Фичестор Городских сервисов Яндекса

Павел Попов
Руководитель группы разработки
передовых технологий, Яндекс Go

infra.conf

1. **Что такое Feature Store?**
2. Рождение Avalon
3. UGCDB Framework
4. Proxy и Data Processor
5. Технические характеристики

Что такое Feature Store?

Feature Store — витрина признаков. Основными её задачами являются:

- Хранение признаков
- Каталогизация признаков

AVALONINT-57

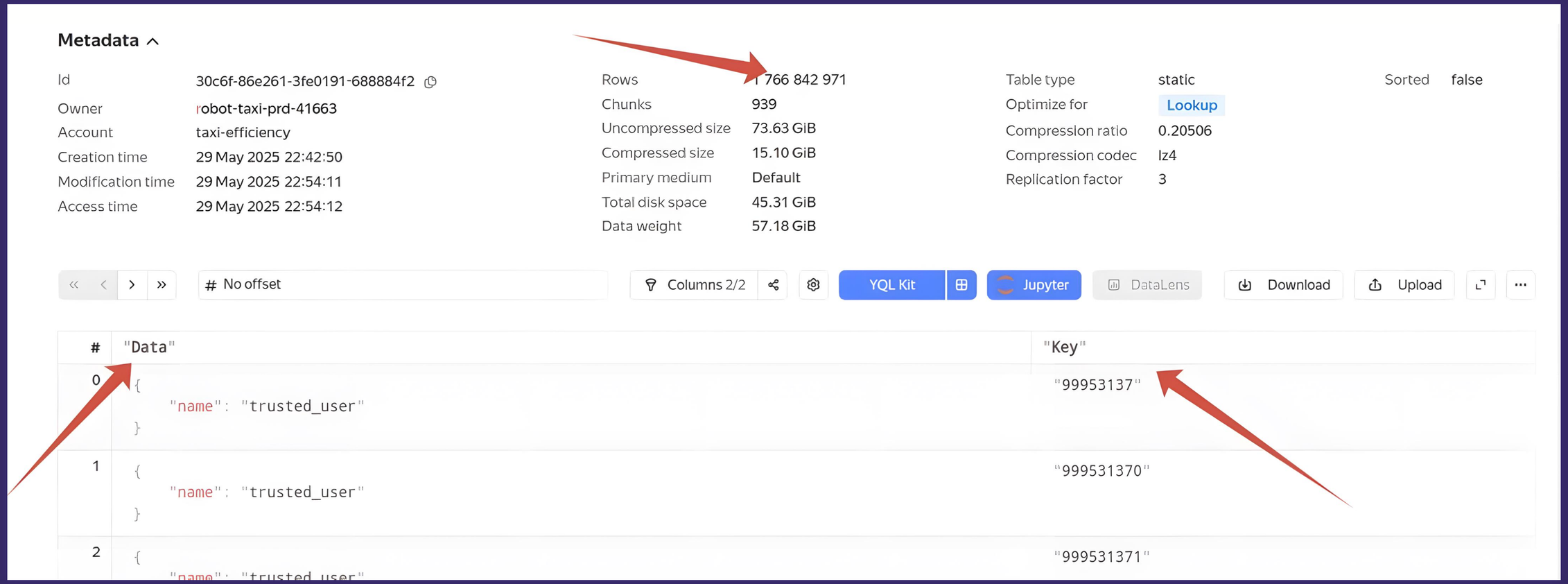
- Сегмент доверенных пользователей Маркета
- Хотим настроить поставку нового свойства доверенного пользователя на Маркете в Авалон, которое затем сможем использовать в новом маркетном перке.
- Антифрод раз в сутки размечает все паспортные аккаунты на предмет доверенности на Маркете, под капотом используя экосистемные фичи и разметку нарушителей сервиса. Хотим поставлять булев признак на пользователе – доверенный он или нет.
- **По объёмам:** чуть более 80% аккаунтов (1,65 млрд) признаём доверенными. В качестве варианта можем поставлять обратное свойство недовверенности на 403 млн пользователей.

AVALONINT-57

- Дифф по доверенным:
  не более 2 млн записей в сутки
- Предполагаемый объём данных (GiB | ключей): **1,65 млрд**
- Для какого идентификатора добавляется признак?: **passport_uid**
- Это экспериментальный признак?: **Нет**
- Сколько дней нужно хранить данные при отсутствии обновлений?: **30**
- Каков источник данных?: **YT**
- Потребуется ли первоначальная наливка историческими данными?: **Нет**

1. Что такое Feature Store?
2. **Рождение Avalon**
3. UGCDB Framework
4. Proxy и Data Processor
5. Технические характеристики

Принесли таблицу



Сервис тегирования аудитории

Прародитель Avalon, рождённый в такси:

- Построен на базе PostgreSQL
- Хранит бинарные признаки
- Удобный способ разметки данными для аналитиков
- Помог запустить не один десяток проектов в Екоме и Райдтехе

Плюсы

- Низкий Time To Market.
- Простой и привычный для аналитиков интерфейс.
- Исторически «доступен» практически в каждом уголке Екома и Райдтеха — для проверки практически любой гипотезы не требуется разработка.

Минусы

- PostgreSQL — ограниченные размеры, необходимо ручное шардирование, дешевле было развернуть новую инсталляцию, чтобы масштабировать.
  Итог — 5 инсталляций.
- Сложно отследить использование признака — сложно поддерживать актуальность признака и искать реальных потребителей.
- Абьюз сервиса: «бесконечные» разметки, «хардкод» в названиях признаков.

Требования к Avalon

- Единая точка входа для потребителей
- «Бесконечное» количество признаков
- Признаки могут быть произвольными данными
- Признаки должны поставляться из Runtime- и Offline-контуров
- Удобный реестр признаков
- Лёгкий и быстрый способ разметки аудитории для проверки гипотез или проведения экспериментов
- Возможность отслеживать потребление признаков

Как получать данные

```
taxi-tvm-curl --env testing avalon-proxy -v avalon-proxy.taxi.tst.yandex.net/v1/fetch-features -X POST -H "Content-Type: application/json" -d '{"id": "1000025037", "features": [“/market-general/{}/features/trusted_user"]}'
```


```
{
  "id": "1000025037",
  "features": {
    "/market-general/{}/features/trusted_user": {
      "data": {
        "@type": "type.googleapis.com/NUgc.NSchema.NAvalon.TGeneralFeature",
        "name": "trusted_user"
      }
    }
  }
}
```

Домен одной ручки

1. Что такое Feature Store?
2. Рождение Avalon
3. **UGCDB Framework**
4. Proxy и Data Processor
5. Технические характеристики

Философия фреймворка

- KV хранилище данных
- если запись должна быть в единственном экземпляре — то это колонка по этому пути
- если данных с типом несколько — это таблица
- внутри таблицы могут быть таблицы

Достоинства UGCDB

Хранилище

- YDB
- Иерархическое Key Value
- Удобное шардирование

Схема данных

- Описывается Protobuf
- Логическая схема данных
- Одна таблица

API

- Обобщённое API генерируется по схеме данных

CDC

- Внешняя реализация
- Возможность гибкой настройки

Физическая схема таблицы

```
Uint32 ShardId;      // Логический номер шарда
String Key;          // Ключ данных
String ValueProto;   // Сообщение, сериализованное в бинарном виде
Json   ValueJson;    // Сообщение, сериализованное в JSON-формате
Timestamp CreateTime; // Время создания записи
Timestamp UpdateTime; // Время последнего обновления записи
Timestamp ExpireAt;   // Время, после которого запись автоматически удалится
Uint64 __tablet;     // Системное, номер таблетки
Uint64 __updateEpoch; // Системное, счётчик (сбрасывается на изменениях таблетки)
Uint64 __updateNo;    // Системное, счётчик (общий для всех записей)
```

Логическая схема данных

```
message TRoot {
  repeated TGeneralFeatureStorage MarketGeneral = 1;

  option (schema) {
    Table: {
      Name: ”market-general”
      Field: ”MarketGeneral”
      Key: ”Name”
    }
  }
};
```

Логическая схема данных

```
message TGeneralFeatureStorage {
    string Id = 1 [json_name = "id"];
    repeated TGeneralFeature Features = 2 [json_name = "features"];

    option (schema) = {
        Table: {
            Name: "features"
            Field: "Features"
            Key: "Name"
        }
    };
}
```

Логическая схема данных

```
message TGeneralFeature {
    string Name = 1 [json_name = "name"];
    google.protobuf.Value Payload = 2 [json_name = "payload"];
}
```

Примеры ключей

/market-general/123/features/trusted_users//
Признак Маркета trusted_users для пользователя 123
/market-general/123/features// Все признаки Маркета
для пользователя с id 123

API

Create/Update

PUT + CAS:

- IfNotExist
- IfEqualTo

Read

GET:

- Точечное
- Range-based

Delete

DELETE:

- Точечное
- Range-based

Batch

POST:
Вариант комбинации операций,
опционально в одной транзакции

Точечное чтение

```
DECLARE $shard_id AS Uint32; // CityHash(root level id -> 123)
DECLARE $key_id AS String;   // ”/market-general/123/features/trusted_users“
SELECT ShardId, Key, ValueProto
FROM `data`
WHERE ShardId = $shard_id AND Key = $key;
```

Примеры ключей:

Доверенные пользователи

/market-general/123/features/trusted_users

Те, кто сейчас не подписан на пуши и не является "недавним" отписчиком.

Будем по этому тэгу показывать поп-ап "подпишись на наши пуши"

/market-general/123/features/crm_exp

```
{
   "name": "crm_exp",
   "payload": {
      "suffixes": [
         "lidogeneration"
      ]
   }
}
```

Range-based-чтение

```
DECLARE $shard_id AS Uint32;    // CityHash(root level id -> 123)
DECLARE $range_start AS String; // “/market-general/123/features/”
DECLARE $range_end AS String;   // “/market-general/123/features/~” – тильда максимальный код ascii
SELECT ShardId, Key, ValueProto
FROM `data`
WHERE ShardId = $shard_id AND Key >= $range_start AND Key < $range_end;
```

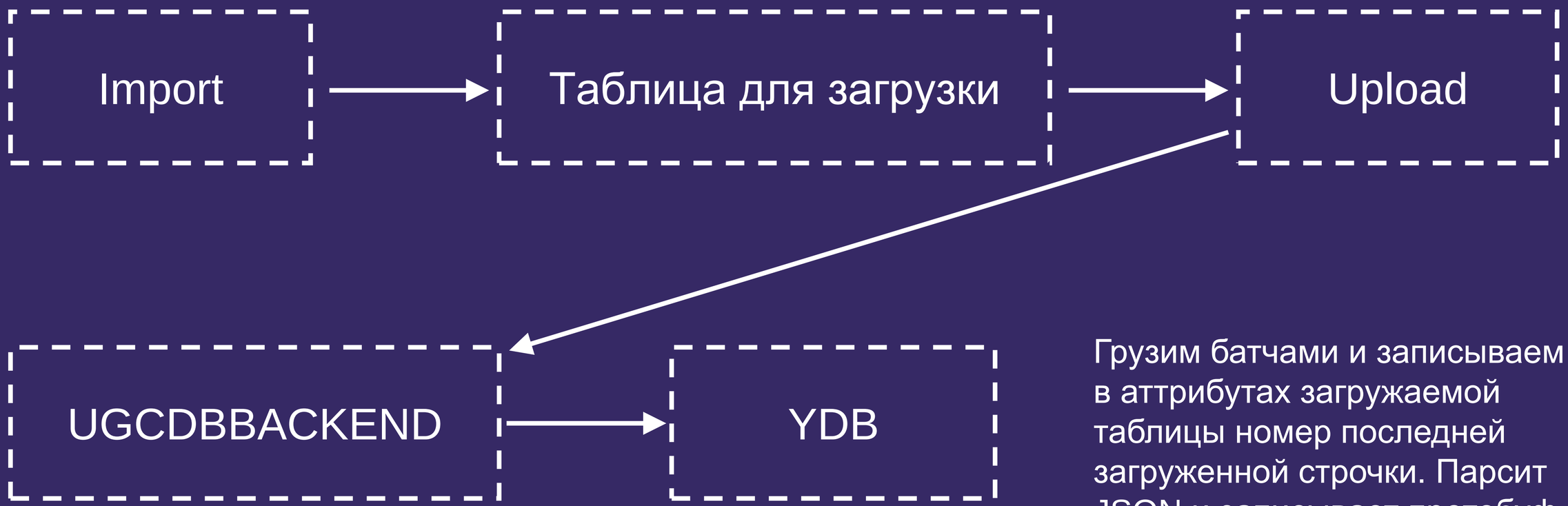
Batch-чтение

```
DECLARE $shard_id AS Uint32;
DECLARE $key_list AS List<Tuple<Uint32?,String?>>;
SELECT ShardId, Key, ValueProto
FROM `data`
WHERE (ShardId, Key) IN $key_list;
```

Модифицирующие запросы

```
// Запись
DECLARE $shard_id AS Uint32;
DECLARE $key AS String;
DECLARE $value AS String;
...
UPSERT INTO `data` (ShardId, Key, ValueProto, ...)
VALUES ($shard_id, $key, $value, ...);
// для IfNotExist и IfNotEqual используются интерактивные транзакции
// логика их обработки реализована на уровне приложения
// Удаление
DECLARE $shard_id AS Uint32;
DECLARE $key AS String;
DELETE FROM `data` WHERE ShardId = $shard_id AND Key = $key;
```

Загрузка данных



Грузим батчами и записываем в аттрибутах загружаемой таблицы номер последней загруженной строчки. Парсит JSON и записывает протобуф

1. Что такое Feature Store?
2. Рождение Avalon
3. UGCDB Framework
4. **Proxy и Data Processor**
5. Технические характеристики

Avalon Proxy

Единая точка входа для чтения

Позволяет инкапсулировать
несколько различных хранилищ

API

Усечённое API для гарантии
одношардовости
пользовательских запросов

ACL

Динамическая конфигурация
доступа потребителей

Nice to have

В будущем хотелось бы работать из неё
напрямую с YDB для лучших таймингов

Avalon Data Processor

Единая точка входа для чтения

Позволяет контролировать поступающие данные

Админка

Хранит данные каталога признаков и внутренние параметры импорта офлайн-данных

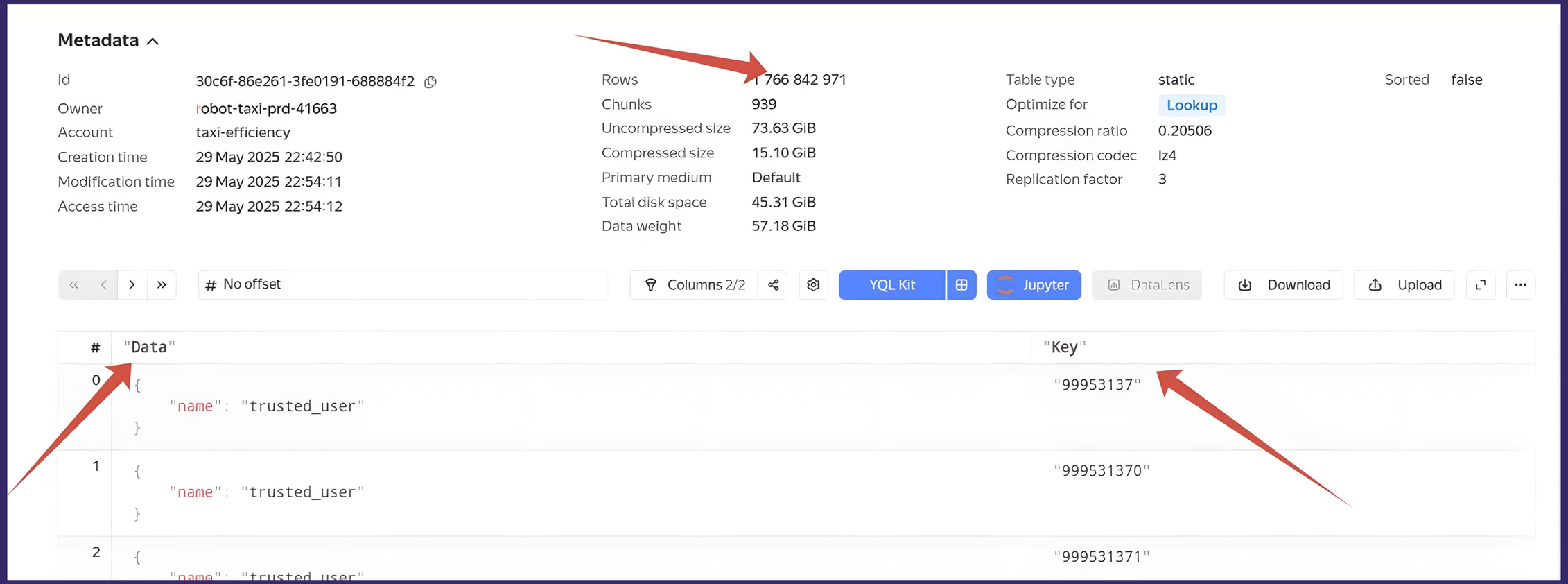
ACL

Динамическая конфигурация доступа потребителей

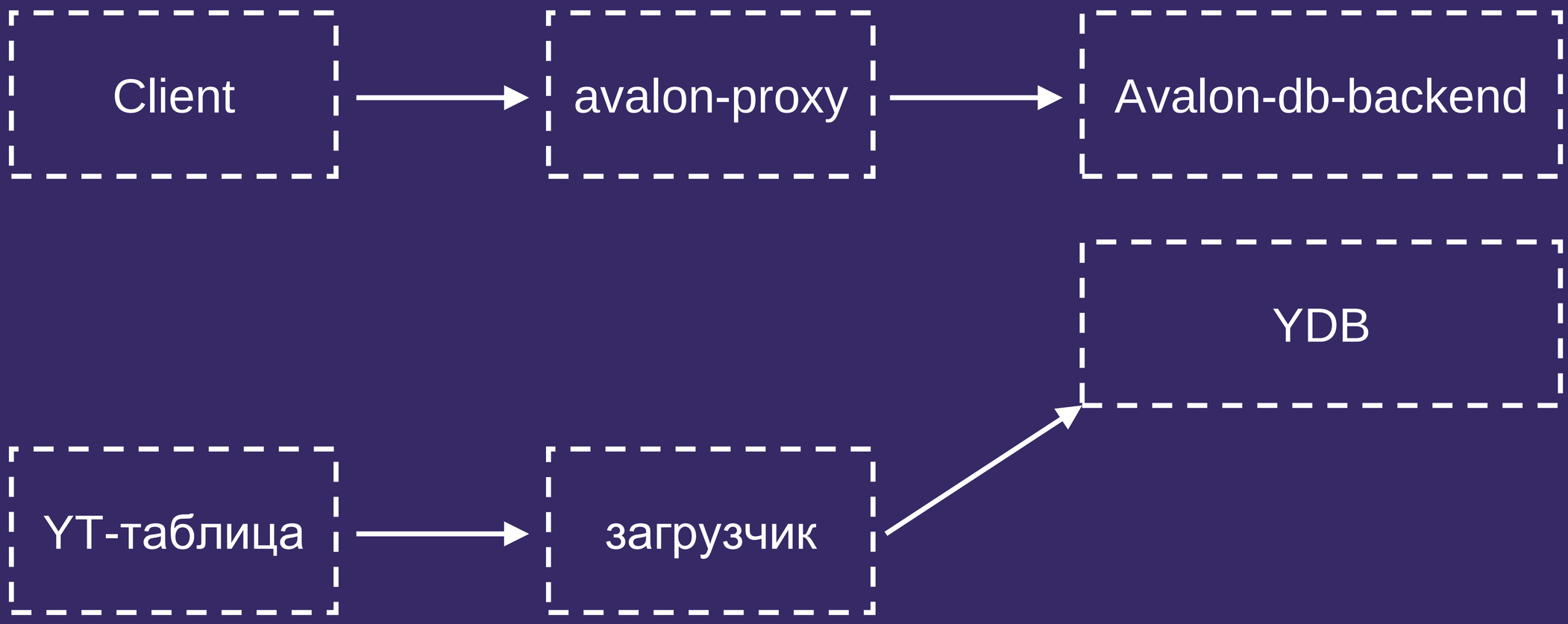
Nice to have

В будущем хотелось бы работать из неё напрямую с YDB для лучших таймингов

Принесли таблицу



Загрузили данные и можем их получать



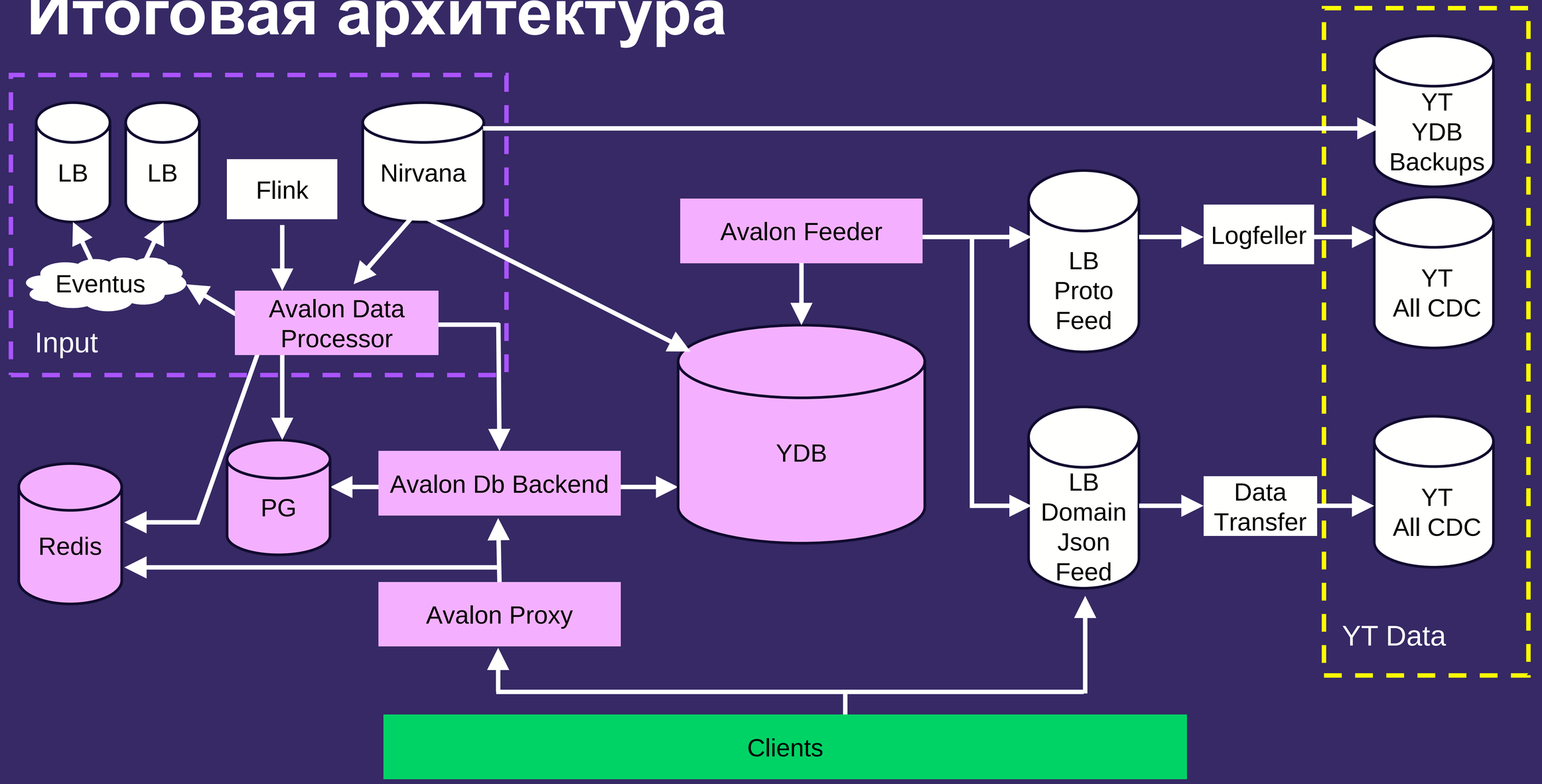
Даём доступы

```
{
"market-loyalty": {
    "/market-general/{}/features": {
    }
  },
  "market-perks": {
    "/market-general/{}/features": {
    },
    "/passport-uids/{}/market-payment-methods": {
    },
    "/passport-uids/{}/market-prev-business-purchases": {
    },
    "/passport-uids/{}/personal-goals": {
    }
  },
}
```

Счастливые пользователи

```
std::vector<std::string> AvalonProxyService::GetAvalonGeneralFeatures() const {
    auto response_opt = GetGeneralFeaturesData(avalon_get_features_task_);
    if (!response_opt) {
        return {};
    }
    auto response = response_opt.value();
    std::vector<std::string> result;
    result.reserve(response.GetFeatures().size());
    for (const auto& feature : response.GetFeatures()) {
        if (feature.GetName() == kMarketGenderPromo) {
            continue;
        }
        if (feature.HasPayload()) {
            const auto& payload = feature.GetPayload();
            if (payload.has_struct_value()) {
                const auto& protoMap = payload.struct_value().fields();
                const auto* suffixes = MapFindPtr(protoMap, "suffixes");
                if (suffixes) {
                    if (suffixes->has_list_value()) {
                        for (const auto& suffix : suffixes->list_value().values()) {
                            if (suffix.has_string_value()) {
                                result.push_back(feature.GetName() + '_' + suffix.string_value());
                                boost::to_upper(result.back());  // For compatibility with tags
                            }
                        }
                    }
                    continue;
                }
            }
        }
        result.push_back(feature.GetName());
        boost::to_upper(result.back());  // For compatibility with tags
    }
    return result;
}
```

Итоговая архитектура



Что если нам нужна история изменений?

```
message TRoot {
repeated TGeneralFeatureStorage DiscountsGeneral = 1 [(topic) = "discounts-general-json"];
 option (schema) {
  Table: {
        Name: "discounts-general"
        Field: "DiscountsGeneral"
        Key: "Id"
    }
}
```

Что если нам нужна история изменений?

```
message TGeneralFeatureStorage {
    string Id = 1 [json_name = "id"];
    repeated TGeneralFeature Features = 2 [json_name = "features"];


    option (schema) = {
        Table: {
            Name: "features"
            Field: "Features"
            Key: "Name"
        }
    };
}
```

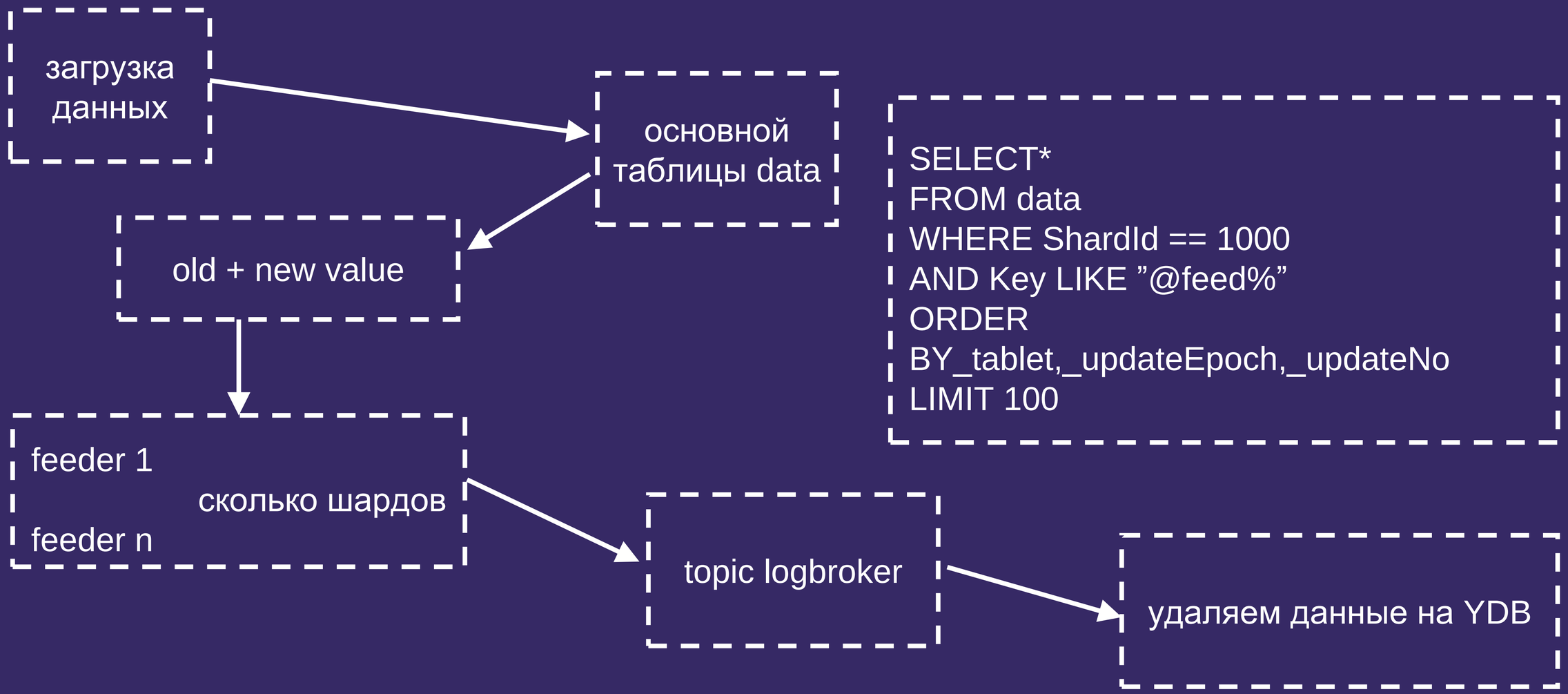
Что если нам нужна история изменений?

```
message TGeneralFeature {
    string Name = 1 [json_name = "name"];
    google.protobuf.Value Payload = 2 [json_name = "payload"];
}
```

Старое и новое значение — тоже признак

```
message TFeedRecord {
    string Key = 1;
    TRecordMeta OldMeta = 2;
    google.protobuf.Any OldValue = 3;
    TRecordMeta NewMeta = 4;
    google.protobuf.Any NewValue = 5;
    TRequestMeta RequestMeta = 6;
}
```

Feeder


загрузка данных
основной таблицы data
old + new value
SELECT*
FROM data
WHERE ShardId == 1000
AND Key LIKE ”@feed%”
ORDER
BY_tablet,_updateEpoch,_updateNo
LIMIT 100
feeder 1
сколько шардов
feeder n
topic logbroker
удаляем данные на YDB

Внешний CDC aka Feeder

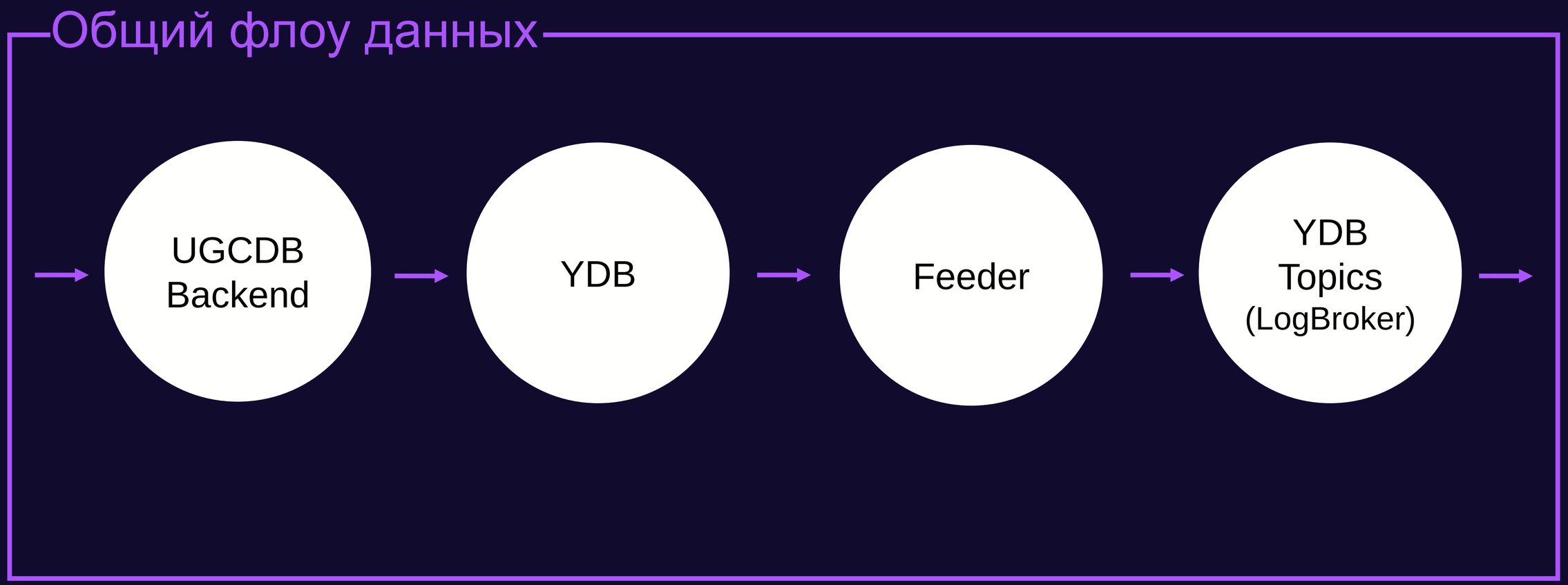
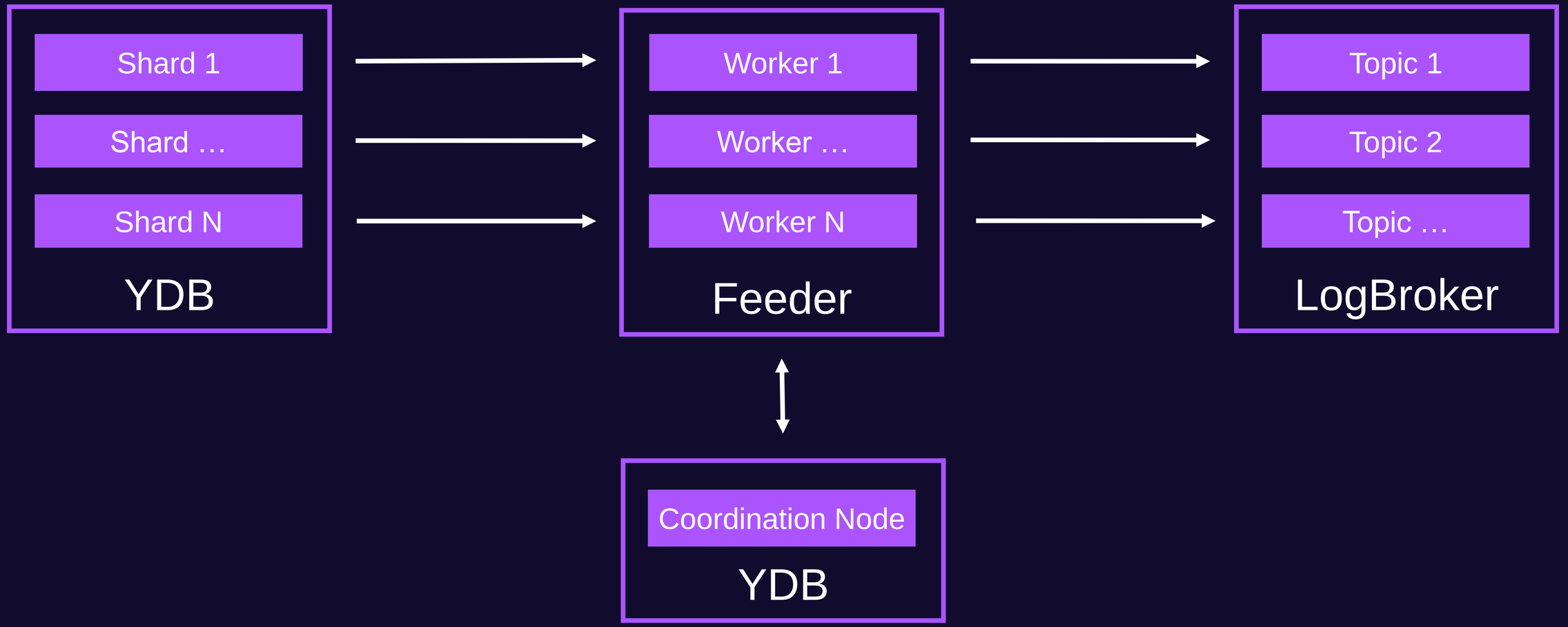


Схема устройства Feeder



Feeder Tips&Tricks

DbBackend

- Гарантирует одновременную запись данных и диффа
- Гарантирует запись данных и диффа в один шард
- Гарантирует, что запись диффа будет иметь минимальное значение ShardId на таблетке

Feeder

- Читает только начало шарда
- Удаляет запись диффа из базы только при успешной записи в LogBroker
- Позволяет писать данные в топик как в бинарном виде, так и в JSON
- Гарантирует Exactly Once за счёт номеров обновления (updateEpoch и updateNo)

Ограничения

Данная реализация фидера не позволяет сохранить возможность шардирования по нагрузке, поэтому остаётся только шардирование по размеру шарда, при этом без обратного мержа, так как в случае мержа мы получим записи фида в середине шарда и не сможем их найти

1. Что такое Feature Store?
2. Рождение Avalon
3. UGCDB Framework
4. Proxy и Data Processor
5. **Технические характеристики**

Данные о нагрузке

~3 ТВ

Объём данных

~6,5 В

Количество ключей

2048

Шардов

Данные о нагрузке

100k

RPS на чтение

5ms

p95 на чтение

30k

RPS на запись,
в т.ч. RW-транзакции

Спасибо за внимание!

Павел Попов
Руководитель
группы разработки
передовых технологий,
Яндекс Go

Задайте вопросы спикеру
в чате мероприятия: