scylla 4.3, which integrates with the Apache Kafka ecosystem. The post covers the basic structure of Scylla's CDC, design decisions, and how it fits into the Debezium framework. It also includes a section on "CDC support in Scylla" and provides a SQL query for creating a CDC log table.

Deep Dive Into a Debezium Community Connector: The Scylla CDC Source Connector

September 22, 2021 by Piotr Grabowski

community kafka scylla

At ScyllaDB, we develop a high-performance NoSQL database Scylla, API-compatible with Apache Cassandra, Amazon DynamoDB and Redis. Earlier this year, we introduced support for Change Data Capture in Scylla 4.3. This new feature seemed like a perfect match for integration with the Apache Kafka ecosystem, so we developed the Scylla CDC Source Connector using the Debezium framework. In this blogpost we will cover the basic structure of Scylla's CDC, reasons we chose the Debezium framework and design decisions we made.

### CDC support in Scylla

Change Data Capture (CDC) allows users to track data modifications in their Scylla database. It can be easily enabled/disabled on any Scylla table. Upon turning it on, a log of all modifications (INSERTs, UPDATEs, DELETEs) will be created and automatically updated.

When we designed our implementation of CDC in Scylla, we wanted to make it easy to consume the CDC log. Therefore, the CDC log is stored in a regular Scylla table, accessible by any existing CQL driver. When a modification is made to a table with CDC enabled, information about that operation is saved to the CDC log table.

Here's a quick demo of it: First, we will create a table with CDC enabled:

```
CREATE TABLE ks.orders(
    user text,
    order_id int,
    order_name text,
    PRIMARY KEY(user, order_id)
) WITH cdc = {'enabled': true};
```

Next, let's perform some operations:

```
INSERT INTO ks.orders(user, order_id, order_name)
VALUES ('Tim', 1, 'apple');

INSERT INTO ks.orders(user, order_id, order_name)
VALUES ('Alice', 2, 'blueberries');

UPDATE ks.orders
SET order_name = 'pineapple'
WHERE user = 'Tim' AND order_id = 1;
```

Finally, let's see the contents of the modified table:

```
SELECT * FROM ks.orders;
```

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

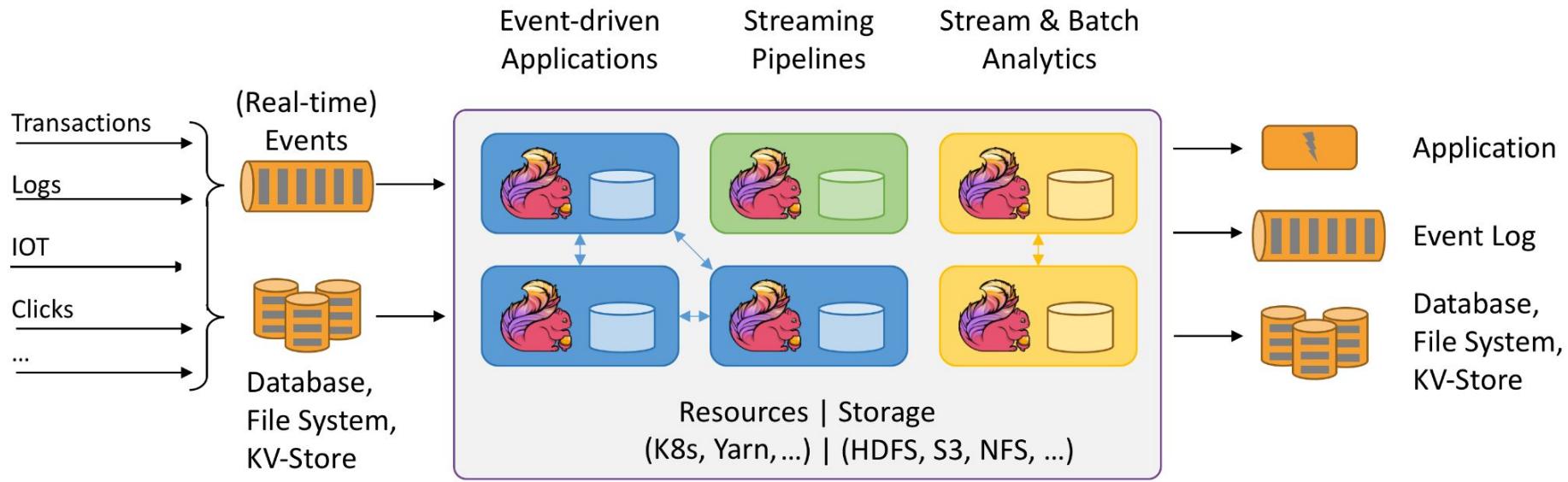


# Apache Flink



# 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/powerdby.html>





# Apache Flink

## APIs for Application Development

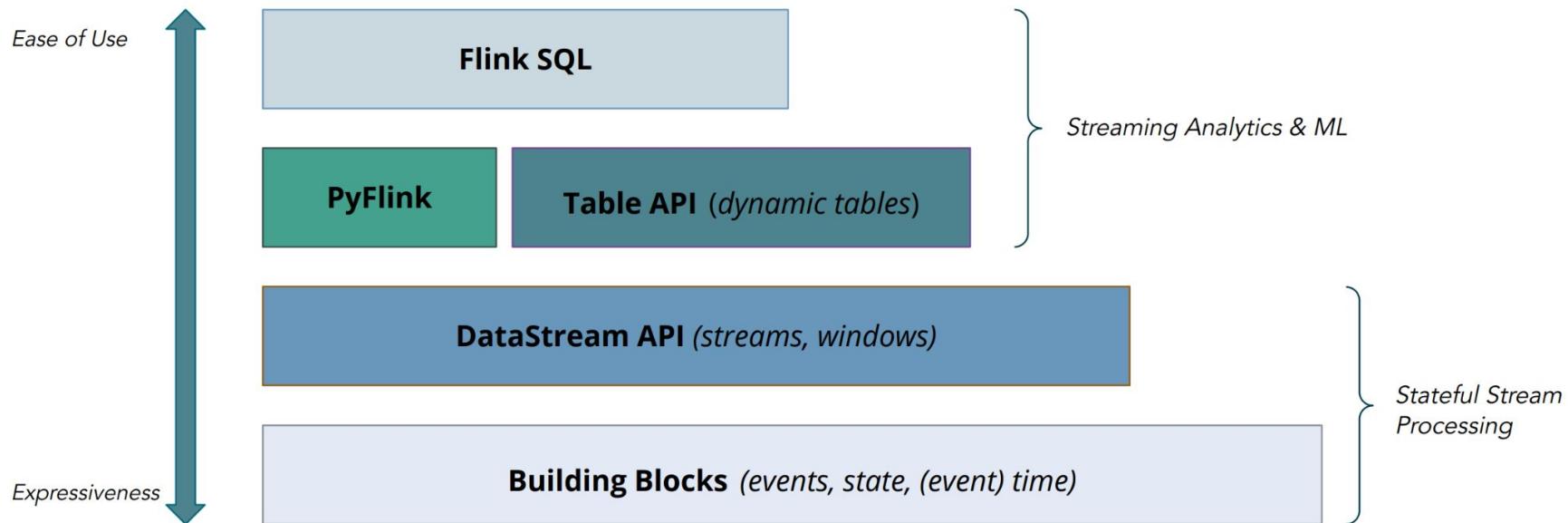


Image source: “[Change Data Capture with Flink SQL and Debezium](#)” by Marta Paes at [DataEngBytes](#) (<https://noti.st/morsapaes/liQzqs/change-data-capture-with-flink-sql-and-debezium>)



# Apache Flink

## APIs for Application Development

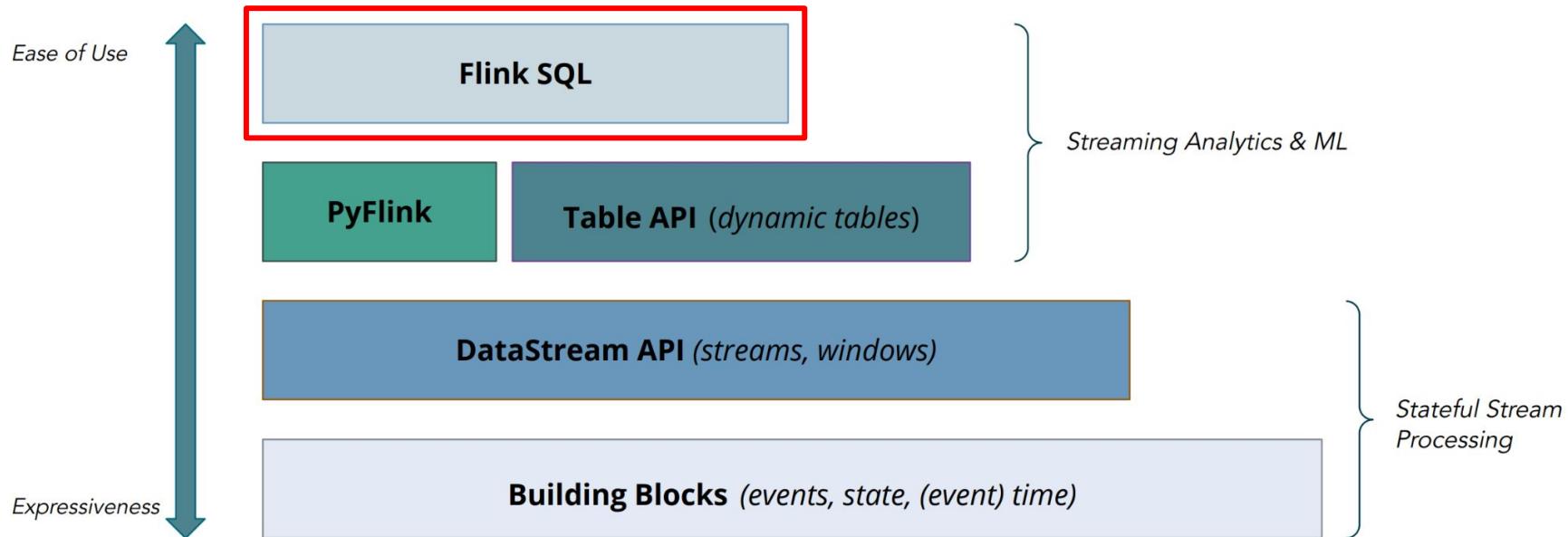
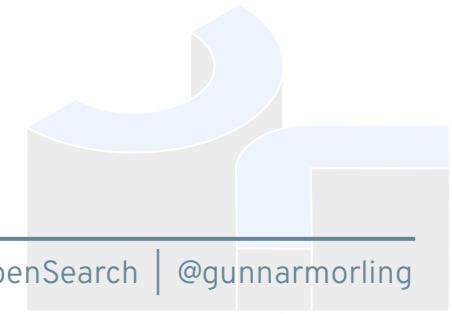
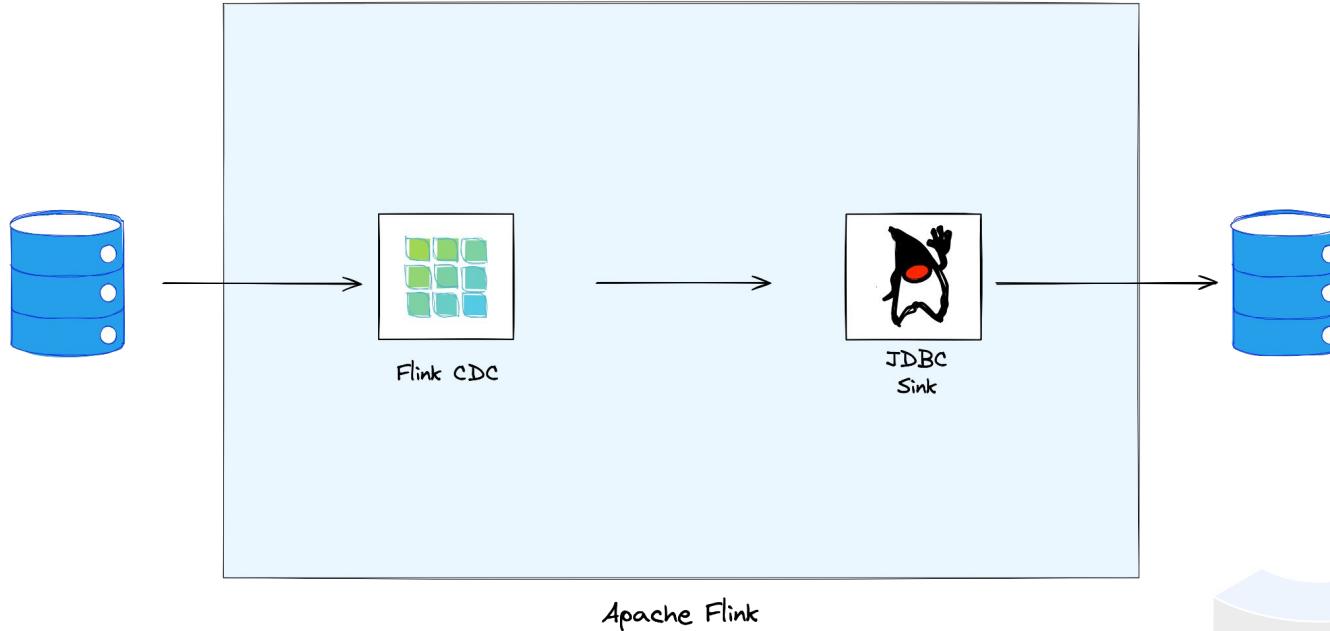


Image source: “[Change Data Capture with Flink SQL and Debezium](#)” by Marta Paes at [DataEngBytes](#) (<https://noti.st/morsapaes/liQzqs/change-data-capture-with-flink-sql-and-debezium>)



# Apache Flink

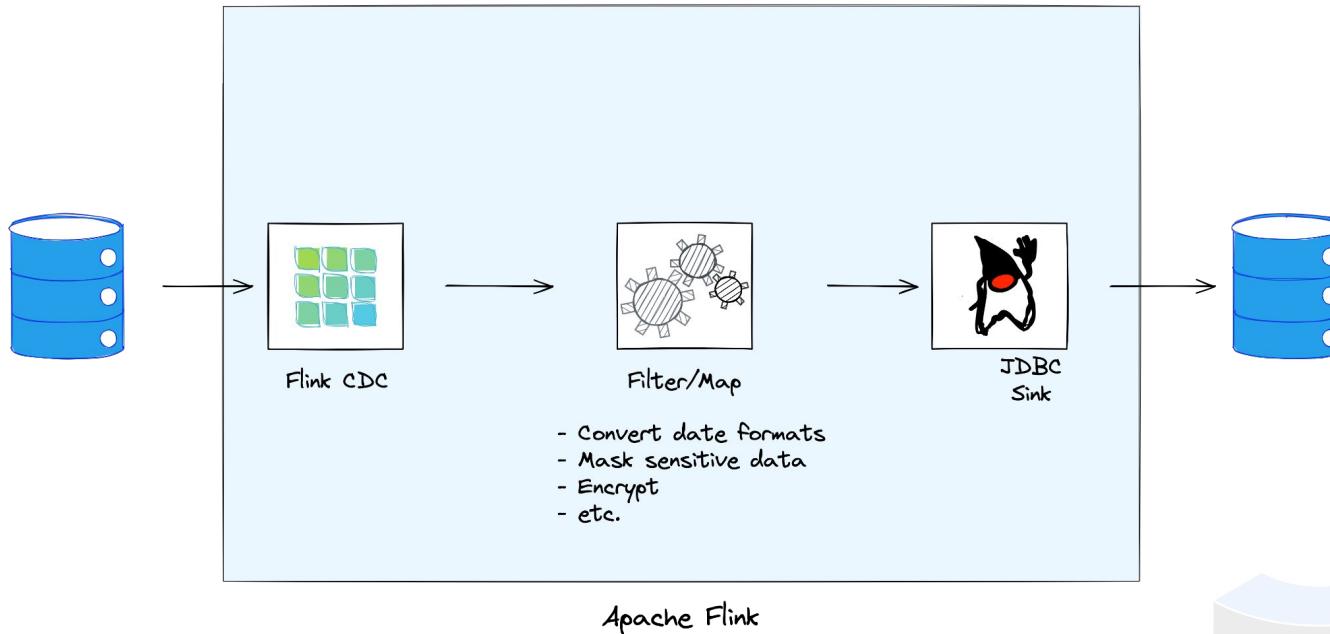
## Stream Processing of Change Data Events





# Apache Flink

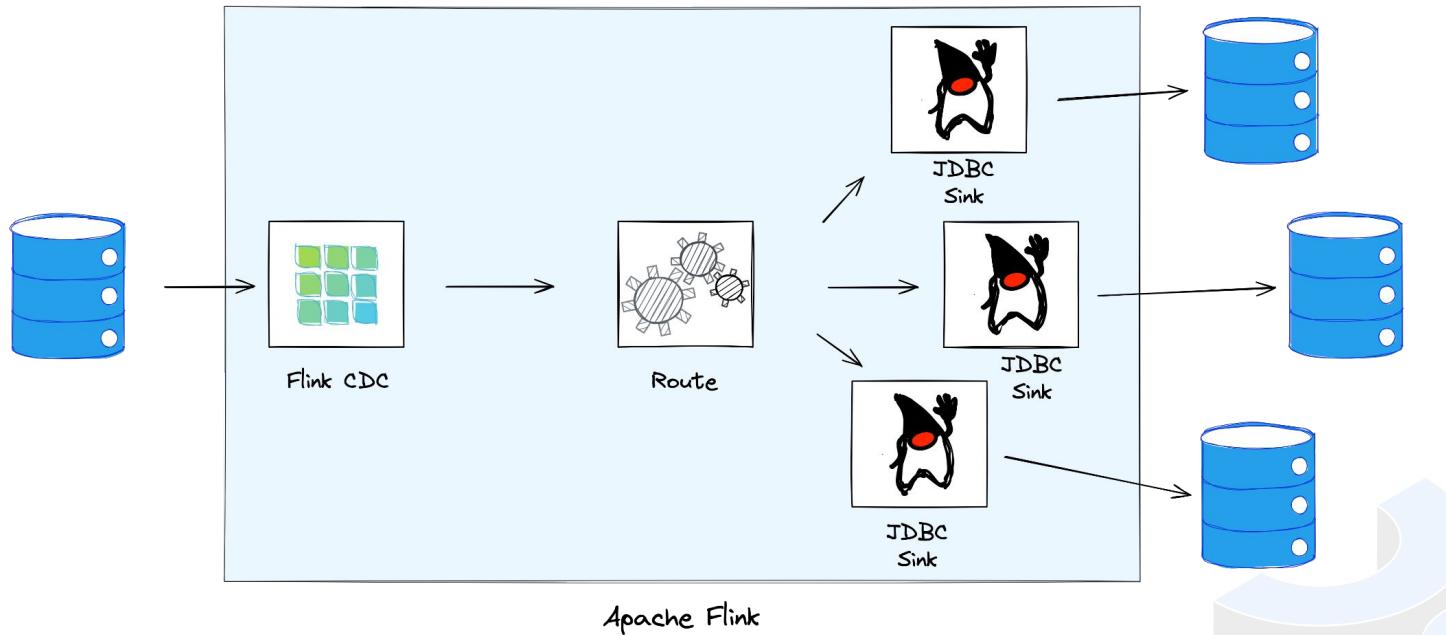
## Stream Processing of Change Data Events





# Apache Flink

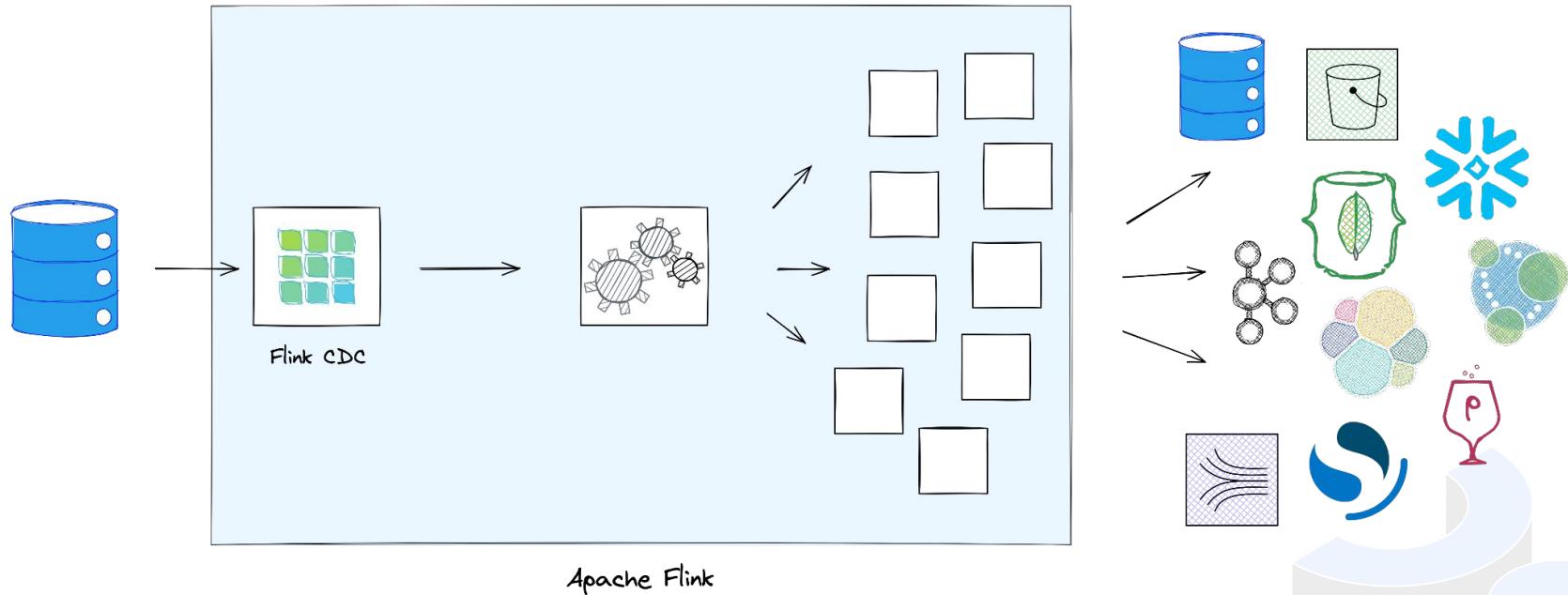
## Stream Processing of Change Data Events





# Apache Flink

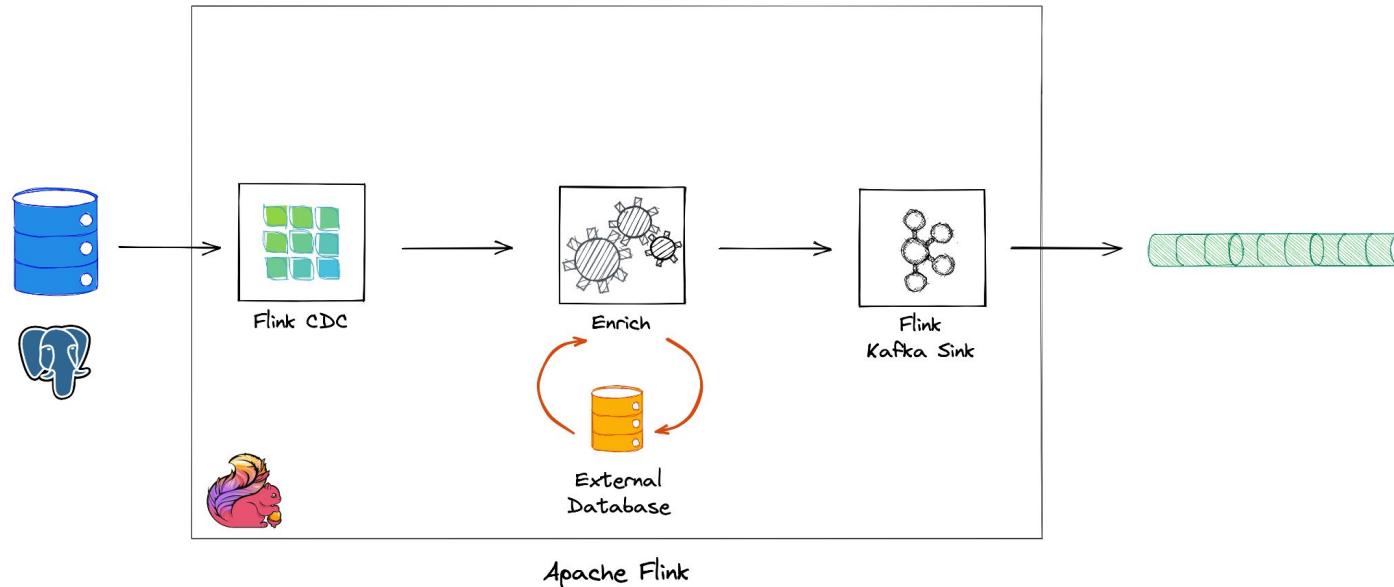
## Stream Processing of Change Data Events





# Apache Flink

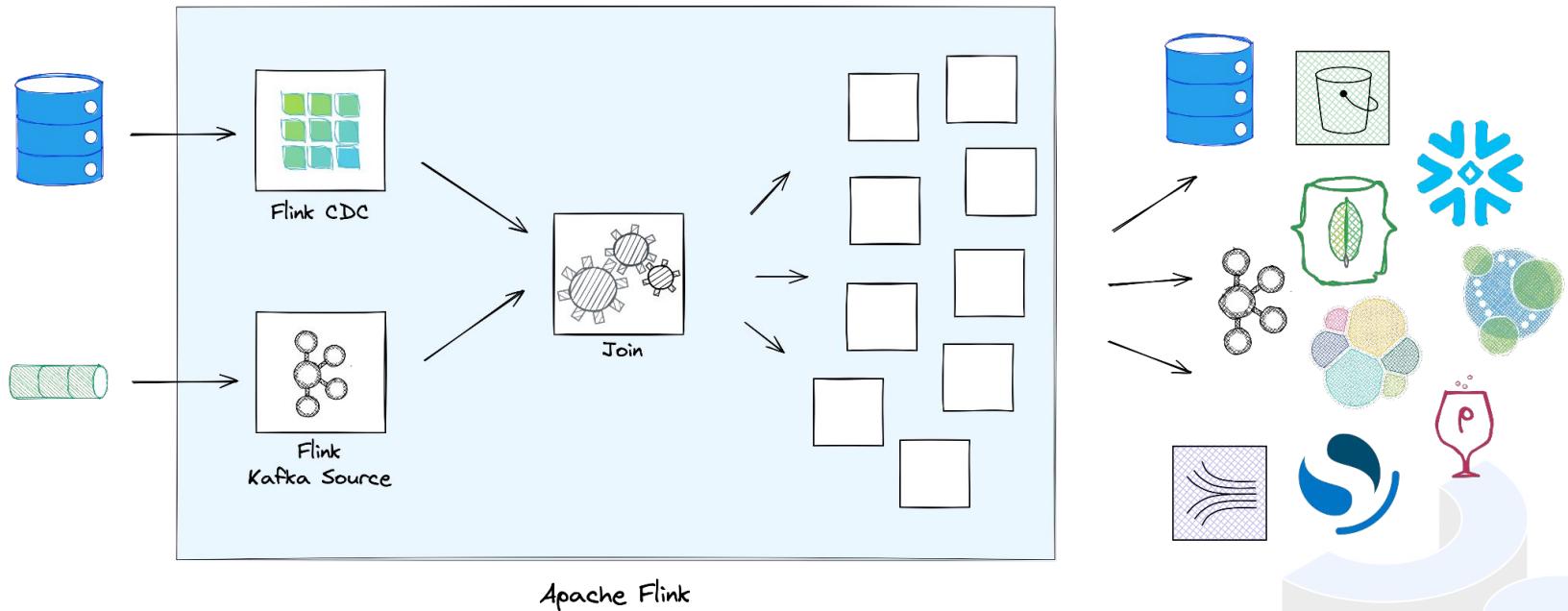
## Stream Processing of Change Data Events





# Apache Flink

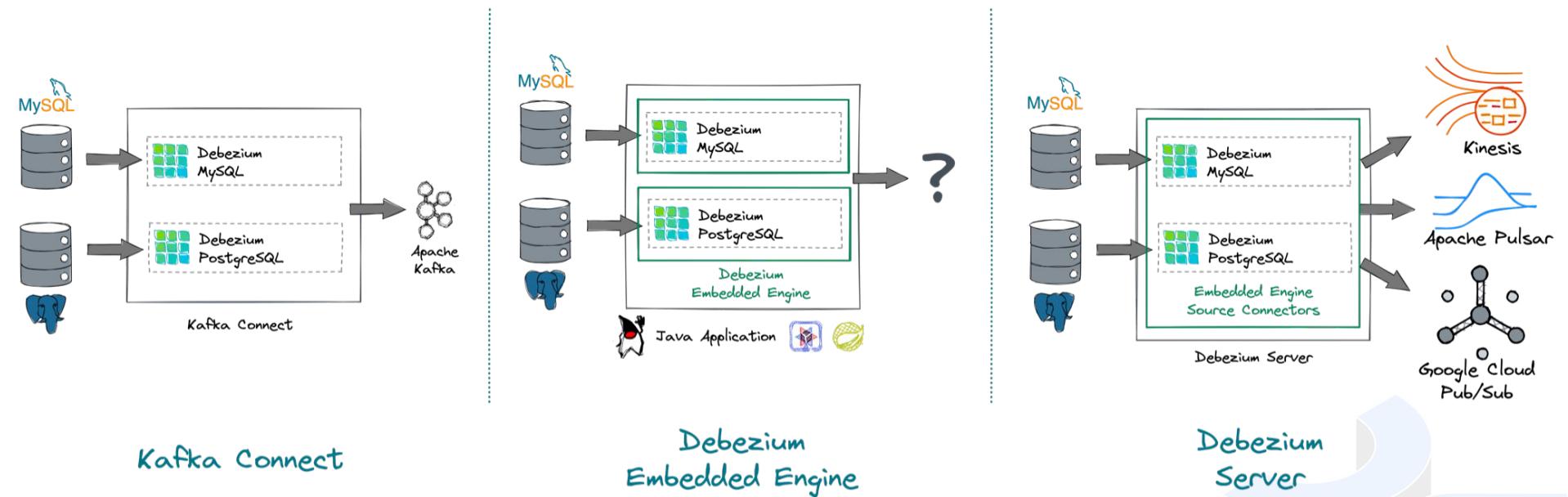
## Stream Processing of Change Data Events





# Debezium and Apache Flink

## Integration Options



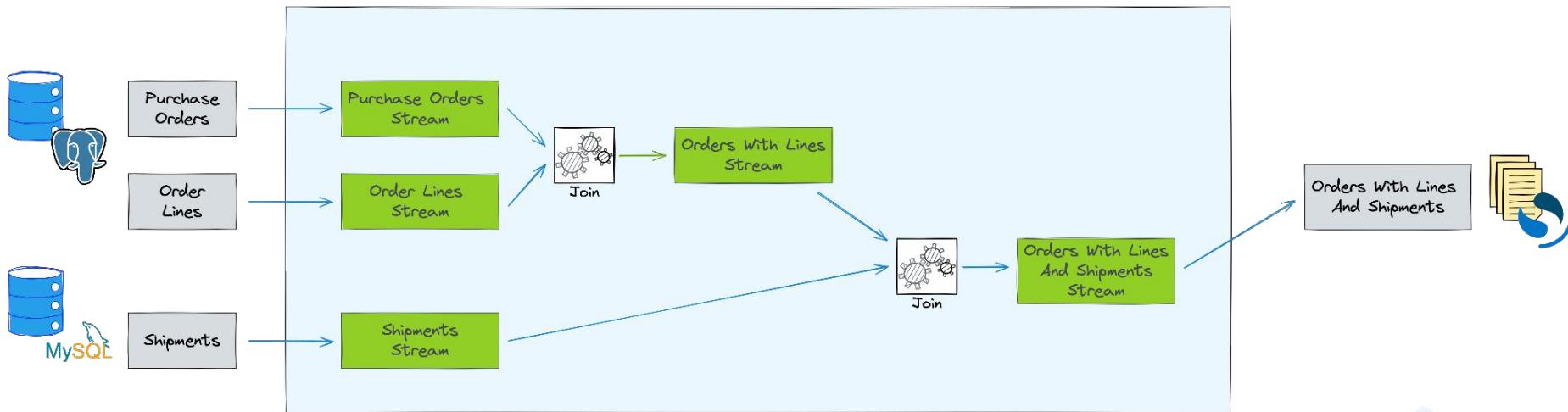


# Demo



# Driving Full-Text Search

## Propagating Joined Data to OpenSearch





# Demo



# 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(
        ROW(po.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) { ... }
}
```





# Demo



# 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

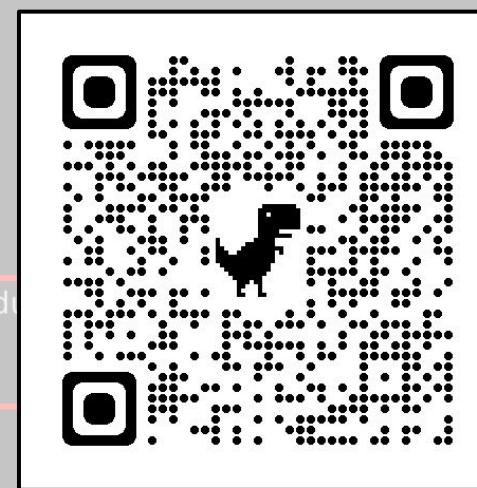
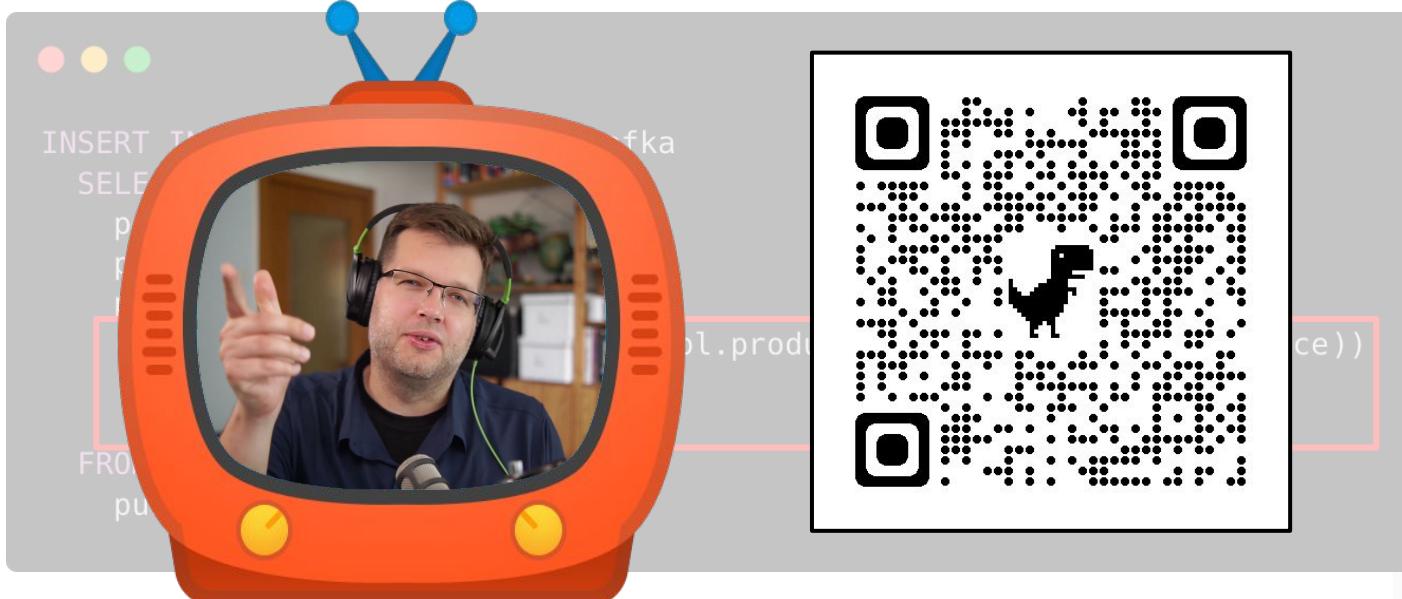


```
INSERT INTO orders_with_lines_kafka
SELECT
    po.id,
    po.order_date,
    po.purchaser_id,
    ( SELECT ARRAY_AGG(ROW(ol.id, ol.product_id, ol.quantity, ol.price))
        FROM order_lines ol
        WHERE ol.order_id = po.id )
FROM
    purchase_orders po;
```



# Nested Data Structures

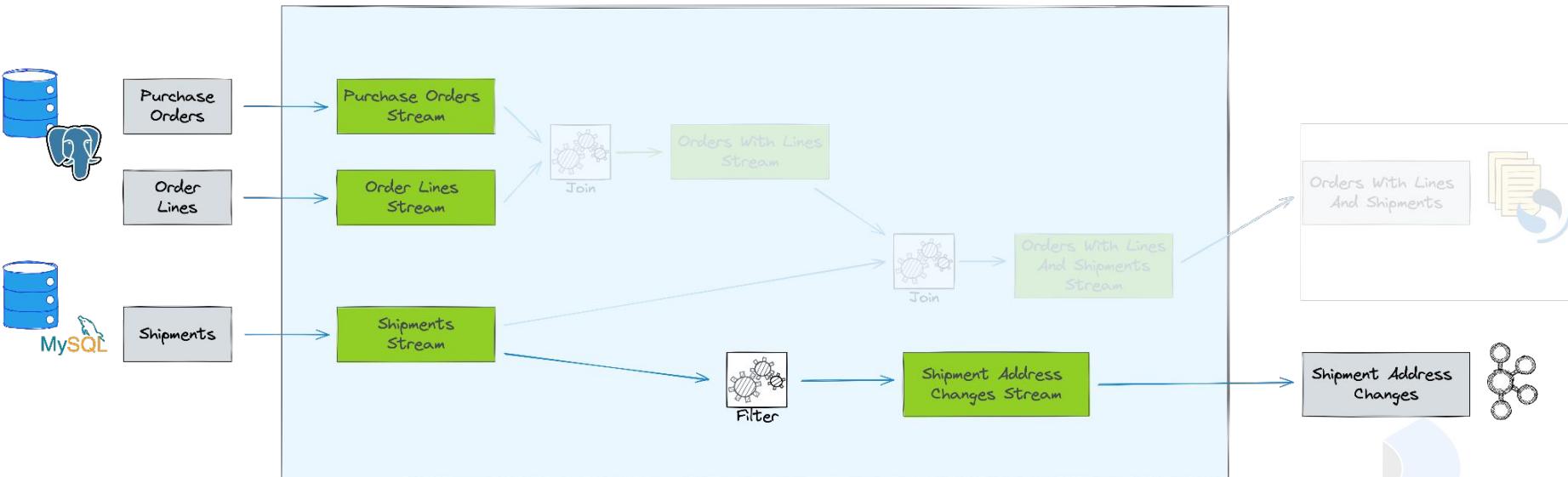
UDFs to the Rescue



<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  
        }  
    ]  
}
```

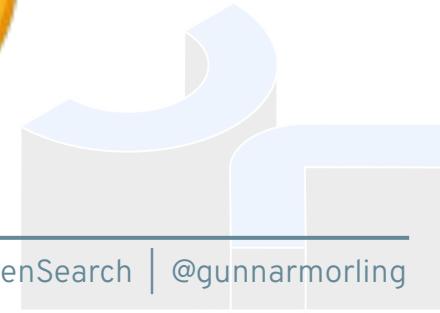
A photograph of a wooden pier extending into the ocean at sunset. The sky is filled with warm orange and yellow hues. In the background, a long bridge stretches across the horizon. The pier's wooden planks lead towards the horizon, and a metal railing runs along its edge.

# 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



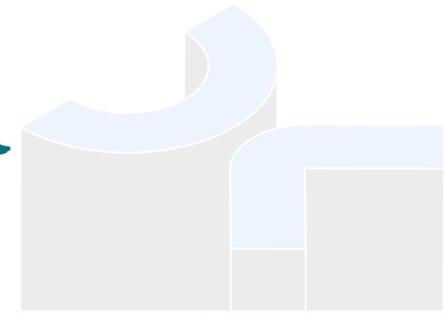
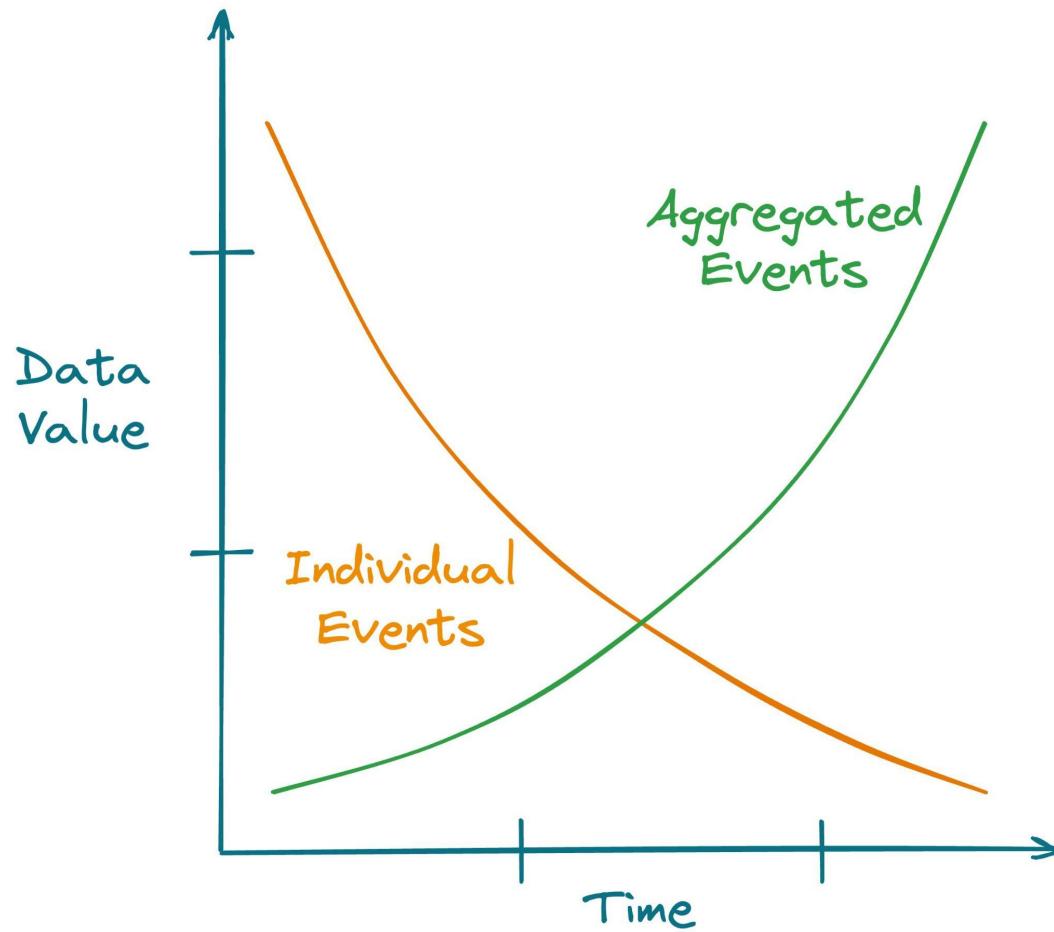


# Towards Production

## 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

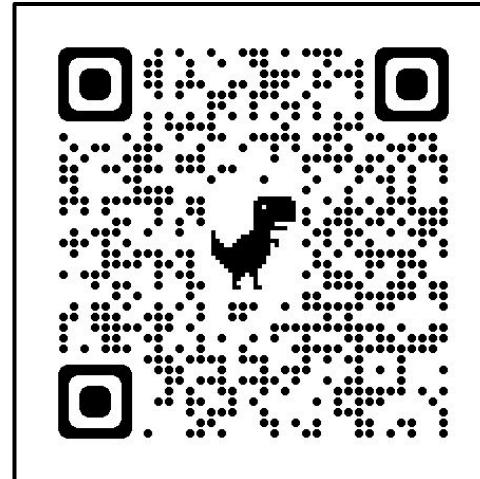






## Learn More

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





# Thank You!

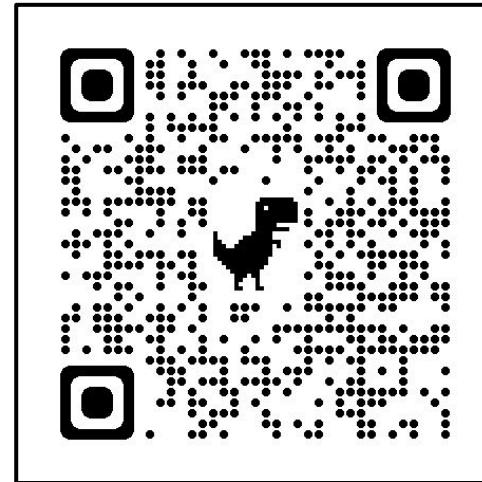
## Q & A

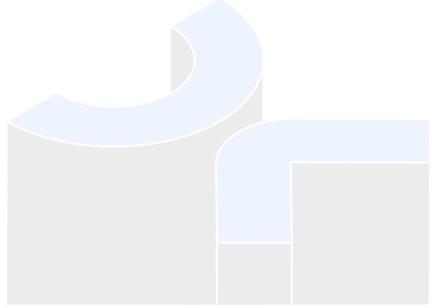


[gunnar@decodable.co](mailto:gunnar@decodable.co)



[@gunnar morling](https://twitter.com/gunnarmorling)





**decodable**