Data Lineage

Как настроить в зоопарке технологий и зачем это нужно

Булат Усманов, Архитектор

купер

Об авторе

- Булат Усманов
- Архитектор в Купере
- 10+ лет в Fintech, E-Com, etc.; AWS Resilience Hub
- usmanovbf@gmail.com, @usmanovbf

План 

01 Что такое **Data Lineage** и его роль в **Data Governance**

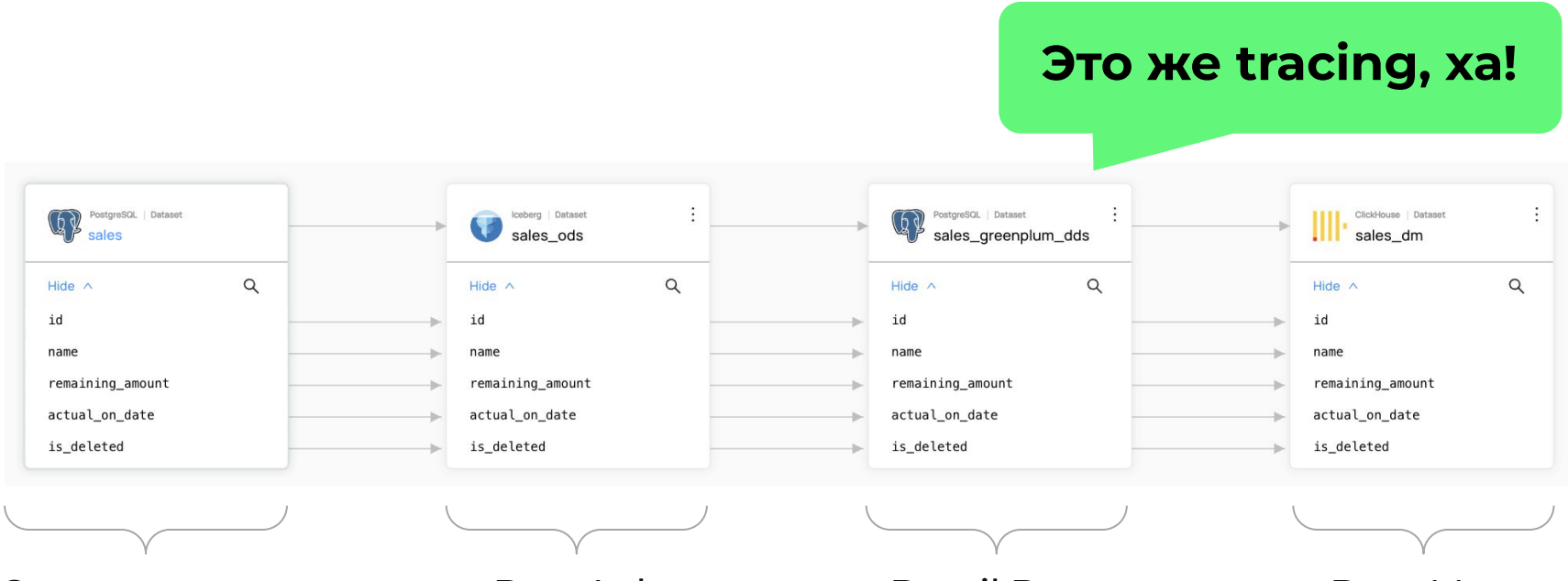
02 Зачем нужен Data Lineage

03 **DataHub** — платформа для Metadata и Data Lineage

04 Простое и сложное внедрение Data Lineage

05 Решения на основе Data Lineage

Что такое Data Lineage?



Операционная БД

Data Lake

Detail Data Store

Data Mart

Data Lineage в Data Governance

- **Data Governance**
  - Data Architecture
  - Data Quality
  - **Metadata**
    - Operational Metadata
    - **Data Lineage**
  - DWH & BI
  - Master Data

Метаданные таблиц, например

Если упростить «Data Management Body of Knowledge»

Зачем Data Lineage нужен?

01  Принятие обоснованных решений и снижение рисков

Зачем Data Lineage нужен?

01 Принятие обоснованных решений и снижение рисков

02 Соответствие регуляторным требованиям

Зачем Data Lineage нужен?

01 Принятие обоснованных решений и снижение рисков

02 Соответствие регуляторным требованиям

03 Улучшение координации между коллегами, командами

Зачем Data Lineage нужен?

01 Принятие обоснованных решений и снижение рисков

02 Соответствие регуляторным требованиям

03 Улучшение координации между коллегами, командами

04 Поддержание Data Mesh, Data Products

Зачем Data Lineage нужен?

01 Принятие обоснованных решений и снижение рисков

02 Соответствие регуляторным требованиям

03 Улучшение координации между коллегами, командами

04 Поддержание Data Mesh, Data Products

05 Ускорение разработки, Time to Market

DataHub

VS

OpenMetadata

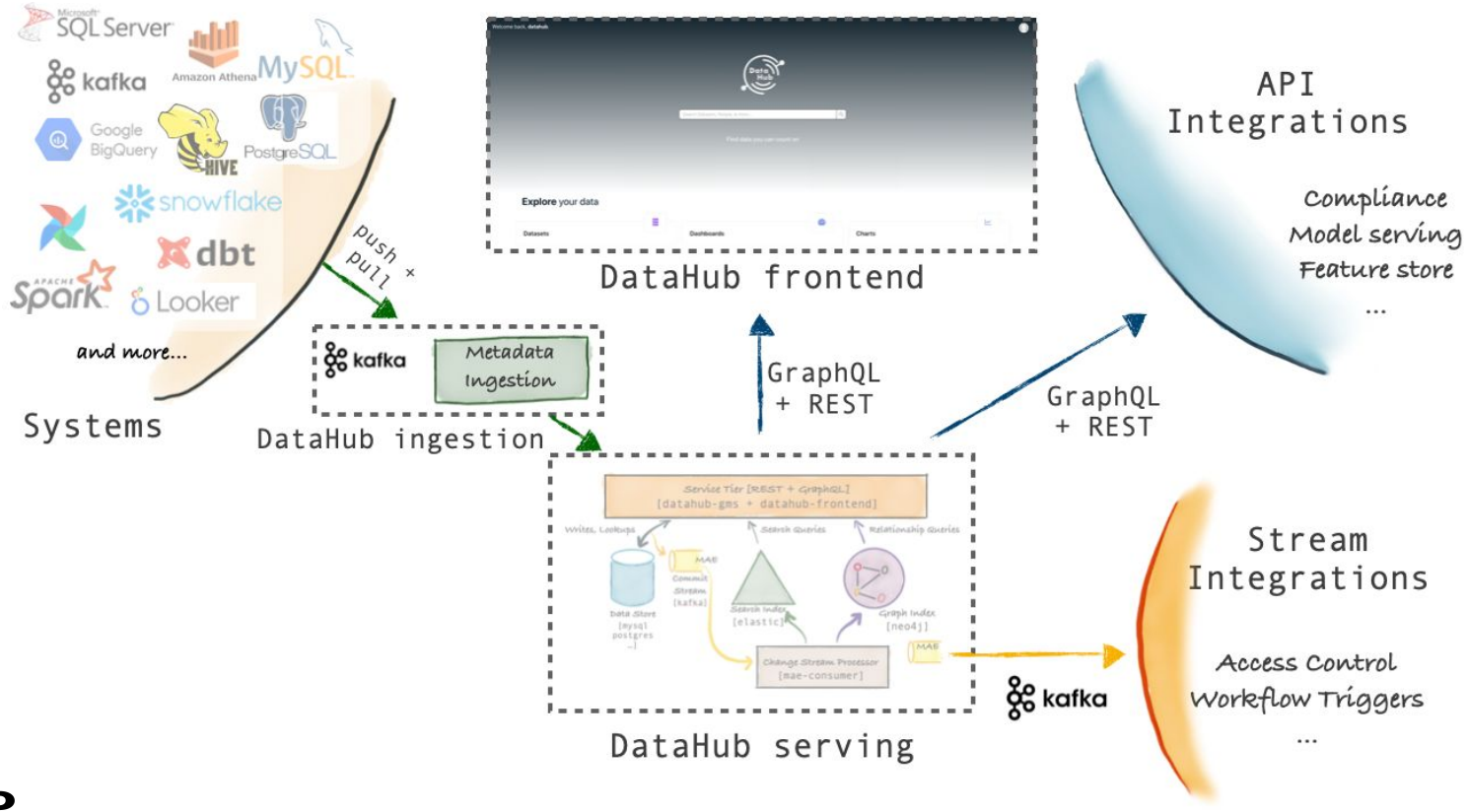
- Больше подключений к источникам
- Богаче API
- Больше community

- Приятнее UX/UI
- Нагляднее и богаче документация
- Отслеживание заполняемости

Приблизились к плюсам OpenMetadata

- Приятнее UX/UI
  - Привыкли к DataHub, но открыты к улучшениям
  - Дополнительный UI с помощью отчетов Tableau
- Нагляднее и богаче документация
  - Мы разве исходники не сможем прочитать?
- Отслеживание заполняемости
  - Отчеты в Tableau, куда данные выгружаются с помощью Airflow

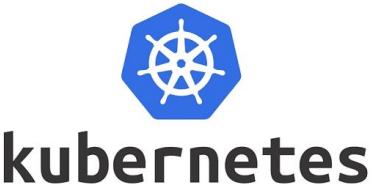
DataHub. Взгляд сверху



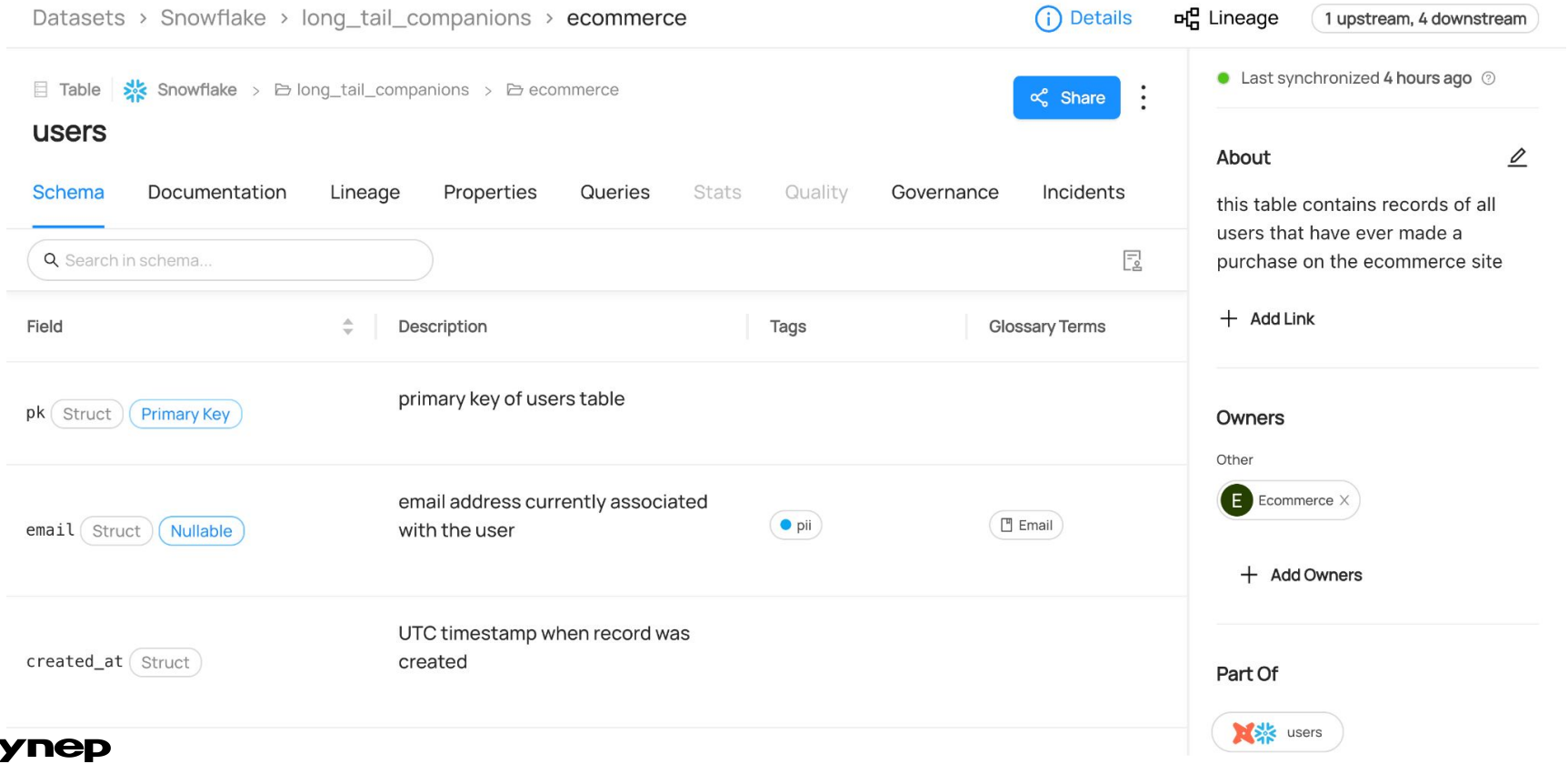
DataHub. Deploy

datahub-gms — Metadata Service. 3 instances при ~250 MAU, 150k items
datahub-ingestion — выгрузка метаданных с помощью “datahub ingest run -c recipe.yaml” по Cron
datahub-mce-consumer — приём новых метаданных из источников
datahub-mae-consumer — отправка готовых метаданных, для интеграций вовне
datahub-actions — фреймворк для кастомизации, для интеграций изнутри

Также ElasticSearch (Neo4j), MySQL, PostgreSQL, Kafka...



DataHub. Пример метаданных



DataHub. Выгрузка метаданных

recipe.yml

```yaml
source:
  type: "mysql"
  config:
    username: "datahub"
    password: "datahub"
    host_port: "localhost:3306"
sink:
  type: "datahub-rest"
  config:
    server: 'http://localhost:8080'
```

```
datahub ingest -c recipe.yml
```

- Pull подход, но есть ещё Push

DataHub. Кастомизация выгрузки

recipe.yml

```yaml
source:
  type: "mysql"
  config:
    username: "datahub"
    password: "datahub"
    host_port: "localhost:3306"
sink:
  type: "datahub-rest"
  config:
    server: 'http://localhost:8080'
```

```
datahub ingest -c recipe.yml
```

А еще:

- SQL профилирование
- Удаление неактуальных метаданных
- Статистика использования таблиц
- Классификация персональных данных по регулярным выражениям
- Пост-трансформация

Внедрение Data Lineage

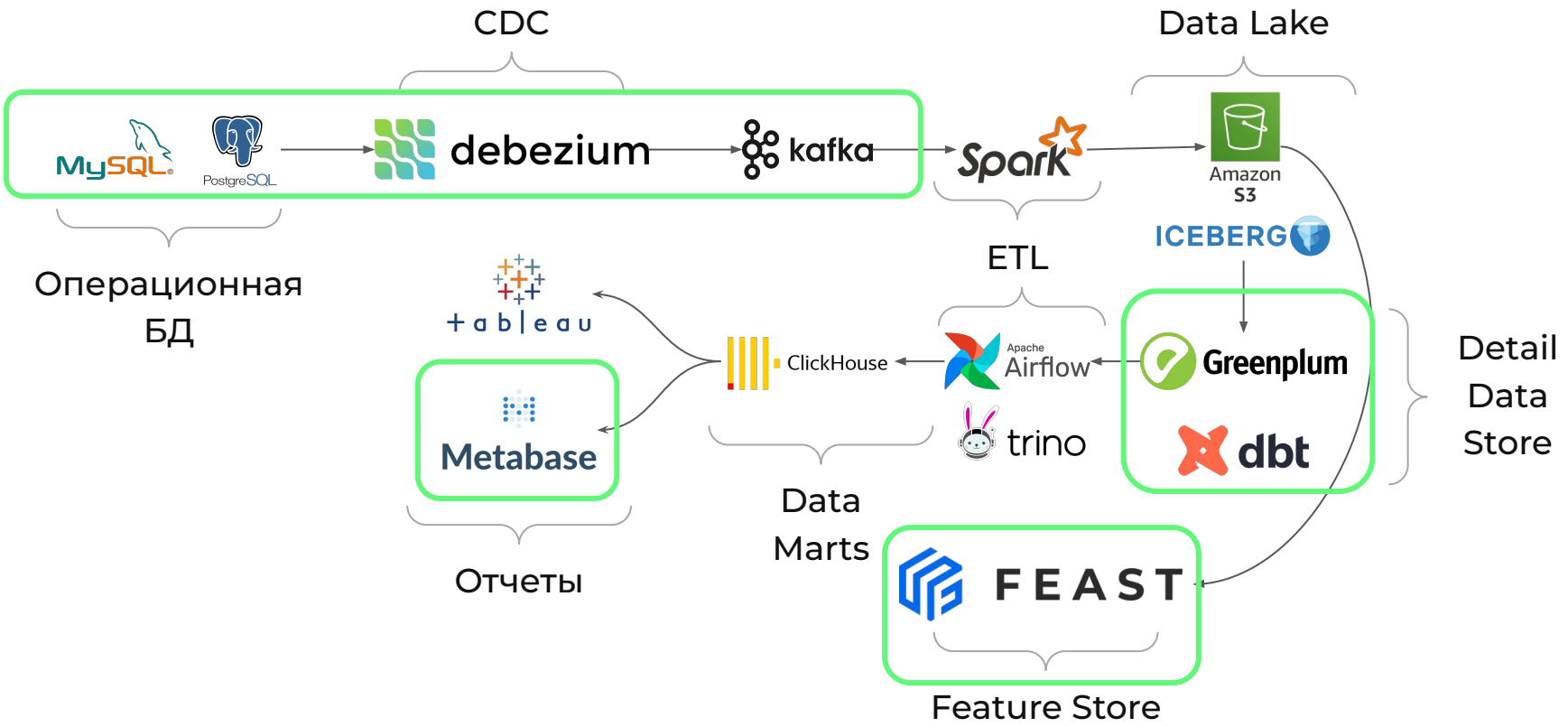
Простое

1. Написать Ingestion recipe.
2. Запустить.
3. Отслеживать — готово!

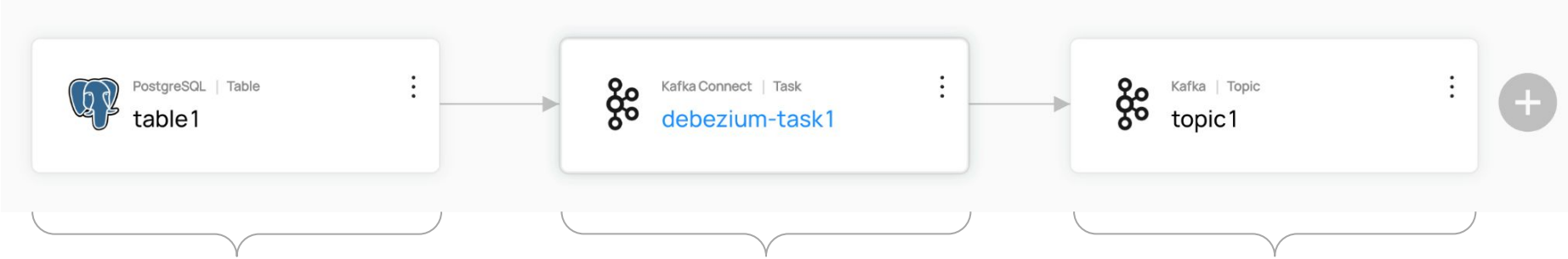
Посложнее

1. Написать Ingestion recipe.
2. **Кастомизировать выгрузку Metadata и Data Lineage (Java, Python).**
3. Запустить.
4. Отслеживать — готово!

Простое внедрение Data Lineage



Lineage для CDC от БД до Kafka

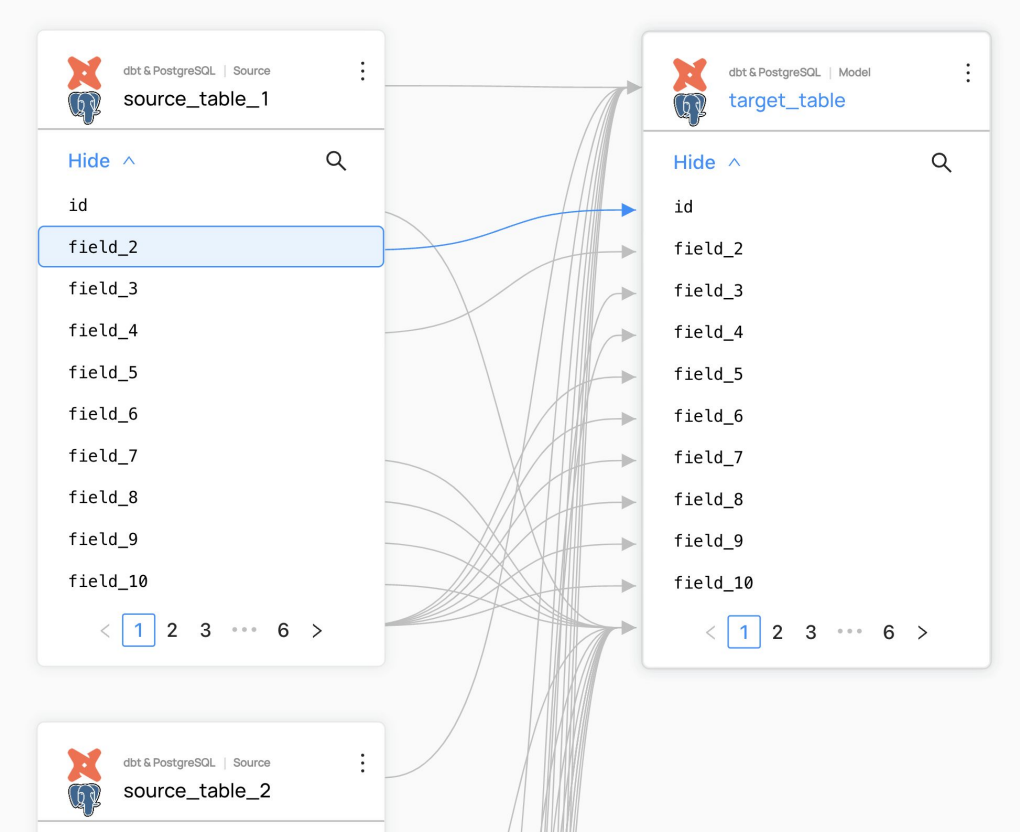


Операционная БД

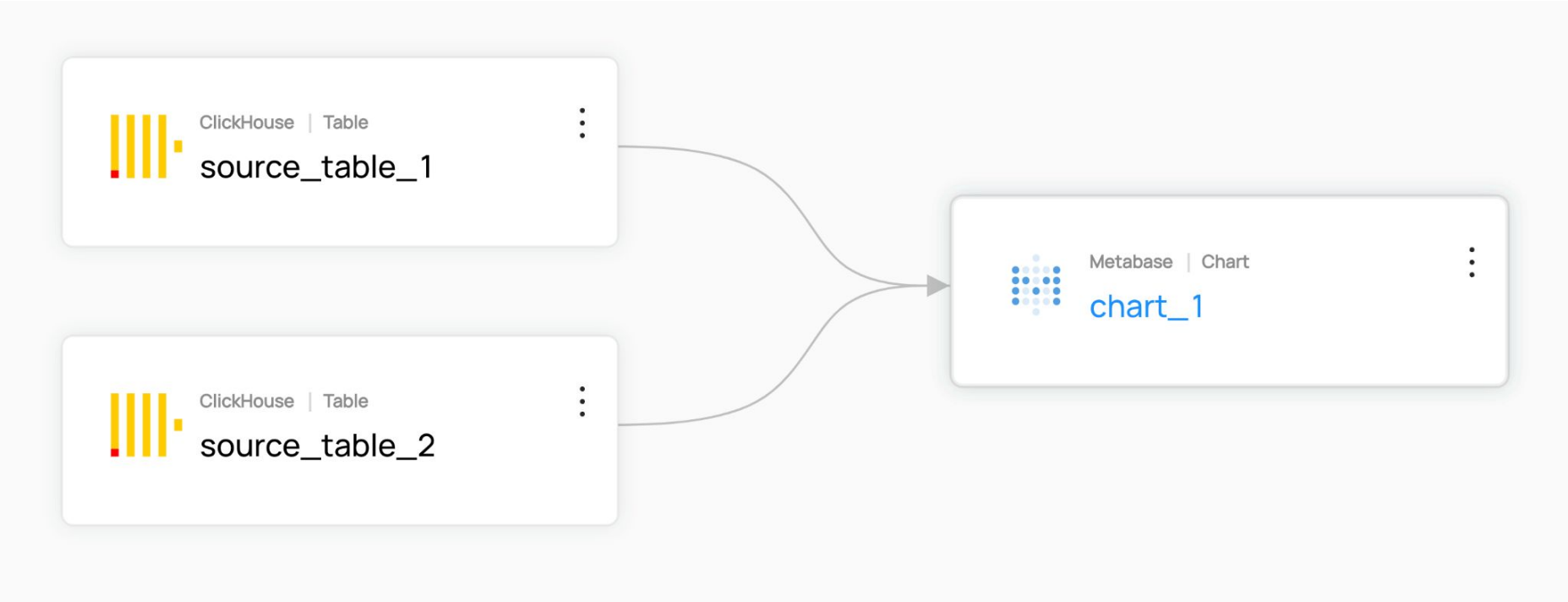
Debezium task (CDC)

CDC Kafka topic

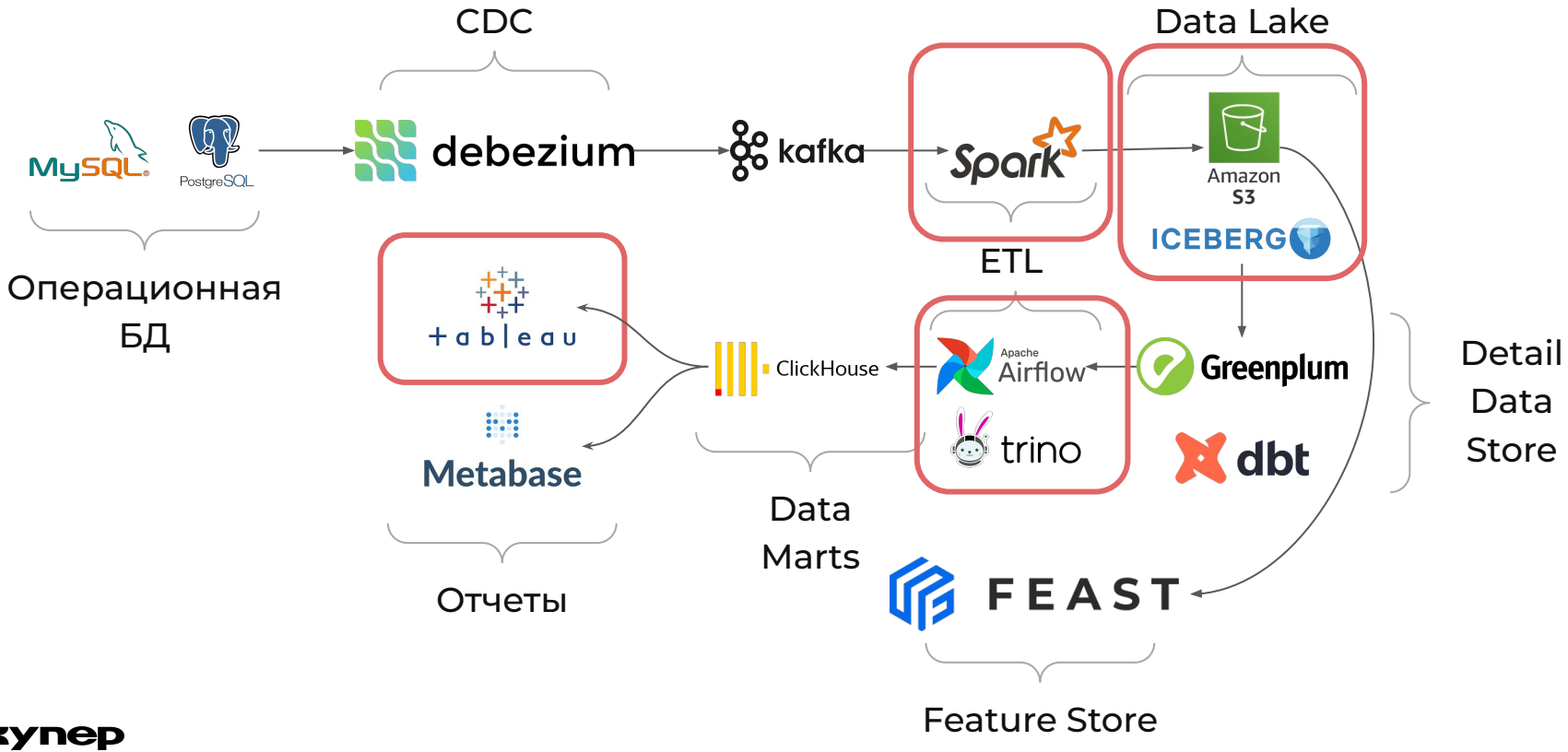
Lineage для dbt (Greenplum)



Lineage для Metabase



Внедрение Data Lineage посложнее



DataHub. Пост-трансформация. Проблема с S3

recipe.yaml

```yaml
source:
  type: s3
  config:
    path_specs:
      -
        include: 's3://bucket-name/{table}/{partition_key[0]}={partition[0]}/platform=ios/*.parquet'
      -
        include: 's3://bucket-name/{table}/{partition_key[0]}={partition[0]}/platform=android/*.parquet'
    aws_config:
      aws_access_key_id: '${ACCESS_KEY}'
      aws_secret_access_key: '${SECRET_KEY}'
```

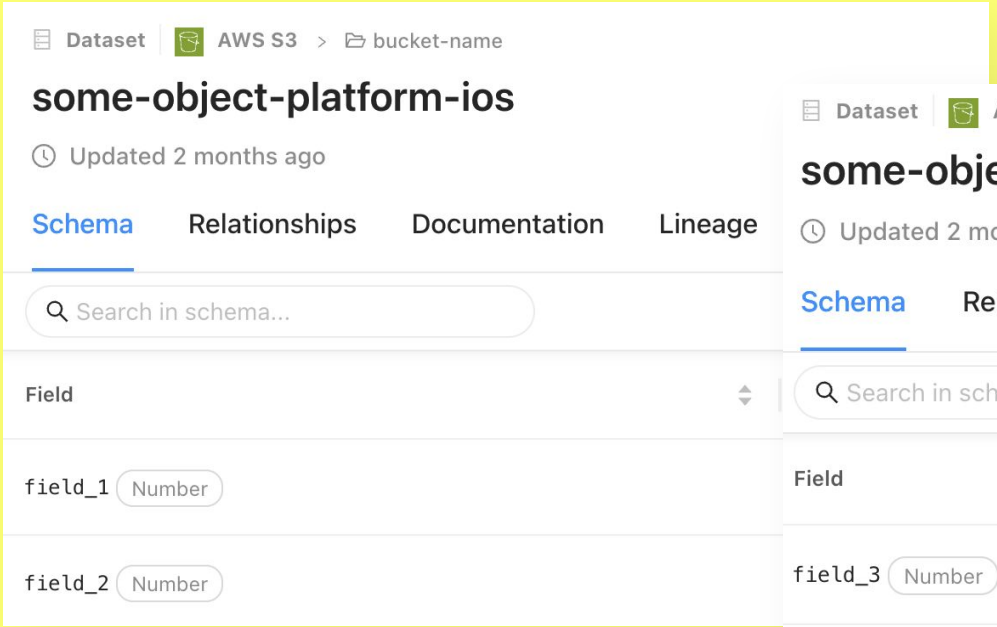
bucket-name.some-object для ios, android

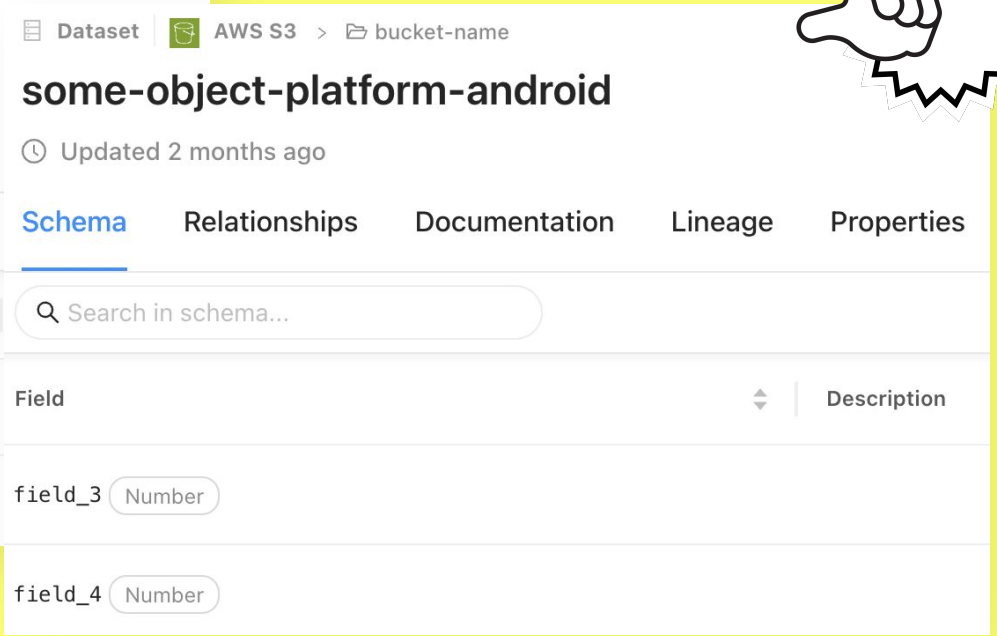
DataHub. Пост-трансформация. Решение

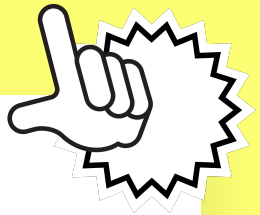
recipe.yaml

```yaml
source:
  type: s3
  config:
    path_specs:
      -
        include: 's3://bucket-name/{table}/{partition_key[0]}={partition[0]}/platform=ios/*.parquet'
      -
        include: 's3://bucket-name/{table}/{partition_key[0]}={partition[0]}/platform=android/*.parquet'
    aws_config:
      aws_access_key_id: '${ACCESS_KEY}'
      aws_secret_access_key: '${SECRET_KEY}'

transform:
  - type: "urn_modifier_transformer.urn_modifier_transformer:UrnModifierTransformer"
    config:
      regex_to_search: "/platform=(.*)?/"
      urn_to_modify: "{parsed_urn.entity_ids[1]}-platform-{match.group(1)}"
```

DataHub. Пост-трансформация. Результат







Data Lineage из Spark. Из коробки

datahub_spark_listener_out_of_the_box.sh

```
spark-submit \
  ... \
  --packages io.acryl:acryl-spark-lineage:0.2.16 \
  --conf "spark.extraListeners=datahub.spark.DatahubSparkListener" \
  your_spark_job_to_run.py # Или через JAR
```

Но не всегда как надо 😕

Много лишних связей в Lineage, много метаданных

Data Lineage из Spark. Кастомизация на Java

```java
public class YourOwnDatahubSparkListener extends org.apache.spark.scheduler.SparkListener {

    @Override
    public void onApplicationStart(SparkListenerApplicationStart applicationStart) {
        saveToContext(applicationStart); // Сохранение данных в контексте для дальнейшей обработки
    }

    @Override
    public void onOtherEvent(SparkListenerEvent input) {
        SparkListenerSQLExecutionStart event = (SparkListenerSQLExecutionStart) input;
        QueryExecution execution = SQLExecution.getQueryExecution(event.executionId());
        LogicalPlan logicalPlan = execution.logical();
        extractLineage(logicalPlan); // Извлечение Lineage из текущего запроса в виде LogicalPlan
    }

    @Override
    public void onApplicationEnd(SparkListenerApplicationEnd applicationEnd) {
        emitToDatahub(applicationEnd); // Отправка извлеченного Lineage в DataHub
    }
}
```

Извлечение Lineage из LogicalPlan

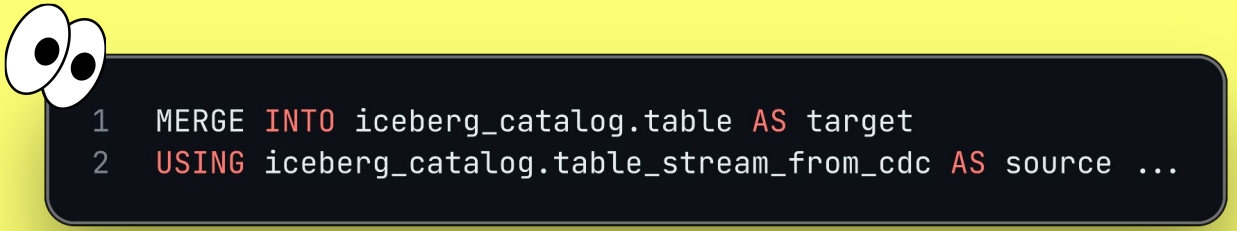
LogicalPlan.scala

```scala
package org.apache.spark.sql.catalyst.plans.logical

abstract class LogicalPlan
  extends QueryPlan[LogicalPlan]
  with AnalysisHelper
  with LogicalPlanStats
  with LogicalPlanDistinctKeys
  with QueryPlanConstraints
  with Logging {}

abstract class QueryPlan[PlanType <: QueryPlan[PlanType]]
  extends TreeNode[PlanType]
```

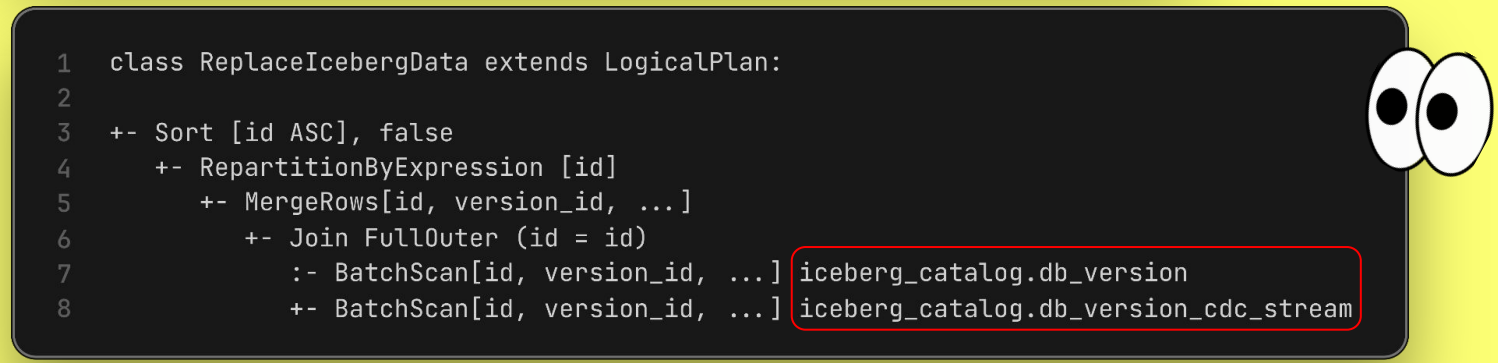
LogicalPlan похож
на результат
от EXPLAIN благодаря
TreeNode

Обойдём дерево!

Извлечение Lineage из LogicalPlan. Пример

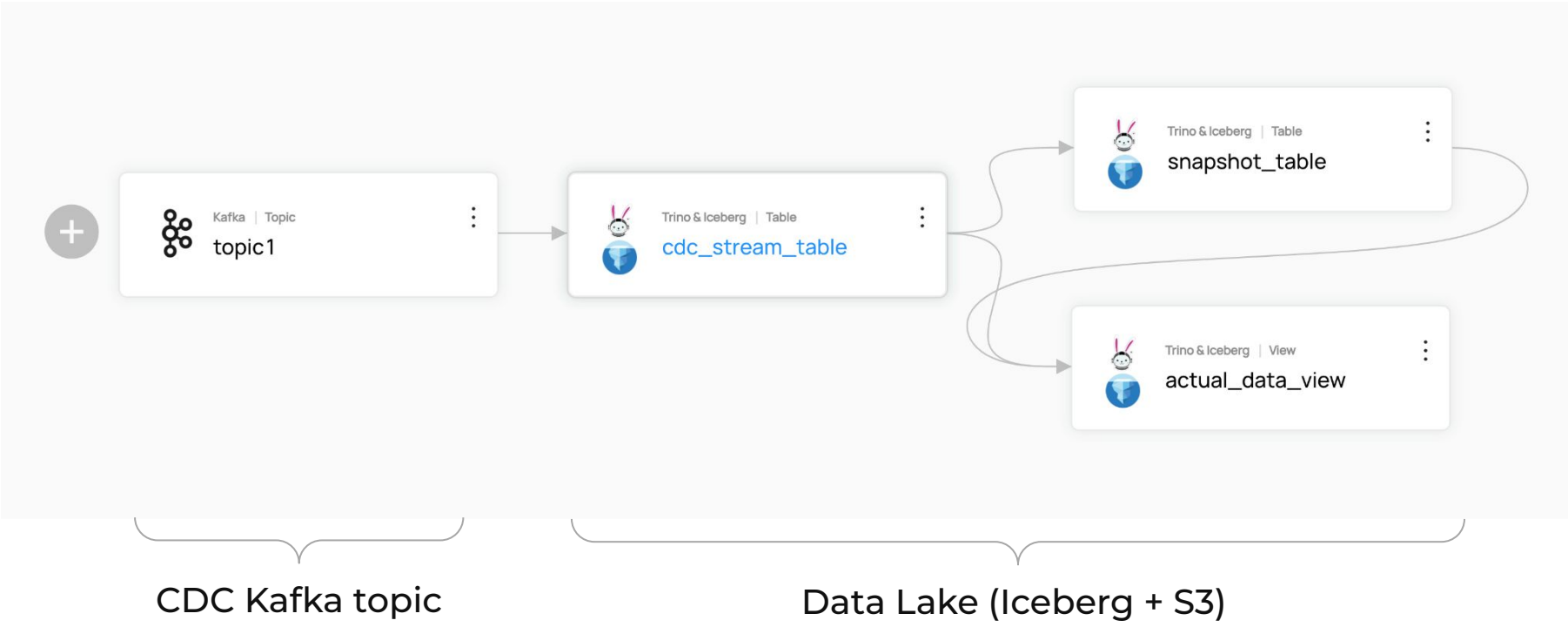


```
MERGE INTO iceberg_catalog.table AS target
USING iceberg_catalog.table_stream_from_cdc AS source ...
```



```
class ReplaceIcebergData extends LogicalPlan:

+- Sort [id ASC], false
   +- RepartitionByExpression [id]
      +- MergeRows[id, version_id, ...]
         +- Join FullOuter (id = id)
            :- BatchScan[id, version_id, ...] iceberg_catalog.db_version
            +- BatchScan[id, version_id, ...] iceberg_catalog.db_version_cdc_stream
```

Lineage для Spark. От Kafka до Data Lake



Data Lineage из Airflow. Из коробки

simpe airflow integration.sh

```
pip install 'acryl-datahub-airflow-plugin[plugin-v2]'
airflow connections add  --conn-type 'datahub-rest' \
  'connection_name' \
  --conn-host 'https://datahub-gms:443' \
  --conn-password 'password'
```

Но не всегда 😕

Работает для некоторых популярных Airflow операторов в Airflow 2.3+

Data Lineage из Airflow. Inlets & outlets

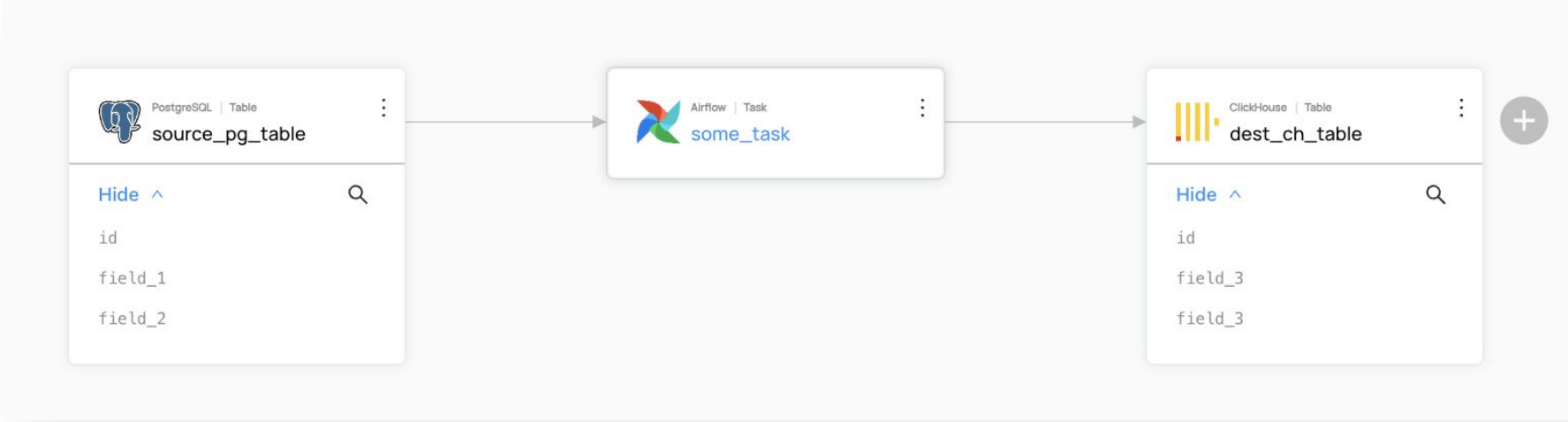
```python
ParamsInletsOutletsOperator(
        task_id="run_data_task", dag=dag,
        inlets=[Dataset(platform="postgres", name="db.schema.table1")],
        outlets=[Dataset("clickhouse", "db.schema.table2")],
)
```

```python
class InnerInletsOutletsOperator(BaseOperator):

    def execute(self, context):
        ...
        inlets, outlets = self._get_lineage()
        context['ti'].task.inlets = self.inlets
        context['ti'].task.outlets = self.outlets

    def _get_lineage(self) -> Tuple[List[str], List[str]]:
        ...
        return inlets, outlets
```

DataHub SDK внутри оператора со своим API

```python
# urn:li:dataset:(urn:li:dataPlatform:postgres,instance-1.db.schema.table1,PROD)
src_entity = DataHubEntity(
    # platform="postgres", platform_instance="instance-1"
    platform_instance=datahub_platform_instances.POSTGRES_PLATFORM_INSTANCE,
    # db.schema.table1
    entity_name=full_src_table
)
dst_entity = DataHubEntity(
    platform_instance=datahub_platform_instances.CLICKHOUSE_PLATFORM_INSTANCE,
    entity_name=full_dst_table
)

lineage_items = [DataHubLineageItem(src=[src_entity], dst=dst_entity)]

datahub_lineage_operator = DataHubLineageOperator(
    task_id=f"lineage_{dst_table}",
    lineage=lineage_items,
    dag=dag
)
```

Data Lineage из Airflow. Результат



А не лучше ли не прописывать руками?

01 Не везде используется SQL – Python, Java ETL ...

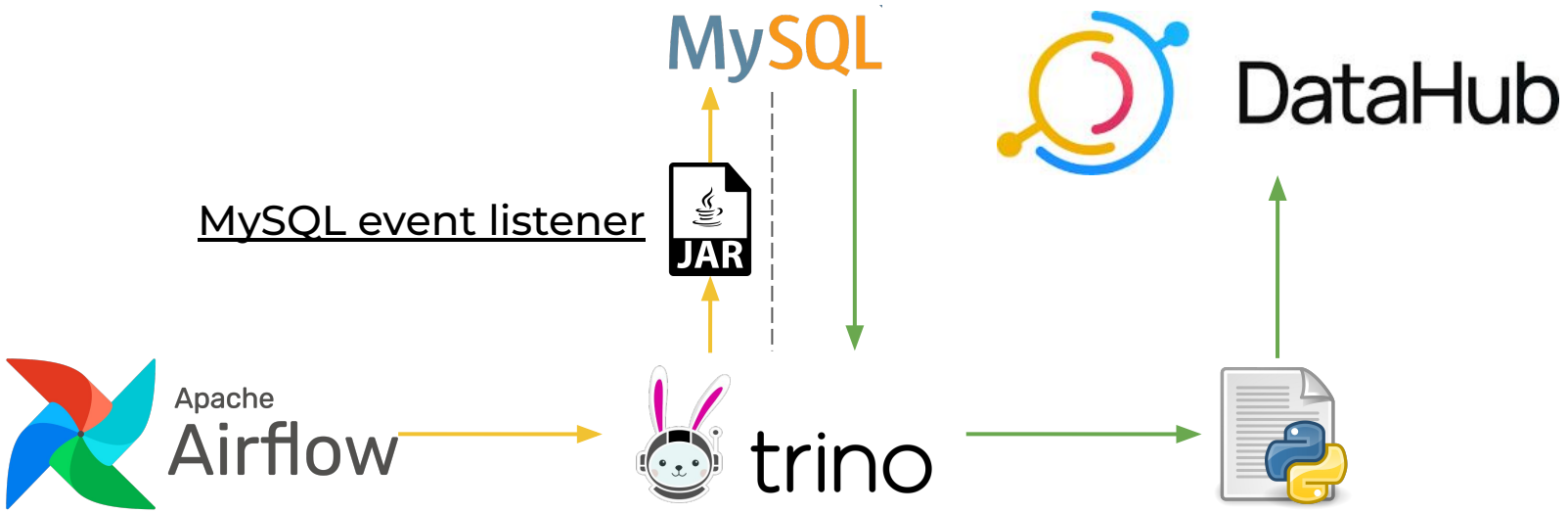
02 Парсер SQL запросов SQLGlot

03 Нужно расширять его диалекты для Postgres, Clickhouse, etc. — покажу на примере Trino

Lineage из Trino

01 Сохранение SQL запросов из Trino в отдельную БД

02 Извлечение Lineage из сохраненных запросов



Lineage из Trino. Кастомизация Trino ingestion

ExtendedTrinoSource.py

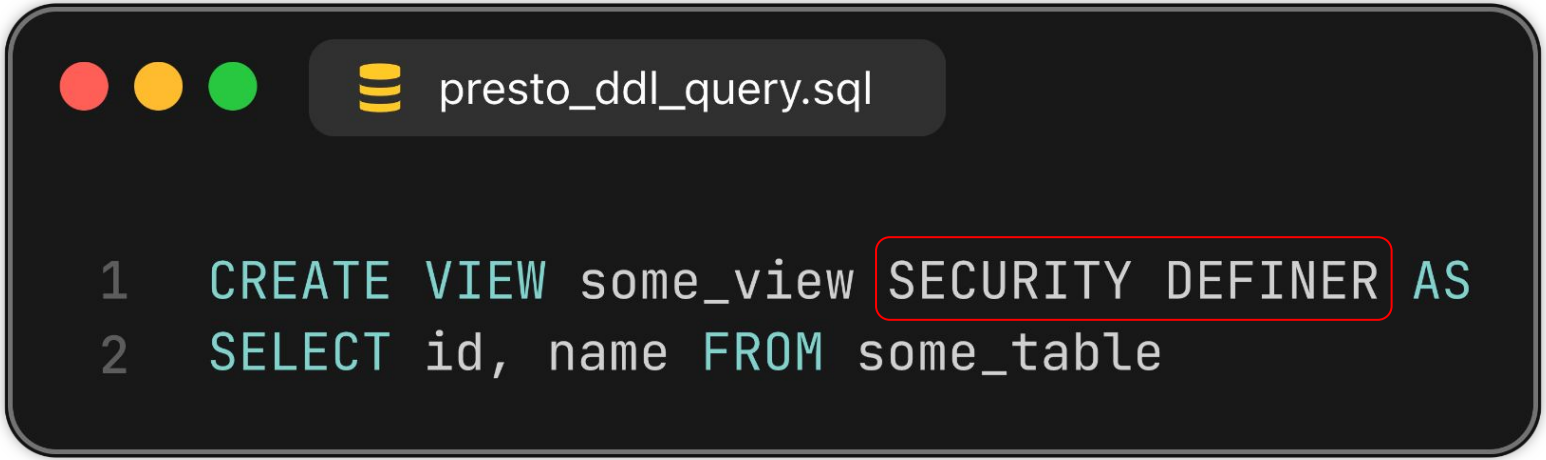
```python
from datahub.ingestion.source.sql.trino import TrinoConfig, TrinoSource
from datahub.sql_parsing.sqlglot_lineage import sqlglot_lineage, SqlParsingResult

class ExtendedTrinoSourceConfig(TrinoConfig):
  some_extra_config_param: str = Field(description="Write you own config param")

class ExtendedTrinoSource(TrinoSource):

  def get_workunits_internal(self) -> typing.Iterable[MetadataWorkUnit]:
    for inspector in self.get_inspectors():
      sql_queries: List[str] = find_queries_from_mysql_db(inspector)

        for sql_query in sql_queries:
          lineage: SqlParsingResult = sqlglot_lineage(sql=sql_query, ...)
          yield extract_metadata_work_units(lineage)
```

Lineage из Trino. Использование SQLGlot

ExtendedTrinoSource.py

```python
from datahub.ingestion.source.sql.trino import TrinoConfig, TrinoSource
from datahub.sql_parsing.sqlglot_lineage import sqlglot_lineage, SqlParsingResult

class ExtendedTrinoSourceConfig(TrinoConfig):
  some_extra_config_param: str = Field(description="Write you own config param")

class ExtendedTrinoSource(TrinoSource):

  def get_workunits_internal(self) -> typing.Iterable[MetadataWorkUnit]:
    for inspector in self.get_inspectors():
      sql_queries: List[str] = find_queries_from_mysql_db(inspector)

        for sql_query in sql_queries:
          lineage: SqlParsingResult = sqlglot_lineage(sql=sql_query, ...)
          yield extract_metadata_work_units(lineage)
```

Расширение диалектов SQLGlot. Пример



Расширение диалекта SQLGlot. Решение

expressions.py

```python
class SecurityProperty(Property):
    arg_types = {"this": True}
```

parser.py

```python
class Parser(metaclass=_Parser):
    PROPERTY_PARSERS: t.Dict[str, t.Callable] = {
        ...
        "SECURITY": lambda self: self._parse_security(),
    }

    def _parse_security(self) -> t.Optional[exp.SecurityProperty]:
        if self._match_texts(("DEFINER", "INVOKER")):
            security_specifier = self._prev.text.upper()
            return self.expression(exp.SecurityProperty, this=security_specifier)
        return None
```

Расширение диалекта SQLGlot. Решение

sqlglot PR #4008 – Merged

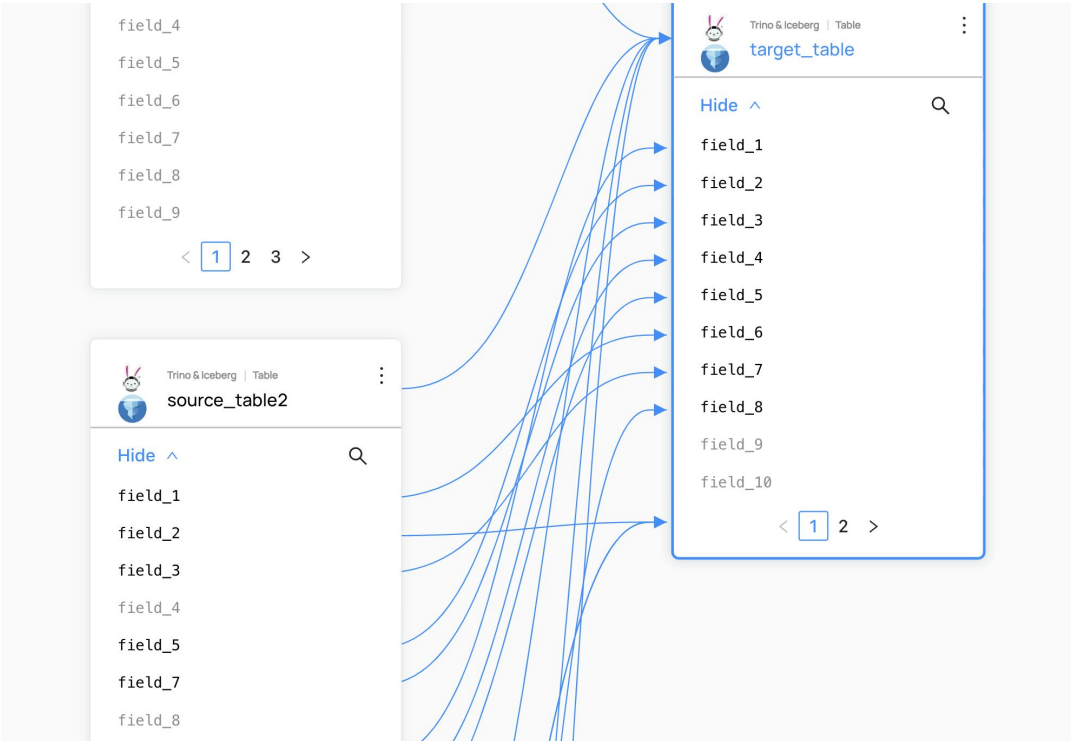
expressions.py

```python
class SecurityProperty(Property):
    arg_types = {"this": True}
```

generator.py

```python
class Generator(metaclass=_Generator):
    TRANSFORMS: t.Dict[t.Type[exp.Expression], t.Callable[..., str]] = {
        ...
        exp.SecurityProperty: lambda self, e: f"SECURITY {self.sql(e, 'this')}",
    }

    PROPERTIES_LOCATION = {
        ...
        exp.SecurityProperty: exp.Properties.Location.POST_SCHEMA,
    }
```

Lineage из Trino. Результат



Решения на основе Data Lineage

01 Предупреждение ломающих DDL изменений

 Kuper-Tech/ddl-breaking-change

Скоро будет код ☺

02 Автоматическое проставление тегов

 Kuper-Tech/datahub-tags-distribution

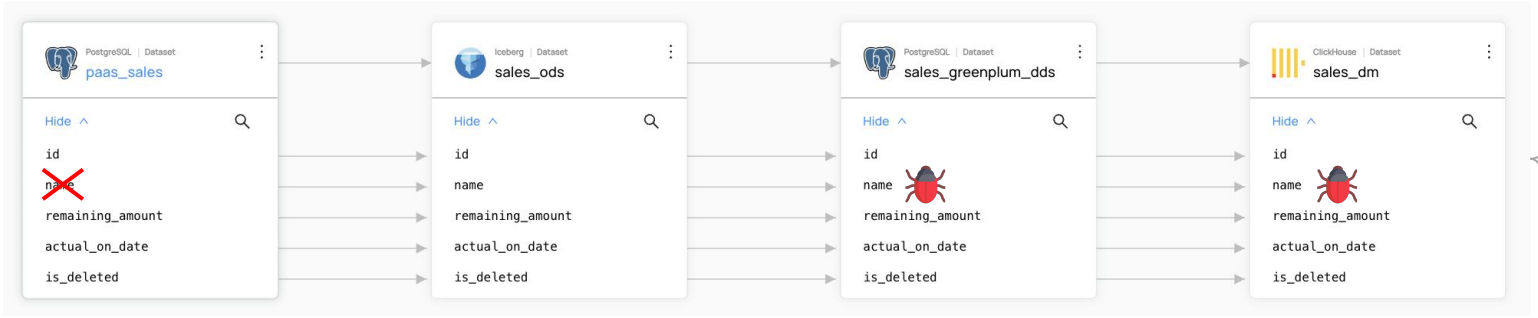
DDL breaking change. Проблема

01 DROP TABLE ...

02 ... DROP COLUMN ...

03 ... RENAME COLUMN ...

04 ... ALTER COLUMN ... TYPE



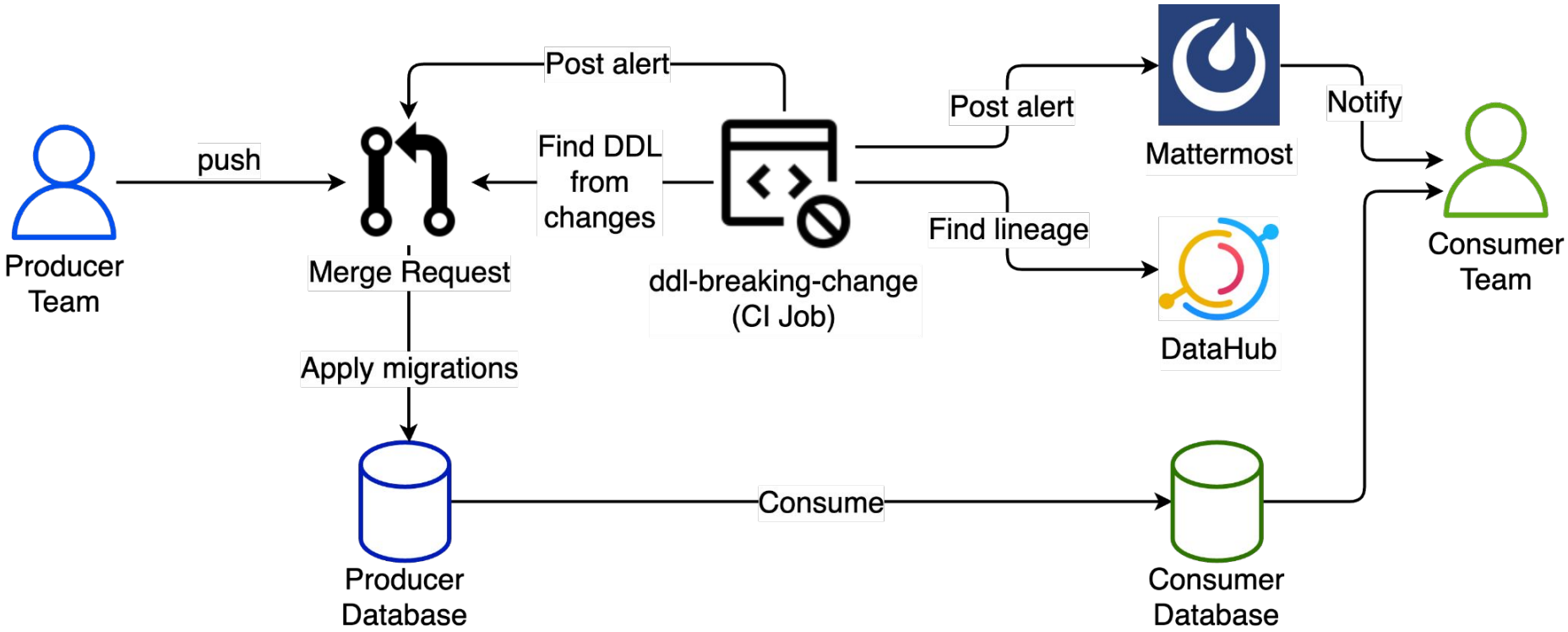
Операционная БД

Data Lake

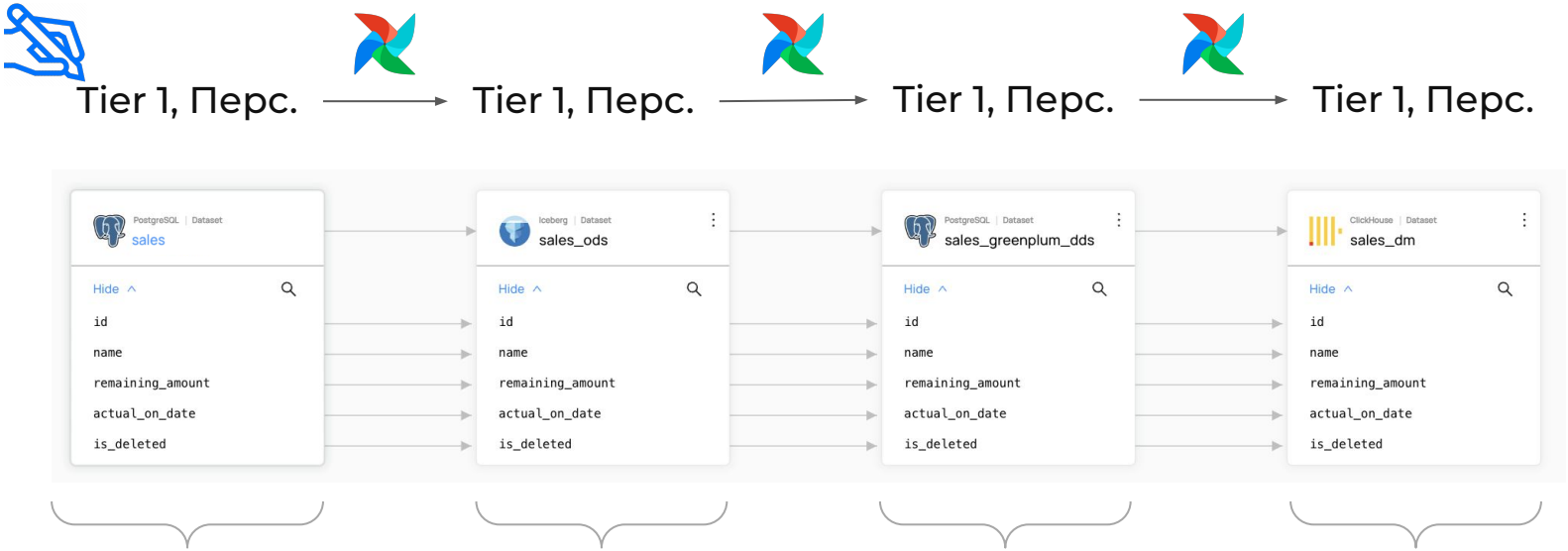
Detail Data Store

Data Mart

DDL breaking change. Решение



Автоматическое проставление тегов



Операционная БД

Data Lake

Detail Data Store

Data Mart

Спасибо! Вопросы?

- Булат Усманов
- Архитектор в Купере
- usmanovbf@gmail.com,
  @usmanovbf