Yandex Cloud

CDC. От баззворда к реализации в Data Transfer

Тимофей Брунько

Разработчик Yandex Cloud

О себе

2020 — Работаю над сервисом Data Transfer в Yandex Cloud

2018 — Присоединяюсь к Яндексу в отдел поставки данных

2012 — Занимаюсь R&D на C++

1. Задачи поиска данных:
   - inverted index
   - in-memory databases
   - nearest neighbor
   - graph search
2. Распознавание образов
3. Кластеризация
4. Оркестрация микросервисов
5. DSL и ANTLR
6. Распределённые системы

2010 — Устраиваюсь Malware Analyst

Что узнает аудитория из доклада

1. Знакомство с паттерном Change Data Capture
2. Реальные примеры использования CDC
3. Технические особенности реализации CDC
4. Особенности работы Debezium

План доклада

1. Что такое CDC
2. Debezium
3. Что такое Data Transfer
4. Чем на практике отличается логическая репликация от CDC
5. Появление CDC в Data Transfer
6. Как НЕ надо делать CDC
7. Как надо делать CDC
8. С чем нужно быть готовым столкнуться на этом пути
9. Что обычно удивляет пользователей сценария CDC
10. Выводы

Вопрос

Чем на практике отличается логическая репликация от CDC?

1. Что такое CDC

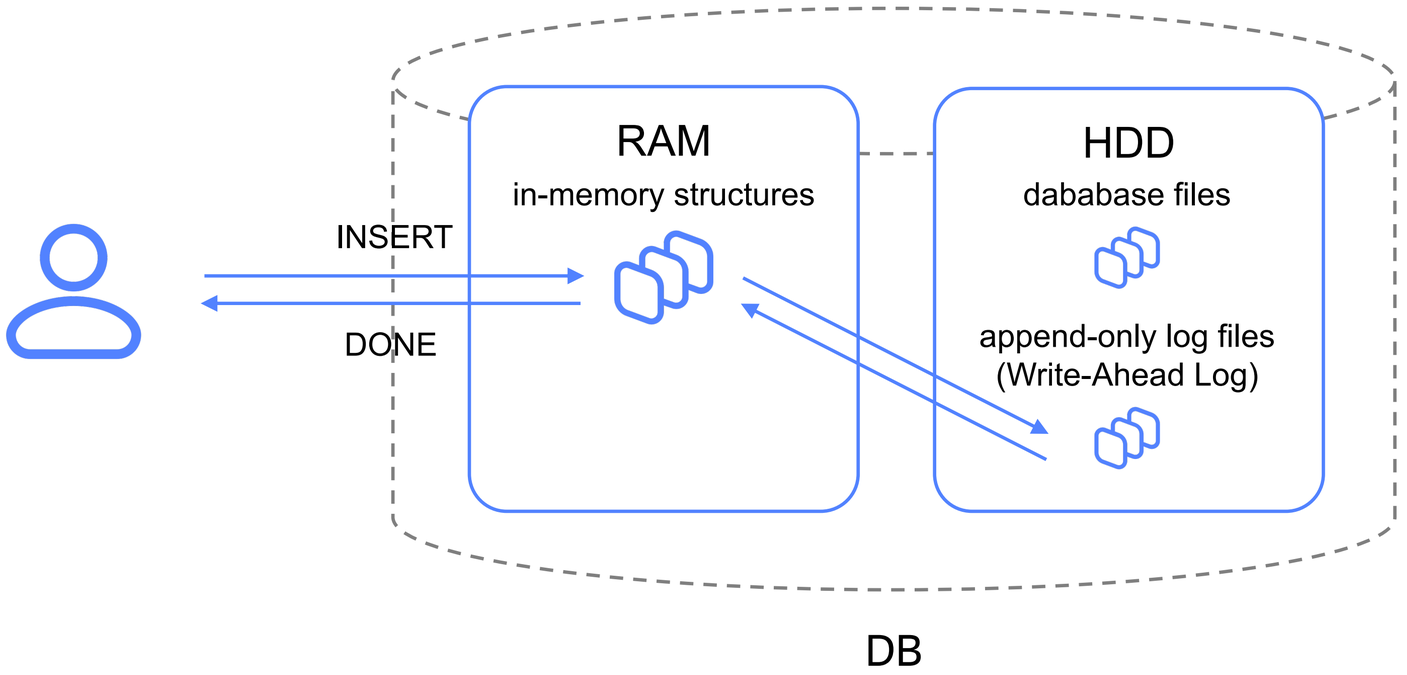
2. Debezium

3. Что такое Data Transfer

4. Чем на практике отличается
   логическая репликация от CDC

5. Появление CDC в Data Transfer

Как работают базы данных. WAL – Write-Ahead Logging



postgres – wal (ex: x-log)

mysql – binlog

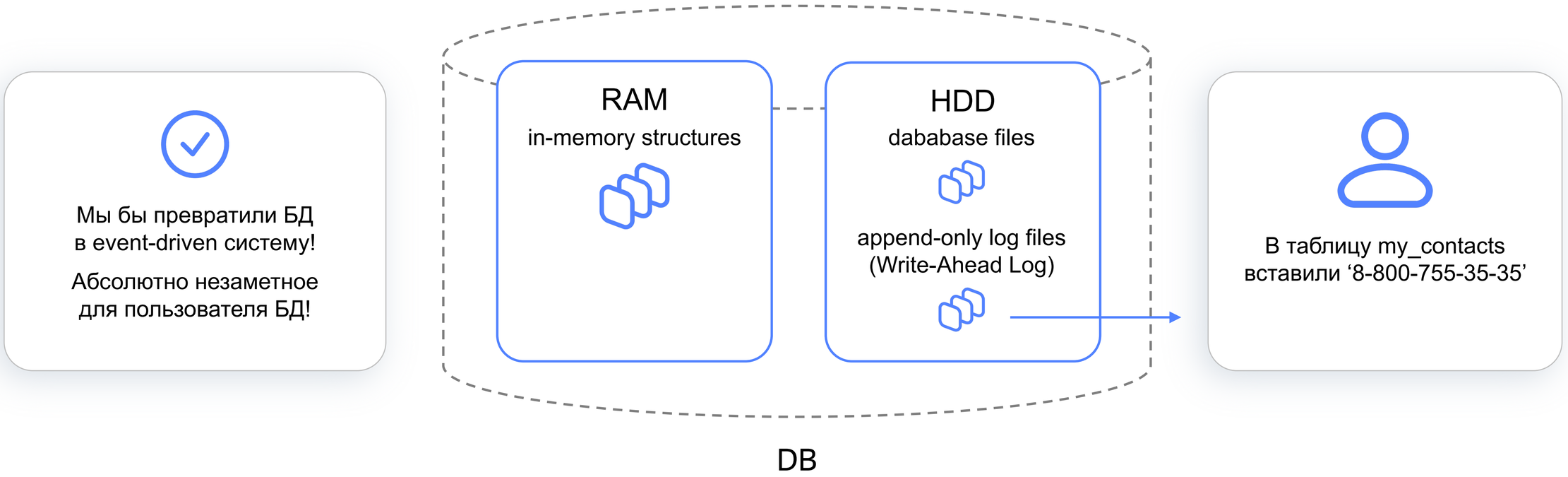
oracle – redo log

mongo – journal files

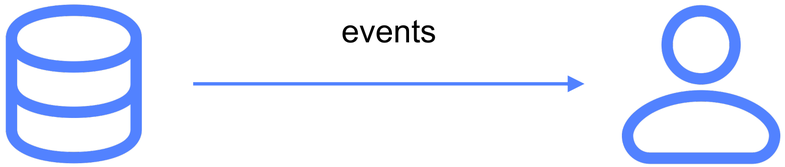
Лог – бинарный

Как работают базы данных. WAL – Write-Ahead Logging

Если бы мы только могли подписаться на обновления WAL-лога…



Что такое CDC



Change Data Capture (CDC)

Это подход, где база данных превращается в event-driven систему и начинает оповещать слушателей о произошедших изменениях.

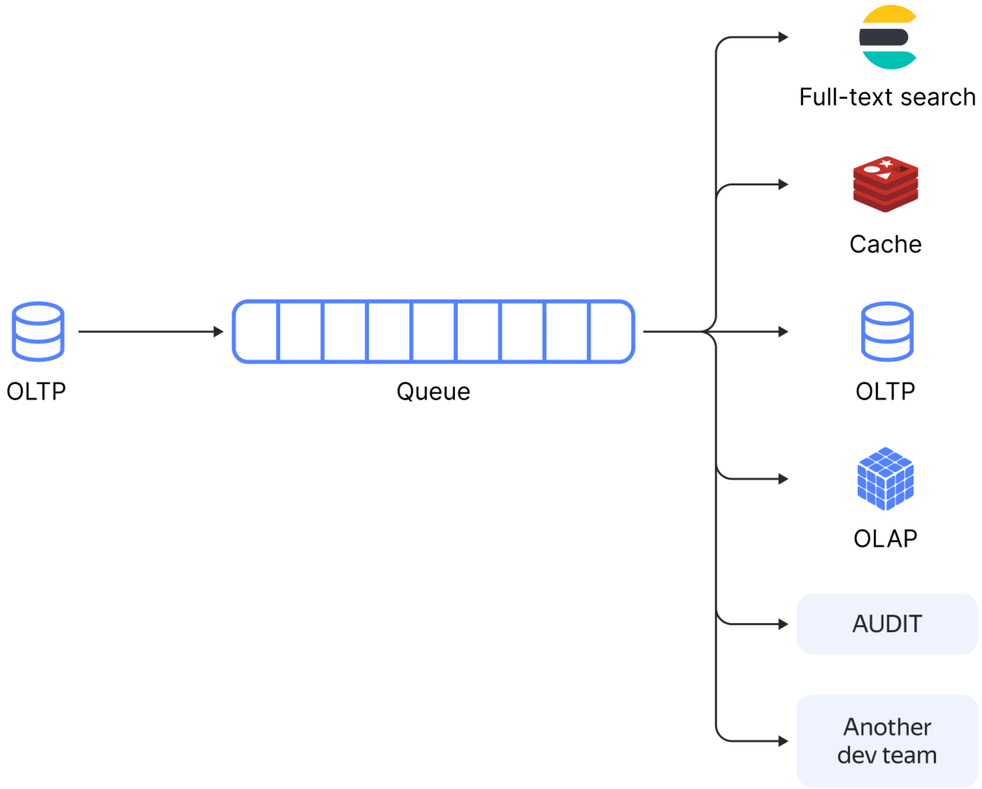
Все современные БД имеют способы отдавать WAL – декодированный, логический

habr.com: Change Data Capture (CDC) в Yandex Data Transfer: гид по технологии с примерами

https://clck.ru/35VGop

Типичная архитектура

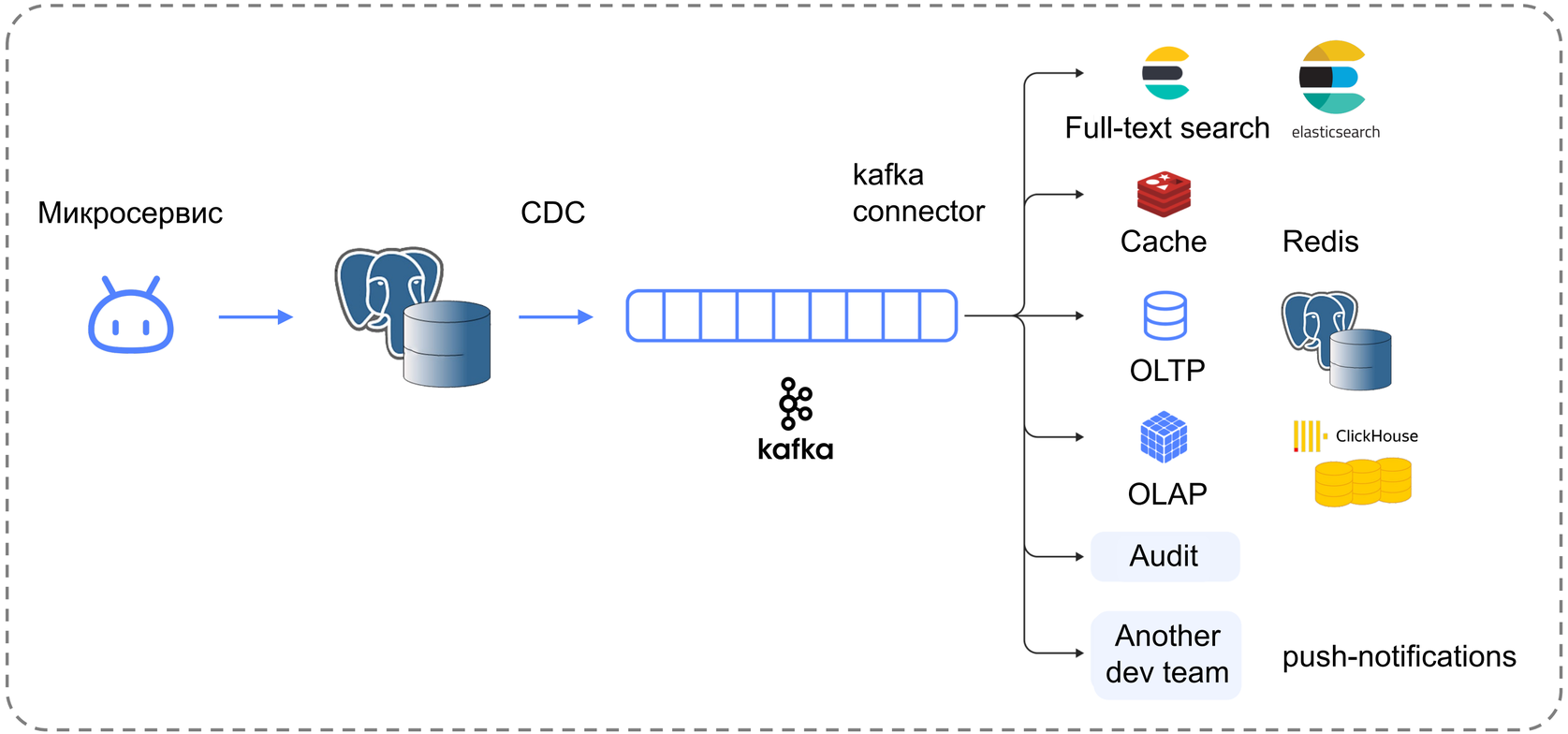
Change Data Capture



Справедливый вопрос: а зачем мне эта информация?

Пример

Change Data Capture



Всё это мы сделали:

- не написав ни строчки кода
- не внеся изменений в код микросервиса

Всё это само автоматически обновляется и всегда актуально!

Наглядный пример 1/3
INSERTS

```
CREATE TABLE my_little_pony (
    id INT PRIMARY KEY,
    name TEXT,
    description TEXT
);

INSERT INTO my_little_pony
    (id, name, description)
VALUES
    (1, 'Twilight Sparkle', ''),
    (2, 'Rainbow Dash', ''),
    (3, 'Spike', 'Spike goes BRRR');
```

```
{
    "op":"c",
    "after":{
        "id":1,
        "name":"Twilight Sparkle",
        "description":""
    }
}

{
    "op":"c",
    "after":{
        "id":2,
        "name":"Rainbow Dash",
        "description":""
    }
}

{
    "op":"c",
    "after":{
        "id":3,
        "name":"Spike",
        "description":"Spike goes BRRR... "
    }
}
```

Наглядный пример 2/3
UPDATES

```
UPDATE my_little_pony
SET description='_'
WHERE id<3;
```

```
{
    "op":"u",
    "after":{
        "id":1,
        "name": "Twilight Sparkle",
        "description":"_"
    }
}

{
    "op":"u",
    "after":{
        "id":2,
        "name":"Rainbow Dash",
        "description":"_"
    }
}
```

Наглядный пример 3/3
DELETES

```
DELETE FROM my_little_pony
WHERE name='Rainbow Dash';
```

```
{
    "op":"d",
    "before":{
        "id":2
    }
}
```

А самая популярная реализация CDC – это…

1. Что такое CDC

2. Debezium

3. Что такое Data Transfer

4. Чем на практике отличается
   логическая репликация от CDC

5. Появление CDC в Data Transfer

Debezium

Debezium
(далее – «ванильный Debezium»)

Опенсорс-проект (Apache-2.0 license), by redhat, имплементирующий CDC для широкого списка источников. Стал стандартом де-факто, написан на Java.

Debezium

1. Формат сериализации

   Один для всех баз, что удобно – можно писать логику обработки, не завязываясь на особенности работы конкретной БД

2. Дефолтный сериализатор

   JSON, также можно подключить Avro & Protobuf

3. Дополнительные возможности

   - Поставка снапшотов в том же формате
   - Поставка событий Debezium в универсальный JDBC
   - Инкрементальные снапшоты, https://debezium.io/blog/2021/10/07/incremental-snapshots/ (2019, Netflix)

Netflix: A Watermark Based Change-Data-Capture Framework

https://clck.ru/35VGqM

Debezium – формат (JSON-вариант)

```
{
        "schema": {...}, // огромное self-describing
        "payload": {
            "before": null,
            "after": {
                "id": 1,
                "first_name": "Anne",
                "last_name": "Kretchmar",
                "email": "annek@noanswer.org"
            },
            "source": {
                // всякие метаданные источника
            },
            "op": "c",
            "ts_ms": 1559033904863
        }
    }
```

С какой же прикладной стороны мы подошли к этому всему?

1. Что такое CDC

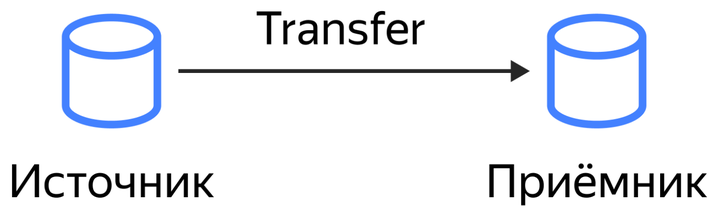
2. Debezium

3. Что такое Data Transfer

4. Чем на практике отличается
   логическая репликация от CDC

5. Появление CDC в Data Transfer

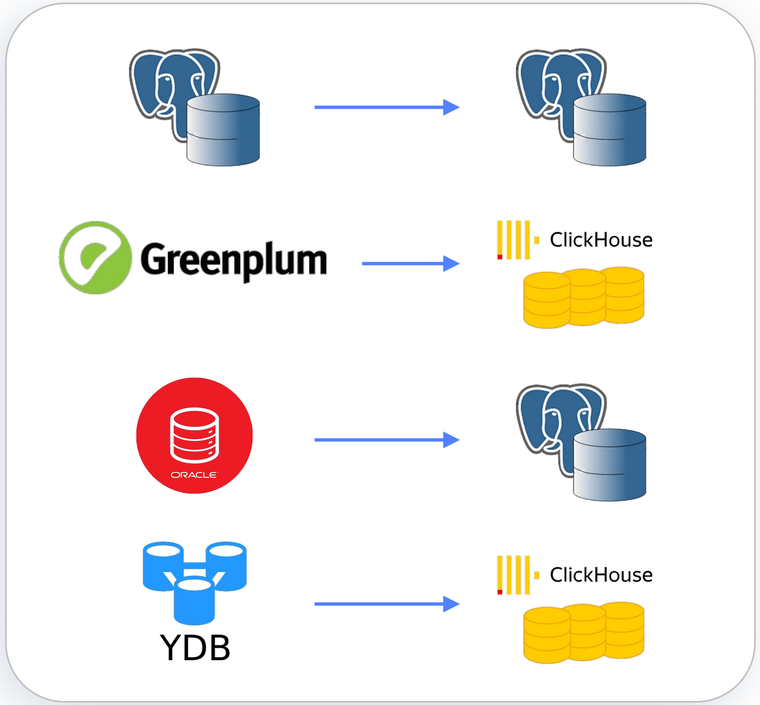
Что такое Data Transfer

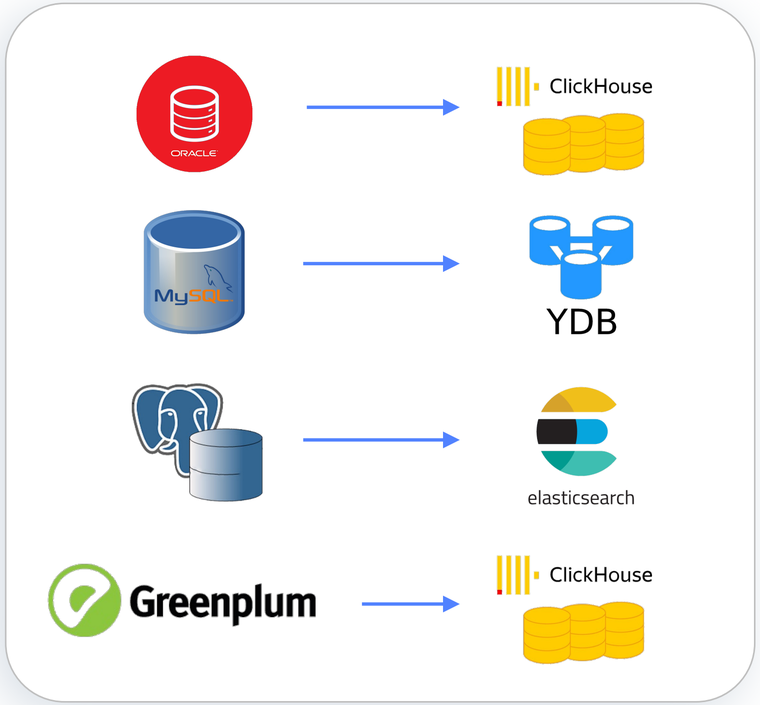


Управляемый сервис для трансфера данных из любого источника в любой приёмник:

- Database → Database
- Queue → Database
- Database → Queue
- S3 → *, *→ S3

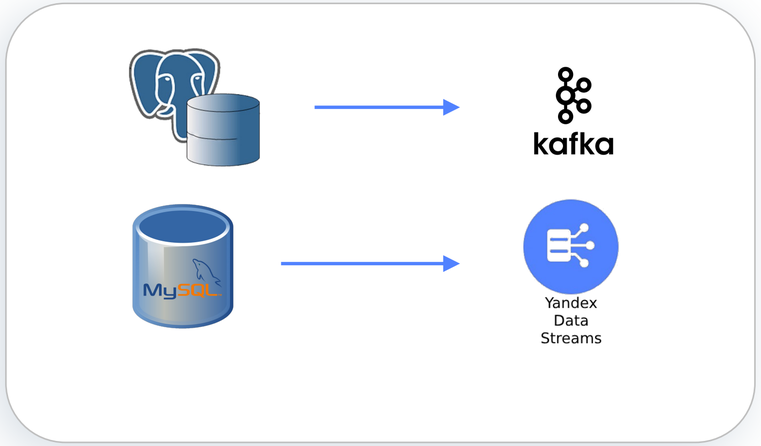
Примеры Database → Database

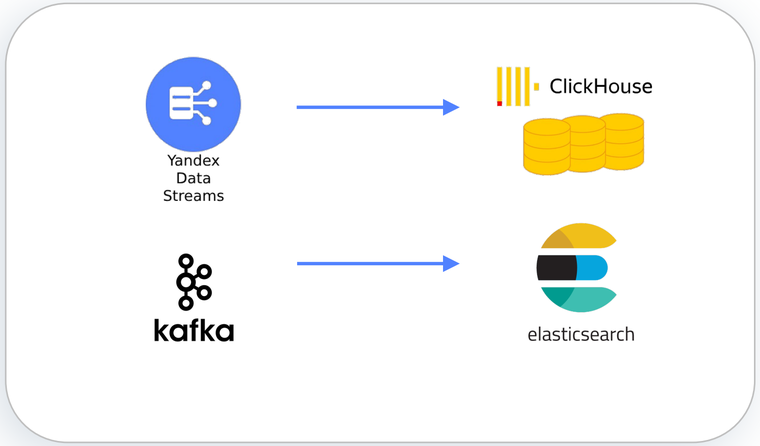




… и это далеко не всё

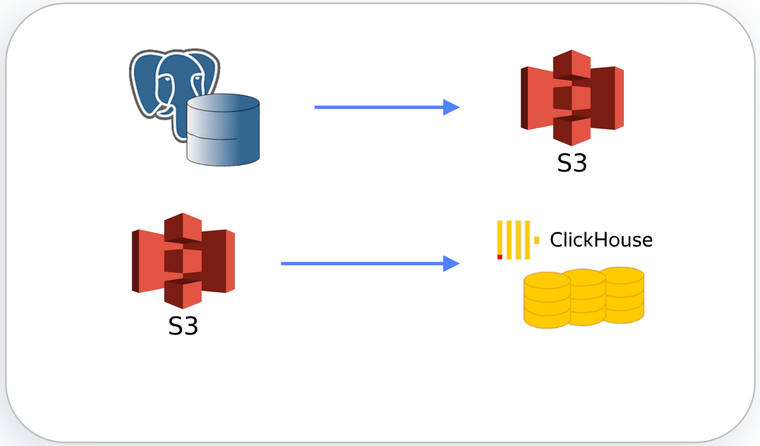
Примеры Database ⟷ Queue

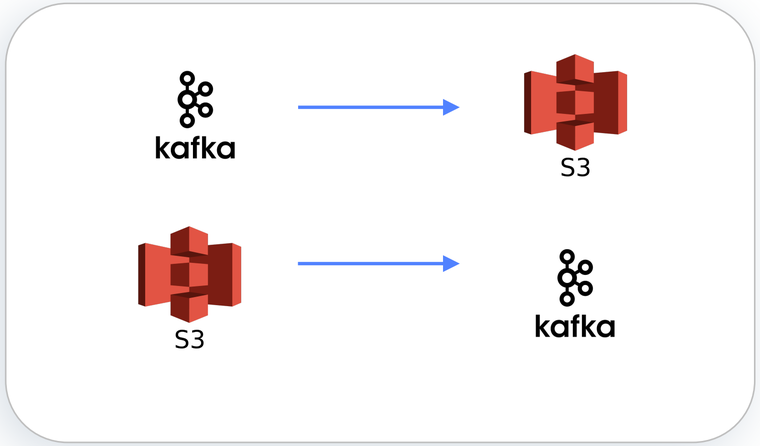




… и это далеко не всё

Примеры S3





… и это далеко не всё

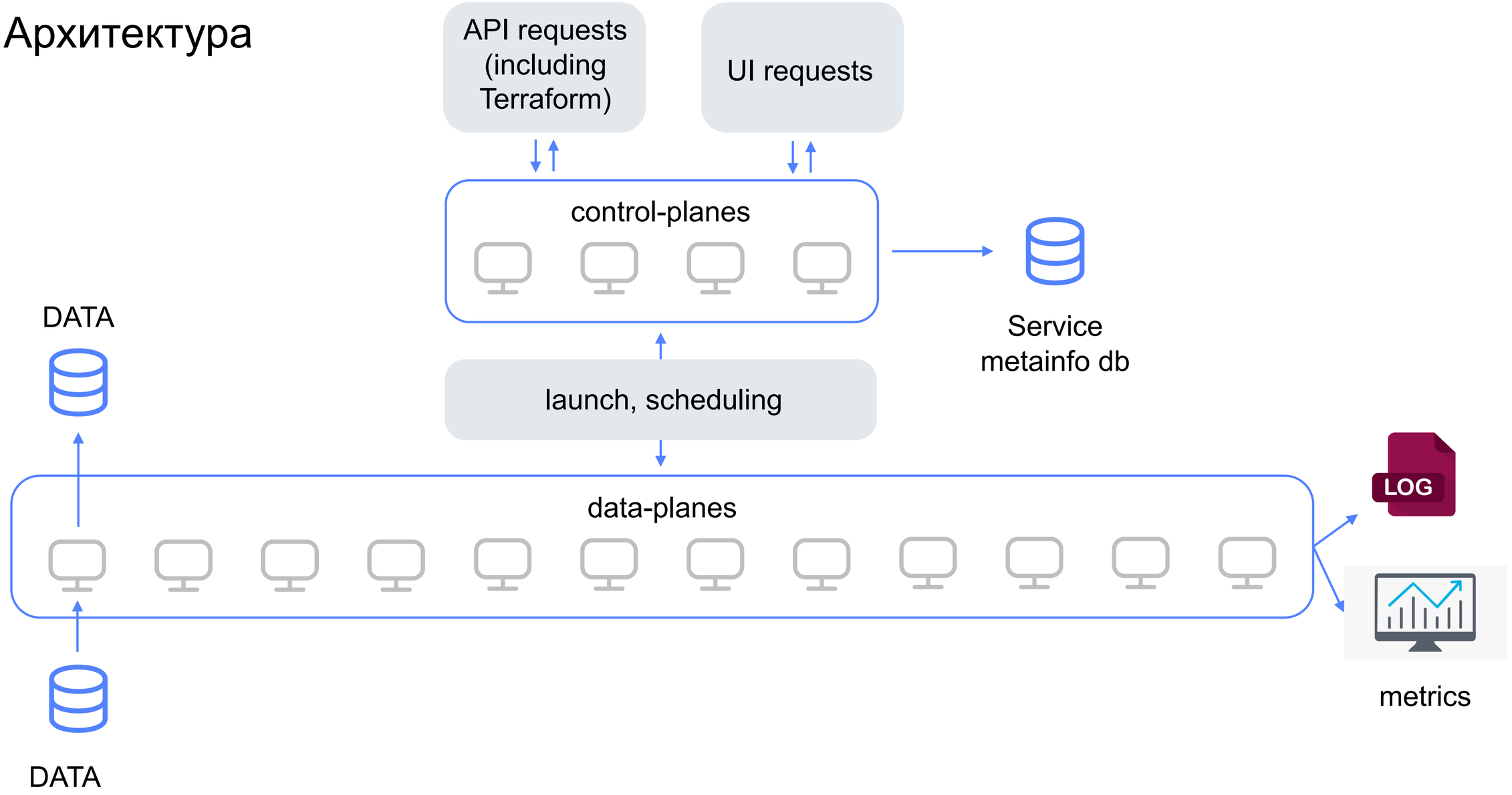
Матрица открытых пар в Yandex Cloud

- Для соединения пары мы не пишем код
- Для открытия пары требуем тесты
- В светлом будущем мы хотим открыть все возможные пары

… также есть коннекторы, не открытые в паблике

Приемник → / Источник ↓	PostgreSQL	MySQL	MongoDB	ClickHouse	Greenplum®	YDB	Object Storage	Apache Kafka	YDS	Elasticsearch	OpenSearch	← Приемник / Источник ↓
PostgreSQL	КР	-	-	КР	К[1]	КР[1]	К[1]	КР	КР[1]	К[2]	К[2]	PostgreSQL
MySQL	-	КР	-	КР	-	КР[1]	К[1]	КР	КР[1]	-	-	MySQL
Oracle	КР[1]	-	-	КР[1]	-	-	-	-	-	-	-	Oracle
MongoDB	-	-	КР	-	-	-	К[1]	-	-	-	-	MongoDB
ClickHouse	-	-	-	К	-	-	-	-	-	-	-	ClickHouse
Greenplum®	К[1]	-	-	К	К[1]	-	-	-	-	-	-	Greenplum®
YDB	-	-	-	КР[1]	-	-	К[1]	Р[2]	Р[2]	-	-	YDB
Metrica	-	-	-	Р[1]	-	-	-	-	-	-	-	Metrica
Yandex Data Streams	Р[1]	Р[1]	Р[1]	Р[1]	Р[1]	Р[1]	Р[1]	Р[1]	Р[1]	Р[2]	Р[2]	Yandex Data Streams
Apache Kafka®	Р[1]	Р[1]	Р[1]	Р[2]	Р[1]	Р[1]	Р[1]	Р[1]	Р[1]	Р[2]	Р[2]	Apache Kafka®
Airbyte®	К[1]	К1	К[1]	К[1]	-	К[1]	-	-	-	-	-	Airbyte®

Архитектура



Архитектура Data Plane

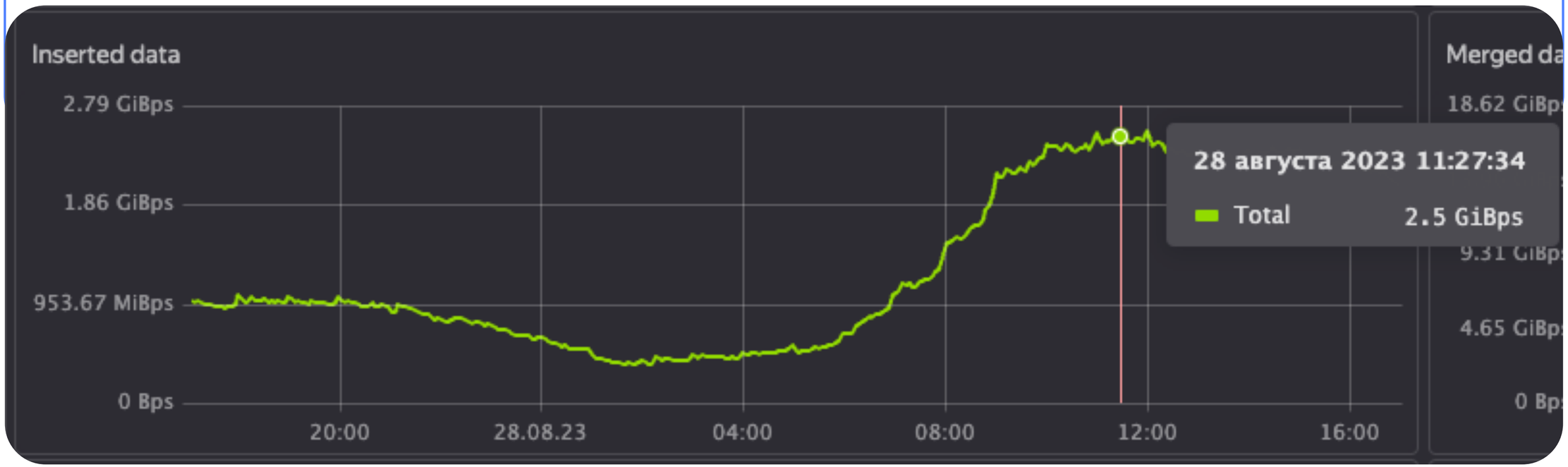
Свойства

1. Поставляем данные эффективно, на больших данных

Свойства

1. Поставляем данные эффективно, на больших данных

Пример трафика всех поставок Queue: ClickHouse одного пользователя в один ClickHouse кластер

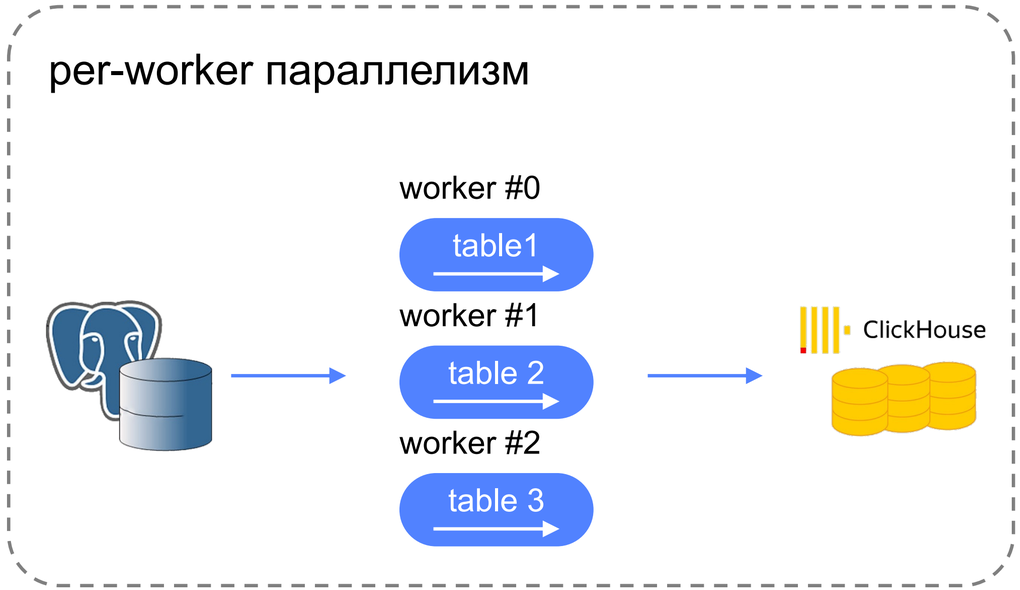


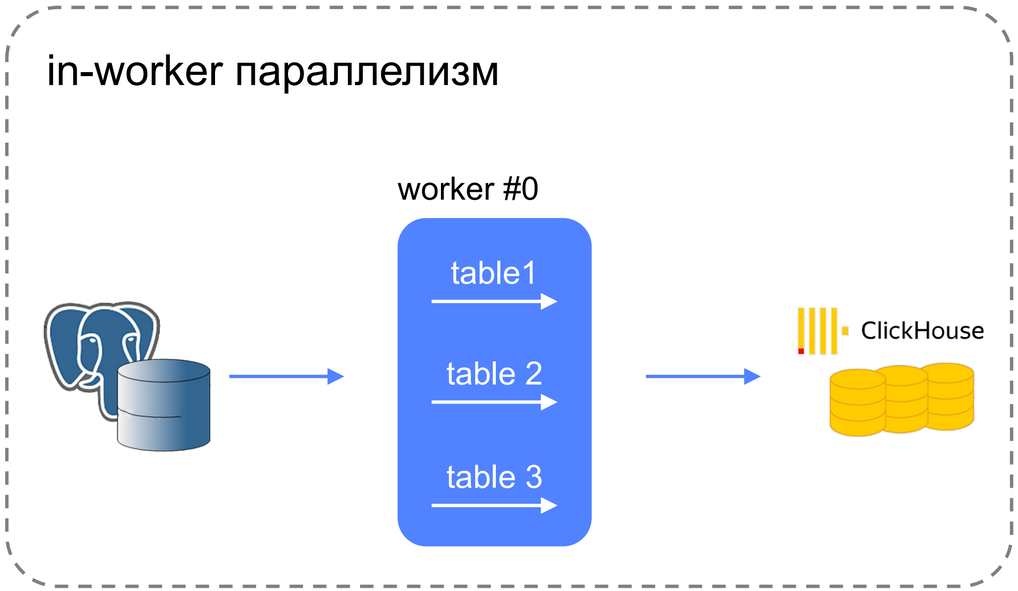
Свойства

1. Поставляем данные эффективно, на больших данных
2. Умеем параллелиться

Свойства

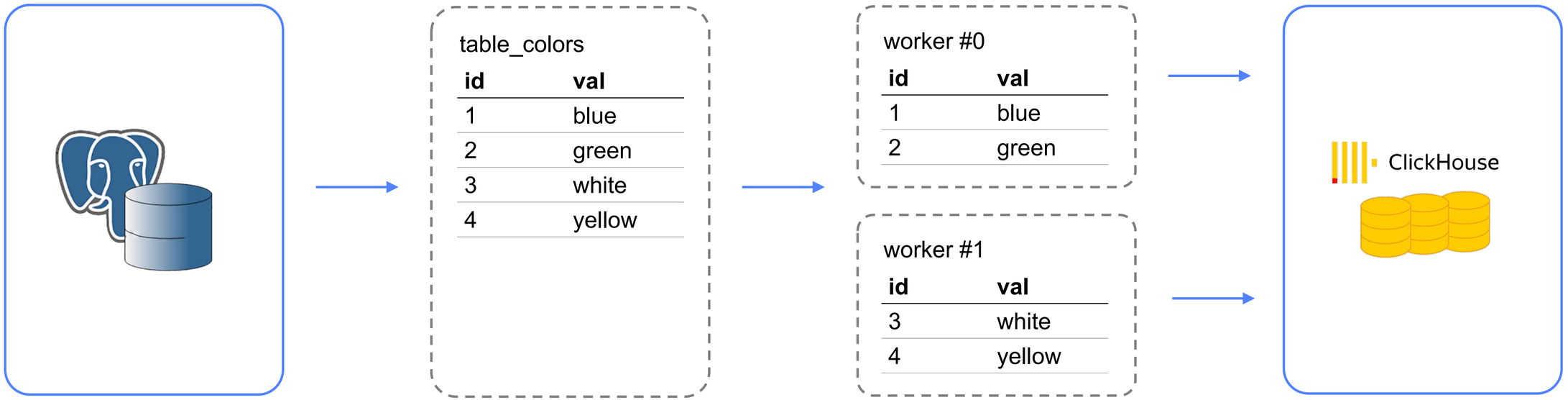
1. Поставляем данные эффективно, на больших данных
2. Умеем параллелиться
   - Параллелимся потаблично





Свойства

1. Поставляем данные эффективно, на больших данных
2. Умеем параллелиться
   - Параллелимся потаблично
   - Параллелимся внутри таблиц



Свойства

1. Поставляем данные эффективно, на больших данных
2. Умеем параллелиться
   - Параллелимся потаблично
   - Параллелимся внутри таблиц
   - Параллелимся одновременно и потаблично, и внутри

Свойства

1. Поставляем данные эффективно, на больших данных
2. Умеем параллелиться
   - Параллелимся потаблично
   - Параллелимся внутри таблиц
   - Параллелимся одновременно и потаблично, и внутри – степень параллелизма каждого уровня настраиваема
3. Поставка данных заводится за минуты
4. Самостоятельный
5. Умеет в трансформации (ETL)
6. Не храним данные – только переносим
7. Умеем автоматически выводить схему таблиц
8. Умеем на приёмниках делать Alter Table

Терминология

Основные понятия: эндпоинт & трансфер

Эндпоинт

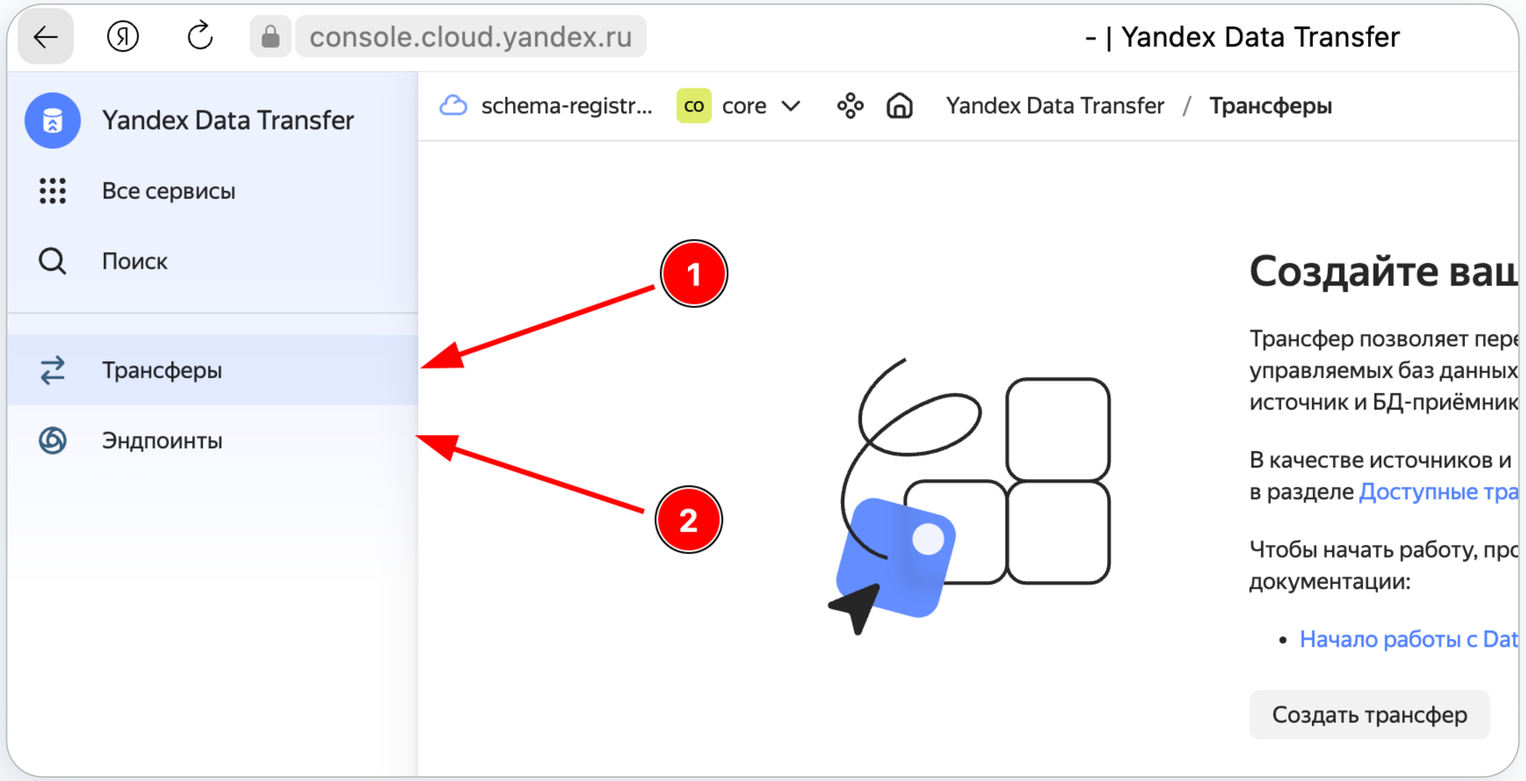
Connection + Дополнительные настройки

Эндпоинт может быть

- Источником
- Приёмником

Терминология

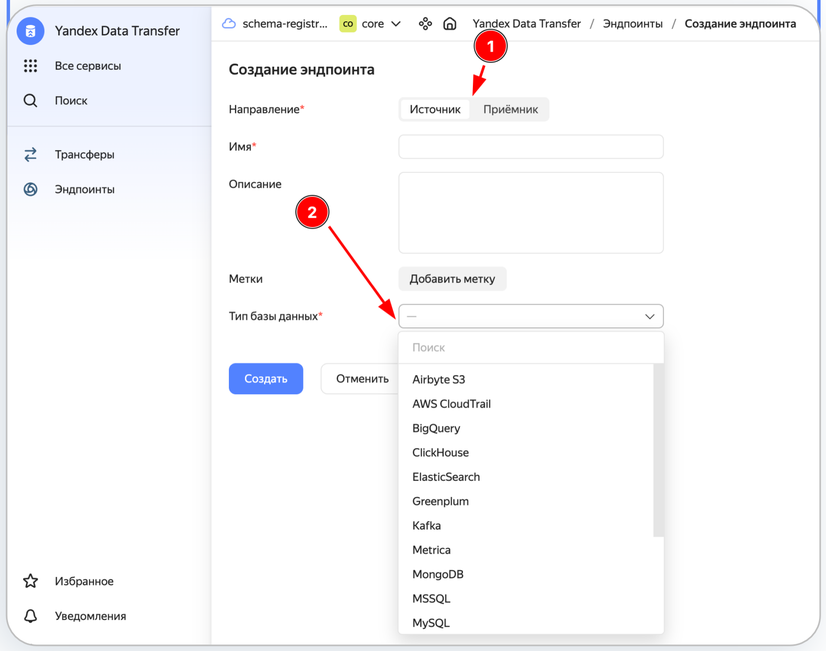
Как это выглядит



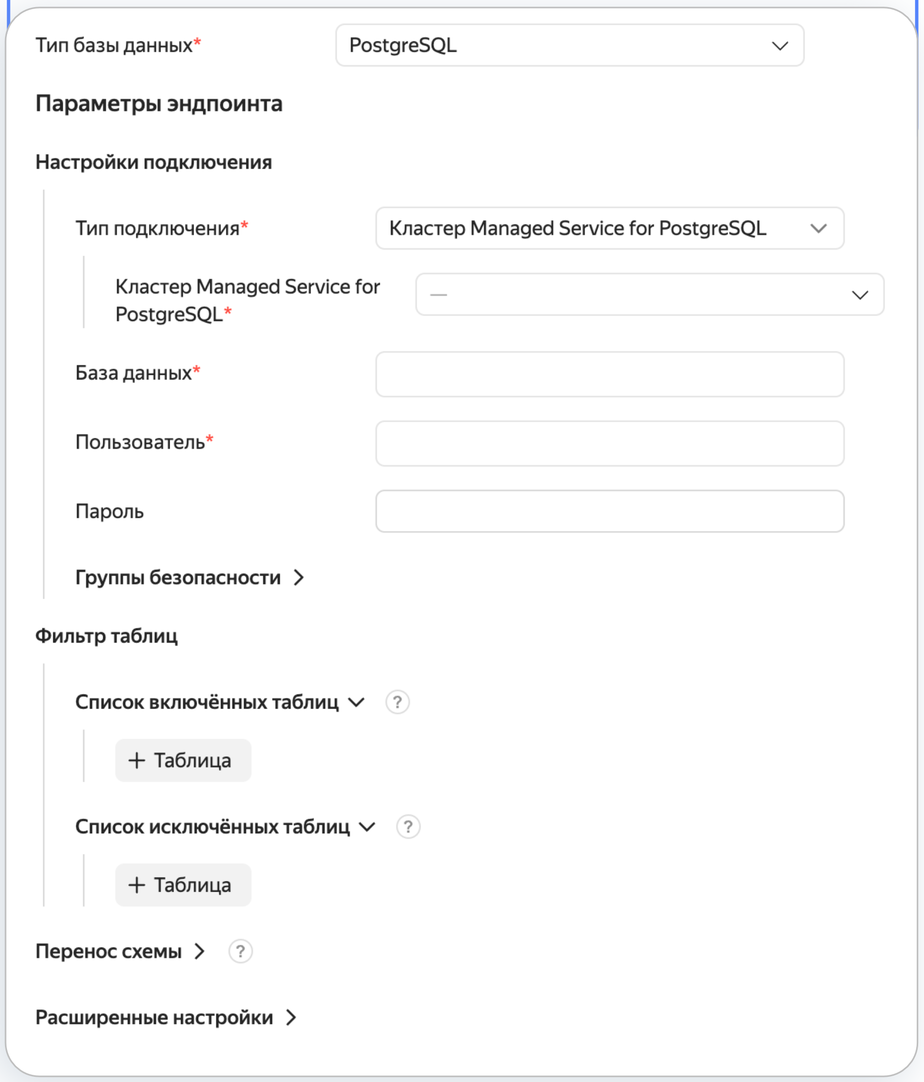
Терминология

Эндпоинт-источник

Выбираем тип



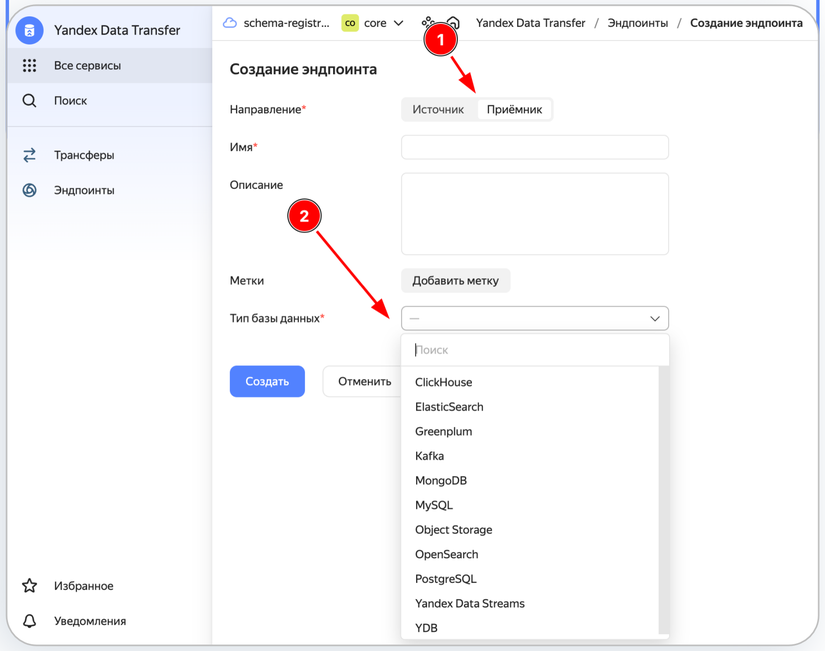
Настраиваем выбранный тип



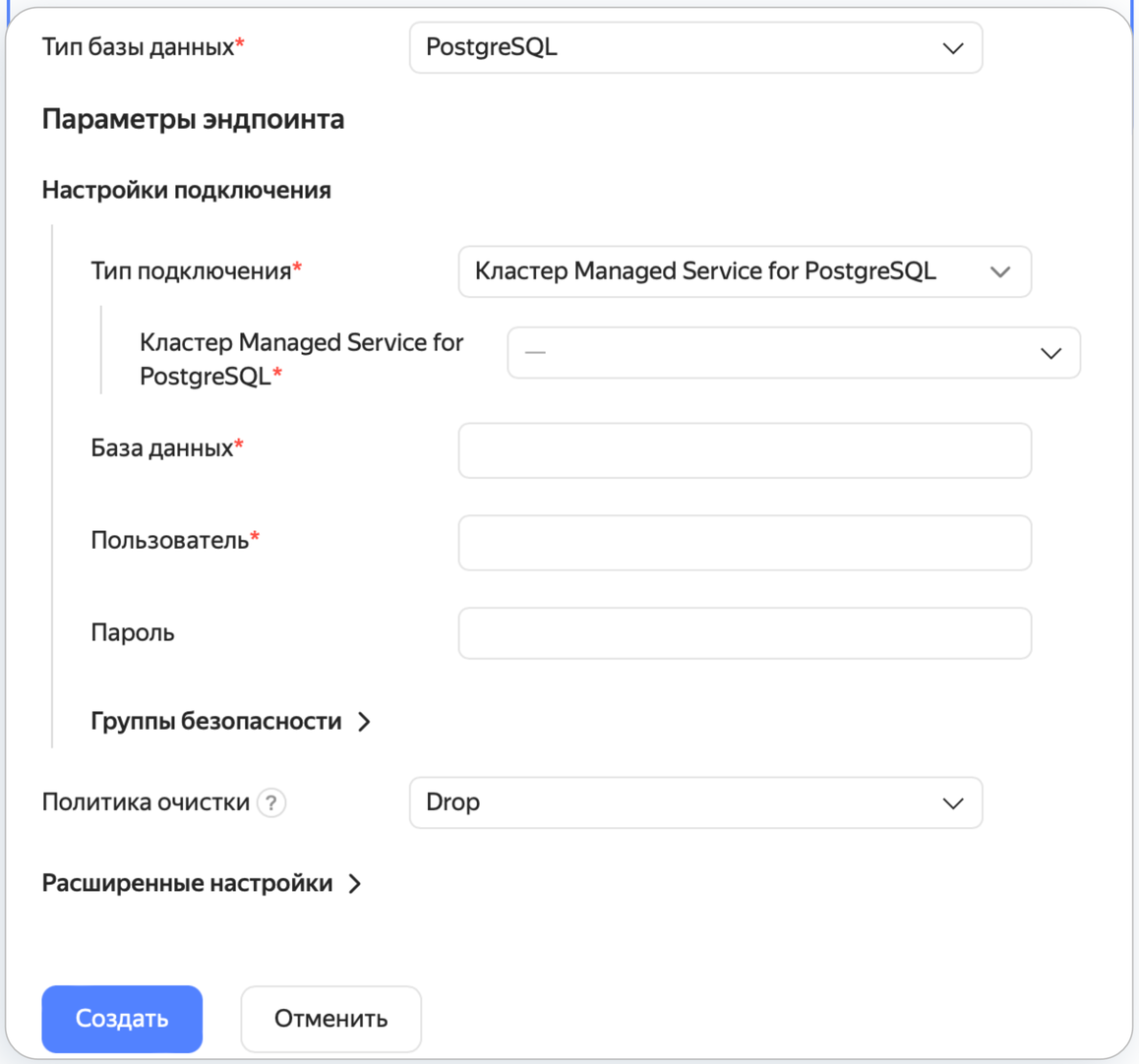
Терминология

Эндпоинт-приёмник

Выбираем тип



Настраиваем выбранный тип



Терминология

Основные понятия: эндпоинт & трансфер

Эндпоинт

Connection +

Дополнительные настройки

Эндпоинт может быть

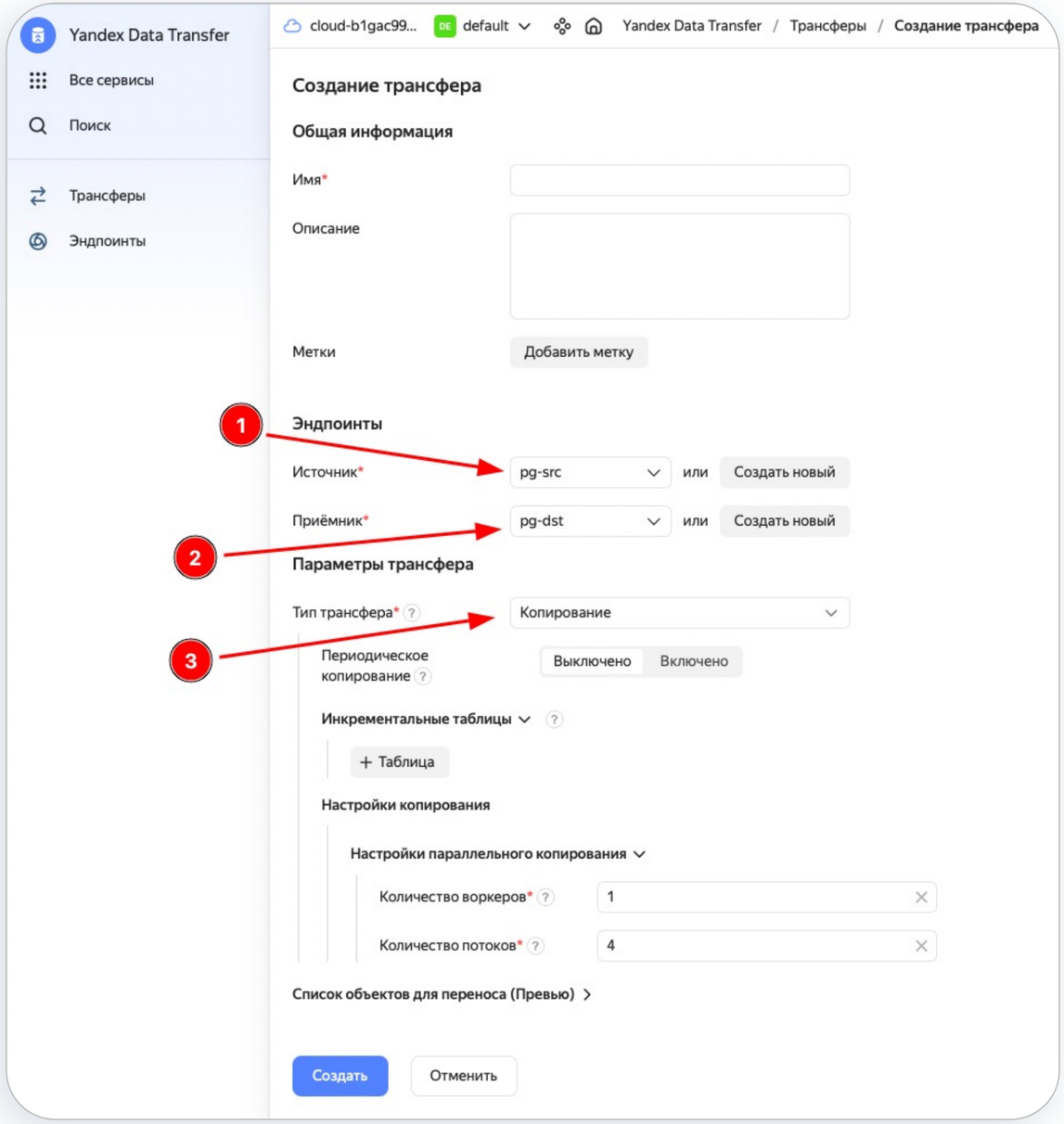
- Источником
- Приёмником

Трансфер

Соединяет эндпоинт-источник и эндпоинт-приёмник, после активации запускает перенос данных

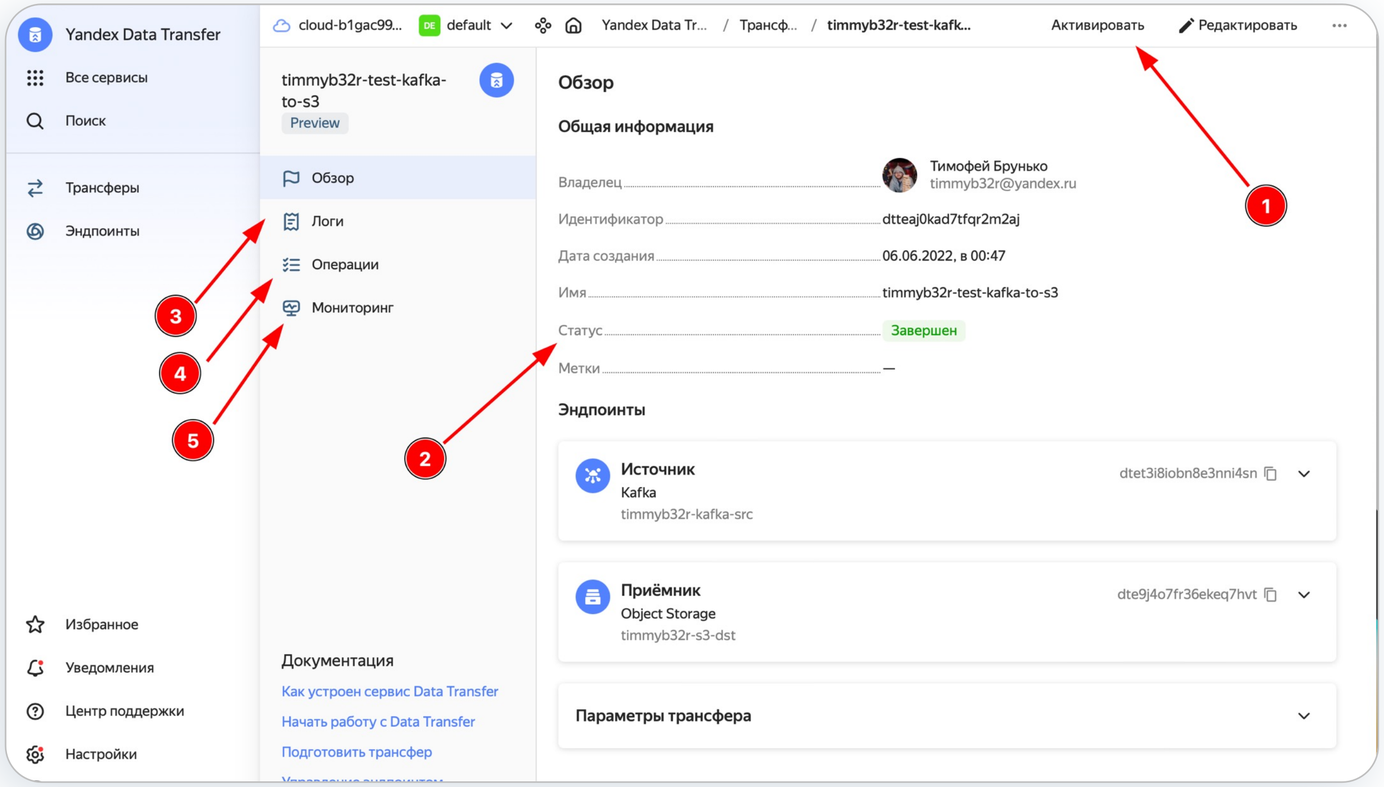
Терминология

Трансфер



Терминология

Трансфер



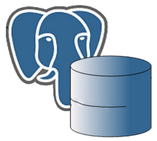
Режим работы

1. COPY (aka SNAPSHOT)
   - Таблицы переносятся, после чего трансфер деактивируется сам

Пример

table_colors

id	val
1	blue
2	green
3	white
4	yellow



1,2,3,4

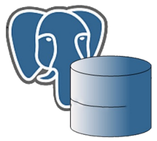
Режим работы

1. COPY (aka SNAPSHOT)
   - Таблицы переносятся, после чего трансфер деактивируется сам
2. REPLICATION
   - Состояние не переносится, переносятся новые изменения, работает на уровне логической (не физической) репликации. Сам не деактивируется. Получается логическая асинхронная реплика без первоначального стейта.

Пример

table_colors

id	val
1	blue
2	green
3	white
4	yellow



INSERT INTO
table_colors
VALUES (5, black);

5

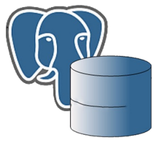
Режим работы

1. COPY (aka SNAPSHOT)
   - Таблицы переносятся, после чего трансфер деактивируется сам
2. REPLICATION
   - Состояние не переносится, переносятся новые изменения, работает на уровне логической (не физической) репликации. Сам не деактивируется. Получается логическая асинхронная реплика без первоначального стейта.
3. COPY + REPLICATION

Пример

table_colors

id	val
1	blue
2	green
3	white
4	yellow



INSERT INTO
table_colors
VALUES (5, black);

1,2,3,4

5

Пользовательские сценарии

Сценарий миграции

- pg → pg
- mysql → mysql

Yandex Data Transfer
в сценариях, о которых
редко говорят

https://clck.ru/35VGnR

Сценарий поставки данных

Преимущественно
для аналитических сценариев

- OLTP → analytical
- Queue → analytical (logs)
- Analytical → analytical
- CDC
- Queue → OLTP
- The stranger things…
  (queue → queue, analytical → queue, …)

Аналоги

Прямые аналоги

- Apache SeaTunnel
- Addax

Непрямые аналоги

- Airbyte
- Fivetran
- Kafka с коннекторами

Аналоги

Чем Data Transfer лучше аналогов

1. Мы даём выбор – с очередью или без
2. Мы интегрированы в экосистему Yandex Cloud
3. Какой профит у работы без очереди:
    - Косты и упрощение схемы поставки
    - Упрощение настройки
    - Performance
    - Лицензирование

Технические факты

1. Написан на Golang
2. Гарантии at least once
   - Там, где есть первичные ключи, – неотличимо от exactly once
   - Там, где нет первичных ключей, – exactly once зачастую тоже реализуемо
3. Исходники не в open source
4. В Yandex Cloud сейчас продукт бесплатен
5. Есть несколько инсталляций сервиса

Эксплуатационные факты

с 2019

Существует продукт

> 20 000

Трансферов заведено за всё время

~1000

Трансферов работают постоянно

1. Что такое CDC

2. Debezium

3. Что такое Data Transfer

4. Чем на практике отличается логическая репликация от CDC

5. Появление CDC в Data Transfer

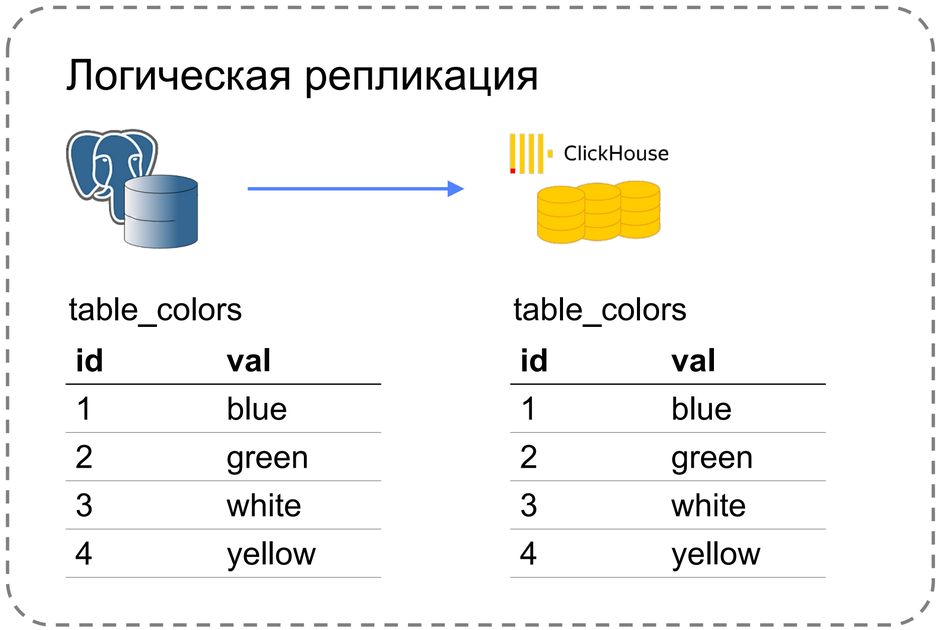
Вопрос

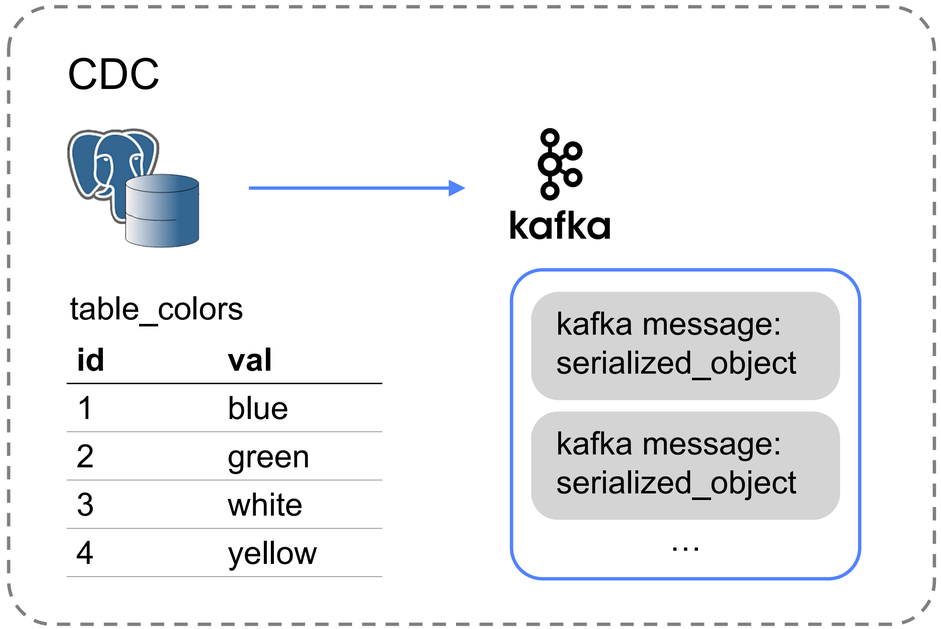
Чем на практике отличается логическая репликация от CDC?

Вопрос

Чем на практике отличается логическая репликация от CDC?

Ответ – сериализацией



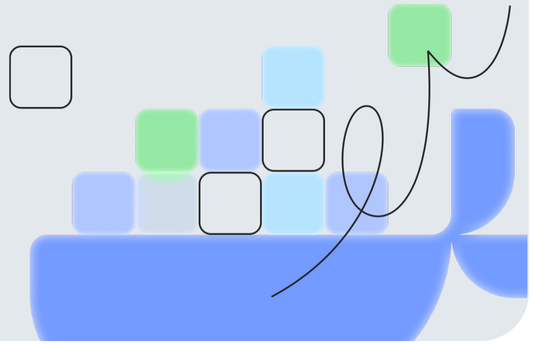


А значит, в Data Transfer было почти всё необходимое для организации CDC:

1. Подключаться к источникам как логическая реплика мы умели
2. Отгружать данные в очередь мы умели

Оставался только один вопрос

Сериализация и запрос пользователей



1. Что такое CDC

2. Debezium

3. Что такое Data Transfer

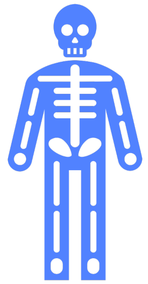
4. Чем на практике отличается
   логическая репликация от CDC

5. Появление CDC в Data Transfer

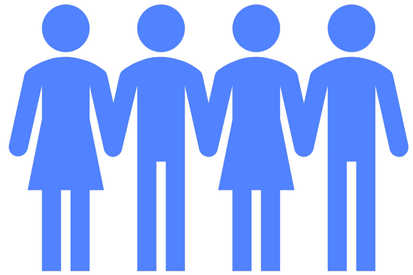
Появление CDC в Data Transfer

Всё это достижимо через отгрузку в очереди – а значит, пришло время для CDC

Хотим забирать данные
из источника только раз,
а поставлять много куда

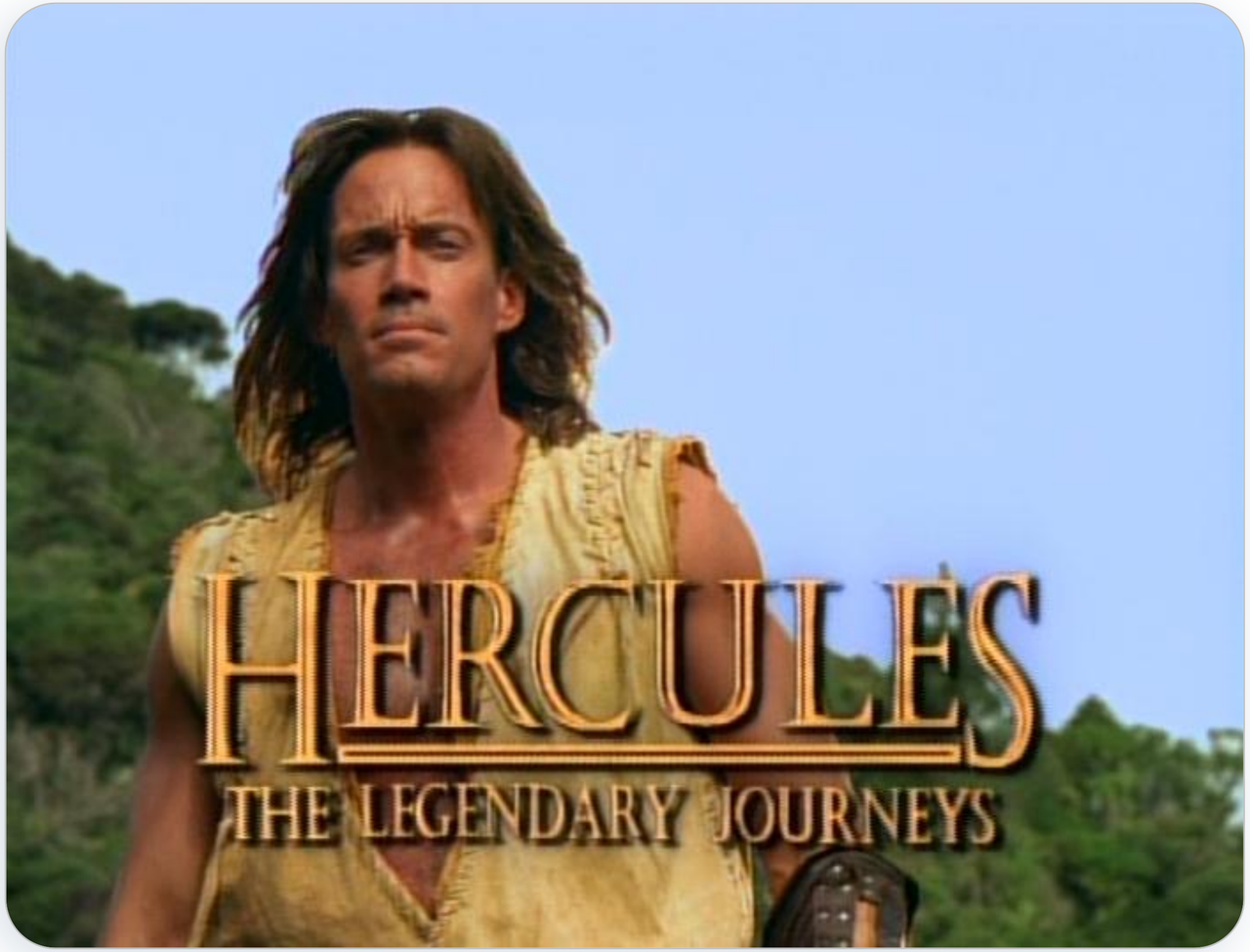


Хочу организовать
event-driven систему



Хотим забирать с источника
WAL лог как можно быстрее,
даже если приёмник кашляет

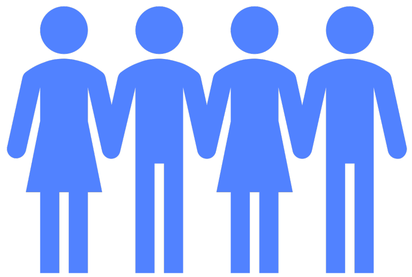
Появление CDC в Data Transfer



6. Как НЕ надо делать CDC
7. Как надо делать CDC
8. С чем нужно быть готовым столкнуться на этом пути
9. Что обычно удивляет пользователей сценария CDC
10. Выводы

Как НЕ надо делать CDC

Хотим забирать данные
из источника только раз,
а поставлять много куда



Хотим забирать из источника
WAL лог как можно быстрее,
даже если приёмник кашляет

DB → queue → DB?

Yes.

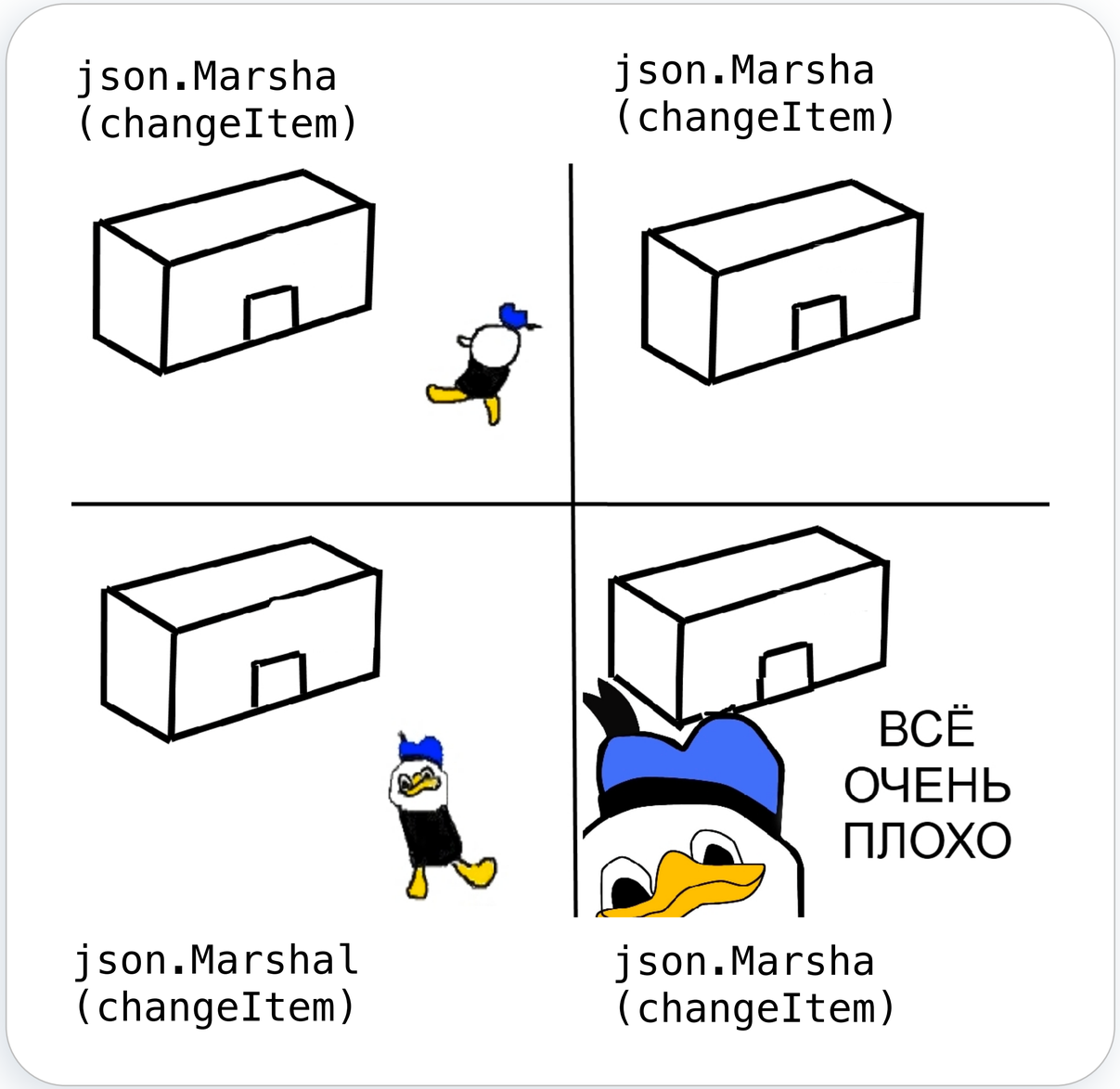
Как НЕ надо делать CDC

```
json.Marshal(changeItem)
```

***changeItem**

это наш внутренний формат
события изменения строки

Как НЕ надо делать CDC

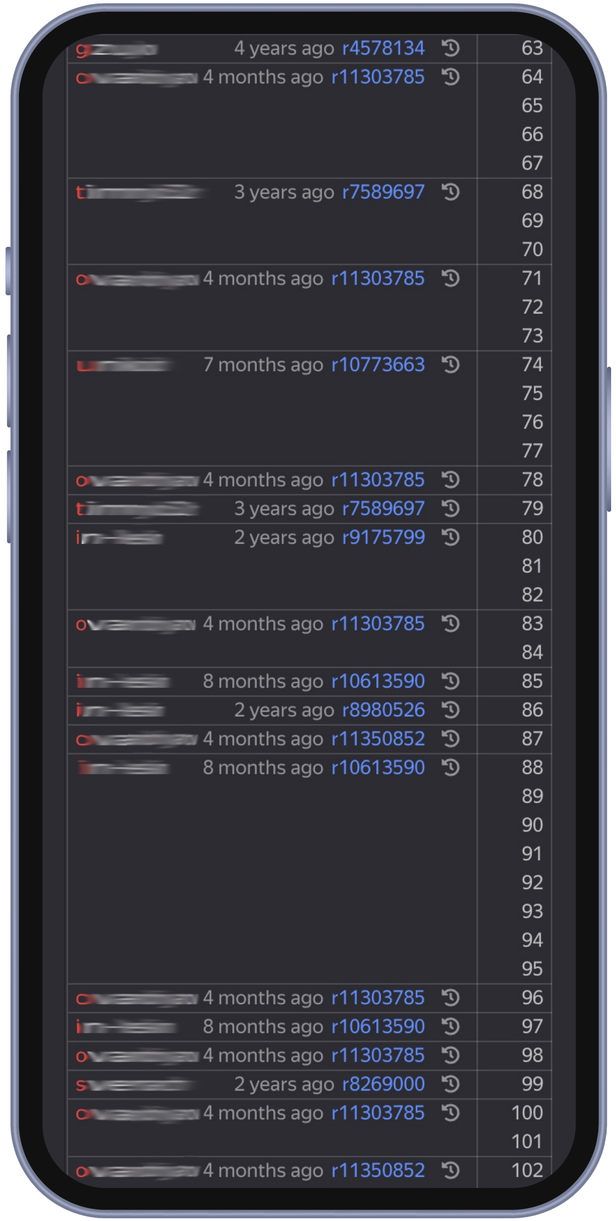


«Всё течёт, всё меняется»

```
json.Marshal(changeItem)
```

Как выстрелить себе в ногу

- Добавить поле
- Удалить поле
- Переименовать пол



Как НЕ надо делать CDC

«Всё течёт, всё меняется»

Вывод № 1

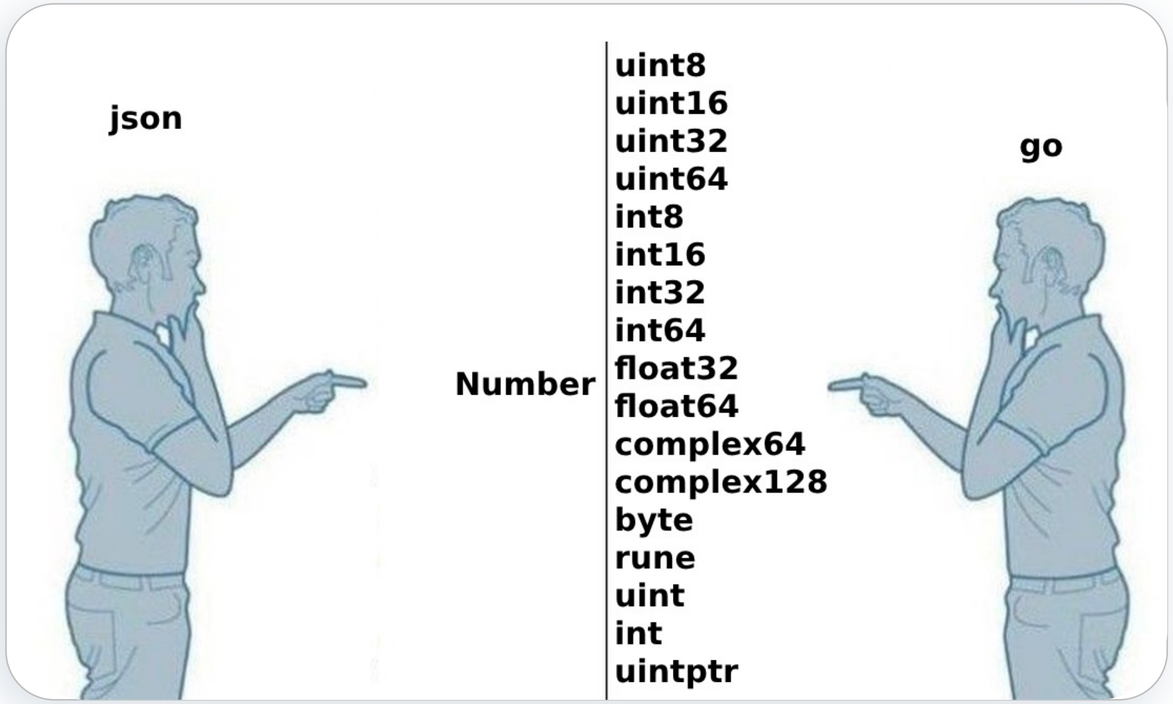
«Отгрузили данные
в очередь – это уже контракт»

Анмаршаллинг JSON в нетипизированное поле

```
type ChangeItem struct {
    //…
    ColumnNames []string
    ColumnValues []any
}
```

Как выстрелить себе в ногу

Заанмаршаллить число
в нетипизированное поле



Как НЕ надо делать CDC

Анмаршаллинг JSON в нетипизированное поле

Вывод № 2

«Числа в json – это прежде всего строки»

Вывод № 3

«Анмаршаллишь json-числа в нетипизированное поле – делай это в строку, а потом руками конверть»

Анмаршаллинг JSON в нетипизированное поле

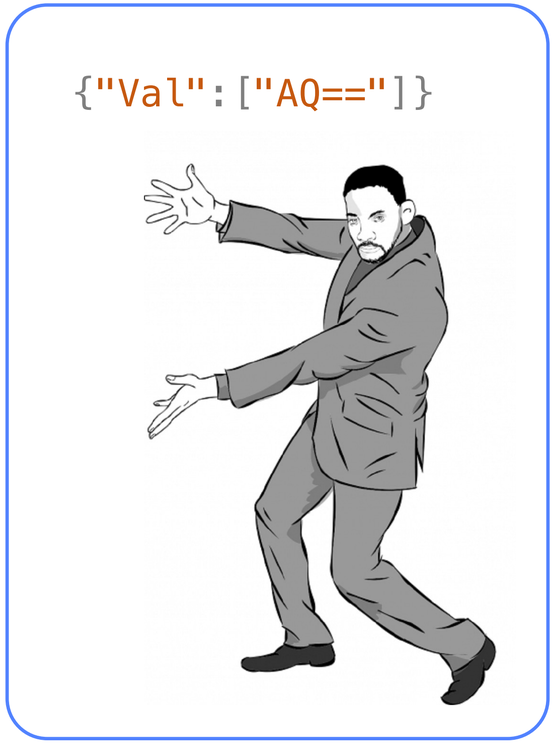
Как выстрелить себе в ногу

Заанмаршаллить []uint8

```
type ChangeItem struct {
    Val any
}


bytesArr0, _ := json.Marshal(ChangeItem{
    Val: any{[]uint8{1}},
})


bytesArr1, _ := json.Marshal(ChangeItem{
    Val: any{"AQ=="},
})
```



Как НЕ надо делать CDC

Анмаршаллинг JSON в нетипизированное поле

Вывод № 4

«Анмаршаллишь json в нетипизированное поле – прикапывай в json тип данных и кастуй явно»

Контракты на нетипизированных полях

```
type ChangeItem struct {
    Val any
    Type DataTypeEnum
}
```

Как выстрелить себе в ногу

- Шаг 1: Не проверяем согласованность
- Шаг 2: Получаем ситуацию, где тип описывает не то, что действительно в переменной

Как НЕ надо делать CDC

Контракты на нетипизированных полях

Вывод № 5

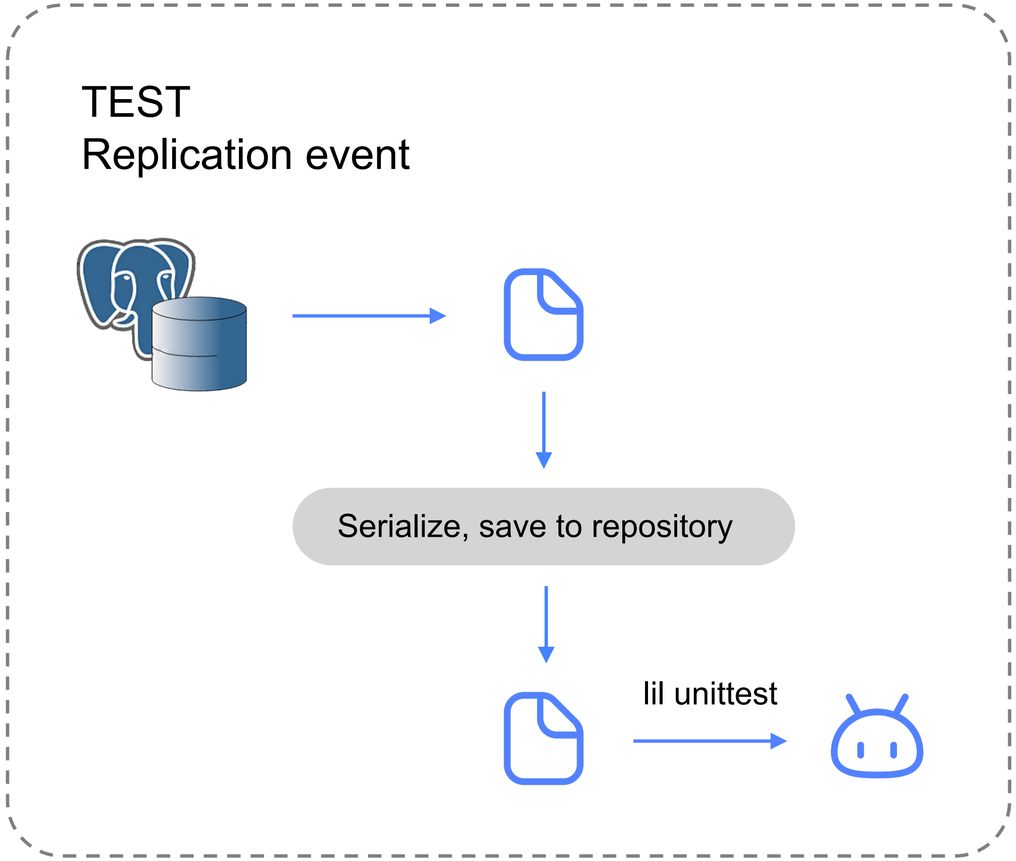
«Есть нетипизированное поле – носи рядом его тип и всегда гарантируй согласованность»

Проблема устаревания юнит-тестов

Проблема устаревания юнит-тестов

Проблема устаревания юнит-тестов

Проблема устаревания юнит-тестов



Как выстрелить себе в ногу

Не сделать тест на то, что сериализованные данные в репозитории всё ещё актуальны

Как НЕ надо делать CDC

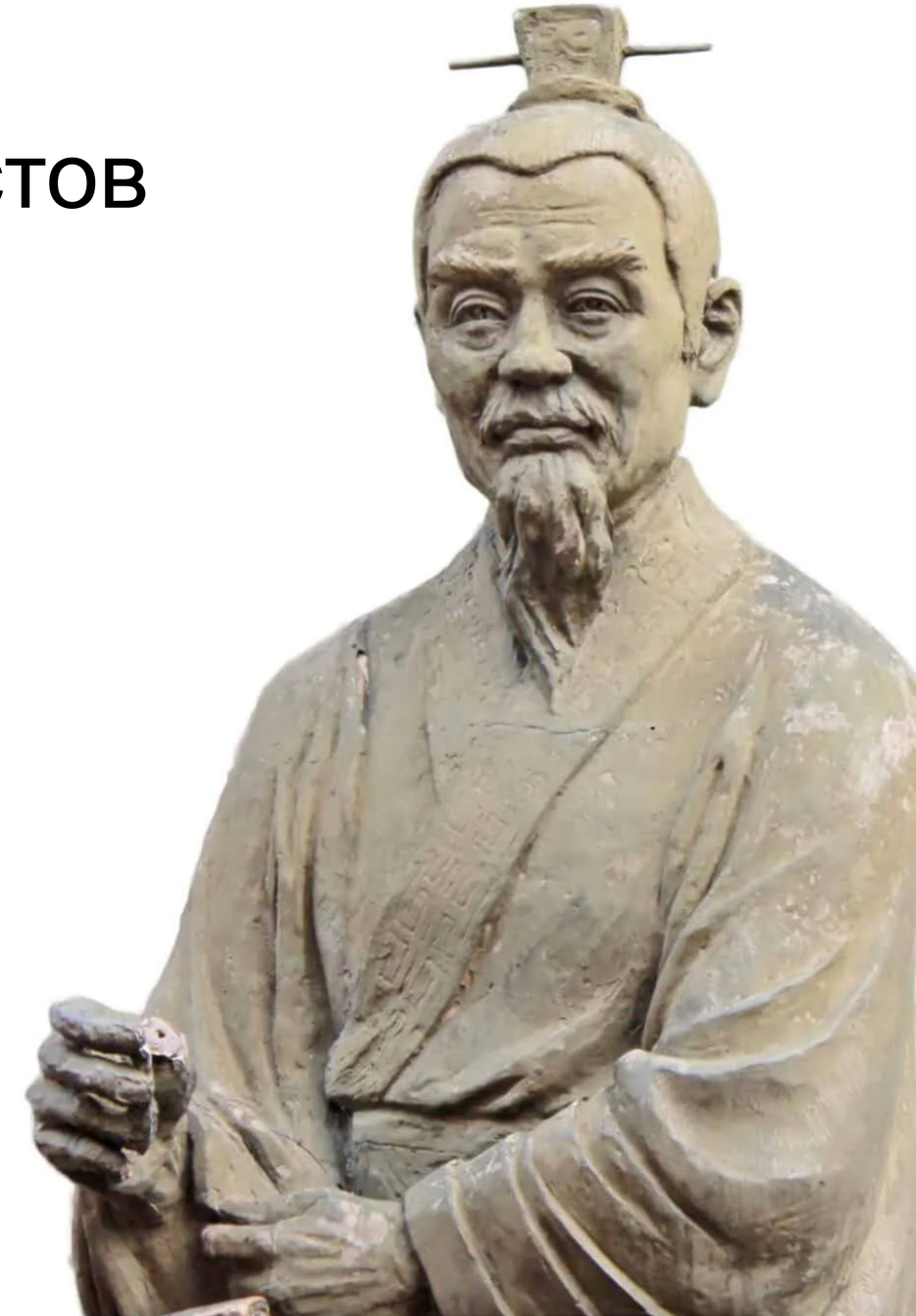
Проблема устаревания юнит-тестов

Вывод № 6

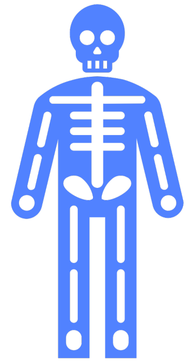
«Если сохраняешь в репозитории сериализованные данные – добавь тест на то, что они актуальны»

Вывод № 7

Сериализация в JSON – это всё ещё сериализация. Если JSON красивый и читаемый – он всё ещё может быть устаревшим относительно структуры.»



Надеяться на то, что пользователи, имея возможность отгрузить данные в очередь, не начнут вычитывать это своим кодом



Хочу организовать event-driven систему

Надеяться на то, что пользователи, имея возможность отгрузить данные в очередь, не начнут вычитывать это своим кодом

Как выстрелить себе в ногу

Понадеяться на то,
что пользователи, настроив
поставку данных в очередь,
не начнут вычитывать
их своим кодом и делать
на этом реактивные системы

Как НЕ надо делать CDC

Надеяться на то, что пользователи читают документацию

Вывод № 8

«Отгрузили данные в очередь – это уже ВНЕШНИЙ контракт»

Главный вывод

Вывод № 9

«Отделяй внешний контракт
от внутреннего представления»

6. Как НЕ надо делать CDC
7. Как надо делать CDC
8. С чем нужно быть готовым столкнуться на этом пути
9. Что обычно удивляет пользователей сценария CDC
10. Выводы

Как надо делать CDC

Какие выводы мы вынесли к текущему моменту:

- Нужно создать внешний (публичный) формат
- Нужно порешать технические нюансы

Как надо делать CDC

Что мы сделали:

- Мы взяли готовый формат у Debezium (drop-in replacement)
- Сделали конвертеры: из нашего формата в Debezium и обратно

Что мы получили

1. CDC для pg/mysql/ydb
2. Well-known формат, настройки
3. А стало быть, и все интеграции Debezium
4. Возможность коннектиться с «ванильным Debezium»
5. Всё это нативно в Data Transfer, со всеми фичами
6. Много дополнительных тестов для самого сервиса
7. Фреймворк для конверторов в Debezium и обратно

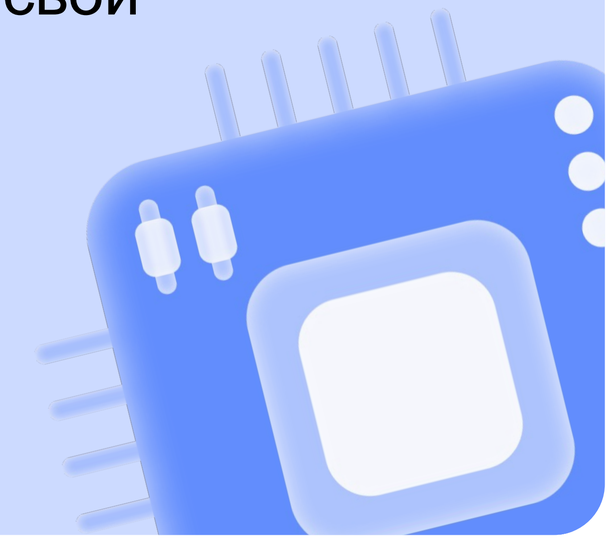
Что мы получили, чего нет в «ванильном Debezium»:

1. PostgreSQL – переживаем переезд мастера
2. PostgreSQL – user-defined types
3. Поддержка баз, которых нет в «ванильном Debezium» – YDB
4. Мы умеем обогащать данные исходными типами
5. Интеграцию с фичами Data Transfer
6. Интеграцию в экосистему Yandex Cloud

6. Как НЕ надо делать CDC
7. Как надо делать CDC
8. С чем нужно быть готовым столкнуться на этом пути
9. Что обычно удивляет пользователей сценария CDC
10. Выводы

Тесты per database

Нужно будет написать много тестов – и для каждой БД свои



Тесты per data types. Пример

Типы данных postgres				Типы данных kafka
bigint	date	macaddr	INT8	INT8
bit(N)	double precision	money	INT16	INT16
bit varying(N)	inet	numeric	INT32	INT32
boolean	int4range	numrange	INT64	INT64
bytea	int8range	oid	FLOAT32	FLOAT32
character(N)	integer	point	FLOAT64	FLOAT64
character	interval	citext	BOOLEAN	BOOLEAN
varying(N)	json	hstore	STRING	STRING
cidr	jsonb	real	BYTES	BYTES
daterange		smallint	ARRAY	ARRAY
		text	MAP	MAP
		tsrange	STRUCT	STRUCT

Тестировать типы данных нужно в 4 режимах

1

Снапшот

2

Репликация

3

Значения внутри массивов

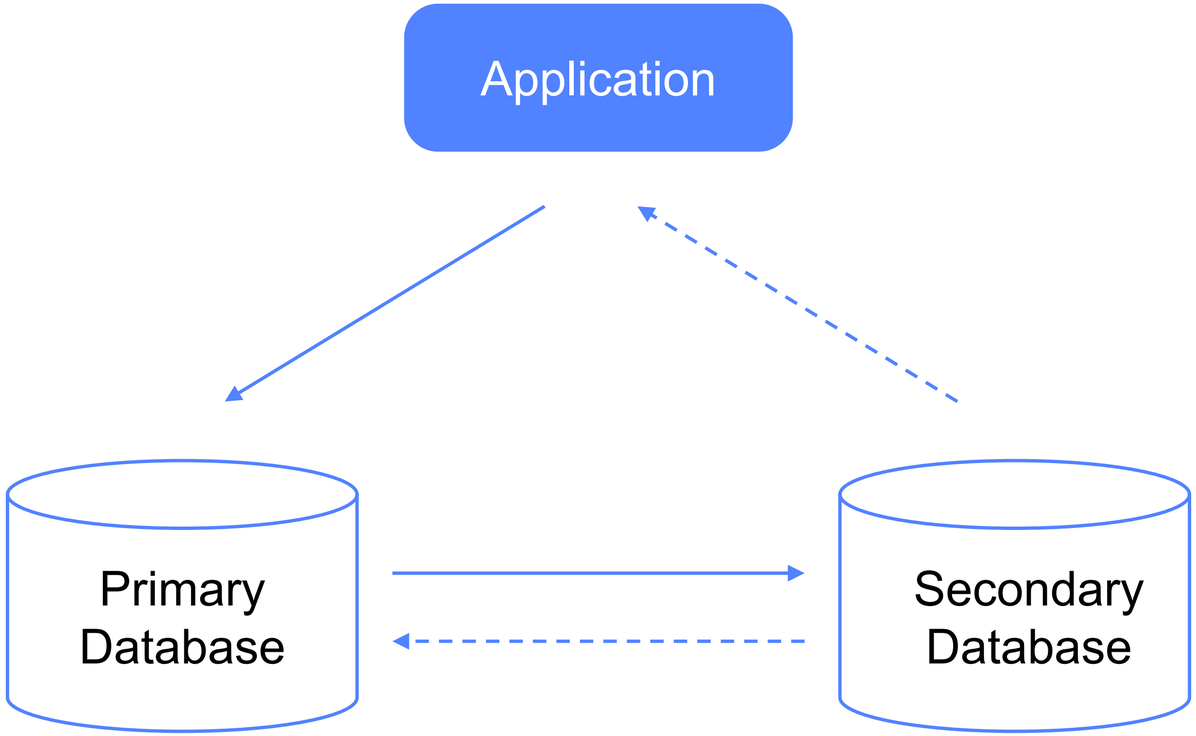
4

После сериализации-десериализации

Debezium не поддерживает типы с timezone – они молча конвертятся в UTC

6. Как НЕ надо делать CDC
7. Как надо делать CDC
8. С чем нужно быть готовым столкнуться на этом пути
9. Что обычно удивляет пользователей сценария CDC
10. Выводы

«Ванильный Debezium» не переживает failover в PostgreSQL



Дефолтное представление значений типа decimal

```
# значение:

decimal(5,2)
123.67
```

```
# значение:
ME8=

# в схеме:
{
    "field": "decimal_5_2",
    "name": "org.apache.kafka.connect.data.Decimal",
    "optional": true,
    "parameters": {
        "connect.decimal.precision": "5",
        "scale": "2"
    },
    "type": "bytes",
    "version": 1
}
```

Дефолтное представление значений типа decimal

Чтобы возить decimal/numeric типы строками

Настраивайте

decimal.handling.mode = string

«Ванильный Debezium» молча пропускает колонки с user-defined типами

Поведение поля before контролируется настройками базы источника

Почему?

Потому что Debezium декодирует
wal, а содержимое wal контролируется
настройками базы данных

Пример

На update из postgres
по дефолту поле before пустое

Но будет заполняться,
если выставить в таблицу:

```
REPLICA IDENTITY = FULL
```

Не забывайте про TOAST

Update, изменяющий primary key, генерирует два ивента: delete+create

6. Как НЕ надо делать CDC
7. Как надо делать CDC
8. С чем нужно быть готовым столкнуться на этом пути
9. Что обычно удивляет пользователей сценария CDC
10. Выводы

Выводы

1

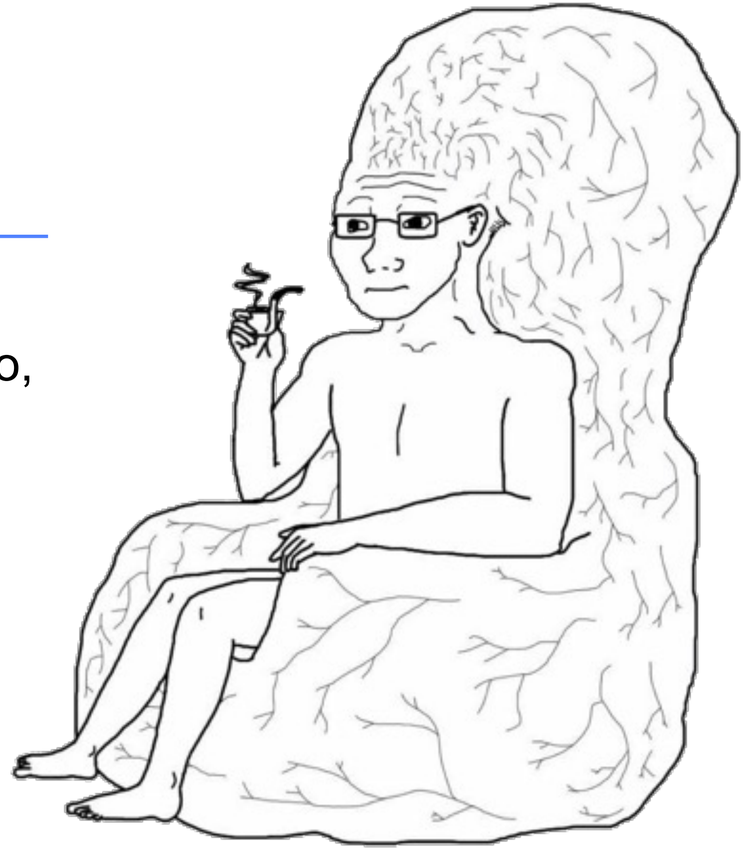
CDC подходит для многих задач и не так страшен

2

JSON как способ сериализации обладает рядом особенностей, помните о них

3

Написать своё решение можно, но не просто. Пользуйтесь готовыми решениями



Yandex Cloud

Вопросы

Тимофей Брунько

Разработчик Yandex Cloud

timmyb32r@gmail.com
telegram.me/timmyb32r