glowbyte

data sapience

Streaming Data Integration — ETL-инструмент для создания near realtime-процессов

Василий Мельник
Ведущий архитектор, Product Owner
SDI Data Sapience

13 лет в IT
11 лет в разработке и проектировании DWH
С 2019 развиваю направление Near Realtime

Команда

10+ разработчиков и аналитиков с опытом разработки Realtime-решений для задач маркетинга, кредитных рисков, online-отчетности

Основное направление деятельности — финтех

Технический стек: Kafka, Kafka Streams/KSQL, Flink, Spark, Hbase, Hive, Impala, HDFS, Spring Boot

Более 10 Near Realtime проектов с 2019 года

План доклада

- Определение NRT
- Практический опыт
- Подход к продукту
- Заключение

ОПРЕДЕЛЕНИЕ NRT

Что мы понимаем под NRT?

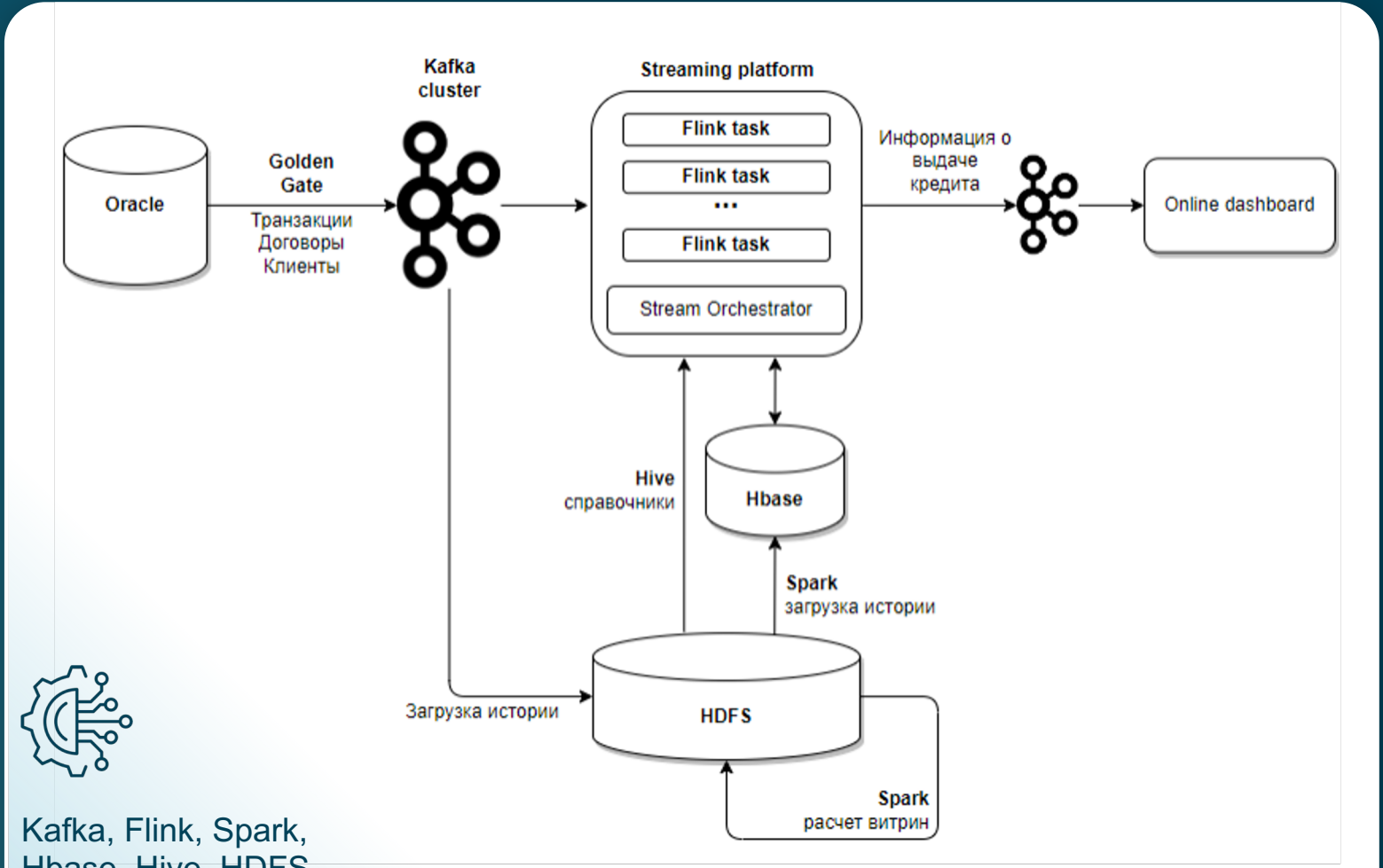
Банк из Топ-10 РФ

Наполнение online-дашбордов по кредитным выдачам

Сигнал - OGG message
Событие - информация о кредитной выдаче
Реакция - отображение на dashboard

Более 20 аналитических разрезов

~60 сообщений в секунду
Latency до 30 секунд



Kafka, Flink, Spark, Hbase, Hive, HDFS

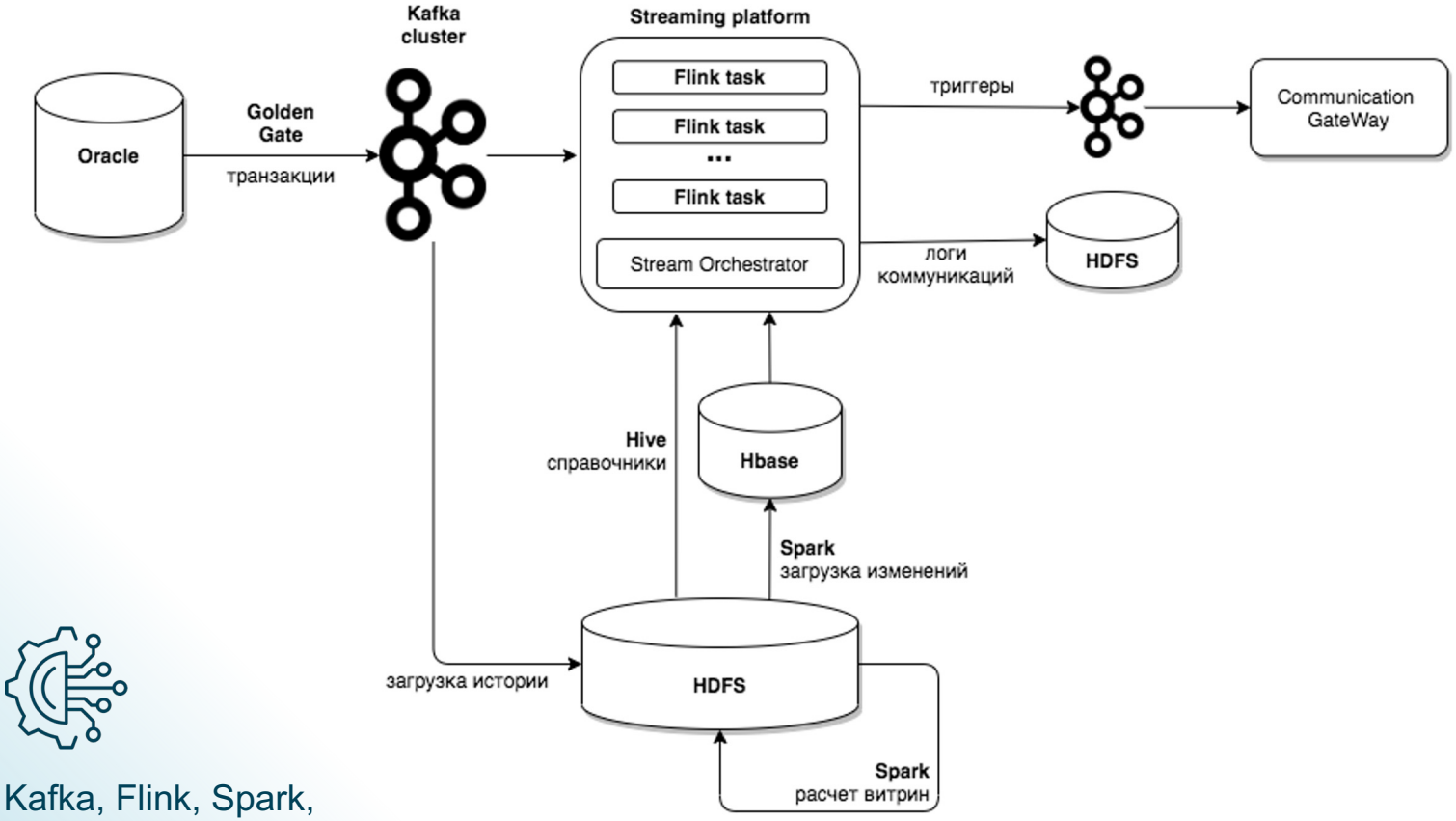
Банк из Топ-10 РФ

Персонализированная продажа страховых продуктов на основе карточных транзакций

Сигнал - OGG message
Событие - превышение пороговой суммы за период
Реакция - отправка предложения по страховому продукту

Более 600 показателей

~1000 сообщений в секунду Latency 1-2 секунд



Kafka, Flink, Spark, Hbase, Hive, HDFS

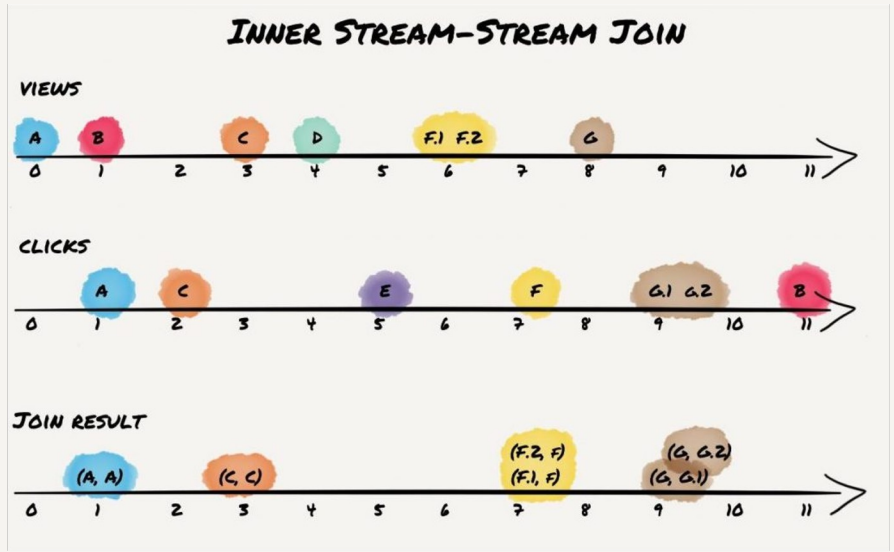
ПРАКТИЧЕСКИЙ ОПЫТ

NRT — это сложно аналитически

ЦЕЛЬ —
КАКУЮ ПОЛЬЗУ ПРИНЕСЕТ REALTIME?

УПОР НА СОБЫТИЯ,
А НЕ ДАННЫЕ

ДРУГАЯ ЛОГИКА ОПИСАНИЯ —
НЕ СТАТИЧНЫЕ ДАННЫЕ, А ПОТОКИ



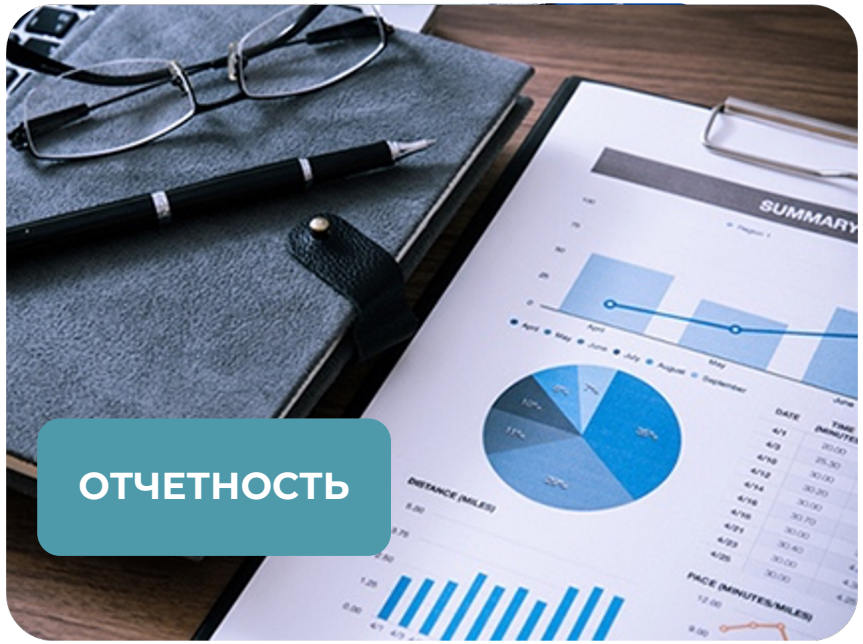
Аналитический NRT как замена “классической” отчетности

ЗАПРОС:

- “Мы получаем отчетность (обязательную/управленческую) через 3 недели после закрытия периода”
- “Давайте пересчитывать отчетность в реальном времени при появлении каждой проводки/договора/etc ”

ОТВЕТ:

Отчетность — не NRT-процесс



Модные миллисекунды

Примеры задач, где требуется секундные задержки:

- Интеграции с фронтом, когда важно выдать клиенту реакцию до того, как он покинул приложение
- Кредитный конвейер в задачах real-time одобрения кредитов
- Real-time AML, когда важно заблокировать подозрительные операции

Не забывать об окружении

- Рассматривая локальное окружение, легко потерять видение всего тракта данных
- Очень может оказаться, что за границами Kafka лежат batch-процессы, не готовые принимать или отдавать real-time
- Источники и приемники должны быть real-time на уровне своего бизнес-процесса

Почему выбрали Flink

Масштабируемость, отказоустойчивость, savepoint “из коробки”

Мощный API для расширения функционала

Поддерживает YARN и Kubernetes

В отличие от KStreams, не привязан к другим инфраструктурным компонентам

“True streaming” Framework в отличие от Spark Streaming

✓

Spark

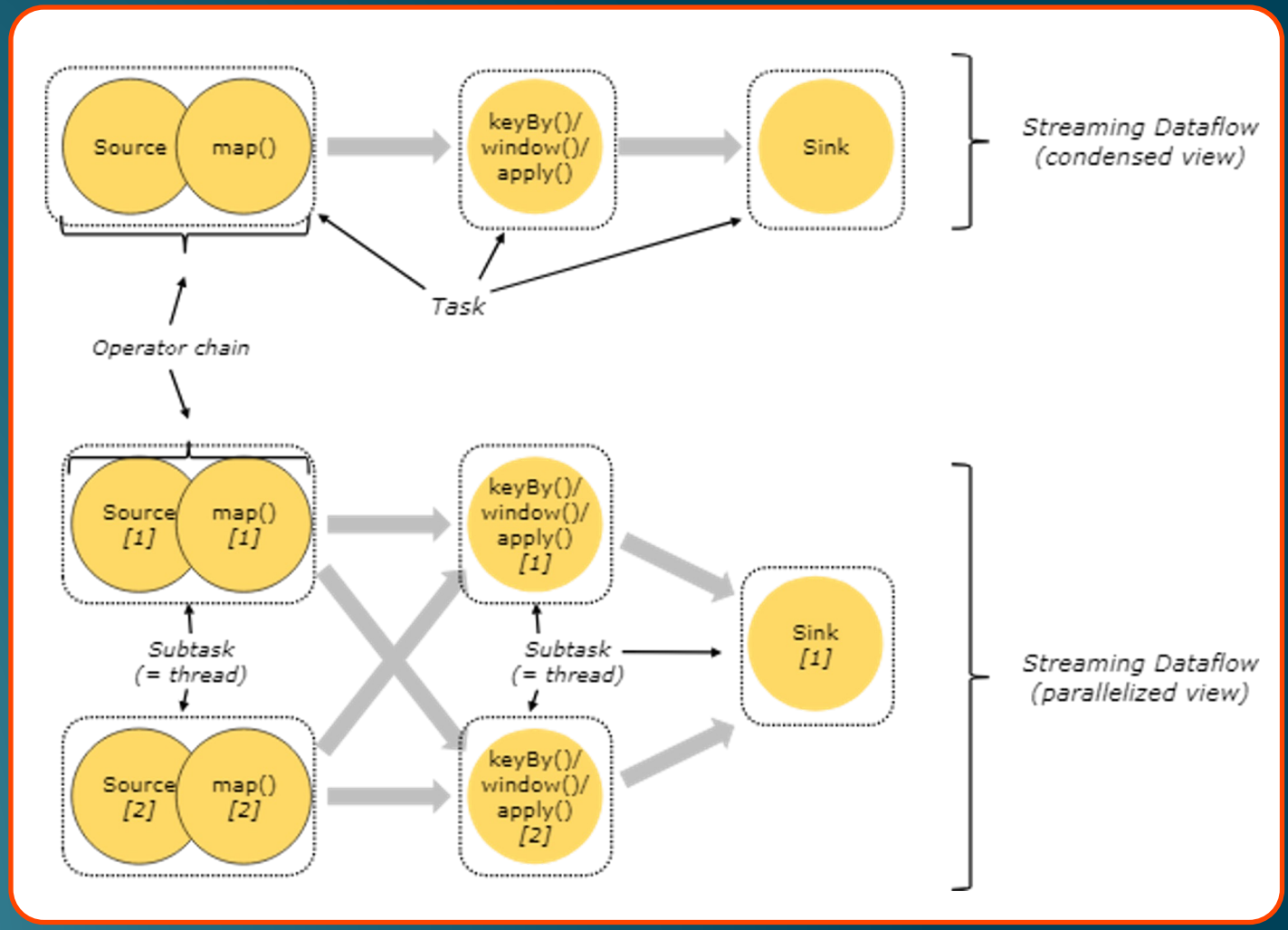
✗

STREAMS

✗

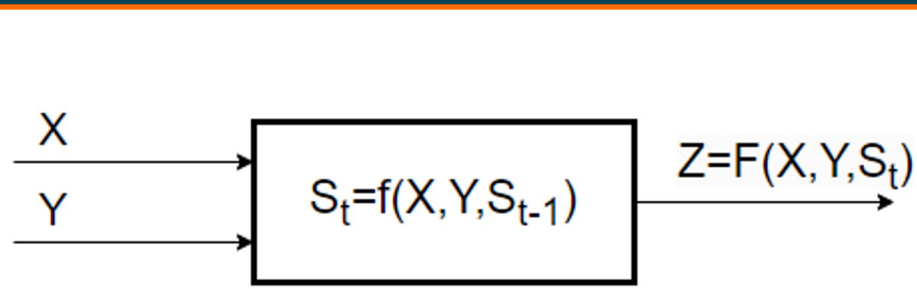
Немного об Apache Flink

- **Flink** — программный фреймворк для создания горизонтально масштабируемых отказоустойчивых приложений
- Можно разворачивать на платформах управления кластерами: **YARN, Kubernetes**
- **Каждое приложение** — ациклический ориентированный граф (DAG) от источников к приемникам через набор трансформаций
- Flink поддерживает **асинхронный подход** к обработке данных



Еще немного об Apache Flink

- Во Flink есть много интерфейсов для определения трансформаций: map, flatMap, окна и пр. пр.
- Но наиболее обще любую трансформацию можно представить в виде автомата с внутренним состоянием.
- Flink использует Heap и RocksDB для хранения состояния.




```
private ValueState<String> saggState;

@Override

public void processElement(String integer, Context context, Collector<String> collector) throws Exception {

    String sagg_val = saggState.value();

    if (sagg_val == null) sagg_val = "Start:";

    sagg_val +="_"+integer;

    saggState.update(sagg_val);

    collector.collect( record: (integer)+"_test "+sagg_val);

}
```

И еще чуть-чуть об Apache Flink

- DAG описывается с помощью DSL
- У каждой трансформации есть выходная схема, используемая для сериализации данных между трансформациями
- Для представления сообщений могут использоваться примитивные типы, POJO-классы или класс Row

```
env.addSource(function, sourceName: "Collection Source", TypeInformation.of(Integer.class) ).setParallelism(1)
        .keyBy(x -> x%12) KeyedStream<Integer, Integer>
        .process(new MyProcessFunction2()).setParallelism(1).startNewChain() SingleOutputStreamOperator<String>
        .keyBy(x-> 3) KeyedStream<String, Integer>
        .process(new MyProcessFunction3()) SingleOutputStreamOperator<String>
        .print();
env.execute();
```

NRT — это сложно технически

ВНЕШНИЕ СЕРВИСЫ

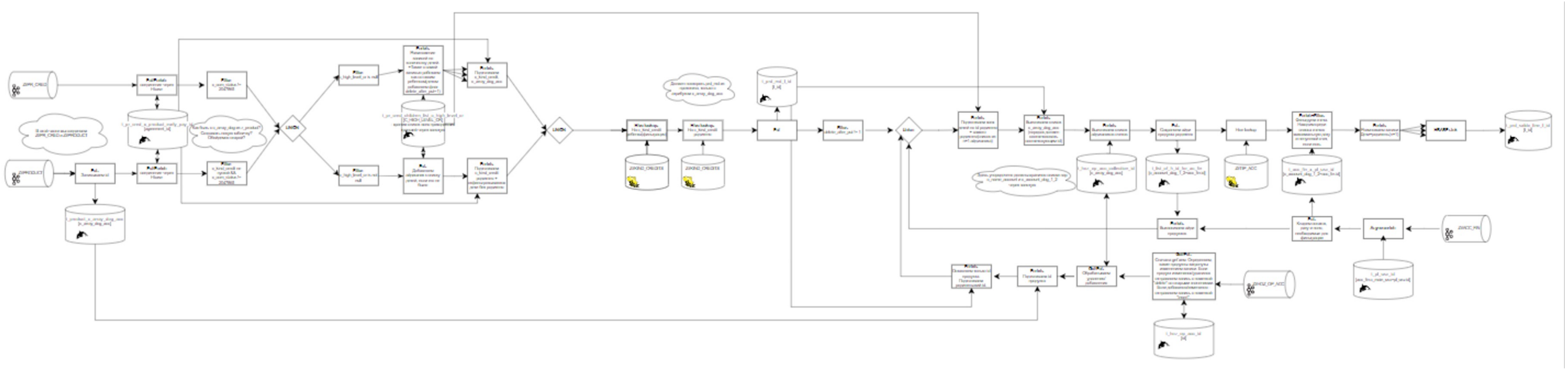
ГАРАНТИИ ДОСТАВКИ

ЗАДЕРЖКИ В ОКРУЖЕНИИ

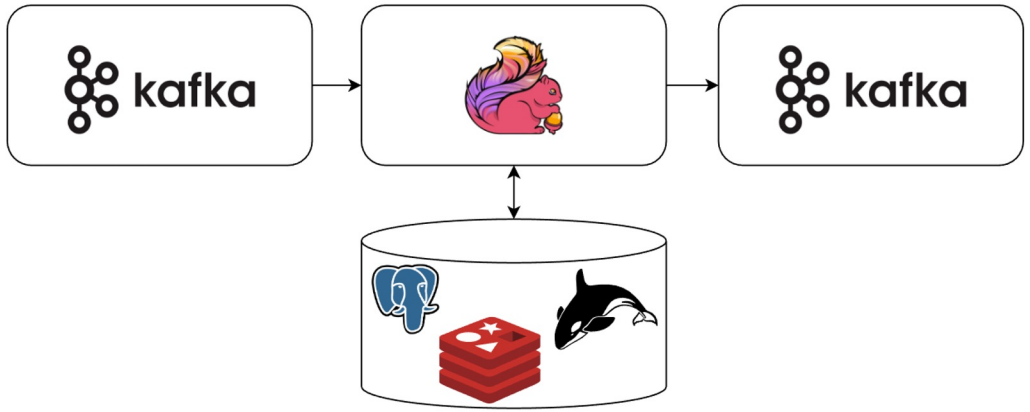
ВНУТРЕННЕЕ СОСТОЯНИЕ ПРОЦЕССА

- РАЗМЕР
- ЧАСТОТА ОБНОВЛЕНИЯ
- ИНИЦИАЛИЗАЦИЯ

+ ВСЕ РЕАЛИЗОВАТЬ



Использование внешней СУБД в качестве state



Внутренний стейт Flink	Внешний стейт
Производительность	Производительность зависит
Горизонтальная масштабируемость	Горизонтальная масштабируемость зависит
Периодические снепшоты	Нет снепшотов
Нельзя(почти) обратиться извне	Легко обратиться извне
Трудно инициализировать	Легко инициализировать
Неразделяемый между процессами	Разделяемый между процессами

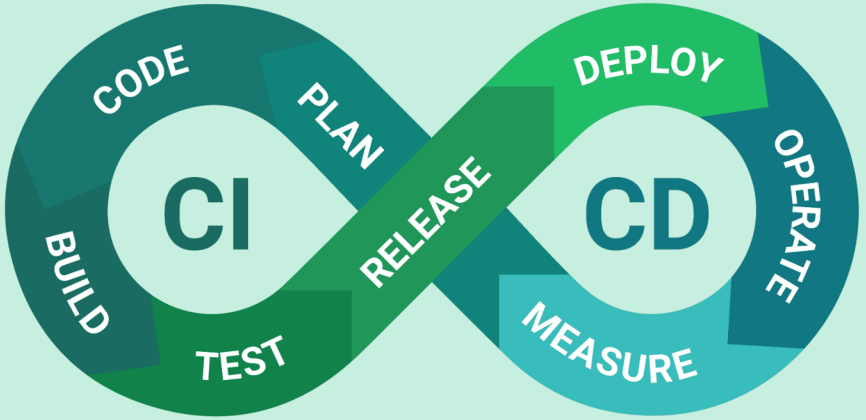
«Забытый» функционал

Тестирование

- Вход/выход
- Промежуточные интеграции

CI/CD

- Обновление версии процесса
- Обновление инфраструктурного ПО



ПОДХОД К ПРОДУКТУ

Чем может и чем не может помочь ЛЮБОЙ продукт?

- Продукт не решит аналитические задачи, потому что они слишком многообразны
- Продукт не продумает за вас весь тракт данных, потому что вариантов поставки и хранения данных слишком много

НО

- **Продукт может облегчить разработку типовых процессов за счет low-code**
- **Продукт может дать интеграции, которых нет в OpenSource-фреймворках**
- **Продукт поможет в работе с внешним state**
- **Продукт обеспечит продуктивизацию(!!!!) — логирование, аутентификация/авторизация, CI/CD**
- **Продукт может упростить тестирование**

Вместо кода — текстовая конфигурация

```java
private JdbcSourceReader getJdbcSourceReader() {
    if (jdbcSourceReader == null) {
        final OperatorConfig operatorConfig = getOperatorConfig();

        final String guidFieldName = getEnvironment().isGuidRedirected()
                ? getEnvironment().getGuidFieldName()
                : null;

        final RowTypeInfo inputTypeInfo = super.getTypeInfo();
        final Boundedness boundedness = getBoundedness();

        if (operatorConfig.get(OptionalParameter.JDBC_SOURCE_INCREMENT_FIELD) != null) {
            this.jdbcSourceReader = new JdbcSourceIncReader<>(
                    operatorConfig,
                    inputTypeInfo,
                    guidFieldName,
                    boundedness
            );
        } else {
            this.jdbcSourceReader = new JdbcSourceReader(
```

```yaml
pipeline:
  sources: # источники
    - name: document # имя оператора
      type: kafka # тип источника
      schema: 'storage/schemas/document.avsc' # схема валидации входных да
      next: # список следующих операторов в pipeline
        - doc_x_balance
      config: # детальная конфигурация
        bootstrap_servers: 'str-pltf-kafka-1:9092,str-pltf-kafka-2:9092'
        topic: 'document'
    - name: balance
      type: kafka
      schema: 'storage/schemas/balance.avsc'
      next:
        - doc_x_balance
      config:
        bootstrap_servers: 'str-pltf-kafka-1:9092,str-pltf-kafka-2:9092'
        topic: 'balance'
  transforms: # трансформации
    - name: doc_x_balance
      type: join
      next:
        - doc_x_balace_enriched
        - join_sink
      config:
```

Каждому оператору в конфигурации соответствует класс одного из 5-ти супертипов:

- Source
- Sink
- OneInputTransformation
- TwoInputTransformation
- MultiInputTransformation

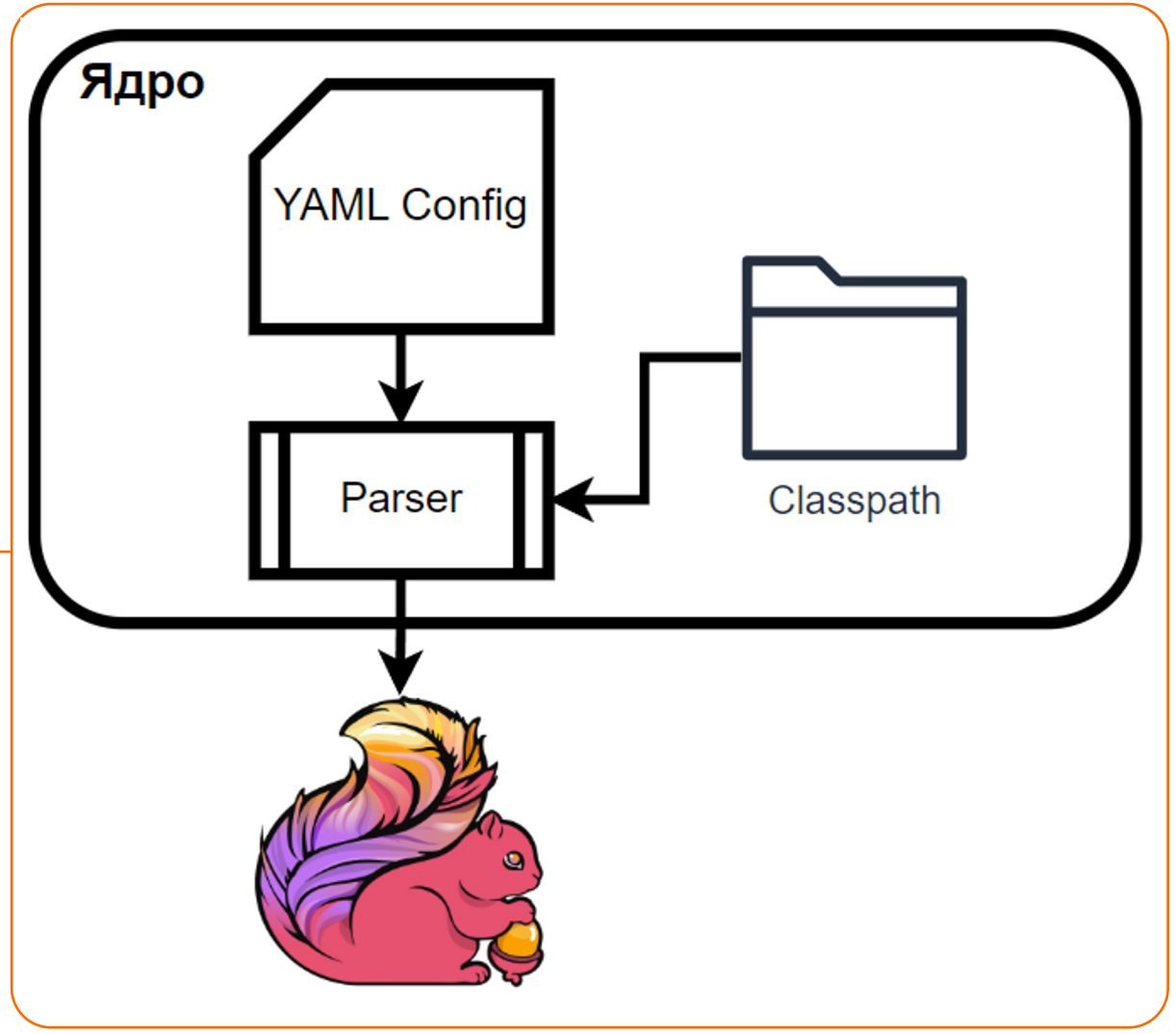
Каждый оператор описывает свой набор параметров и содержит одно или несколько Flink-преобразований:

```
@RequiredArgsConstructor
@Accessors(fluent = true)
@Getter
public enum RequiredParameter implements OperatorConfig.RequiredParameter {
    3 usages
    ZOOKEEPER_QUORUM( parameterName: "hbase.zookeeper_quorum", String.class, description: "Zookeeper quorum"),
    TABLE( parameterName: "hbase.table", String.class, description: "HBase table name"),
    // Список column family. Структура column family описана в ColumnFamilyStructure.
    3 usages
    COLUMN_FAMILIES( parameterName: "hbase.column_families", List.class, description: "List of column families. " +
            "Column family structure should contain `name` (required), `alias` (optional) and `qualifiers` (required). " +
            "Parameter `qualifiers`, in turn, should contain list of qualifier structure, that should contain `name` (required) and `type`
```

Зачем такие сложности?

Не всегда можно реализовать логику оператора одной Flink-функцией, например:

- скользящие окна в реальном времени требуют генерации “отменяющих сообщений”
- во многих операторах требуется задать ключ дистрибуции.



Парсер конфигурации становится тривиальным и не меняется при расширении набора трансформаций, т.к. вся информация об именовании операторов и наборе параметров содержится в Java-коде и подтягивается в райнтайме

Промежуточная сериализация

- Используем Row для сериализации промежуточных сообщений
- Эффективнее сериализуется по сравнению с POJO
- Удобен для SQL API
- Требует **явного** указания схемы для эффективной сериализации

```
static void prepareDataStream(String name, RowTypeInfo typeInfo, DataStream<Row> dataStream) {
    final Transformation<Row> transformation = dataStream.getTransformation();
    try {
        transformation.setOutputType(typeInfo);
    } catch (IllegalStateException ex) {
        if (!typeInfo.schemaEquals(dataStream.getType())) {
            throw new CoreRuntimeException("Inconsistent type information for stream `" + name + "`.");
        }
    }
    prepareDataStream(name, dataStream);
}
```

Unit-тестирование

- Предназначено для тестирования операторов на этапе разработки
- Тестирование производится путем сравнения фактического и ожидаемого выходного результата на заданном наборе входных значений
- Используются test-утилиты Flink для создания локального кластера
- Потребовалось реализовать возможность запуска как в BOUNDED, так и в UNBOUNDED-режимах, т.к. для части операторов важно дождаться корректного срабатывания окна и таймеров

```
private static void delayedJobCancel(JobClient jobClient, Time.TimeValue delay) {

    try {
        Thread.sleep(delay.millis());
        final JobStatus jobStatus = jobClient.getJobStatus().get();
        if (!jobStatus.isGloballyTerminalState()) {
            jobClient.cancel().get();
            return; // Job finished with cancellation means, there were no errors.
        }
    } catch (InterruptedException cause) {
        throw new RuntimeException("Failed while waiting job cancellation.", cause);
    } catch (ExecutionException cause) {
        throw new RuntimeException("Failed to cancel job.", cause);
    }
```

Маленький список операторов

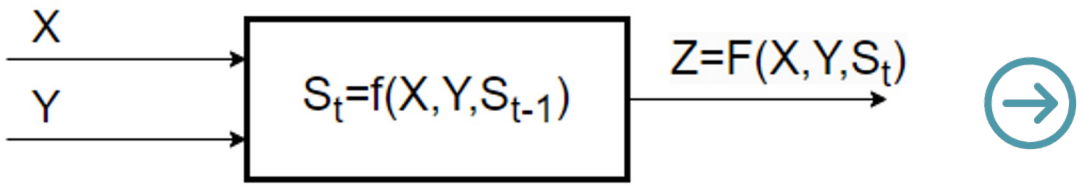
Скользящее окно	«Сборка» сообщения из нескольких потоков	Оператор Complex Event Processing (CEP)
Оператор обновления внешнего состояния	Groovy- и Python-скрипты	Hbase / Tarantool / Redis lookup
Lookup на периодически обновляющийся справочник	Скалярное преобразование	Фильтр

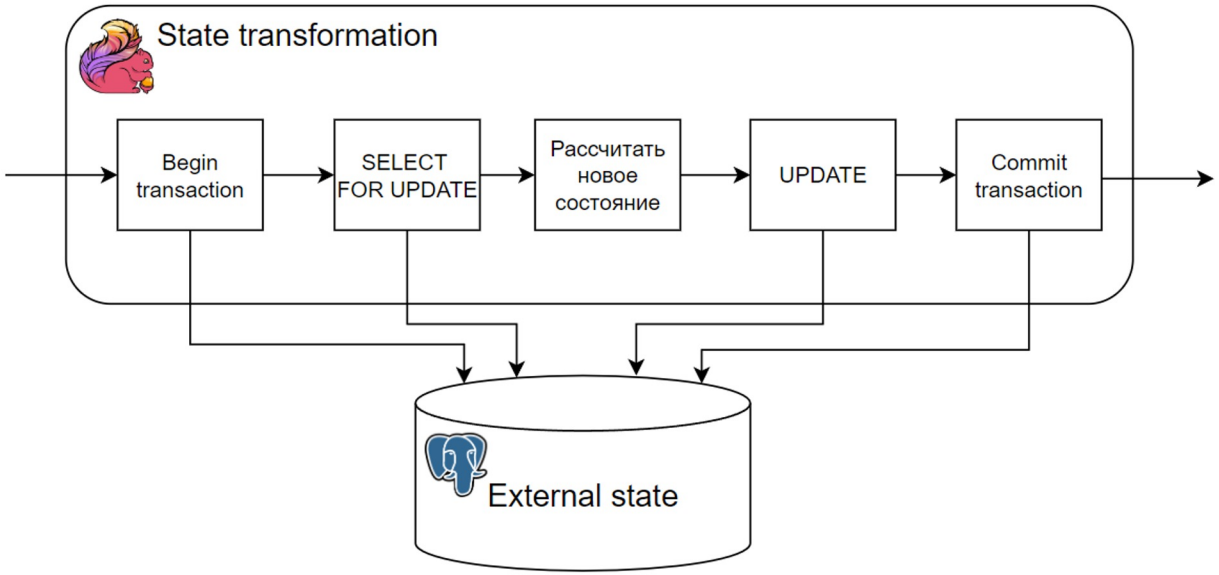
Что придумали с внешним стейтом?

01 Реализовали автомат на «новых физических принципах»

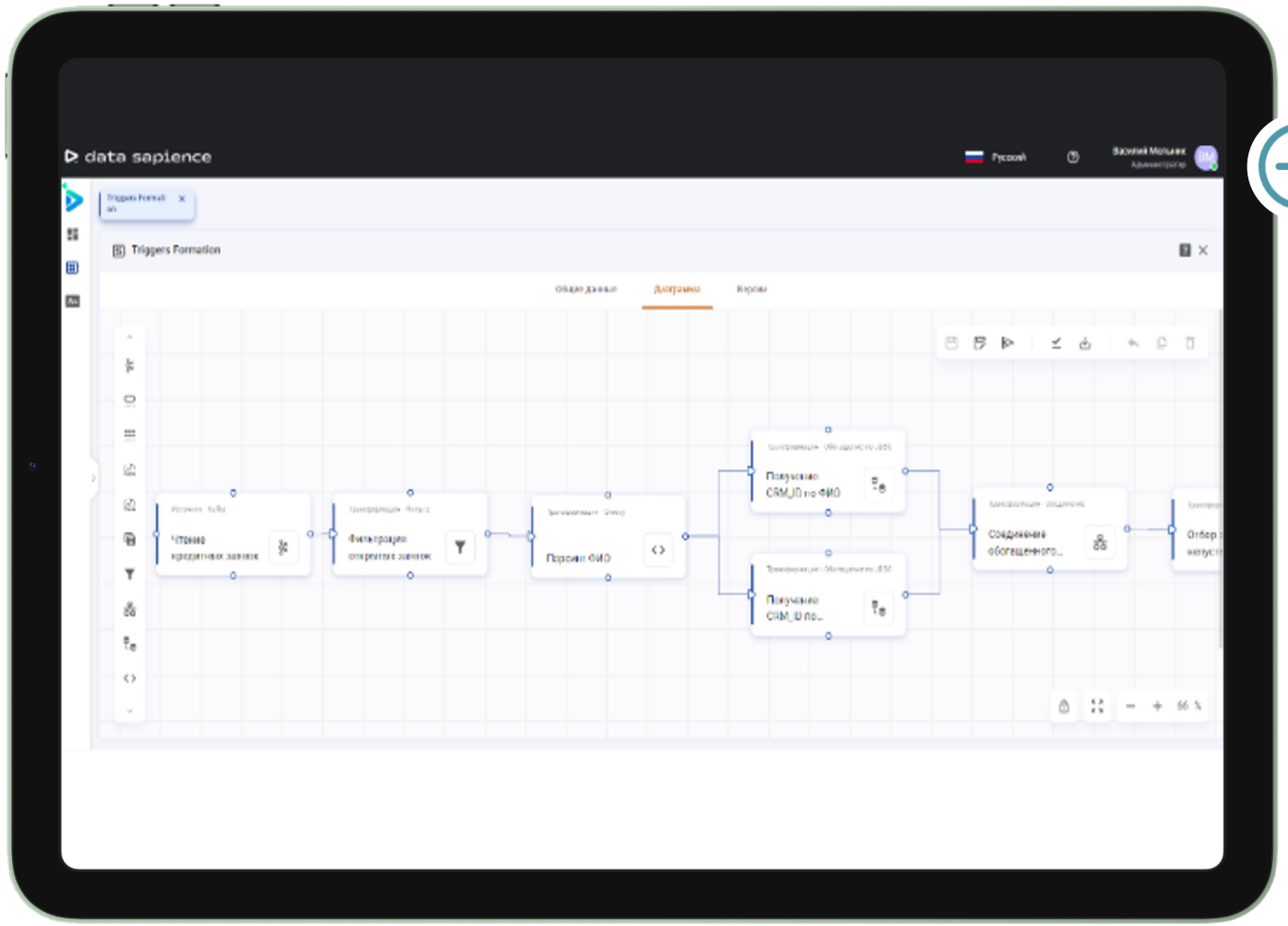
02 Чтобы обеспечить преимущества внешнего стейта и избежать аномалий сериализации используем транзакционные механизмы RDBMS

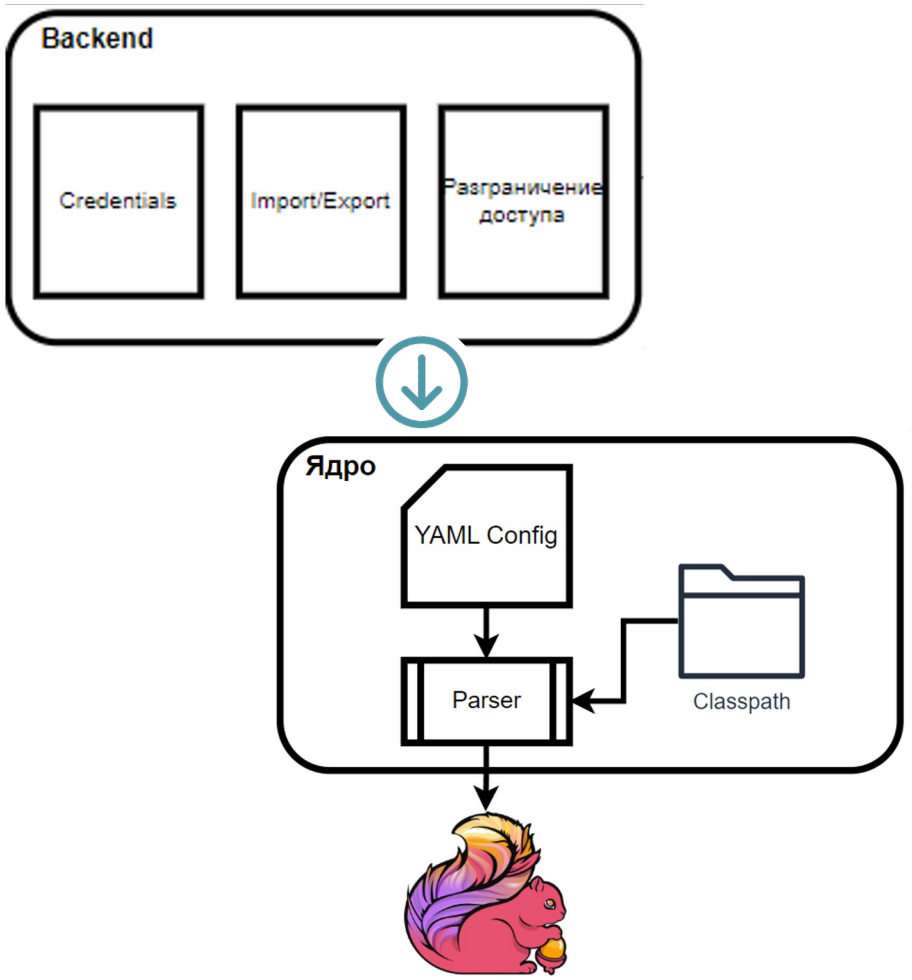
03 СУБД — узкое место, рассматриваем возможности перехода на горизонтально масштабируемые решения с транзакционными гарантиями



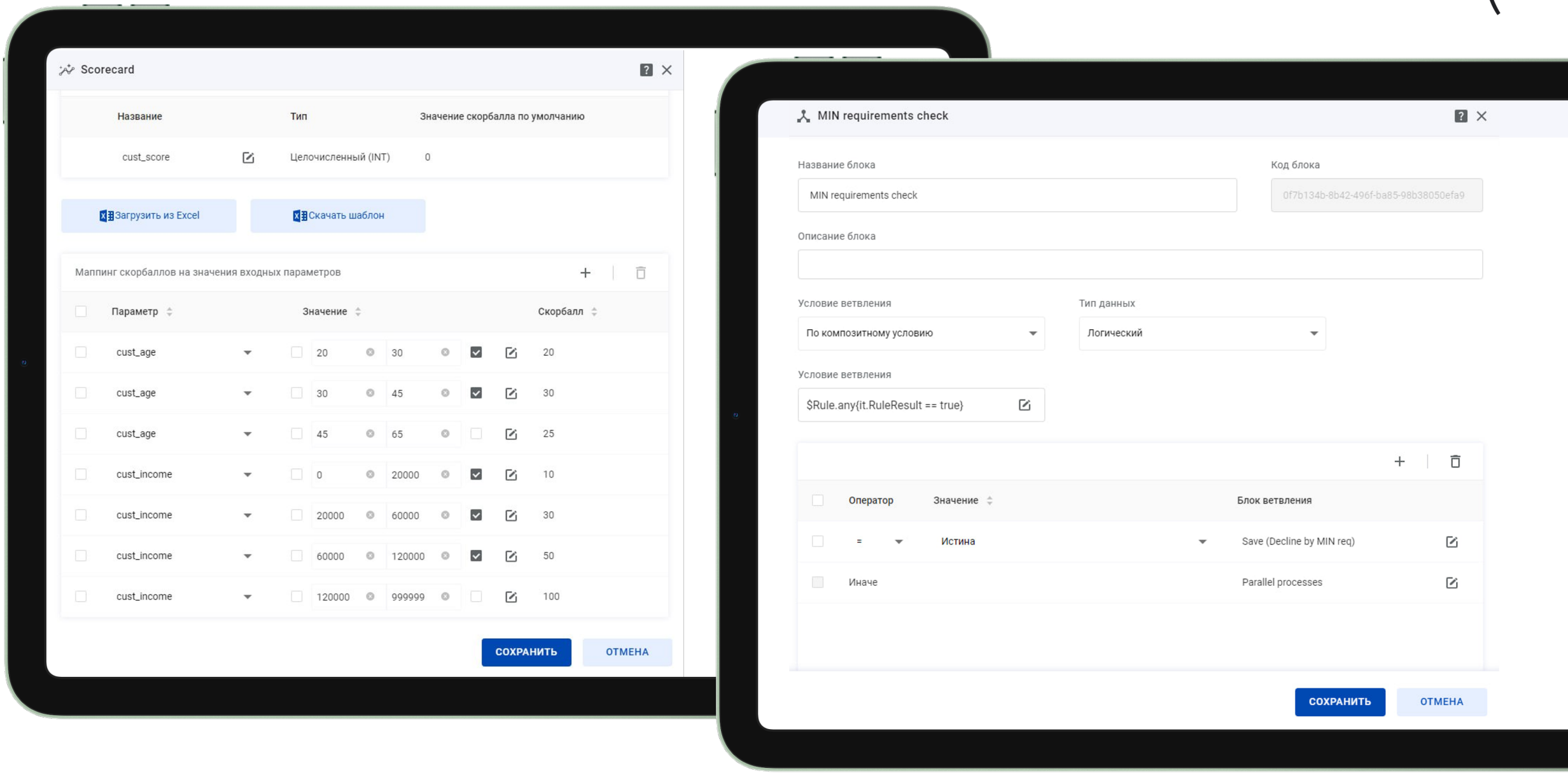


А как работать пользователю?



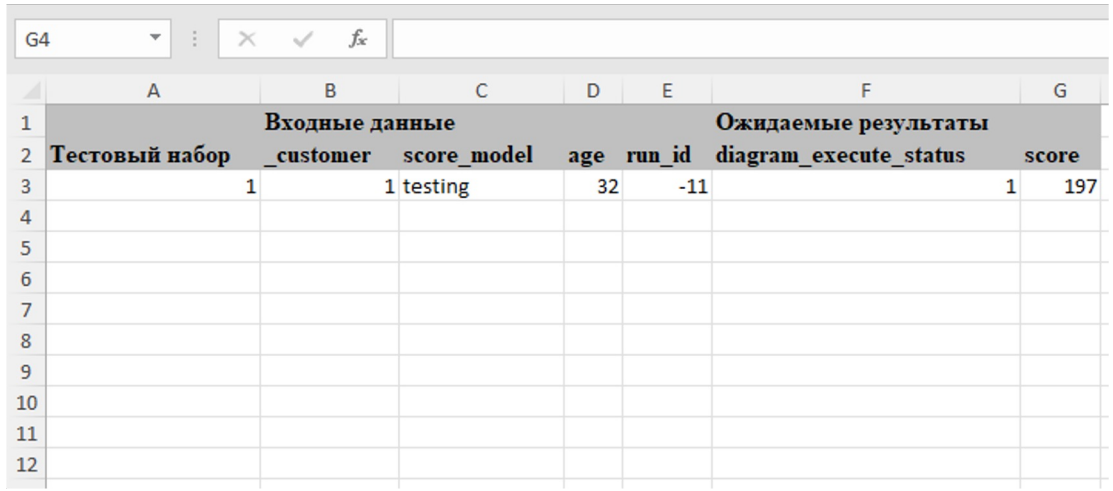


Пользователь может больше!

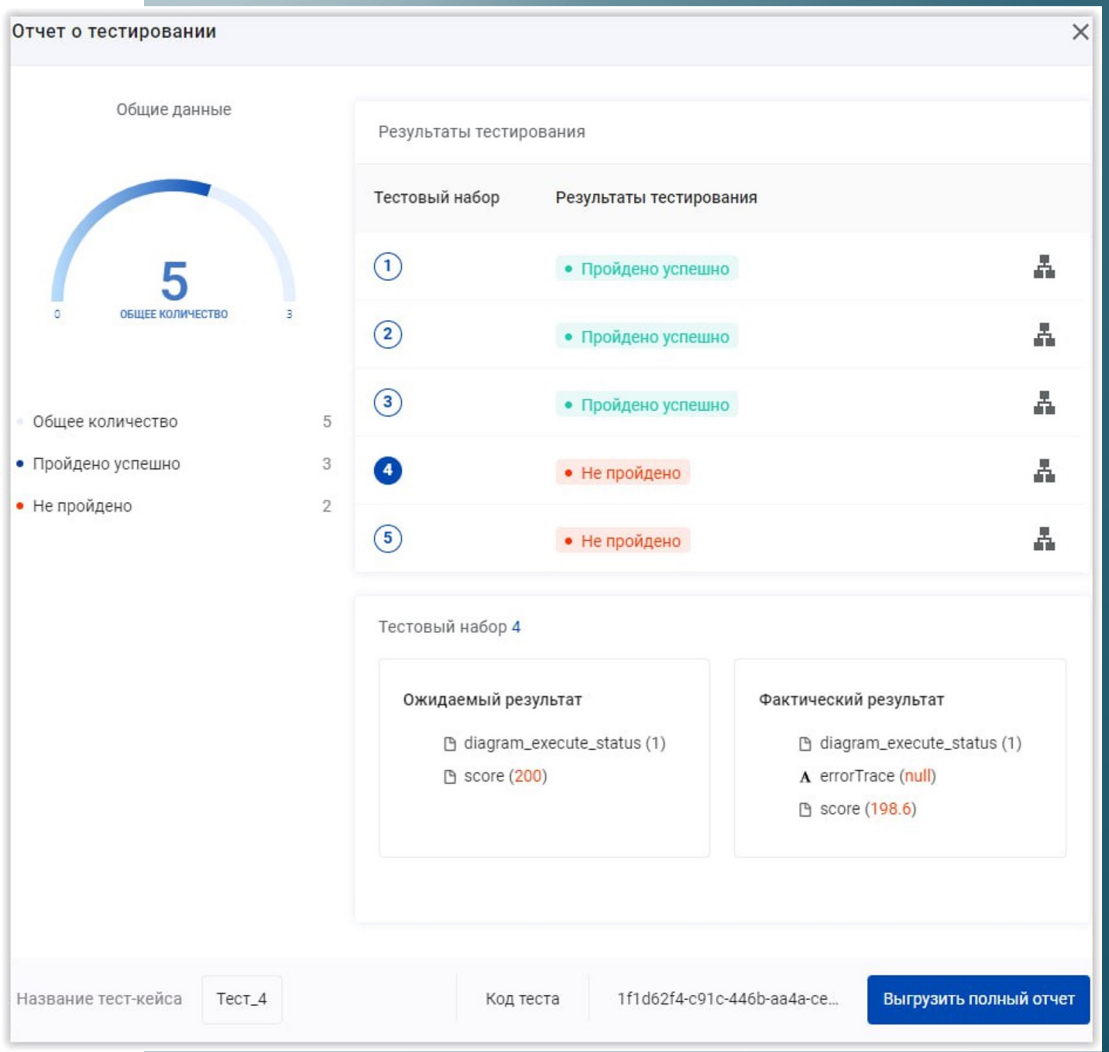


Функциональное тестирование

- Предназначено для тестирования готовых ETL-процессов
- В текущий момент доступно только для Kafka Source и реализуется путем подмены топика-источника на топик с синтетическими данными
- Для отслеживания промежуточных результатов используется промежуточный вывод в stdout и парсинг логов в OpenSearch



	A	B	C	D	E	F	G
1		Входные данные				Ожидаемые результаты	
2	Тестовый набор	_customer	score_model	age	run_id	diagram_execute_status	score
3	1	1	testing	32	-11	1	197
4							
5							
6							
7							
8							
9							
10							
11							
12							



ЗАКЛЮЧЕНИЕ

Заключение

- Задачи NRT удивительно многообразны благодаря IT-ландшафту и бизнес-задачам
- Мы начали с продуктивизации базовых технических моментов
- Впереди - расширение возможностей тестирования и реализация analyst-friendly подхода