



Онлайн образование



Проверить, идет ли запись

Меня хорошо видно && слышно?



Тема вебинара

ksqlDB



Заигрин Вадим

Ведущий эксперт по технологиям, Сбербанк

vzaigrin@yandex.ru

<https://t.me/vzaigrin>



Преподаватель



Вадим Заигрин

Более 30 лет в ИТ:

- Big Data
 - Data Engineer
 - Data Science
- Разработка
 - Scala, Java, Python, C, Lisp
- IT Infrastructure
 - Администрирование
 - Сопровождение
 - Архитектура

Big Data проекты в банках, телекоме и в рознице.



Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в Telegram



Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения



Индивидуально



Время, необходимое
на активность



Пишем в чат



Говорим голосом

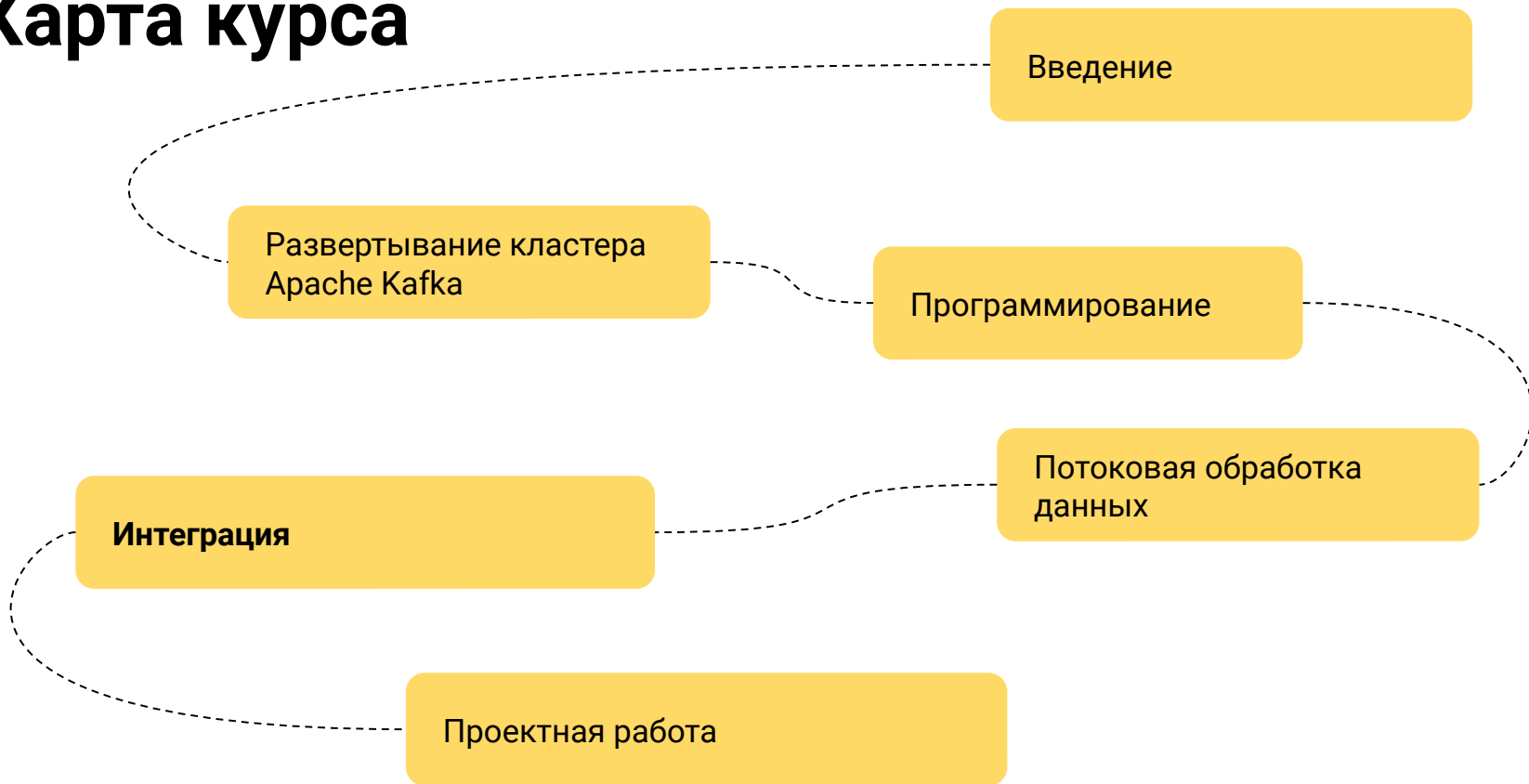


Документ



Ответьте себе или
задайте вопрос

Карта курса



Маршрут вебинара



Конвейер данных

Знакомство с ksqlDB

Основные понятия

Интеграция данных

Рефлексия

Цели вебинара

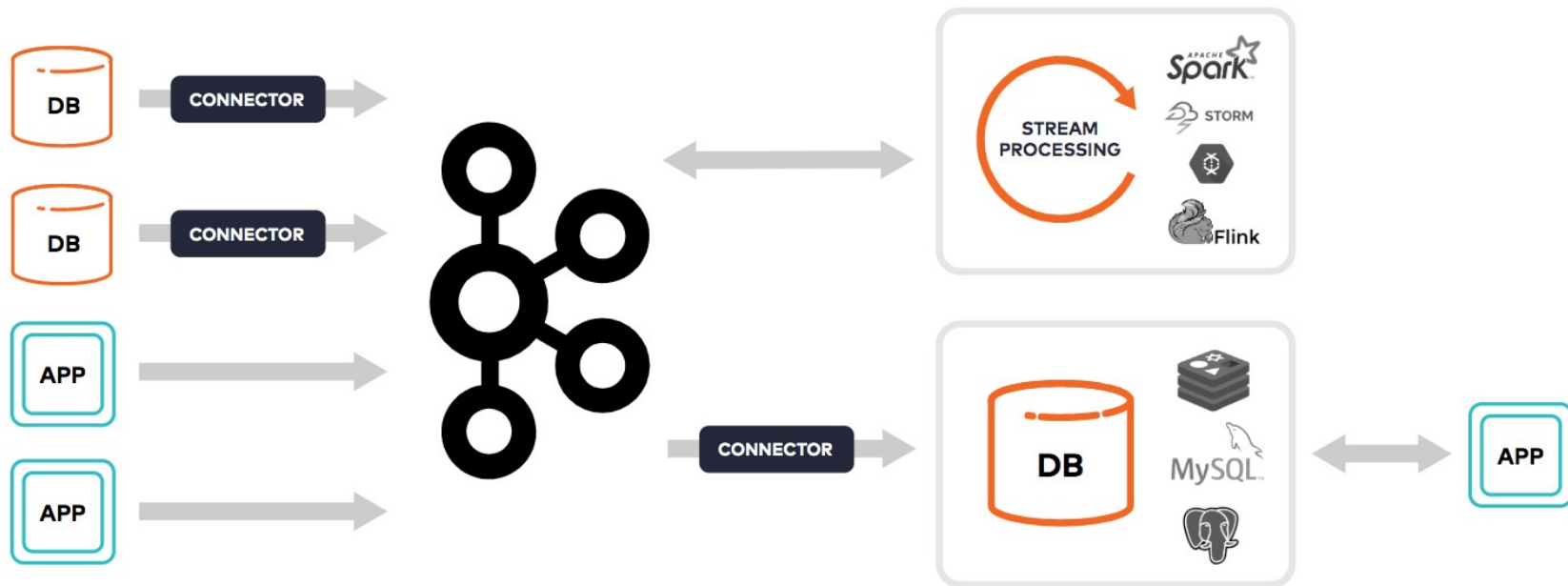
- | | |
|----|---------------------------------------|
| 1. | Вспомним конвейер данных |
| 2. | Познакомимся с ksqlDB |
| 3. | Рассмотрим интеграцию данных в ksqlDB |

Смысл

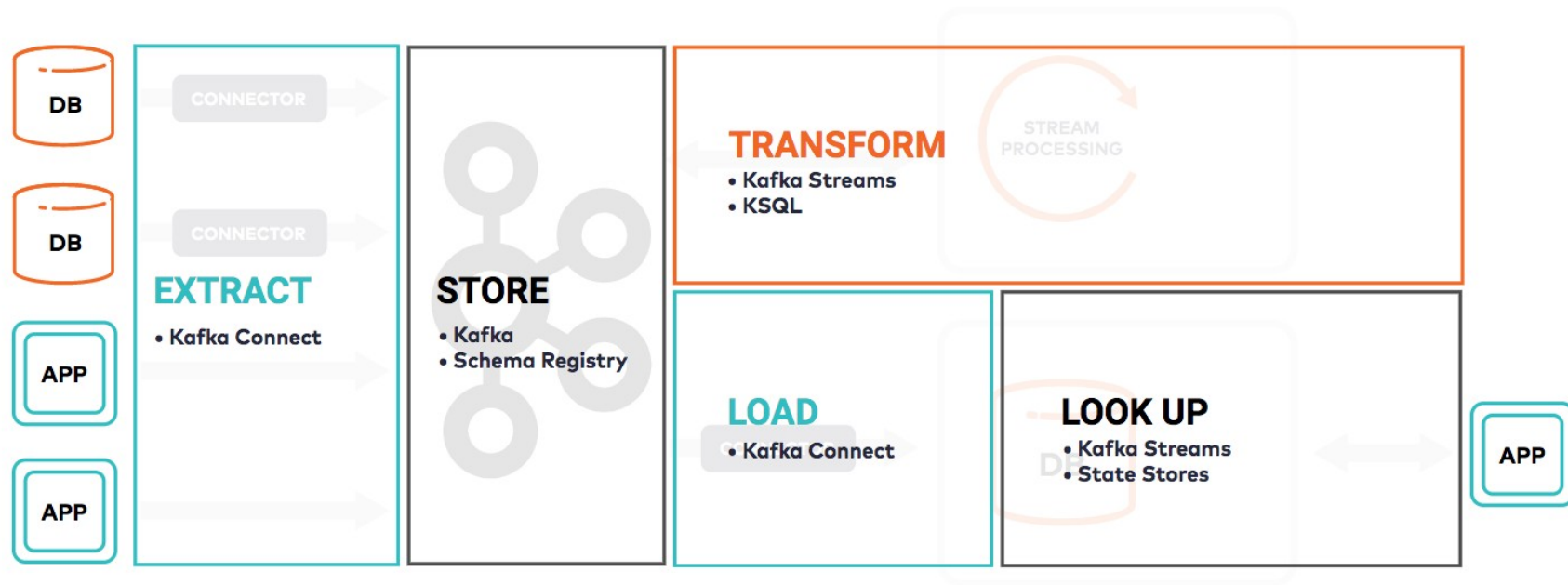
1. Сможем запустить и использовать ksqlDB
2. Сможем интегрировать ksqlDB с источниками данных

Конвейер данных

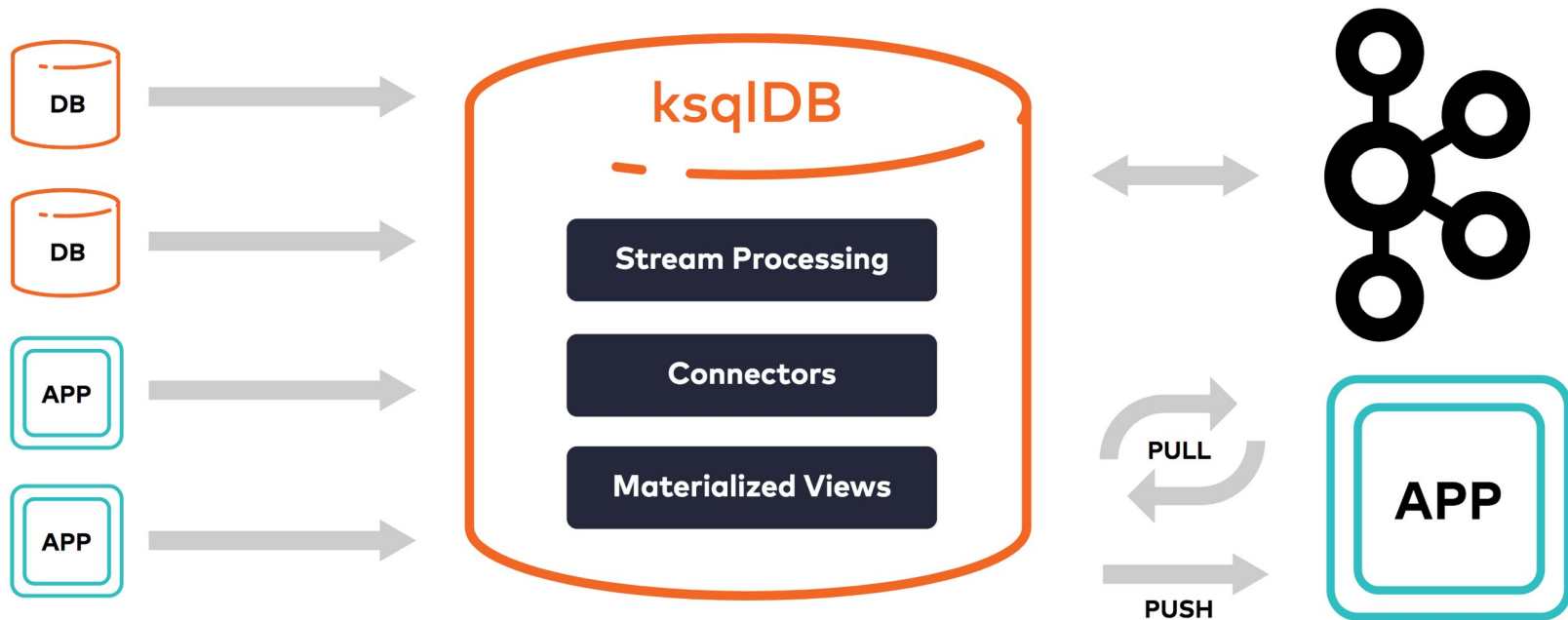
Потоковый конвейер данных



Потоковый ETL



Одна система для всего



ksqlDB

Что такое ksqlDB?

ksqlDB — база данных потоковой передачи сообщений

Возможности:

- Моделирование данных в виде потоков или таблиц посредством SQL
- Применение конструкций SQL для создания новых производных представлений данных
- Выполнение запросов на передачу данных (*push-запросов*)
- Создание *материализованных представлений* из потоков и таблиц
- Запрос представлений посредством запросов на получение данных (*pull-запросов*)
- Определение коннекторов для интеграции с внешними хранилищами данных

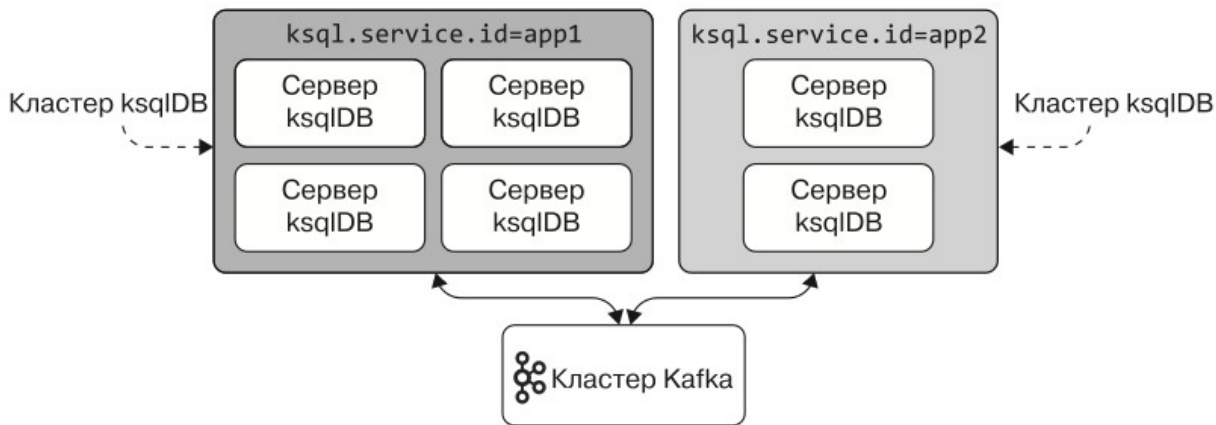
Сценарии использования

- Материализованные кэши
<https://docs.ksqldb.io/en/latest/tutorials/materialized/>
- Поточковые конвейеры ETL
<https://docs.ksqldb.io/en/latest/tutorials/etl/>
- Микросервисы, управляемые событиями
<https://docs.ksqldb.io/en/latest/tutorials/event-driven-microservice/>

Архитектура

Сервер ksqlDB

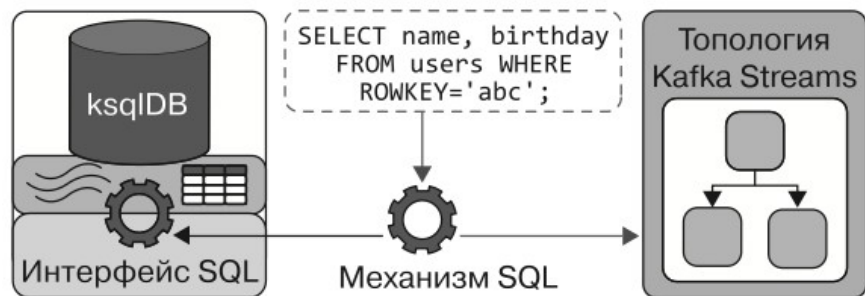
- **Сервер ksqlDB** отвечает за выполнение приложений потоковой обработки
- **Кластер ksqlDB** — это группа взаимодействующих серверов ksqlDB
- Сервер ksqlDB состоит из:
 - ядро SQL
 - служба REST



Ядро SQL

Механизм SQL отвечает за:

- синтаксический анализ операторов SQL
- преобразование операторов в одну или несколько топологий Kafka Streams
- запуск приложений Kafka Stream



Служба REST

Интерфейс REST предназначен для взаимодействия клиентов с механизмом SQL

Пример:

```
curl -X "POST" "http://localhost:8088/query" \  
      -H "Content-Type: application/vnd.ksql.v1+json; charset=utf-8" \  
      -d $'{  
    "ksql": "SELECT USERNAME FROM users EMIT CHANGES;",  
    "streamsProperties": {}  
  }'
```

ksqlDB CLI

ksqlDB CLI — это приложение, позволяющее взаимодействовать с сервером ksqlDB

Пример:

ksql http://localhost:8088

```
=====
=                                     =
=      _   _   _   _   _   _   _   =
=      | |   | |   | |   | |   | |   =
=      | | / _/ _/ _/ _/ _/ _/ _/   =
=      |  < \ \ ( | | | | | | | | ) | =
=      | | \ \ \ \ \ \ \ \ \ \ \ \ \ | =
=      | |                                     | | =
=      The Database purpose-built         =
=      for stream processing apps         =
=====
```

Copyright 2017-2022 Confluent Inc.

CLI v0.29.0, Server v0.29.0 located at http://ksqldb-server:8088

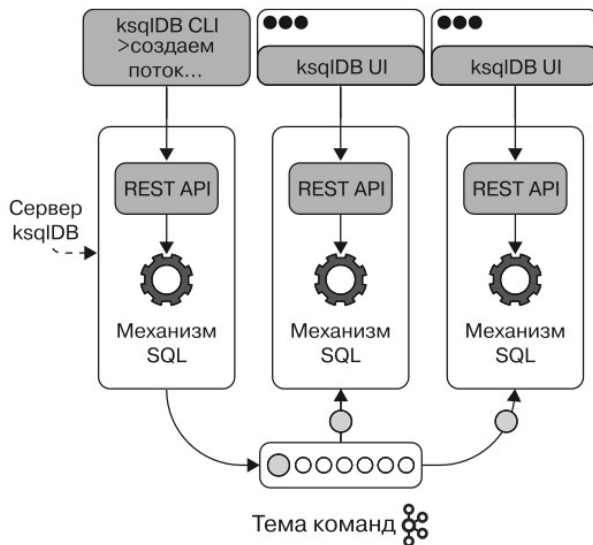
Server Status: RUNNING



Режимы развертывания

Интерактивный режим

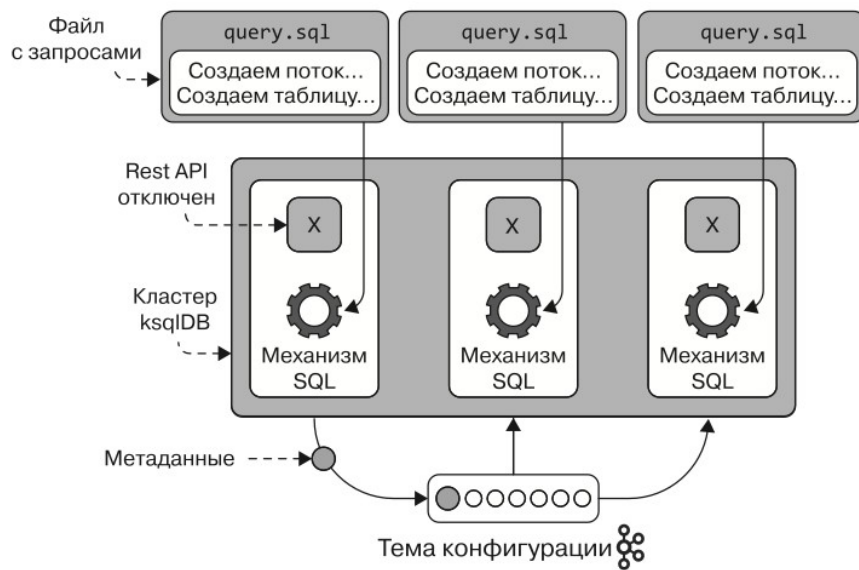
- Клиенты могут отправлять новые запросы в любое время посредством REST API
- Все запросы, отправленные в механизм SQL через REST API, записываются во внутреннюю тему — *тему команд*



Автономный режим

Автономный режим отключает REST API и гарантирует невозможность изменения выполняемых запросов

`queries.file=/path/to/query.sql`



Основные понятия

Событие

Событие — основная единица данных, «строка» таблицы:

- Данные (key, value)
- HEADERS (если есть)
- Метаданные (псевдоколонки):
 - ROWTIME — время события в ms
 - ROWPARTITION — партиция в топике
 - ROWOFFSET — смещение в топике

Потоки

Поток — это разделенная на разделы, *неизменяемая* коллекция, доступная только для добавления

```
CREATE [OR REPLACE] [SOURCE] STREAM [IF NOT EXISTS] stream_name
  ( { column_name data_type [KEY | HEADERS | HEADER(key)] } [, ...] )
  WITH ( property_name = expression [, ...] );
```

-- stream with a page_id column loaded from the kafka message value:

```
CREATE STREAM pageviews (
  page_id BIGINT,
  viewtime BIGINT,
  user_id VARCHAR
) WITH (
  KAFKA_TOPIC = 'keyless-pageviews-topic',
  VALUE_FORMAT = 'JSON'
);
```

-- stream with a page_id column loaded from the kafka message key:

```
CREATE STREAM pageviews (
  page_id BIGINT KEY,
  viewtime BIGINT,
  user_id VARCHAR
) WITH (
  KAFKA_TOPIC = 'keyed-pageviews-topic',
  VALUE_FORMAT = 'JSON'
);
```



Таблицы

Таблица — это изменяемая, разделенная на разделы коллекция, модели которой меняются с течением времени

```
CREATE [OR REPLACE] [SOURCE] TABLE [IF NOT EXISTS] table_name
( { column_name data_type [PRIMARY KEY] } [, ...] )
WITH ( property_name = expression [, ...] );
```

```
-- table with declared columns:
CREATE TABLE users (
  id BIGINT PRIMARY KEY,
  usertimestamp BIGINT,
  gender VARCHAR,
  region_id VARCHAR
) WITH (
  KAFKA_TOPIC = 'my-users-topic',
  VALUE_FORMAT = 'JSON'
);
```



Работа с потоками и таблицами

- Вывод списка потоков и таблиц

```
{ LIST | SHOW } { STREAMS | TABLES } [EXTENDED];
```

- Получение описаний потоков и таблиц

```
DESCRIBE [EXTENDED] <идентификатор>;
```

- Изменение потоков и таблиц

```
ALTER { STREAM | TABLE } <идентификатор> параметры [, ...]
```

- Удаление потоков и таблиц

```
DROP { STREAM | TABLE } [ IF EXISTS ] <идентификатор> [DELETE TOPIC]
```



Материализованные представления

Материализованные представления — потоки и таблицы, производные от другой коллекции

```
CREATE TABLE season_length_change_counts
WITH (
  KAFKA_TOPIC = 'season_length_change_counts',
  VALUE_FORMAT = 'AVRO',
  PARTITIONS = 1
) AS
SELECT
  title_id,
  season_id,
  COUNT(*) AS change_count,
  LATEST_BY_OFFSET(new_episode_count) AS episode_count
FROM season_length_changes_enriched
WINDOW TUMBLING (
  SIZE 1 HOUR,
  RETENTION 2 DAYS,
  GRACE PERIOD 10 MINUTES
)
GROUP BY title_id, season_id
EMIT CHANGES ;
```

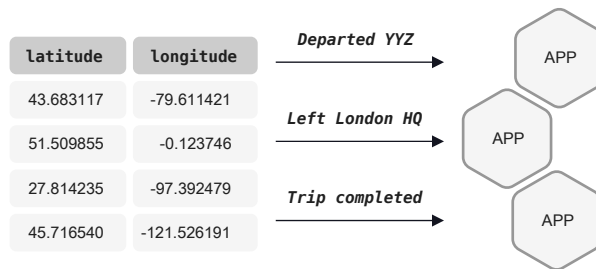
Потоковая обработка

Потоковая обработка — преобразование, фильтрация, объединение и агрегирование событий

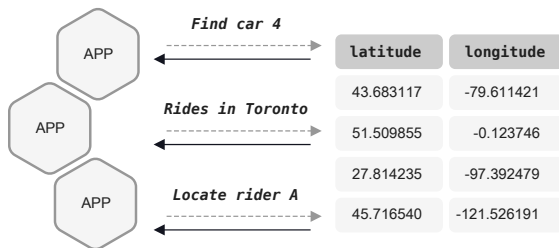
```
SELECT select_expr [, ...]  
FROM from_item  
[ LEFT JOIN коллекция_для_соединения ON критерий_соединения ]  
[ WINDOW оконное_выражение ]  
[ WHERE условие ]  
[ GROUP BY выражение_группировки ]  
[ PARTITION BY выражение_разделения ]  
[ HAVING условие_для_агрегированных_значений ]  
EMIT CHANGES  
[ LIMIT количество ];
```

Запросы

- **Persistent** – запросы на стороне сервера, которые выполняются бесконечно
- **Push** — подписка на обновления



- **Pull** – это результат "сейчас", как запрос к традиционной СУБД



Работа с запросами

- Вывод списка запросов

```
{ LIST | SHOW } QUERIES [EXTENDED];
```

- Получение описаний запросов

```
EXPLAIN { идентификатор_запроса | инструкция_запроса };
```

- Завершение запросов

```
TERMINATE { query_id | ALL };
```

Функции

- Вывод списка доступных функций

`SHOW FUNCTIONS;`

- Получение описаний функций

`DESCRIBE FUNCTION <идентификатор>;`

- Пользовательские функции:

- *Скалярные* – не имеют состояния и возвращают точно одно значение
- *Агрегатные* – имеют состояние и возвращают точно одно значение
- *Табличные* – не имеют состояния и возвращают ноль или более значений

<https://docs.ksqldb.io/en/latest/how-to-guides/create-a-user-defined-function/>

Соединения

Соединения (Join)

Соединения — объединения потоков событий в режиме реального времени

Type		INNER	LEFT OUTER	RIGHT OUTER	FULL OUTER
Stream-Stream	Windowed	Supported	Supported	Supported	Supported
Table-Table	Non-windowed	Supported	Supported	Supported	Supported
Stream-Table	Non-windowed	Supported	Supported	Not Supported	Not Supported



Примеры соединений

- Поток и таблица

```
CREATE STREAM pageviews_enriched AS
SELECT
  users.userid AS userid,
  pageid,
  regionid,
  gender
FROM pageviews
LEFT JOIN users ON pageviews.userid = users.userid
EMIT CHANGES;
```

- Два потока

```
CREATE STREAM shipped_orders AS
SELECT
  o.id as orderId,
  o.itemid as itemId,
  s.id as shipmentId,
  p.id as paymentId
FROM orders o
INNER JOIN payments p WITHIN 1 HOURS ON p.id = o.id
INNER JOIN shipments s WITHIN 2 HOURS ON s.id = o.id;
```



Требования к партиционированию

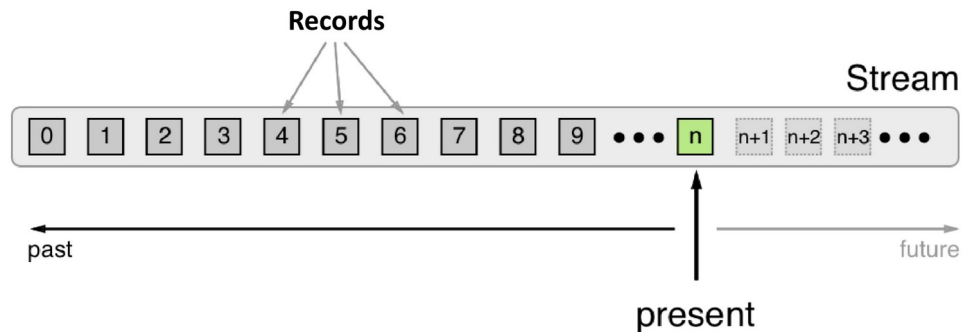
- События должны иметь одинаковый тип ключа
- Коллекции в соединении должны иметь одинаковое количество партиций
- Коллекции должны иметь одинаковую стратегию партиционирования

Время и окна

Семантика времени

- **Event-time** (*Время события*) – время, когда запись создается источником данных
- **Ingestion-time** (*Время приема*) – время, когда запись сохраняется в топике брокером
- **Processing-time** (*Время обработки*) – время, когда запись используется приложением потоковой обработки
- **Stream-time** (*Время потока*) – максимальная временная метка, видимая во всех обработанных записях на данный момент

The Stream



Назначение метки времени

Метка времени устанавливается либо производителем, либо брокером Kafka, в зависимости от параметра `message.timestamp.type`:

- **CreateTime**: Брокер использует временную метку записи, установленную производителем
- **LogAppendTime**: Брокер перезаписывает временную метку записи на местное время брокера при добавлении записи в топик

При создании коллекции можно использовать любое поле в качестве метки времени:

```
WITH(TIMESTAMP='some-field')
```

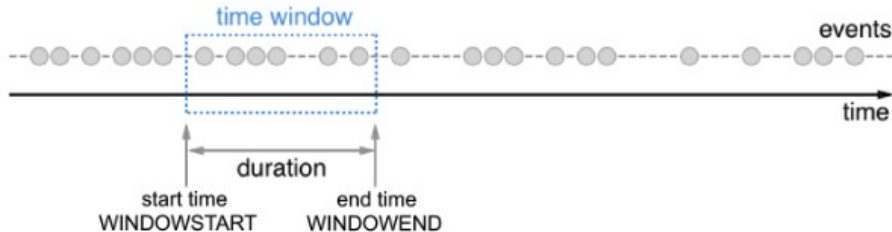
Оptionальное свойство `TIMESTAMP_FORMAT` описывает формат поля

Окна

Окно — интервал на линии времени

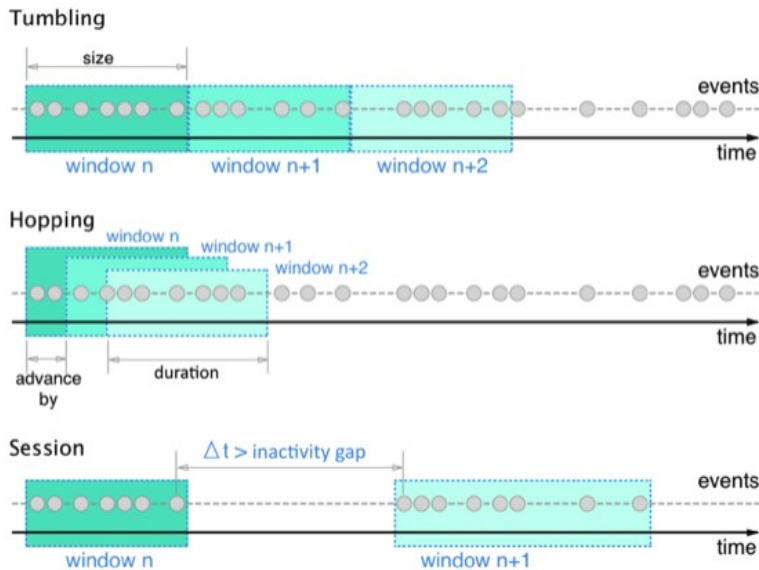
- WINDOWSTART — начало окна
- WINDOWEND — окончание окна

Использование окон позволяет определять, как группировать записи с одинаковым ключом для операций с отслеживанием состояния, таких как агрегирование или объединение, во временные интервалы

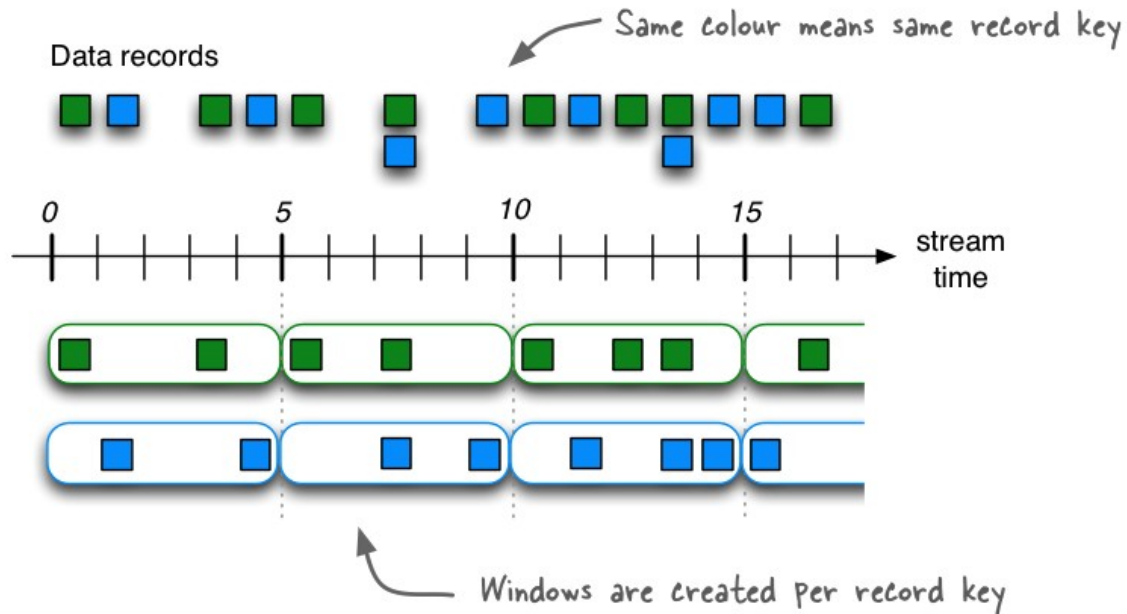


Типы окон

- **Tumbling** – окна фиксированной продолжительности, не перекрывающиеся
- **Hopping Window** – окна с фиксированной продолжительностью, перекрывающиеся
- **Session Window** – окна с динамическим размером, не перекрывающиеся, размер определяется данными



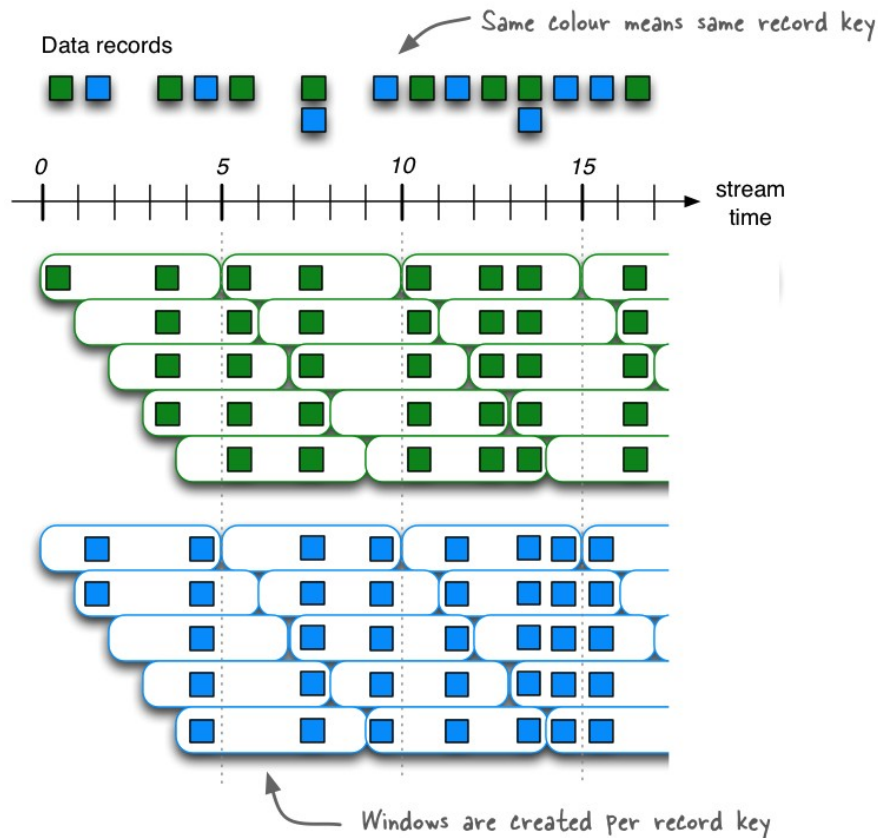
Tumbling window



```
SELECT card_number, count(*) FROM authorization_attempts
WINDOW TUMBLING (SIZE 5 SECONDS)
GROUP BY card_number HAVING COUNT(*) > 3
EMIT CHANGES;
```

Hopping Window

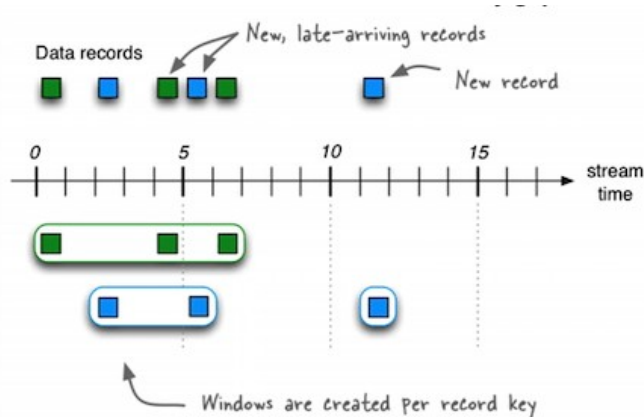
```
SELECT regionid, COUNT(*) FROM pageviews
WINDOW HOPPING (SIZE 30 SECONDS, ADVANCE BY 10 SECONDS)
WHERE UCASE(gender)='FEMALE' AND LCASE (regionid) LIKE '%_6'
GROUP BY regionid
EMIT CHANGES;
```



Session Window

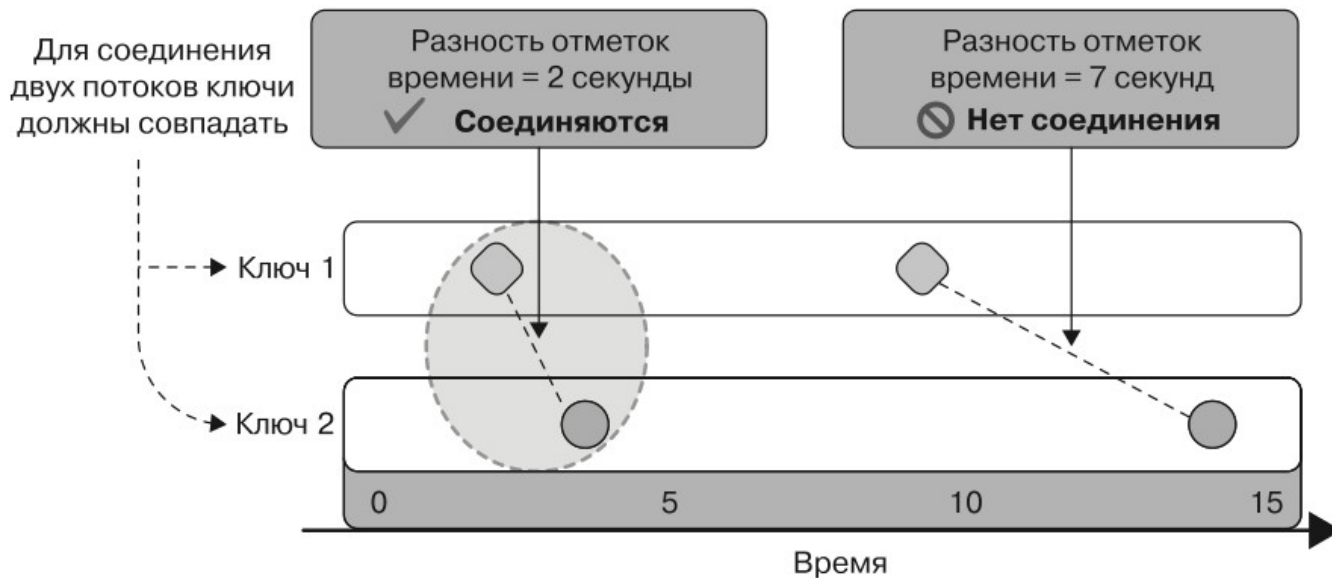
Сессионное окно объединяет записи в сеанс, который представляет период активности, разделенный указанным промежутком бездействия

- Сессионные окна отслеживаются независимо по ключам, поэтому окна с разными ключами обычно имеют разное время начала и окончания
- Продолжительность сессионных окон различается



```
SELECT regionid, COUNT(*) FROM pageviews  
WINDOW SESSION (60 SECONDS)  
GROUP BY regionid  
EMIT CHANGES;
```

Оконные соединения



Задержавшиеся события

Льготный период – дополнительное время для приёма «опоздавших» данных
По умолчанию 24 часа

```
SELECT order_zipcode, TOPK(order_total, 5) FROM orders
WINDOW TUMBLING (SIZE 1 HOUR, GRACE PERIOD 2 HOURS)
GROUP BY order_zipcode
EMIT CHANGES;
```

Сохранение окон

- Можно настроить количество окон в прошлом, которые сохраняются в ksqlDB
- Период хранения должен быть больше суммы размера окна и льготного периода

```
CREATE TABLE pageviews_per_region AS
  SELECT regionid, COUNT(*) FROM pageviews
  WINDOW HOPPING (SIZE 30 SECONDS, ADVANCE BY 10 SECONDS,
                  RETENTION 7 DAYS, GRACE PERIOD 30 MINUTES)
  WHERE UCASE(gender)='FEMALE' AND LCASE (regionid) LIKE '%_6'
  GROUP BY regionid
  EMIT CHANGES;
```

Пример

Пример 1

- `docker-compose up -d`
- `docker exec broker kafka-topics --create --topic users --bootstrap-server localhost:9092`
- `docker exec -ti ksqldb-cli ksql http://ksqldb-server:8088`
 - `SET 'auto.offset.reset' = 'earliest';`
 - `SHOW TOPICS;`
 - `CREATE STREAM users (ROWKEY INT KEY, USERNAME VARCHAR) WITH (KAFKA_TOPIC='users', VALUE_FORMAT='JSON');`
 - `INSERT INTO users (username) VALUES ('izzy');`
 - `INSERT INTO users (username) VALUES ('elyse');`
 - `INSERT INTO users (username) VALUES ('mitch');`
 - `SELECT 'Hello, ' + USERNAME AS GREETING FROM users EMIT CHANGES;`

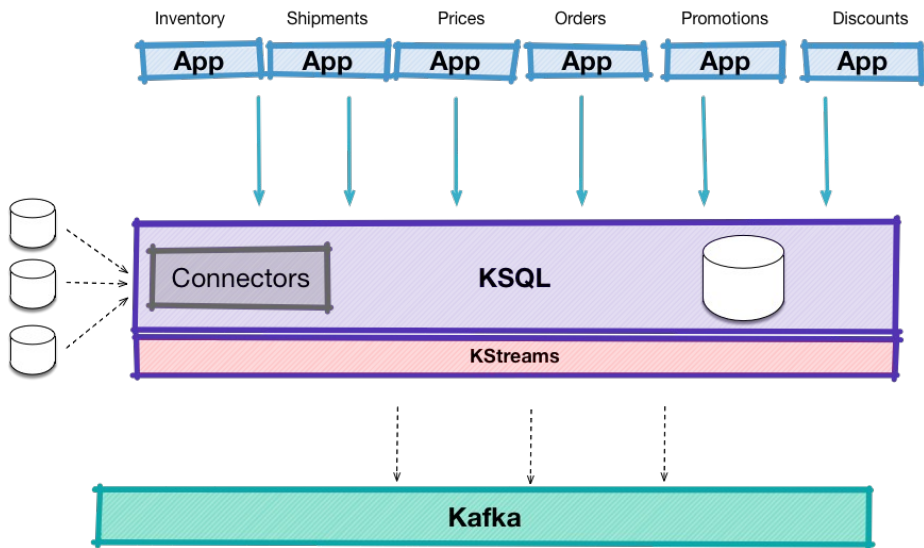
LIVE

Интеграция данных

Connectors

ksqlDB предоставляет возможности для управления и интеграции с Connect:

- Создание коннекторов
- Описание коннекторов
- Импорт топиков, созданных с помощью Connect в ksqlDB

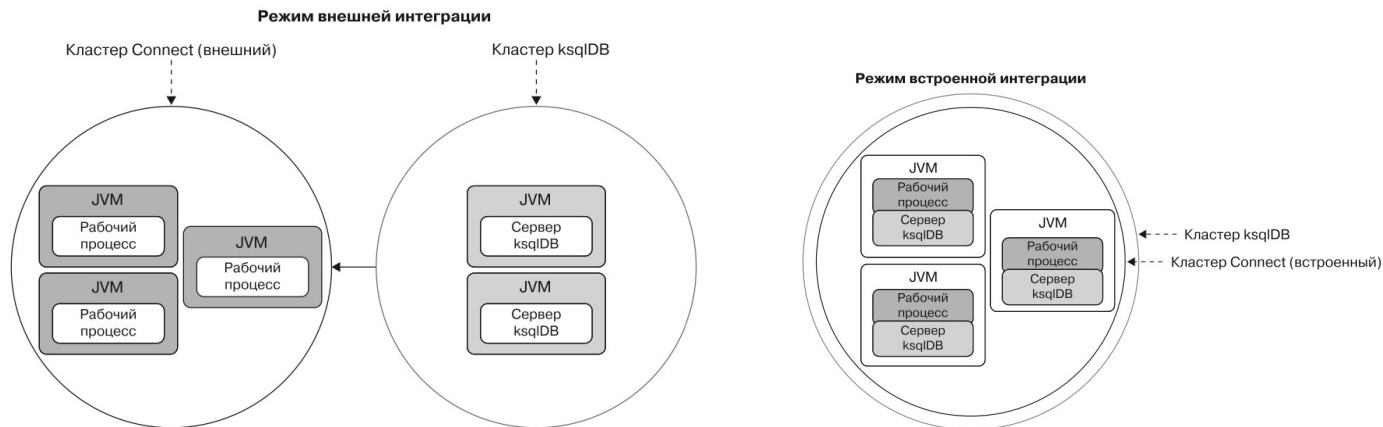


Режимы интеграции ksqlDB с Connect

Интеграция с Kafka Connect в ksqlDB может работать в двух разных режимах:

- **External (Внешняя):** подключение к существующему кластеру Connect `ksql.connect.url=http://localhost:8083`
- **Embedded (Встроенная):** рабочий процесс Kafka Connect выполняется под управлением той же JVM, что и сервер ksqlDB

`ksql.connect.worker.config=/etc/ksqldb-server/connect.properties`



API для коннекторов

- CREATE SOURCE | SINK CONNECTOR [IF NOT EXISTS] connector_name WITH(property_name = expression [, ...]);

```
CREATE SOURCE CONNECTOR `jdbc-connector` WITH(  
  "connector.class"='io.confluent.connect.jdbc.JdbcSourceConnector',  
  "connection.url"='jdbc:postgresql://localhost:5432/my.db',  
  "mode"='bulk',  
  "topic.prefix"='jdbc-',  
  "table.whitelist"='users',  
  "key"='username');
```

- DESCRIBE CONNECTOR connector_name;
- DROP CONNECTOR [IF EXISTS] connector_name;
- SHOW | LIST [SOURCE | SINK] CONNECTORS;

Пример

Пример 2

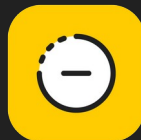
- `docker-compose up -d`
- `docker exec -ti postgres psql -U postgres`
 - `CREATE TABLE titles (id SERIAL PRIMARY KEY, title VARCHAR(120));`
 - `INSERT INTO titles (title) values ('Stranger Things');`
 - `INSERT INTO titles (title) values ('Black Mirror');`
 - `INSERT INTO titles (title) values ('The Office');`
- `docker exec -ti ksqldb-cli ksql http://ksqldb-server:8088`
 - `CREATE SOURCE CONNECTOR `postgres-source` WITH (...)`
 - `SHOW CONNECTORS;`
 - `DESCRIBE CONNECTOR `postgres-source`;`
 - `SHOW TOPICS;`
 - `PRINT `postgres.titles` FROM BEGINNING;`

LIVE

Вопросы?



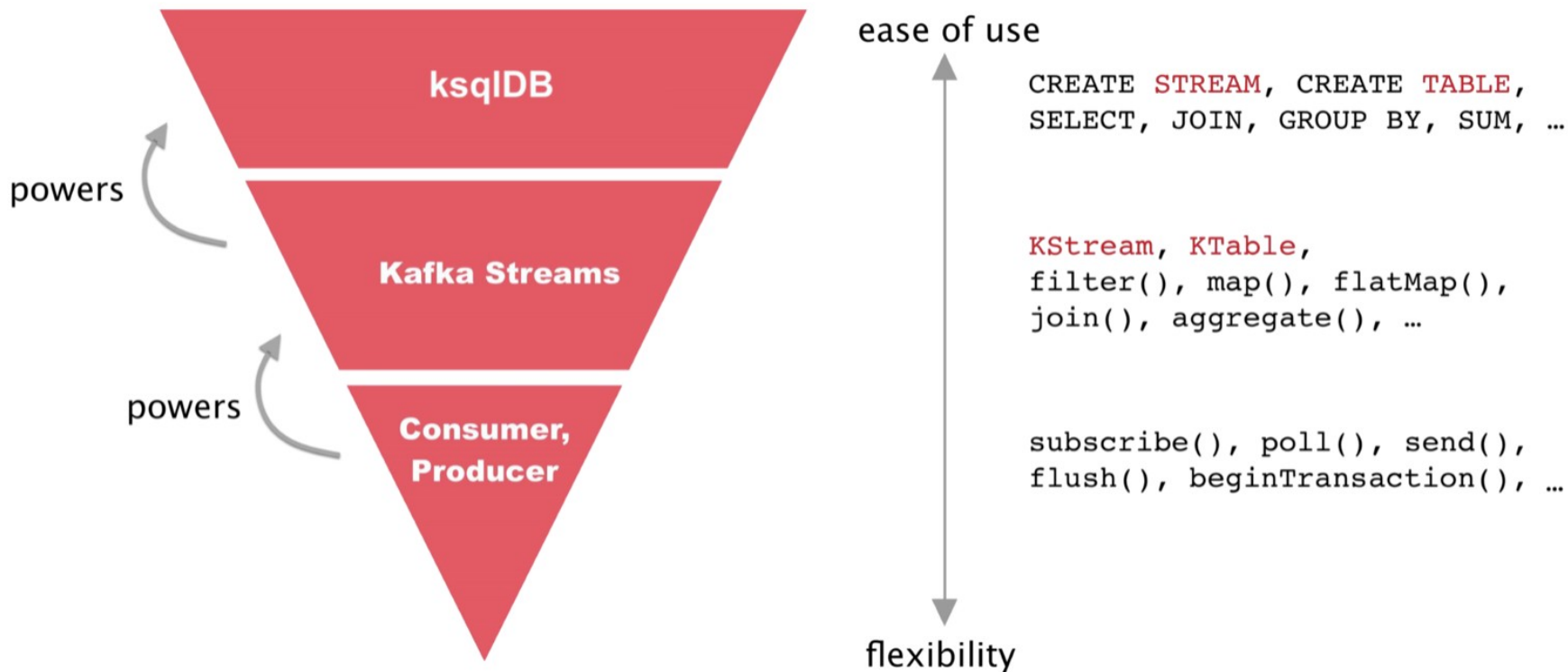
Ставим "+",
если вопросы есть



Ставим "-",
если вопросов нет

Итого

KsqlDB vs Kafka Streams vs API



Преимущества ksqlDB

- Интерактивность
- Улучшенная поддержка анализа данных
- Меньший объём кода
- Более низкий порог входа
- Упрощённая архитектура

Литература

Список материалов для изучения

1. Kafka Streams и ksqlDB: данные в реальном времени
2. <https://ksqldb.io>
3. <https://docs.confluent.io/platform/current/ksqldb/index.html>
4. <https://github.com/confluentinc/ksql>

Рефлексия

Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то, что узнали на вебинаре?



Следующий вебинар



Выбор темы и организация проектной работы



Ссылка на вебинар
будет в ЛК за 15 минут



Материалы
к занятию в ЛК —
можно изучать



Обязательный материал
обозначен красной
лентой



**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Спасибо за внимание!

Приходите на следующие вебинары



Заигрин Вадим

Ведущий эксперт по технологиям, Сбербанк

vzaigrin@yandex.ru

<https://t.me/vzaigrin>

