Yandex Cloud

Schema Registry: Ultimate Guide

Тимофей Брунько,
разработчик Yandex Cloud

Тимофей Брунько

2010

Устраиваюсь malware analyst

2012

Занимаюсь R&D на C++. Задачи поиска данных

- Inverted index
- In-memory databases
- Nearest neighbor
- Graph search
- Распознавание образов
- Кластеризация
- Оркестрация микросервисов
- DSL и ANTL
- Распределённые системы

2018

Присоединяюсь к Yandex, в отдел поставки данных

2020

Начинаю работать над Data Transfer в Yandex Cloud

2023

Разрабатываю Yandex MetaData Hub (Schema Registry)

Что узнает аудитория из доклада

- Как люди, решая разные проблемы, подошли с разных сторон к идее Schema Registry, и что получилось
- Какие проекты, реализующие концепцию Schema Registry существуют
- Как бинарно устроены форматы: avro, protobuf
- Что такое «совместимость» для avro, protobuf, json_schema
- Алгоритмы проверок обратной совместимости

Какой боли поможет избежать Schema Registry для дата-инженера?

Пример №1

- У вас есть работающий пайплайн обработки данных
- Схема данных ODS-слоя (aka staging-area) описывается, например, protobuf’ом
- Сотрудник решает, что существующее поле типа int64 никогда не содержит отрицательных значений — и меняет его на uint64

Если использовать Schema Registry — подобного не произойдёт

Какой боли поможет избежать Schema Registry для дата-инженера?

Пример №2

- У вас есть поставка данных: kafka→custom_consumer, которая работает годами
- Custom_consumer — самописный, который предполагает, что данные в очереди всегда валидны, и если что-то не так — retry
- Сотрудник что-то перепутал, и записал в очередь строчку «test»
- Весь последующий пайплайн встаёт, потому что тут бесконечно ретраят парсинг строчки «test»

Если использовать Schema Registry — подобного не произойдёт

Какой боли поможет избежать Schema Registry для дата-инженера?

Пример №3

- Нужные вам данные лежат в очереди, и вы хотите сделать обработку потока
- Вы долго выясняете, каким же способом эти данные сериализованы
- Оказывается, что данные сериализованы кастомным форматом, парсер этого формата реализован только для C++

Если использовать Schema Registry — подобного не произойдёт

План доклада

1. Схемы и форматы сериализации данных
2. Schemaless- и schemaful-подходы
3. Что такое Schema Registry (SR)
4. Применения SR
5. Существующие SR
6. Нюансы
7. Форматы
   - Avro
   - Protobuf
   - JSON schema
   - Other
8. Что использовать в очередях — Avro, Protobuf или JSON?
9. О своей разработке
10. Дополнительные материалы

Условное обозначение

1. Схемы и форматы сериализации данных

1. Схемы и форматы сериализации данных

Если упростить, то:

- Схема данных — это описание структуры документов
- Обычно это логическая структура данных — какие поля у нас есть в данных, и каких они типов
- Формат сериализации данных — это способ конвертации сообщения в байтики

1. Схемы и форматы сериализации данных

Схема данных на примере golang-структуры

```
type Record struct {
    Name string
    Age int64
    City string
}
```

1. Схемы и форматы сериализации данных

Схема данных на примере golang-структуры

Как только вы объявили структуру данных:

- Можно вывести типизированную схему RDBSM/map-reduce таблицы, где каждый инстанс структуры — это строчка в таблице
- Можно гонять на этой таблице SQL-запросы

```
type Record struct {
    Name string
    Age int64
    City string
}
```

Name	Age	City
Gordon Freeman	47	city-17
G-man	49	city-18

```
SELECT * FROM Record WHERE City='city-17';
```

1. Схемы и форматы сериализации данных

Схема данных и формат сериализации на примере AVRO (.Avsc файл)

```
{
"type": "record",
 "name": "Record",
 "fields": [
  {
   "name": "Name",
   "type": "string"
  },
  {
   "name": "Age",
   "type": "long"
  },
  {
   "name": "City",
   "type": "string"
  }
 ]
}
```

1. Схемы и форматы сериализации данных

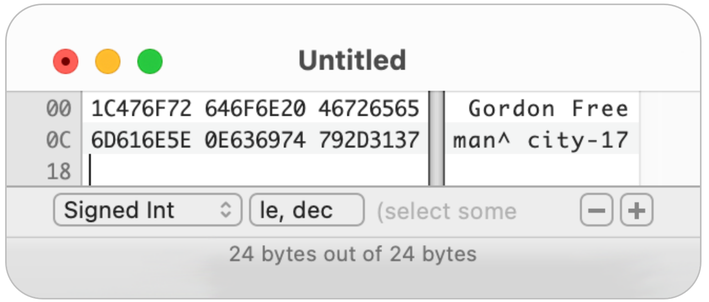
Схема данных и формат сериализации на примере AVRO

```
{
"type": "record",
 "name": "Record",
 "fields": [
  {
   "name": "Name",
   "type": "string"
  },
  {
   "name": "Age",
   "type": "long"
  },
  {
   "name": "City",
   "type": "string"
  }
 ]
}
```

- Создаём инстанс такой структуры
- Заполняем следующими значениями

```
{
  "Name": "Gordon Freeman",
  "Age": 47,
  "City": "city-17"
}
```

Сериализуем



1. Схемы и форматы сериализации данных

Схема данных и формат сериализации на примере protobuf (.proto файл)

```
syntax = "proto3";

message Record {
    string Name = 1;
    int64 Age = 2;
    string City = 3;
}
```

1. Схемы и форматы сериализации данных

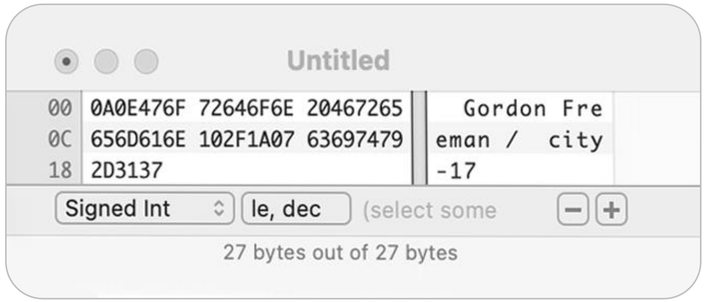
Схема данных и формат сериализации на примере protobuf

```
syntax = "proto3";

message Record {
    string Name = 1;
    int64 Age = 2;
    string City = 3;
}
```

- Создаём инстанс такой структуры
- Заполняем следующими значениями

```
{
    "Name": "Gordon Freeman",
    "Age": 47,
    "City": "city-17"
}
```

Сериализуем



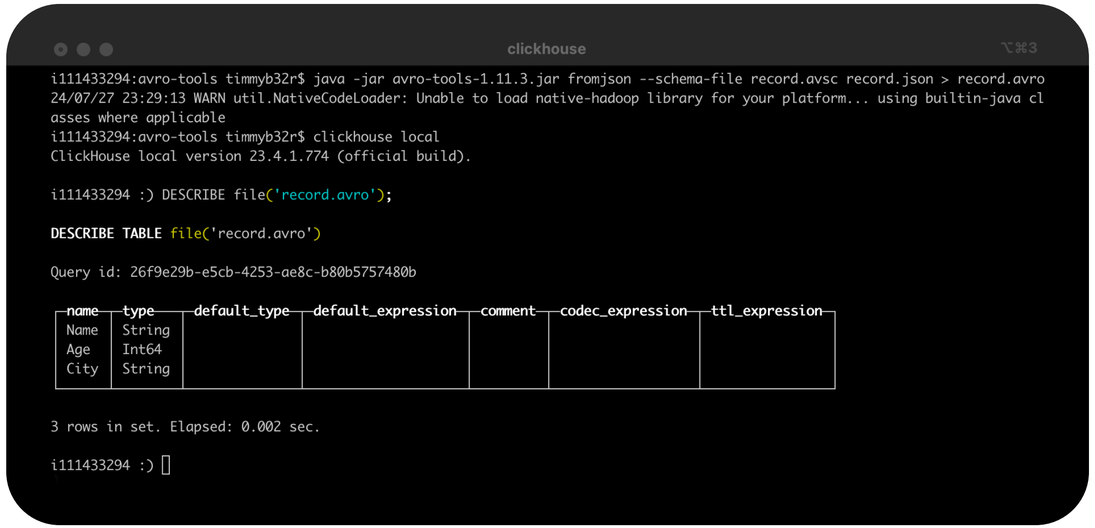
1. Схемы и форматы сериализации данных

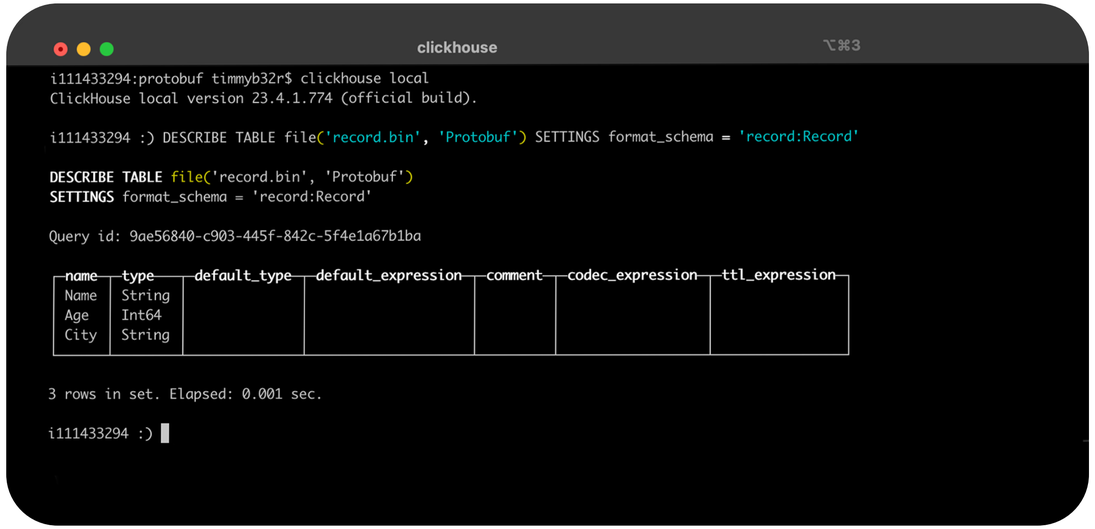
Итого:

- Мы увидели 3 разных способа описать одну и ту же схему данных
- Одна и та же схема данных, в разных форматах сериализовалась по-разному

1. Схемы и форматы сериализации данных

При переходе в DWH различия между форматами сериализации исчезают, важна лишь схема данных





Схематизация данных и возможность использовать SQL тесно связаны

На уровне SQL — деталей форматов сериализации уже нет

Wire —
это про байтики

2. Schemaless и schemaful-подходы

2. Schemaless и schemaful-подходы

2010-е — как всё начиналось:

- Появление MongoDB
- Пик популярности NoSQL как подхода
- Hadoop как стандарт де-факто big data
- Набирает популярность python, dynamic typing is new metaprogramming

2. Schemaless и schemaful-подходы

2020-е — как всё закончилось:

- MongoDB — в 2015-м научилась валидировать схемы документов
- NoSQL пришел к NewSQL
- Для Hadoop стандартом стал hive (SQL), Apache Pig забыт
- Python стал типизированным

2. Schemaless и schemaful-подходы

Почему так?

- Схема — это контракт
- Контракты делают наши системы:

  Надёжными

  Предсказуемыми

  Проверяемыми

2. Schemaless и schemaful-подходы

Дата-инженеры делятся на два типа:

- Те, кто пока ещё не схематизирует данные
- Те, кто уже схематизирует данные

3. Что такое Schema Registry?

3.1 Что такое Schema Registry — определение

Дословный перевод — реестр схем

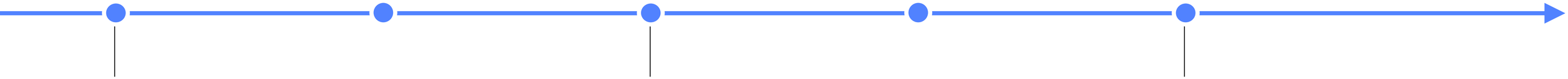
Сервис, реализующий две функции:

- Хранение схем
- Проверку обратной совместимости изменений

CRUD, где на «U» есть проверка совместимости изменений

3.2 Что такое Schema Registry — история

- К 2012–2014 годам в индустрии сформировался запрос на подобный инструмент
- Что этому помогло — AVRO формат вообще невозможно однозначно десериализовать не зная схемы, а AVRO стал стандартом де-факто передачи схематизированных данных в очередях



2012 Jun.11

В jira проекта Apache Avro появляется тикет «Restful service for holding schemas»

2013 Jan.24

Появляется LinkedIn camus schema registry (confluent был интегрирован с camus и контрибьютили в него)

2014 Jul.01

Релиз snowplow Iglu Schema Registry

2014 Dec.09

Первый коммит в Confluent Schema Registry

2015 Sep.24

Статья «Yes, Virginia, you really do need a Schema Registry» — confluent начинает рекламировать свой Schema Registry

4. Применения Schema Registry

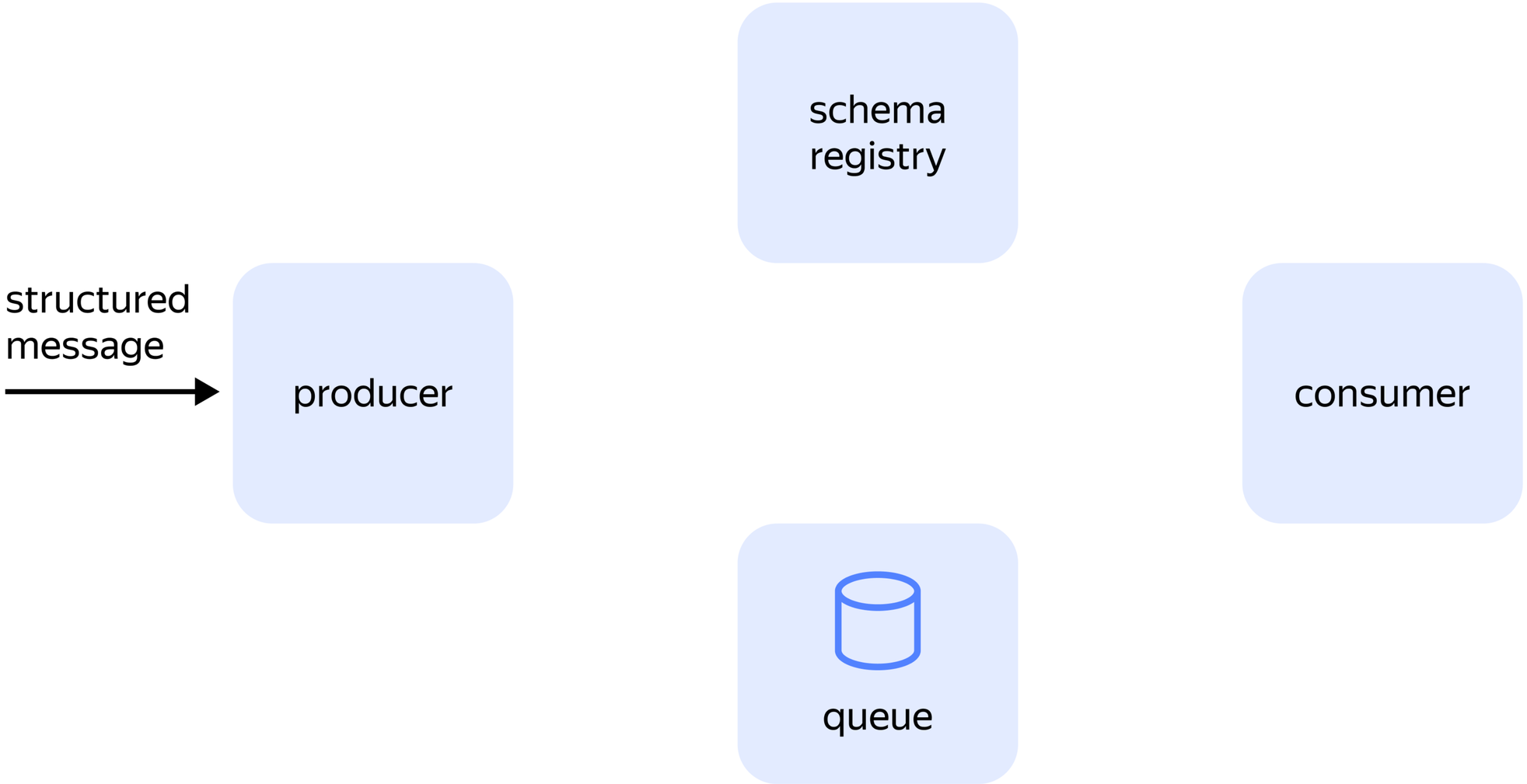
4. Применения Schema Registry

Все schema registry
на рынке делятся
на 2 области применения:

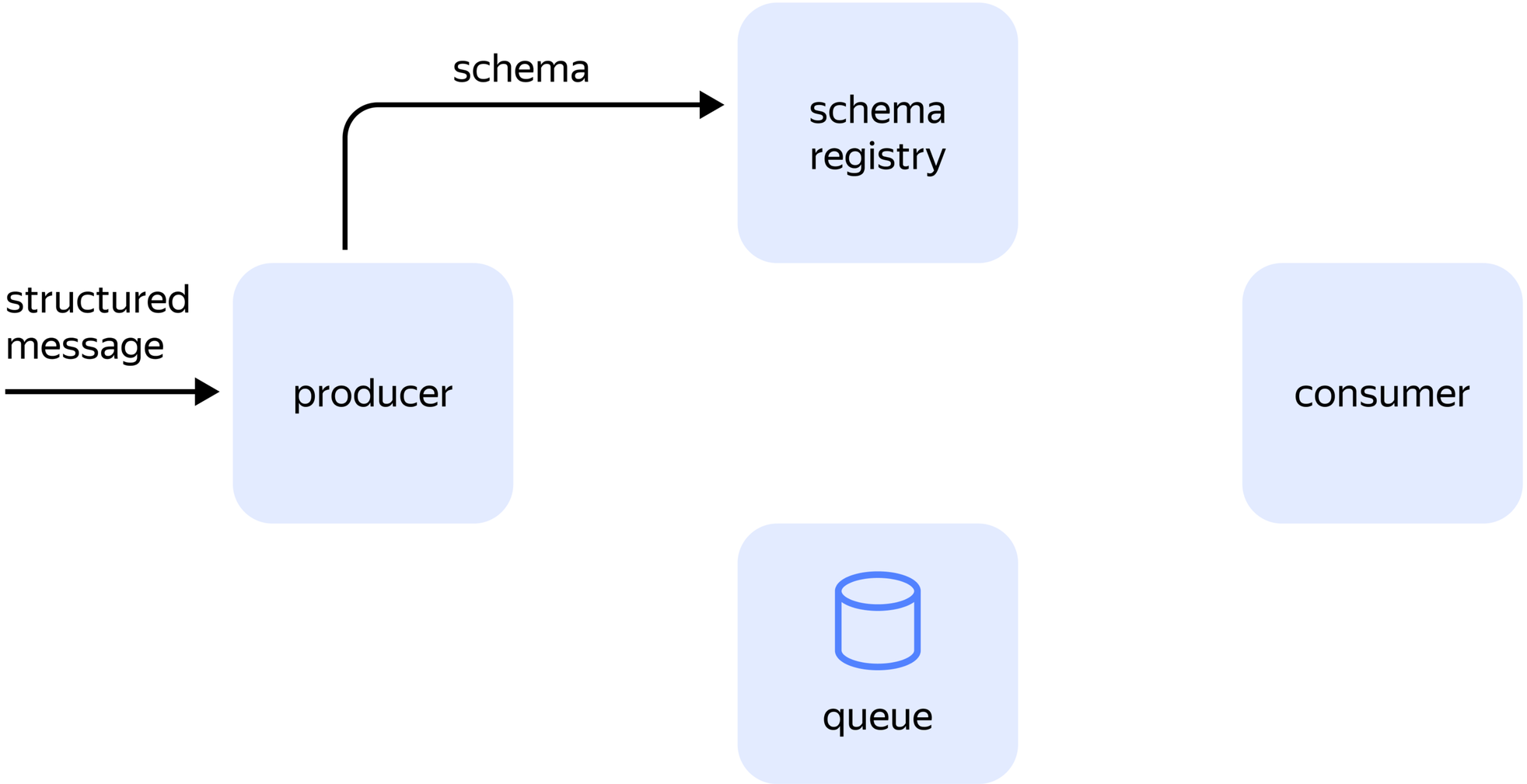
- Schema Registry
  для очередей
- Schema Registry для API

4.1 Schema Registry для очередей

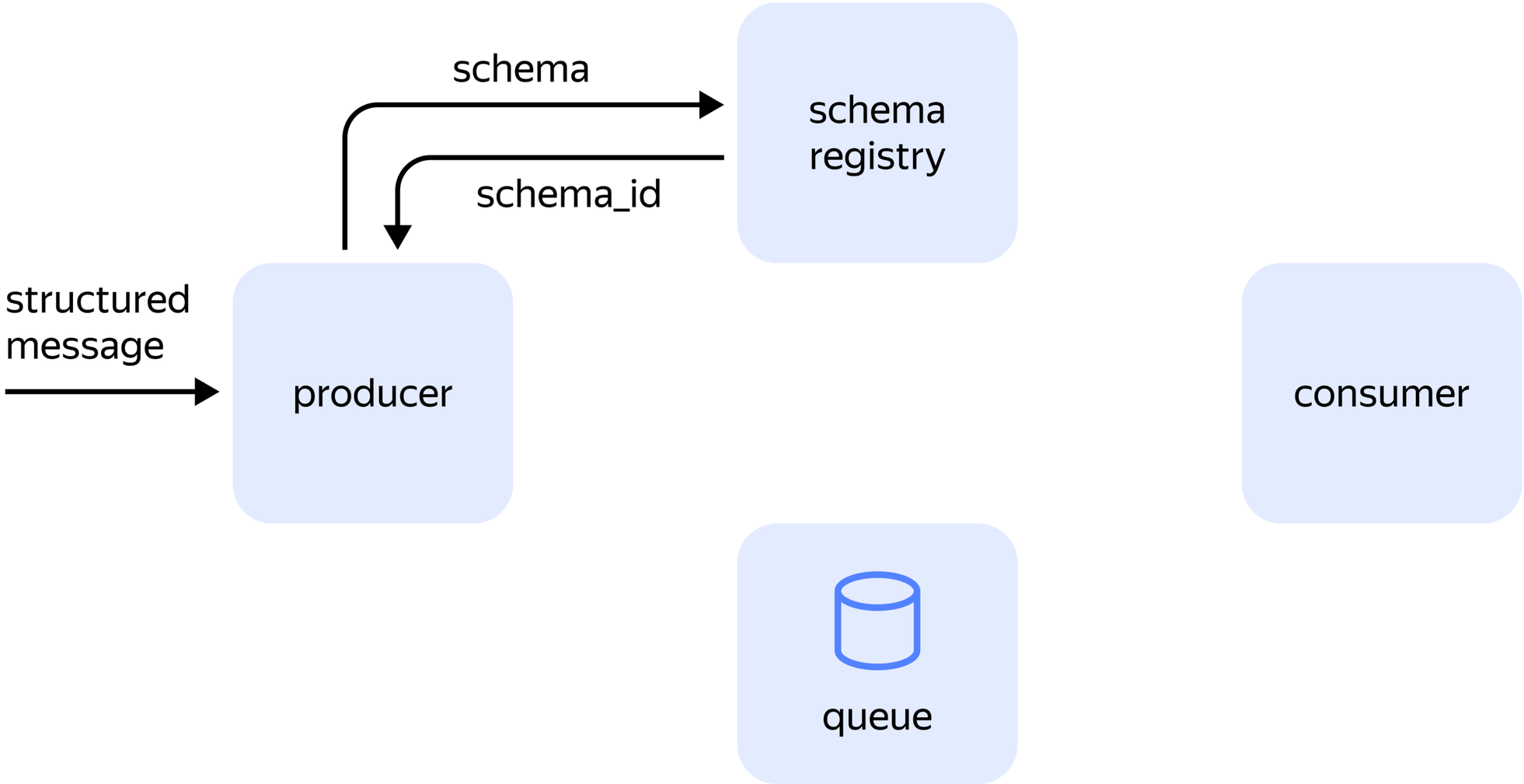
4.1 Schema Registry для очередей



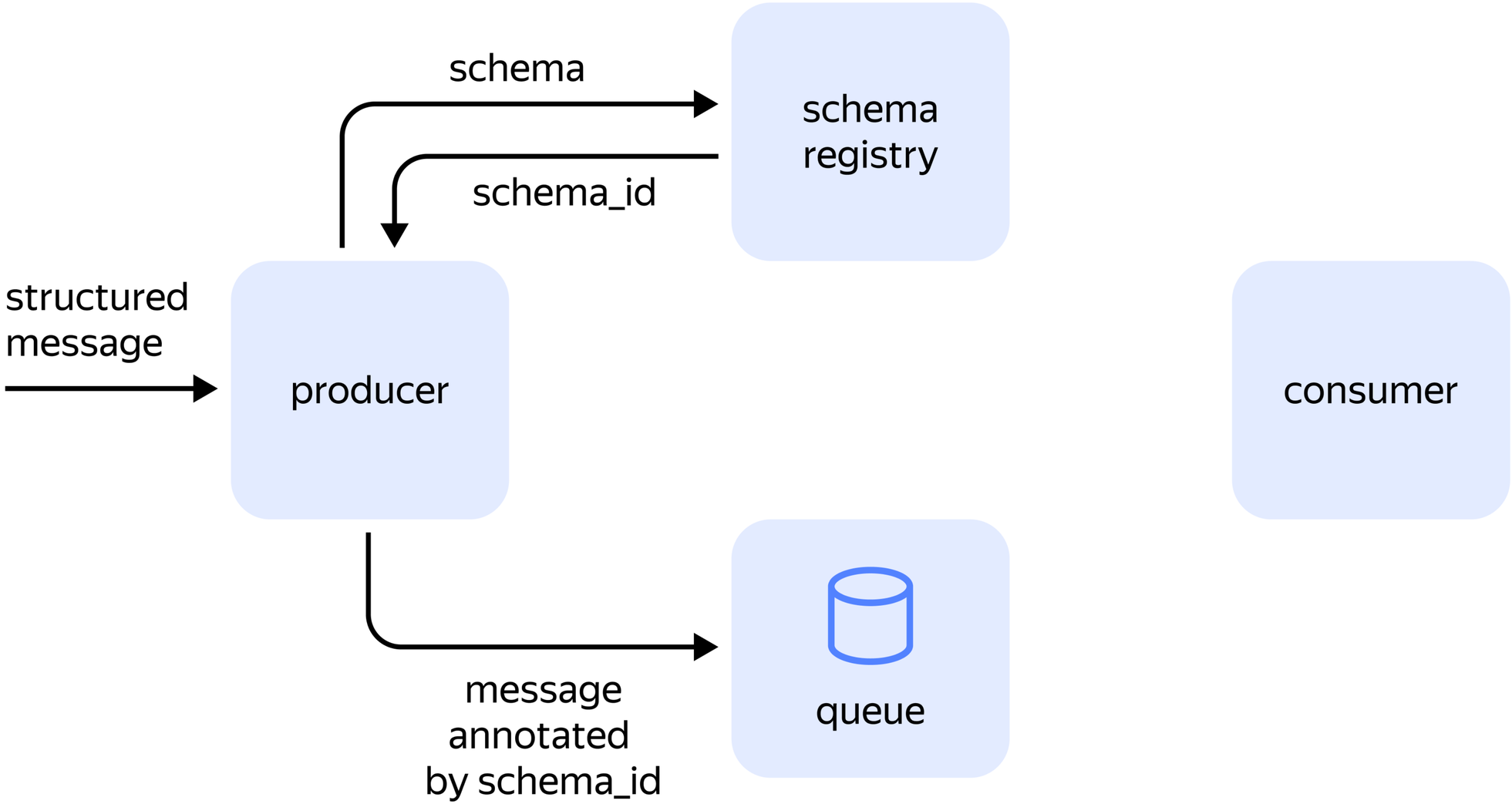
4.1 Schema Registry для очередей



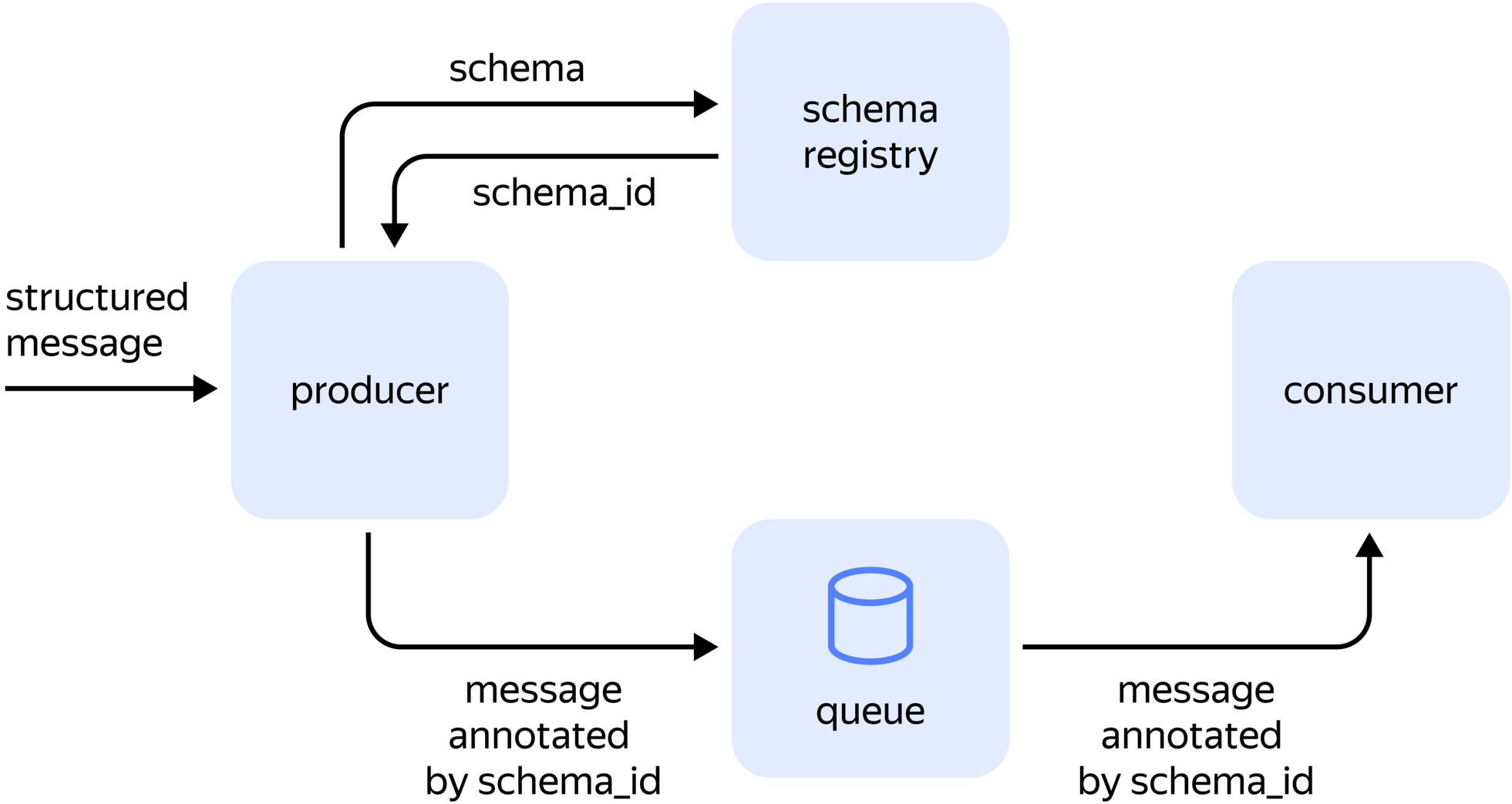
4.1 Schema Registry для очередей



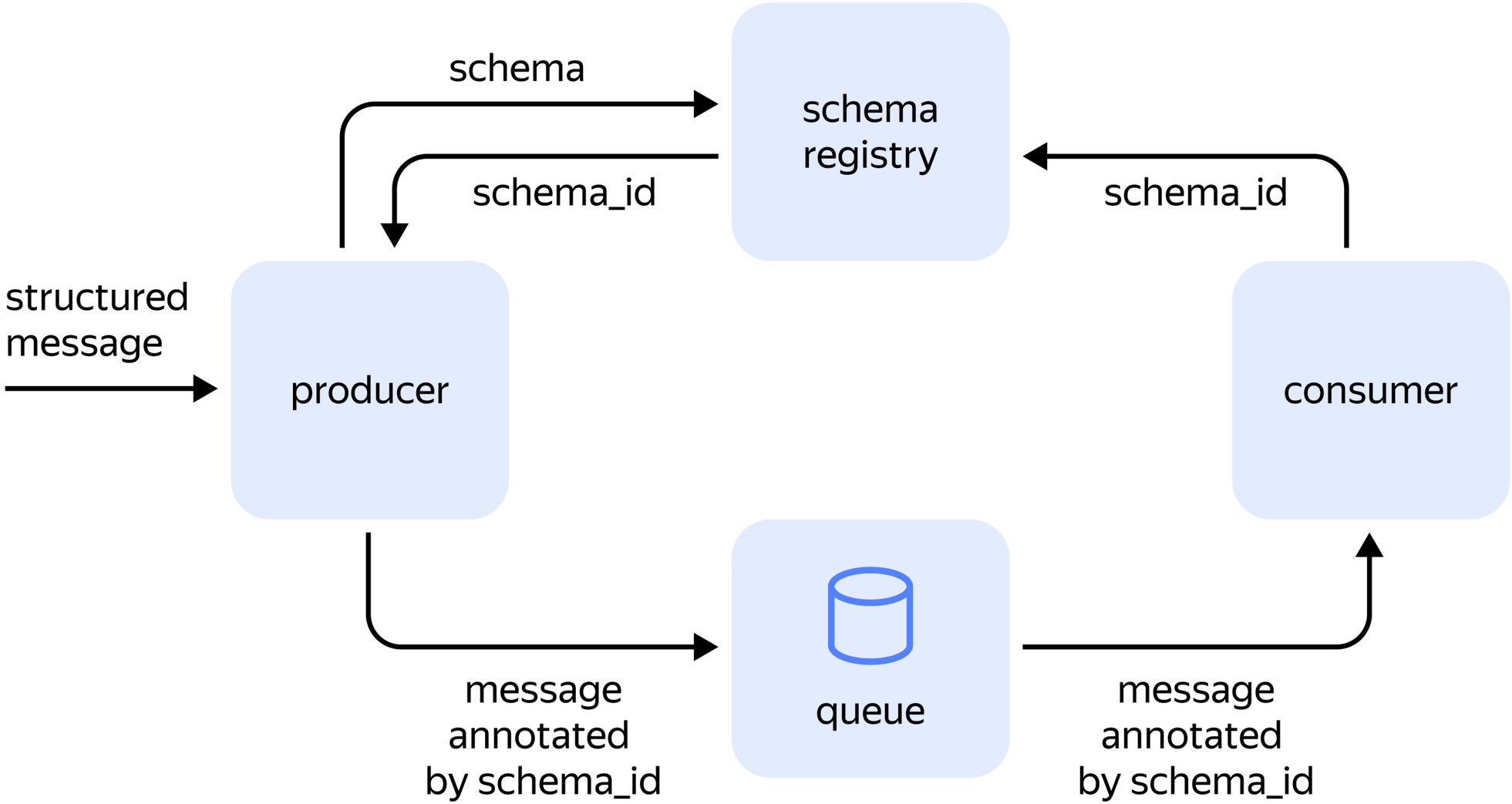
4.1 Schema Registry для очередей



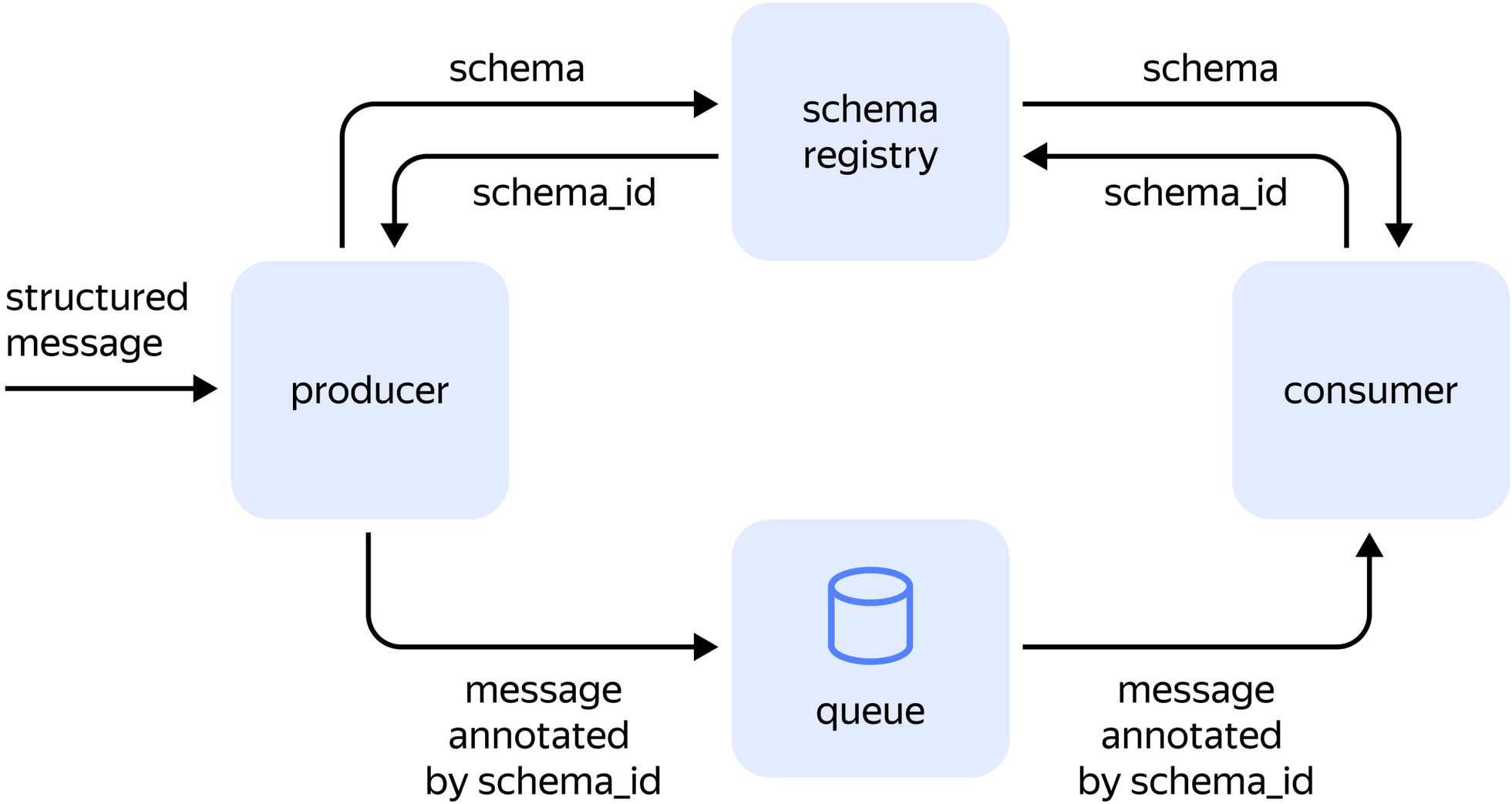
4.1 Schema Registry для очередей



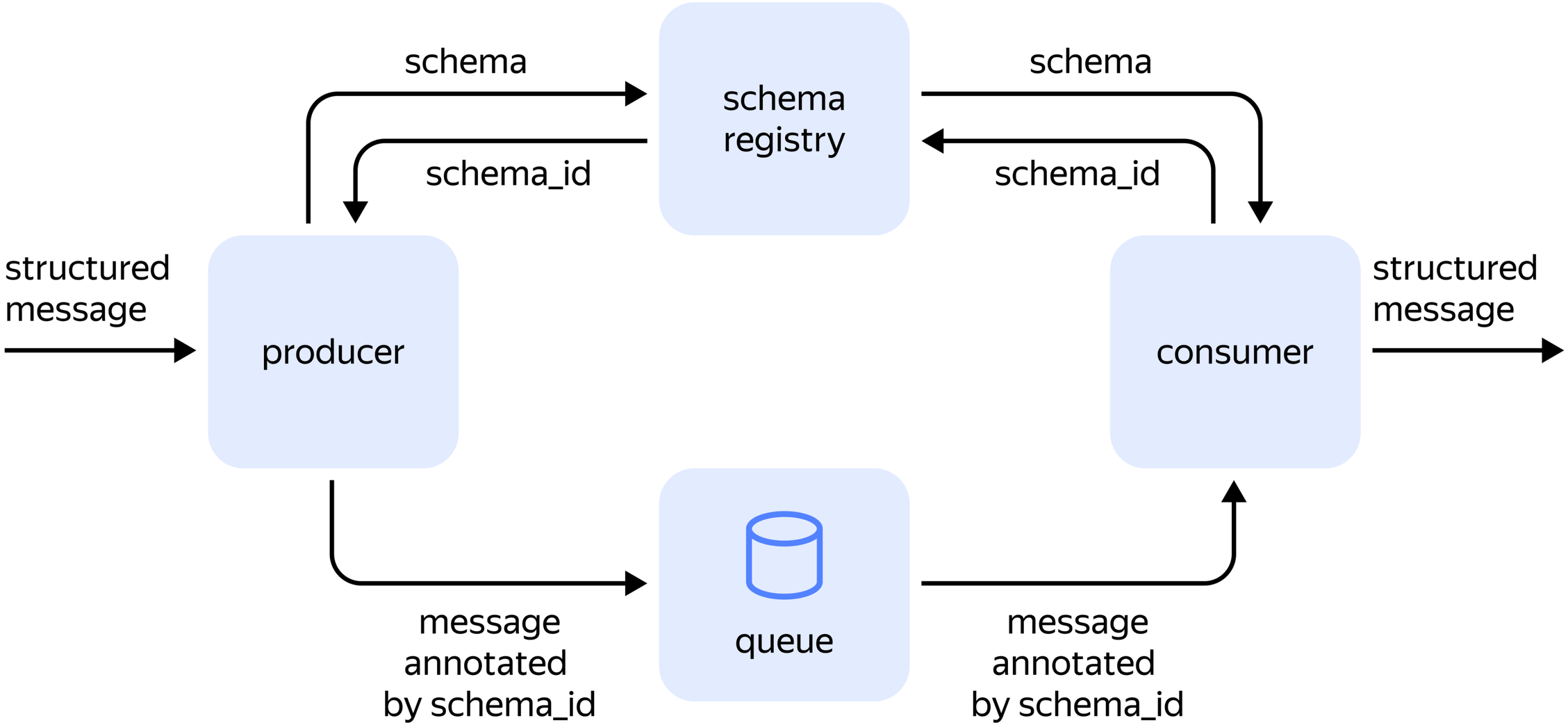
4.1 Schema Registry для очередей



4.1 Schema Registry для очередей

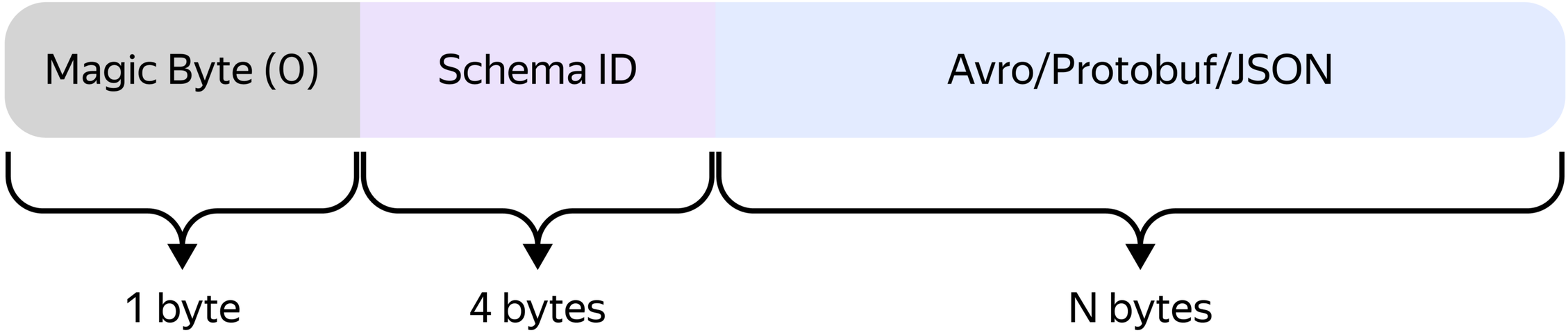


4.1 Schema Registry для очередей



4.1 Schema Registry для очередей

Default Wire Format (Confluent)



4.1 Schema Registry для очередей

Польза от Schema Registry в такой схеме:

- Появляется репозиторий версионированных схем
- Безопасная эволюция схем
- Надёжная поставка данных, shift left
- Экономия трафика
- Появляется data discovery
- Из коробки появляется десериализатор
- Если схема на источнике изменилась, она применяется по всему пайплайну дальше

4.1 Schema Registry для очередей

Форматы, с которыми работают Schema Registry такого класса:

1

Avro

2

Protobuf

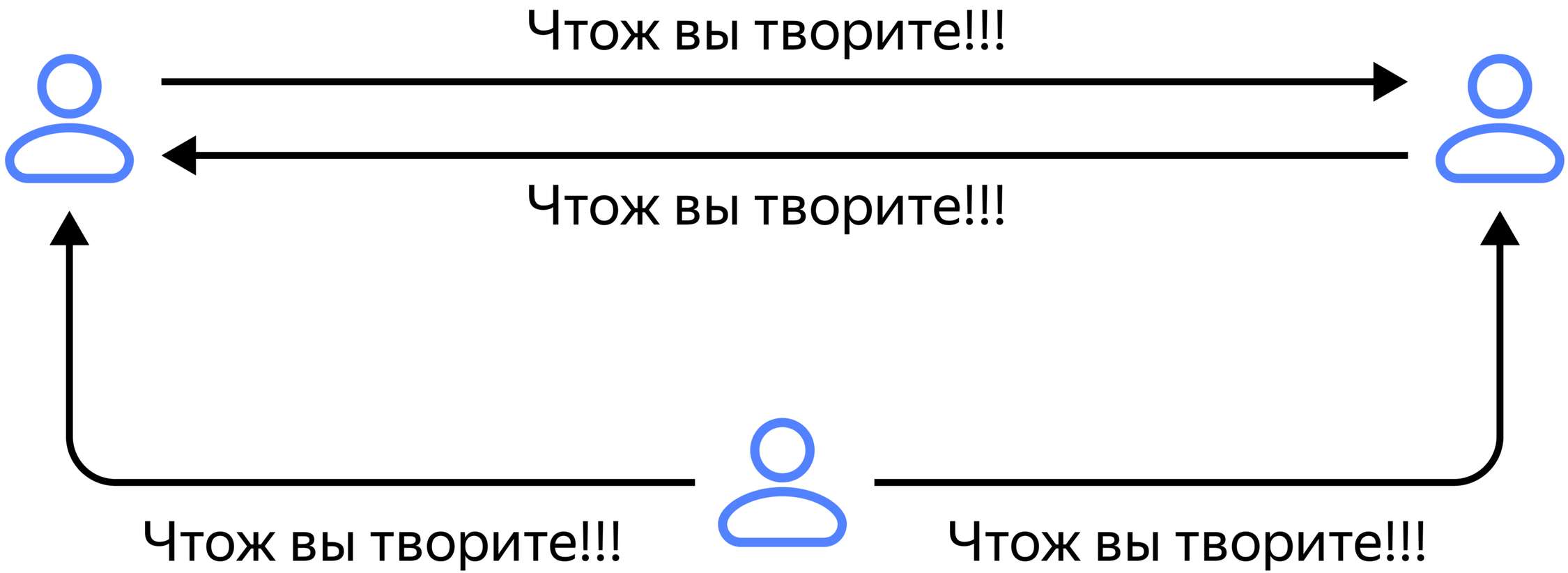
3

Json schema

4.2 Schema Registry для API

ЦА этого применения — участники разработки микросервисной архитектуры

4.2 Schema Registry для API — иллюстрация проблемы



4.2 Schema Registry для API

Schema Registry такого класса проверяют спецификации API на предмет несовместимых изменений

Форматы, с которыми работают Schema Registry такого класса:

- OpenAPI
- gRPC
- GraphQL

Если вы поставляете данные через очередь — используйте Schema Registry

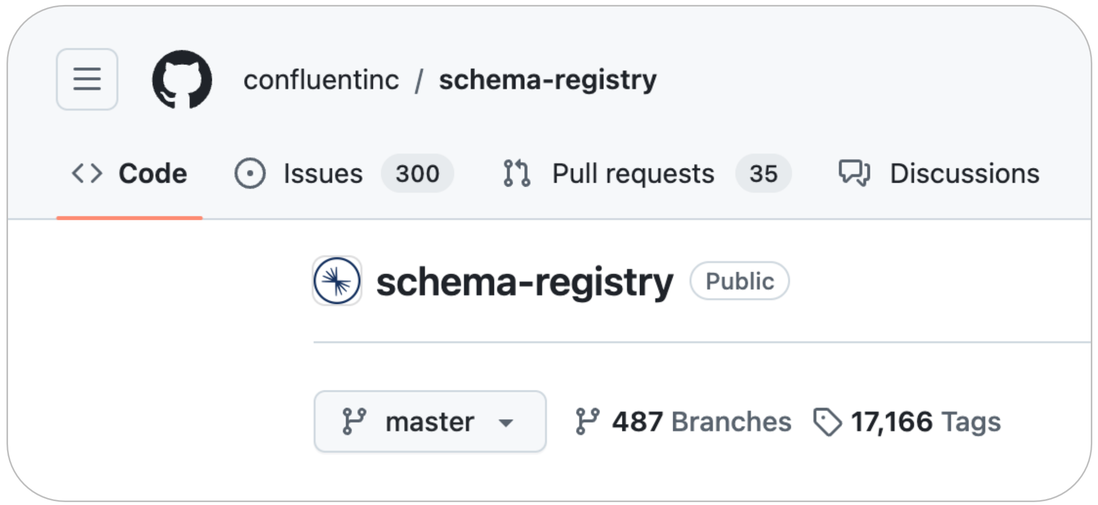
API

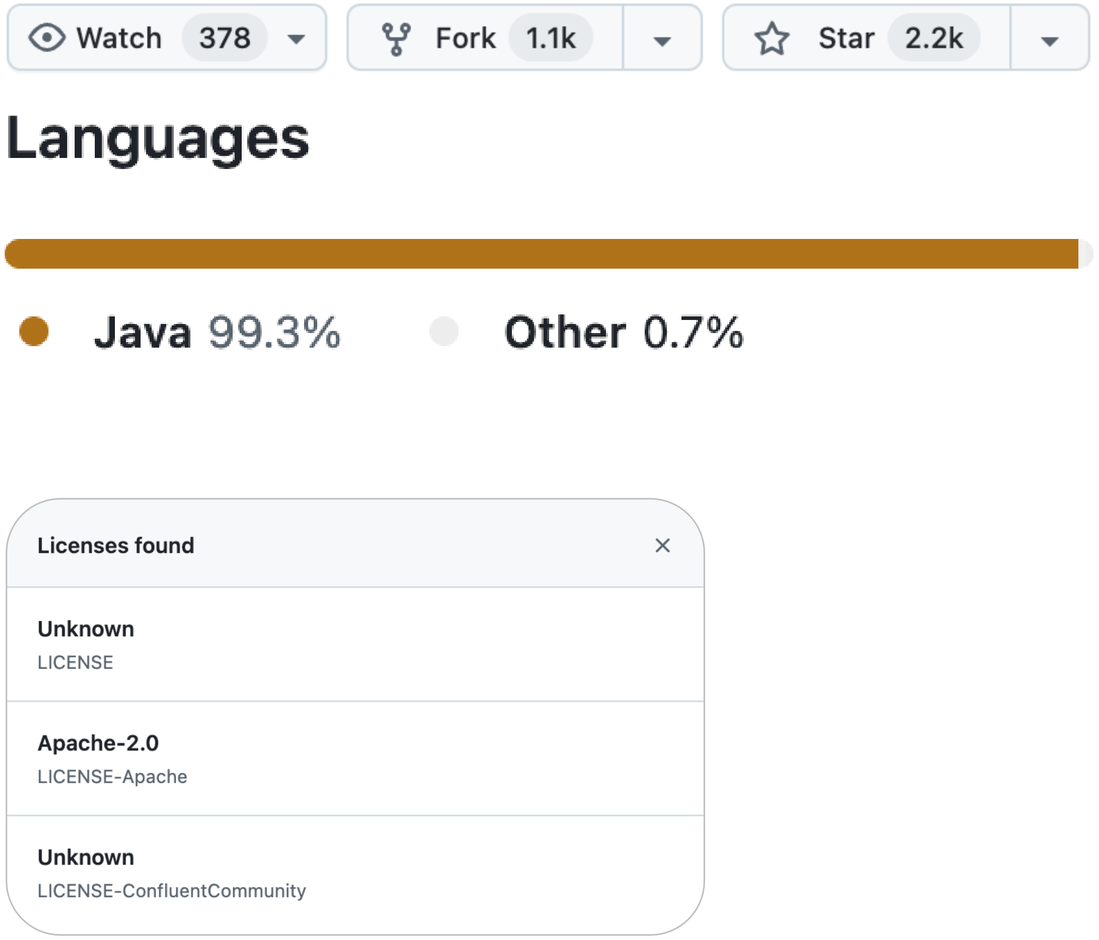
Если вы разрабатываете микросервисное API — используйте Schema Registry

5. Существующие Schema Registry

5.1 Существующие Schema Registry

Confluent Schema Registry

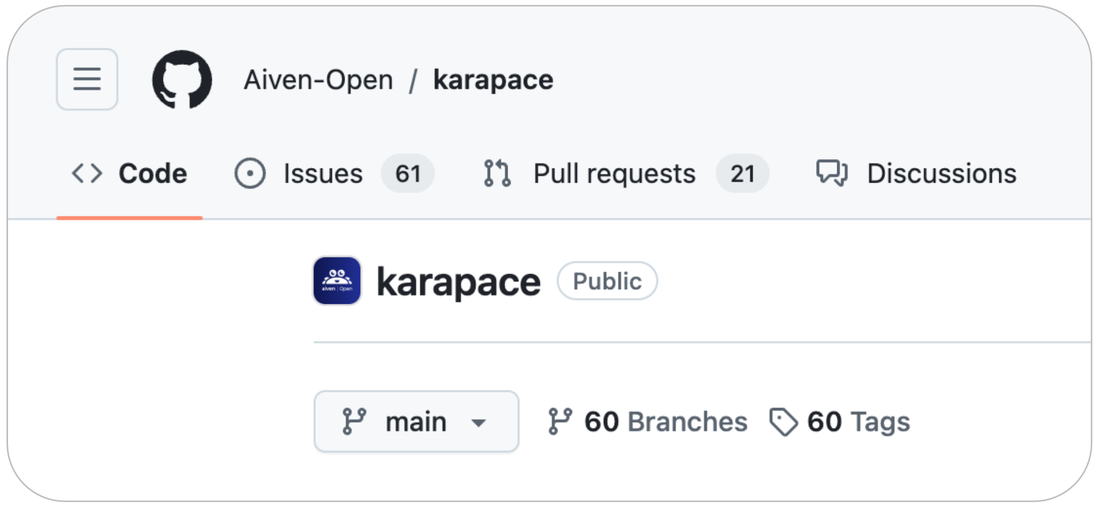


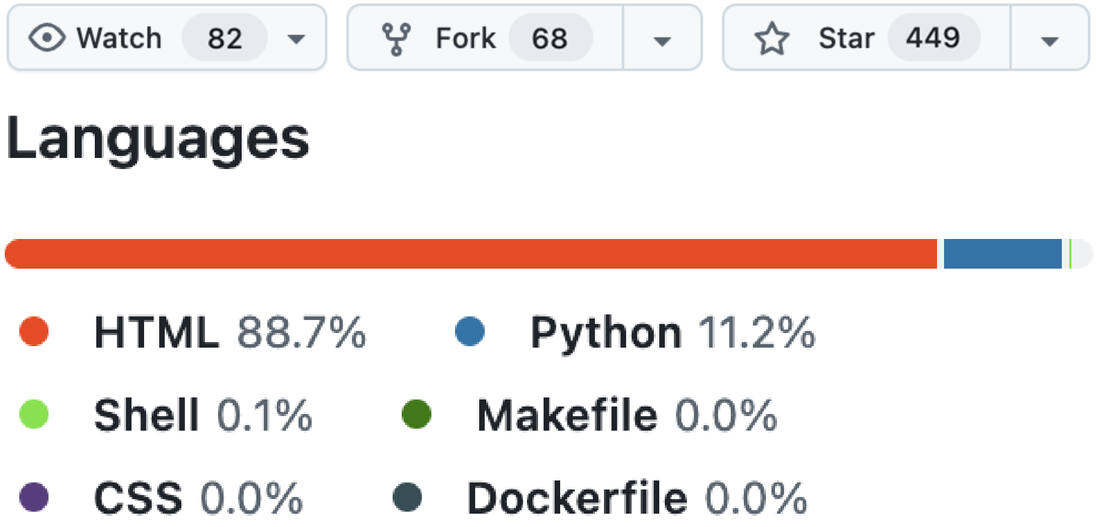


- Стандарт де-факто
- В 2023м появилось Data Quality

5.2 Существующие Schema Registry

Karapace



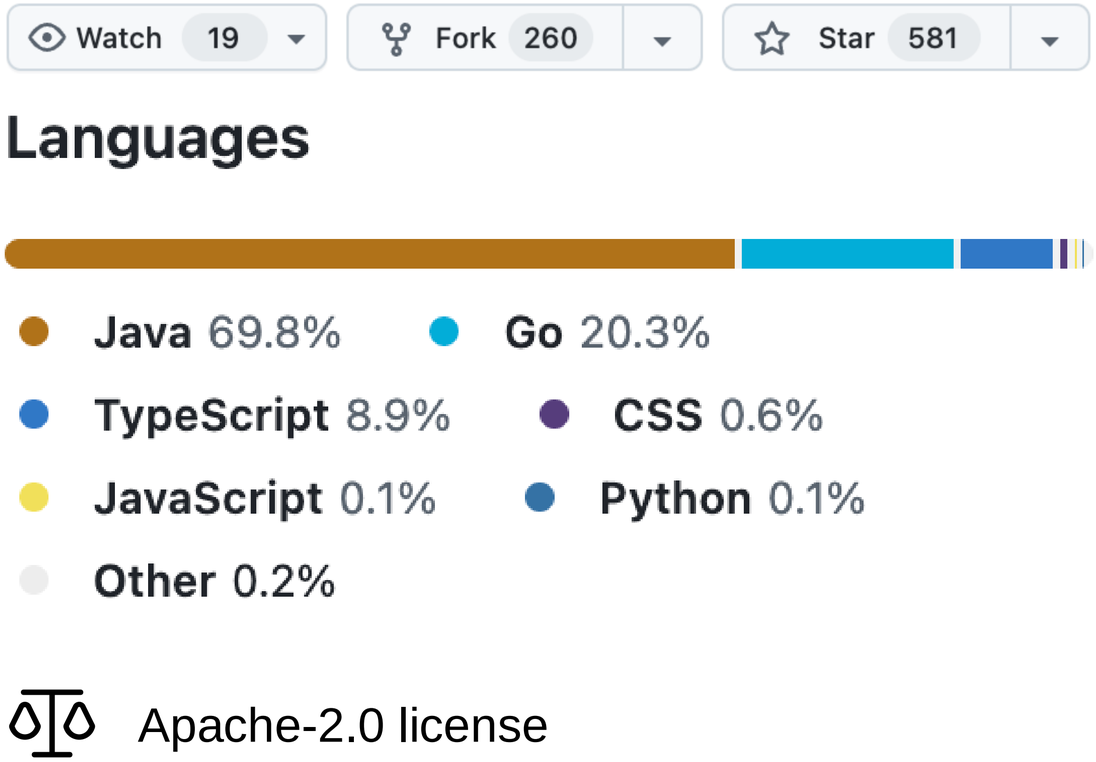


Свободная имплементация
confluent schema registry API

Apache-2.0 license

5.3 Существующие Schema Registry

Apicurio

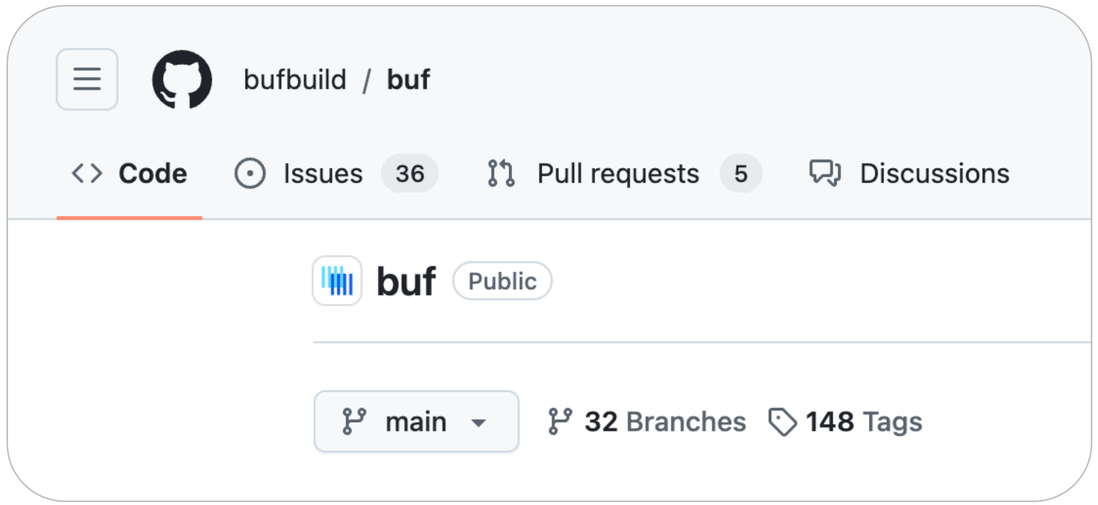


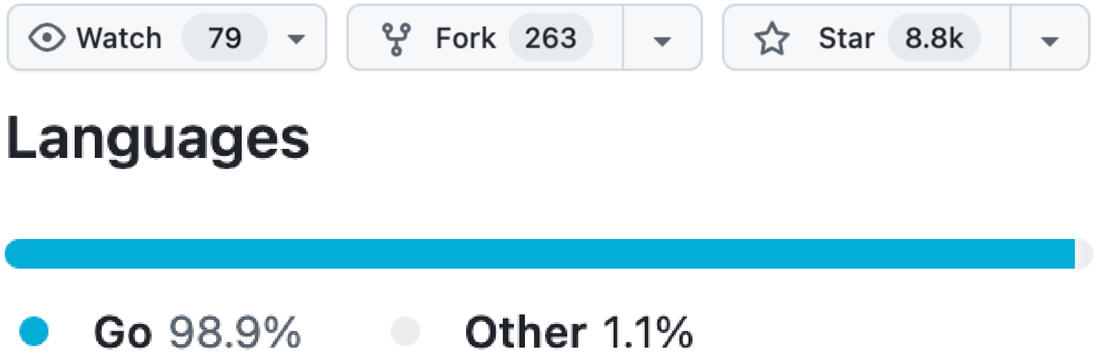
- Изначально сделано для проверки API
- Поддержан confluent schema registry API

Apache-2.0 license

5.4 Существующие Schema Registry

Buf





- Работает только с protobuf,
  но зато лучше всех
- Apache 2.0 только cli,
  confluent schema registry API
  поддержан в проприетарном saas

Apache-2.0 license

5.5 Существующие Schema Registry

Other

- Cloudera/Hortonworks
- Redpanda
- Облачные:

  Azure Schema (фича evenhub’a, wire-формат в очереди собственный, поддержан формат AVRO, в sdk есть намек на JsonSchema)

  Google pub/sub (schema навешивается на топик; форматы: avro/protobuf, compatibility не проверяется)

  AWS Glue (wire-формат свой, единственный wire со сжатием; форматы: avro/json_schema/protobuf)

- Queue-specific

  Apache Pulsar

  Pravega

  RocketMQ

- Stencil

5.6 Существующие Schema Registry

Other other

1

GraphQL

github.com/tot-ra

github.com/kamilkisiela/graphql-inspector

github.com/StarpTech/graphql-registry (last commit: 9 months ago)

Также есть проприетарные решения: hive (the-guild.dev), apollo, wundergraph, grafbase

2

gRPC

github.com/nilslice/protolock — утилита для проверки совместимости изменений в .proto-файлах

3

OpenAPI

github.com/Tufin/oasdiff — утилита для проверки совместимости изменений

6. Нюансы

6.1 Нюансы — Compatibility modes

- BACKWARD (default)
- BACKWARD_TRANSITIVE
- FORWARD
- FORWARD_TRANSITIVE
- FULL
- FULL_TRANSITIVE
- NONE

6.1 Нюансы — Compatibility modes

Schema version

mode	check
backward	check(1,2)
backward_ transitive	check(1,2) && check(0,2)

mode	check
forward	check(2,1)
forward_ transitive	check(2,1) && check(2,0)

mode	check
full	check(1,2) and check(2,1)
full_ transitive	check(1,2) && check(0,2) && check(2,1) && check(2,0)

6.1 Нюансы — Compatibility modes

Теоретическое следствие:

Если схема 1 backward совместимо со схемой 2 — то схема 2 forward совместима со схемой 1

6.1 Нюансы — Compatibility modes

Пример, где TRANSITIVE лучше, чем не-TRANSITIVE:

0

Version 0:

Добавили опциональное поле "my_field" типа INT

1

Version 1:

Удалили опциональное поле "my_field"

2

Version 2:

Добавили опциональное поле "my_field" типа STRING

6.1 Нюансы — Compatibility modes

По возможности используйте FULL_TRANSITIVE:

Плюсы:

Так вы сможете обновлять продьюсеров
и консьюмеров независимо друг
от друга, без ограничений

Минусы:

Но вы не сможете использовать
oneof & расширять енумы

6.1 Нюансы — Compatibility modes

Если использовать FULL_TRANSITIVE невозможно — используйте BACKWARD_TRANSITIVE:

Плюсы:

Вы сможете использовать oneof & расширять енумы

Минусы:

Консьюмер не должен быть свежее продьюсера

Кажется нет причин использовать BACKWARD вместо BACKWARD_TRANSITIVE

6.1 Нюансы — Compatibility modes

Когда вы поставляете данные через очередь — ограничьтесь следующими двумя типами изменений схемы:

- Добавление опционального поля
- Удаление опционального поля

Руководствуясь этим правилом:

- Ваши поставки не сломаются
- Даже если вы не используете Schema Registry
- Это правило независимо от формата данных

6.2 Нюансы — API

Есть один стандарт де-факто: confluent schema registry API, поддержан:

- Confluent schema registry
- Karapace
- Apicurio
- Buf (saas)
- Redpanda

Лучшая дока:
clck.ru/3D2meA

6.2 Нюансы — API

	Confluent SR API
Confluent	⊕
Karapace	⊕
Redpanda	⊕
Apicurio	⊕
Buf	⊕
Stencil	

6.2 Нюансы — API — интеграции

Кто поддерживает confluent schema registry API

- Apache Kafka Connectors
- Debezium
- Apache NIFI
- Apache Flink
- Apache Spark
- KSQL (Kafka SQL)
- ClickHouse
- Apache Storm
- Apache Druid
- Apache Pinot

За счёт широкой поддержки confluent schema registry API — вы можете собрать пайплайн поставки данных на Open Source продуктах, не написав ни строчки данных

6.2 Нюансы — API — confluent schema registry API

```
> curl -X POST \
  --data '{"schemaType": ”PROTOBUF", "schema": "..."}' \
  $URI/subjects/my-subject-name/versions
{"id":1}
```

```
> curl -X GET $URI/schemas/ids/1
{"schemaType": "PROTOBUF", "schema": "..."}
```

6.3 Нюансы — разные данные в одном топике

Martin Kleppmann — Should You Put Several Event Types in the Same Kafka Topic?

clck.ru/3D2nXn

6.4 Нюансы — размещение сервиса Schema Registry

	Confluent SR API	Storage
Confluent	➕	Kafka-only
Karapace	➕	Kafka-only
Redpanda	➕	Kafka-only
Apicurio	➕	Kafka, PostgreSQL, Microsoft SQL Server
Buf	➕	Proprietary commercial saas
Stencil		PostgreSQL

6.5 Нюансы — алгоритмы сравнения

Все schema registry на рынке оперируют понятием наборов «правил» (rules) — rule либо задает ломающее изменение, либо задаёт инвариант, нарушение которого является ломающим изменением

★

Confluent / karapace / stencil / redpanda / avro — в один обход распаршенного дерева описания схемы осуществляет все проверки

★

Buf — каждый rule — это callback на определенный вид вершины, каждый rules это отдельный обход распаршенного дерева, каждый rule запускается в отдельной горутине

★

Apicurio — каждый rule это метод, в котором собственный обход дерева, вызываются последовательно. Для JsonSchema это так же, но через pattern visitor

7. Форматы

7.1 Форматы — AVRO

7.1 Форматы — AVRO

- Формат компактного построчного хранения схематизированных данных
- Эволюция схемы встроена в дизайн формата
- Также AVRO позволяет описывать RPC (AVRO protocol)
- Можно использовать в двух видах:

  With code generation

  Without code generation
- Confluent рекомендует использовать AVRO в очередях
- Есть 2 вида сериализации: binary & JSON

7.1 Форматы — AVRO

Схема данных описывается форматом JSON

```
 "type": "record",
 "name": "Record",
 "fields": [
  {
   "name": "Name",
   "type": "string"
  },
  {
   "name": "Age",
   "type": "long"
  },
  {
   "name": "City",
   "type": "string"
  }
 ]
}
```

7.1 Форматы — AVRO

Схема данных также может описываться IDL
(Avro IDL, Interface Definition Language)

```
record Record {
    string Name;
    long Age;
    string City;
}
```

7.1 Форматы — AVRO

Типы данных:

- Целые числа (int32/int64)
- Floating-point числа (4bytes/8bytes)
- Строки/байты
- Bool
- Null
- Decimal
- Time-related
- Complex: union/record/enum/array/map

7.1 Форматы — AVRO

Variable-length zig-zag encoding (int/long)

Vint Encoding Example

Value	First byte	Second byte
0	00000000	
1	00000001	
2	00000010	
…		
127	01111111	
128	10000000	00000001
129	10000001	00000001
130	10000010	00000001

7.1 Форматы — AVRO

- Кодирование variable-length encoding даёт вам эффективное хранение чисел
- Variable-length encoding есть и в AVRO и в protobuf

7.1 Форматы — AVRO

Optional fields

```
{
   "name": "color",
   "type": ["null", "string"],
   "default": null
}
```

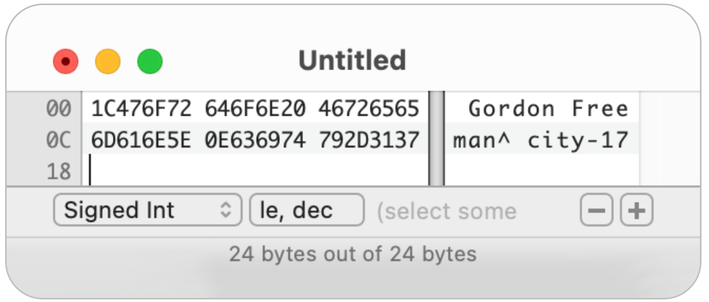
7.1 Форматы — AVRO

```
{
 "type": "record",
 "name": "Record",
 "fields": [
  {
   "name": "Name",
   "type": "string"
  },
  {
   "name": "Age",
   "type": "long"
  },
  {
   "name": "City",
   "type": "string"
  }
 ]
}
```

- Создаём инстанс такой структуры
- Заполняем следующими значениями

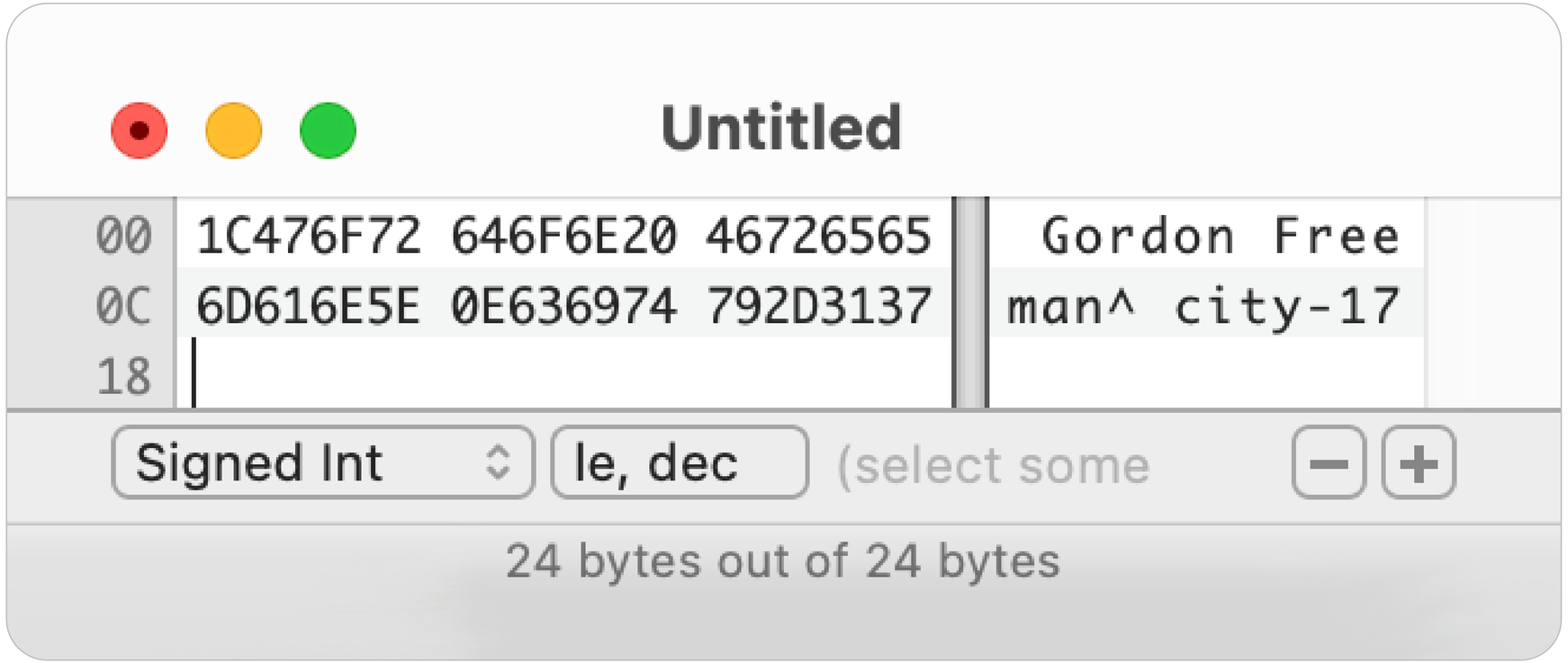
```
{
  "Name": "Gordon Freeman",
  "Age": 47,
  "City": "city-17"
}
```

Сериализуем



7.1 Форматы — AVRO

Рассмотрим



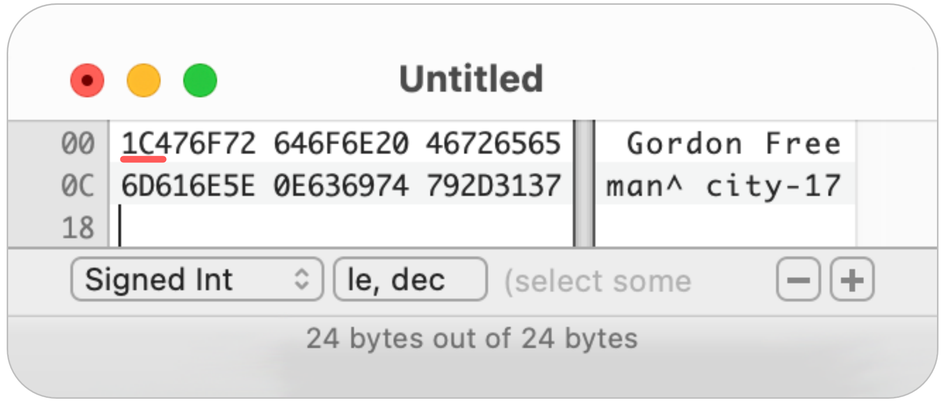
7.1 Форматы — AVRO

0x1C -> 0001.1100 -> 0001110.0

0 - sign

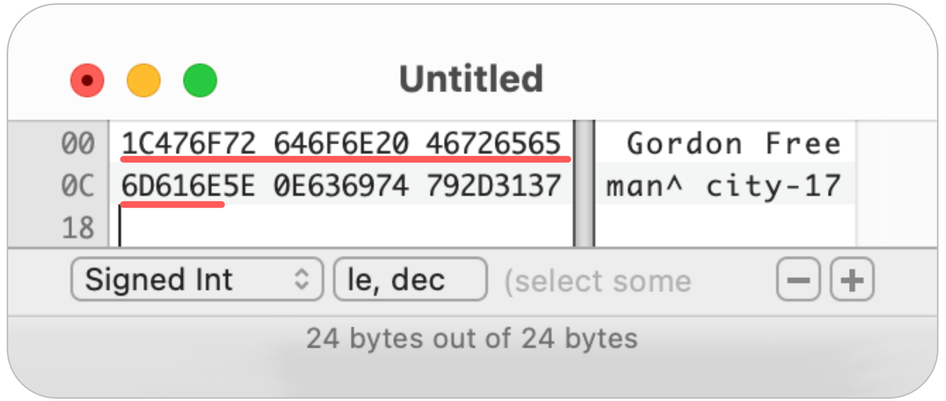
0001110 –> 0xE -> 14(dec)

Длина строки: 14



7.1 Форматы — AVRO

Строка «Gordon Freeman»,
длиной 14 байт

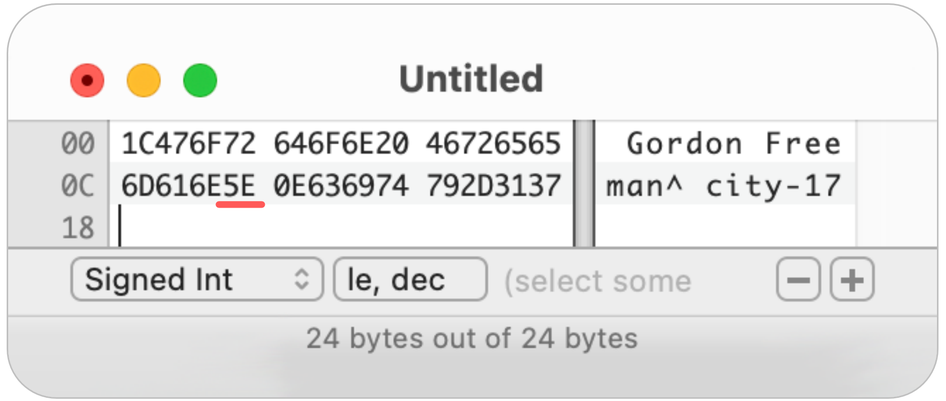


7.1 Форматы — AVRO

long – variable-length zig-zag encoding

variable length:

0x5E -> 0101.1110 ->
0.1011110 -> length: 1 byte



zig-zag:

1011110 -> 0x5E in zig-zag
encoding -> 0x2F -> 47(dec)

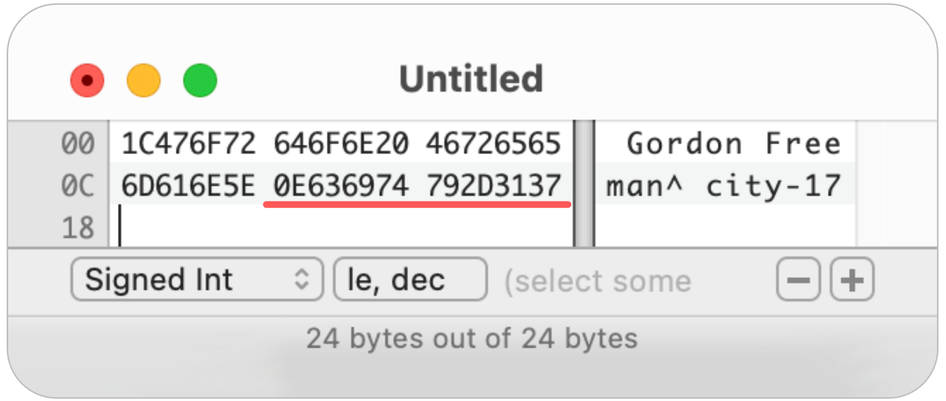
Итого: число 47

7.1 Форматы — AVRO

0x0E -> 0000.1110 -> 0000111.0

0000111 -> 7(dec)

Итого: строка «city-17»
длиной 7 байт



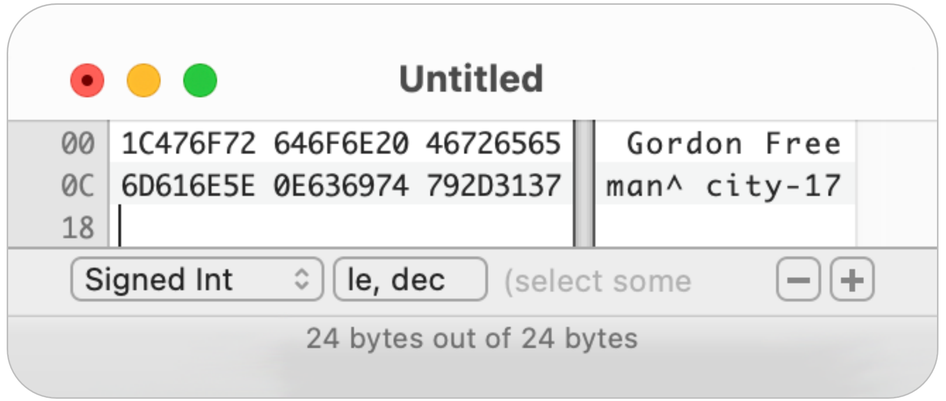
7.1 Форматы — AVRO

В сообщении закодировано:

- Строка «Gordon Freeman» длиной 14 байт
- Число «47»
- Строка «city-17» длиной 7 байт

Чего в сообщении не закодировано:

- Какому полю соответствует закодированное значение
- Тип данных следующего поля



7.1 Форматы — AVRO

Как кодируются вложенные сообщения AVRO

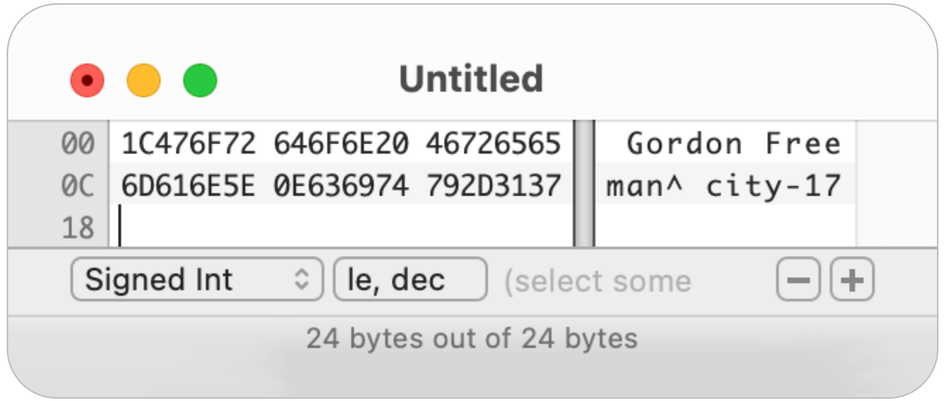
```
record RecordInner {
    string FirstName;
    string SecondName;
}

record Record {
    RecordInner rec;
    long Age;
}
```

```
record Record {
    string FirstName;
    string SecondName;
    long Age;
}
```

7.1 Форматы — AVRO

Как кодируется AVRO:

- Разворачиваем вложенные рекорды
- Arrays: count + blocks
- Maps: count + k/v pairs
- Unions: array index + value
- Опциональные поля —
  union типов с default type null



7.1 Форматы — AVRO

```
{
   "type": "record",
   "name": "Record",
   "fields": [
      {
         "name": "String",
         "type": "string"
      }
   ]
}
```

String: "^"

value: 0x025E

```
{
   "type": "record",
   "name": "Record",
   "fields": [
      {
         "name": "Field1",
         "type": "long"
      },
      {
         "name": "Field2",
         "type": "long"
      }
   ]
}
```

Field1: 1

Field2: 47

Без знания AVRO-схемы, невозможно однозначно декодировать сообщение

Вопрос — как же при такой организации изменять схему?

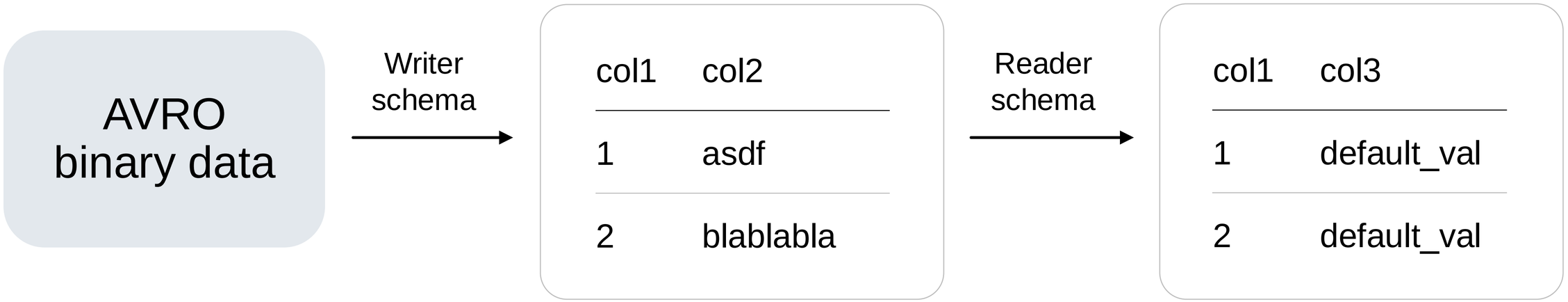
7.1 Форматы — AVRO

Для работы с эволюцией AVRO вам нужны 2 схемы:

- Writer schema — ровно та схема, по которой сериализовано
- Reader schema — это уже та схема, какую таблицу мы хотим получить из исходной

7.1 Форматы — AVRO

Как это можно воспринимать



- По Writer schema вы получаете таблицу
- Reader schema — показывает какую таблицу вы из неё хотите получить:

  Удалить одни колонки

  Добавить новые колонки с default value

  Переименовать колонки (через alias)

7.1 Форматы — AVRO

«Avro schema evolution is broken»

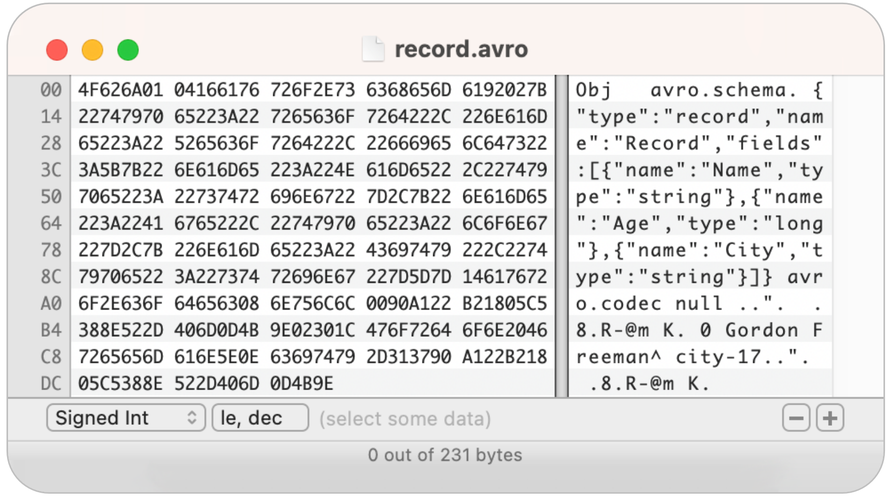
«One great solution would be augmenting the binary encoding with a simple field number identifier. Let's imagine an Avro 2.x that had this feature, and would support schema evolution»

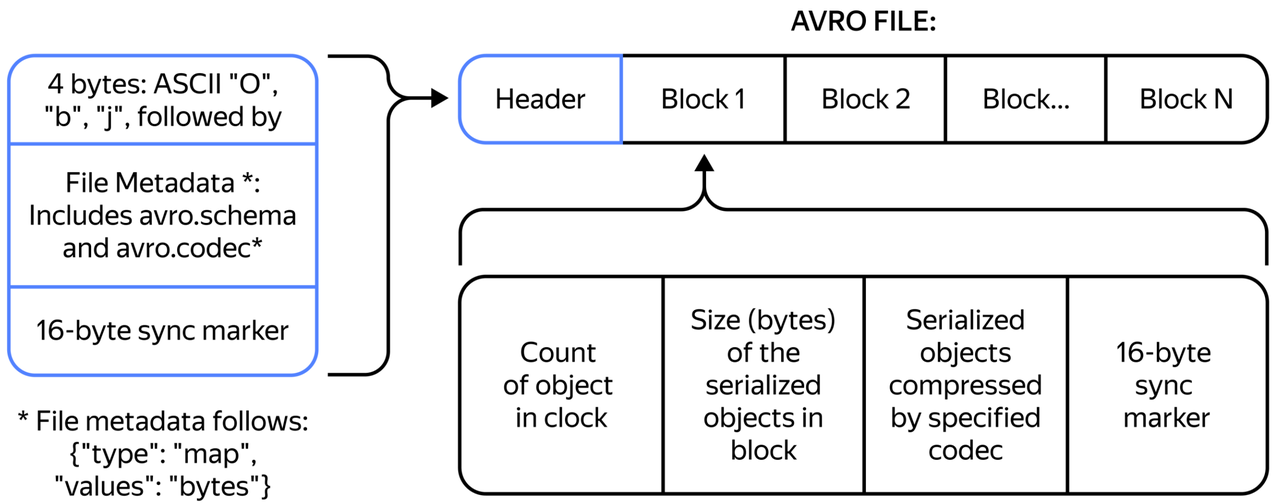
Fun story: clck.ru/3D3bcM

7.1 Форматы — AVRO

AVRO file

> java -jar avro-tools-1.11.3.jar fromjson --schema-file record.avsc record.json > record.avro





7.1 Форматы — AVRO

Compatibility — описывается самим форматом AVRO. Schema Registry остается только вызвать метод calculateCompatibility из библиотеки AVRO

```
public enum SchemaIncompatibilityType {
    NAME_MISMATCH,
    FIXED_SIZE_MISMATCH,
    MISSING_ENUM_SYMBOLS,
    READER_FIELD_MISSING_DEFAULT_VALUE,
    TYPE_MISMATCH,
    MISSING_UNION_BRANCH
}
```

github.com/apache/avro

7.1 Форматы — AVRO

Всего 6 способов выстрелить себе в ногу

- NAME_MISMATCH — изменилось имя (name/namespace) схемы
- FIXED_SIZE_MISMATCH — изменилась длина поля с типом fixed (буфер фиксированной длины)
- MISSING_ENUM_SYMBOLS — исчезли какие-то значения енума
- READER_FIELD_MISSING_DEFAULT_VALUE — для новой колонки нет default value
- TYPE_MISMATCH — изменили тип
- MISSING_UNION_BRANCH — исчезли какие-то типы из union’ов

7.1 Форматы — AVRO

Удобные инструменты:

- Avro-tools
- Avrocat

7.1 Форматы — AVRO

	Confluent SR API	Storage	AVRO
Confluent	⊕	Kafka-only	⊕
Karapace	⊕	Kafka-only	⊕
Redpanda	⊕	Kafka-only	⊕
Apicurio	⊕	Kafka, PostgreSQL, Microsoft SQL Server	⊕
Buf	⊕	Proprietary commercial saas	
Stencil		PostgreSQL	⊕

7.2 Форматы — Protobuf

7.2 Форматы — Protobuf

- Формат компактного хранения схематизированных данных
- Также на нём можно описывать RPC сервисы — gRPC
- Хранение построчное
- История создания protobuf — protobuf.dev/history
- Особенности:

  Компактность

  Эффективность

  Поддержка всех популярных ЯП

  Строгая типизация схемы данных

  Возможность эволюции схемы (**эволюция — не часть дизайна формата**)

  Есть неочевидные ограничения: reasonablypolymorphic.com/blog/protos-are-wrong

7.2 Форматы — Protobuf

- Формат компактного хранения схематизированных данных
- Также на нём можно описывать RPC сервисы — gRPC
- Хранение построчное
- История создания protobuf — protobuf.dev/history

- Особенности:

  Компактность

  Эффективность

  Поддержка всех популярных ЯП

  Строгая типизация схемы данных

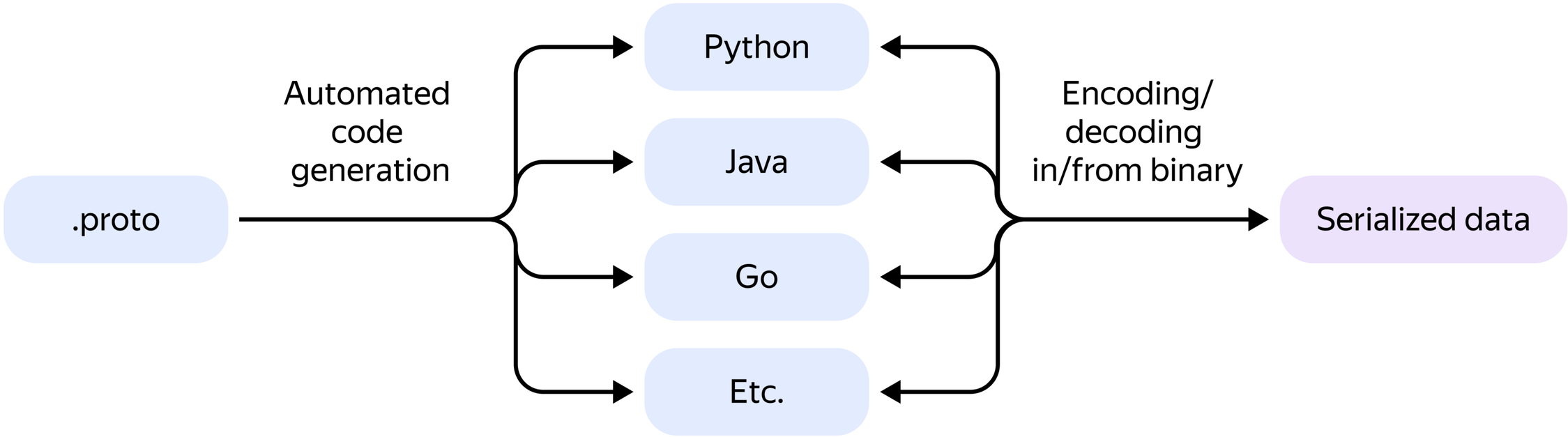
  Возможность эволюции схемы (эволюция — не часть дизайна формата)

  Есть неочевидные ограничения: reasonablypolymorphic.com/blog/protos-are-wrong

7.2 Форматы — Protobuf

Основной способ работы — через кодогенерацию, эффективно реализующую getters/setters и marshal/unmarshal — для всех популярных ЯП.
Но есть и возможность работать без кодогенерации

Data Serialization with Protocol Buffers



7.2 Форматы — Protobuf

Сейчас поддерживается 2 версии протобуфа:

- Proto2
- Proto3

Важнейшее различие — убрали тэг required

7.2 Форматы — Protobuf

Схема данных может описываться только IDL

```
syntax = "proto3";


message Record {
    string Name = 1;
    int64 Age = 2;
    string City = 3;
}
```

7.2 Форматы — Protobuf

Пример синтаксиса описания gRPC сервисов:

```
syntax = "proto3";


service MyService {
    rpc MyMethod(Record) returns (Record) {}
}


message Record {
    string Name = 1;
    int64 Age = 2;
    string City = 3;
}
```

7.2 Форматы — Protobuf

```
# protobuf – описывает структуру
message Record {
   string my_name = 1;
}

# protojson – описывает инстанс
{
   "myName": "Billy Harrington",
}

# prototext – описывает инстанс
Record {
   my_name: "Billy Harrington"
}
```

Prototext
clck.ru/3D3ggj

JSON Mapping
clck.ru/3D3h49

- Protojson и prototext — два альтернативных способа сериализации proto-мессаджей
- Protojson — является JSON сообщением

7.2 Форматы — Protobuf

Типы данных:

- Целые числа (int32/sint32/sfixed32/int64/ sing64/sfixed64/uint32/fixed32/uint64/fixed64)
- Floating-point числа (4bytes/8bytes)
- Строки/байты
- Bool
- Map (разворачивается в массив пар key-value)
- Oneof (enum типов)
- Enum

Модификаторы для полей:

- Optional
- Required
- Repeated — делают массивы

7.2 Форматы — Protobuf

- Reserved — это только механизм удаления ненужных полей, не нужно резервировать поля на будущее!!!
- Документация: Field numbers should never be reused. Never take a field number out of the reserved list for reuse with a new field definition

Reserved fields: clck.ru/3D3kPZ

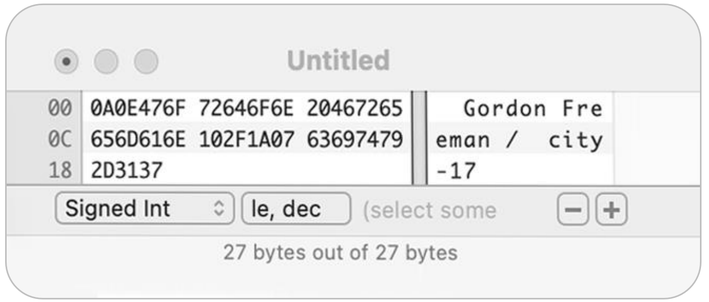
7.2 Форматы — Protobuf

```
syntax = "proto3";

message Record {
    string Name = 1;
    int64 Age = 2;
    string City = 3;
}
```

- Создаём инстанс такой структуры
- Заполняем следующими значениями

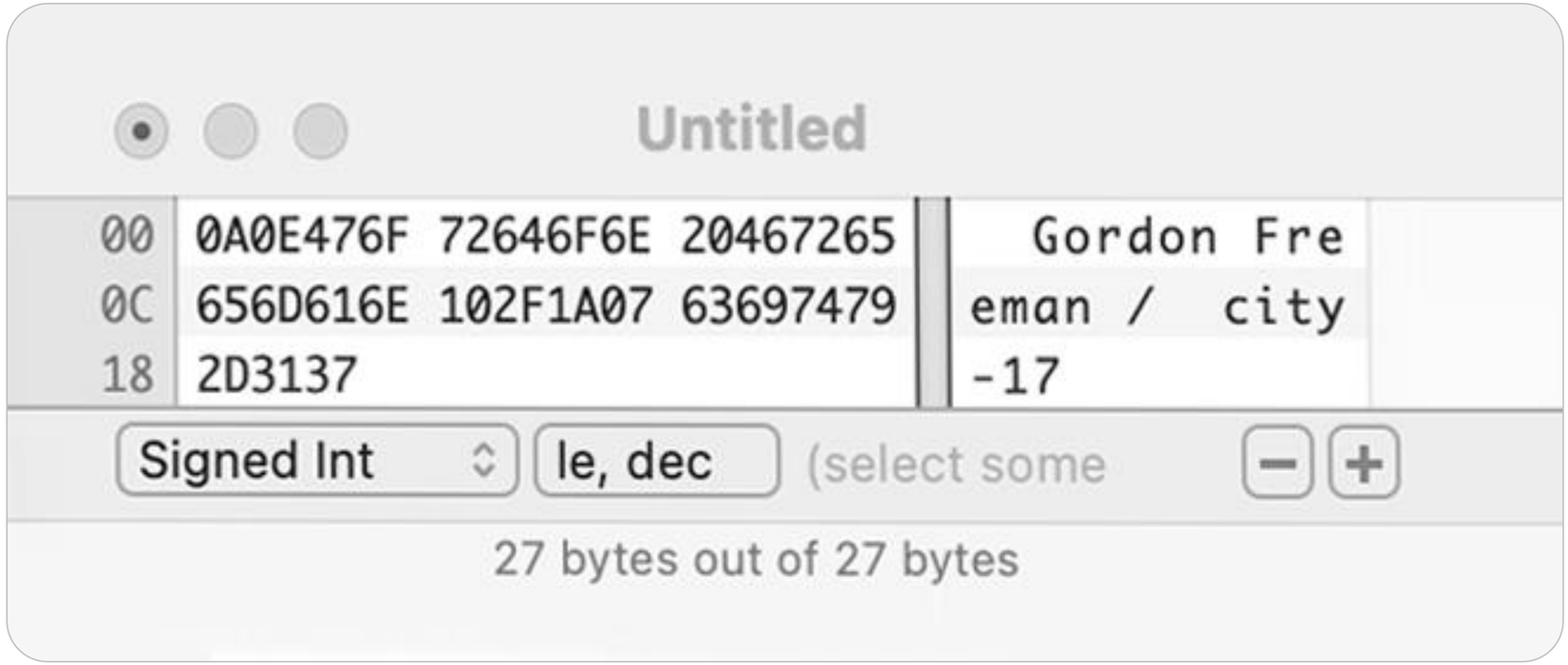
```
{
    "Name": "Gordon Freeman",
    "Age": 47,
    "City": "city-17"
}
```

Сериализуем



7.2 Форматы — Protobuf

Рассмотрим



7.2 Форматы — Protobuf

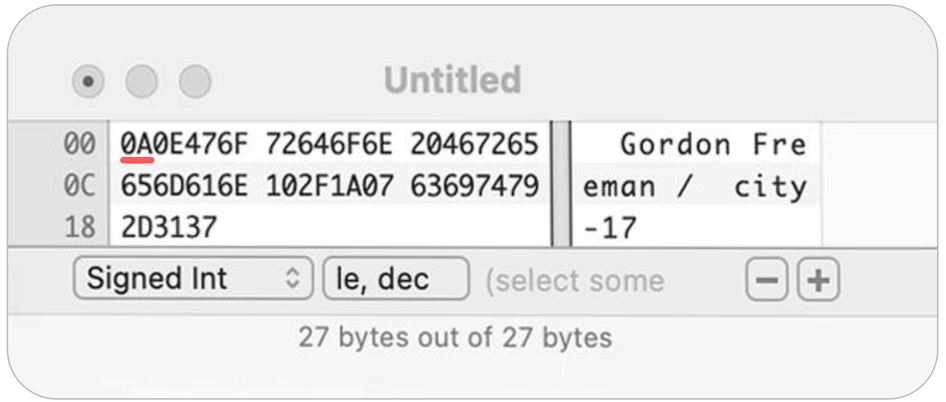
0x0A -> 0000.1010

0000.1010 -> 00001.010

00001 - Поле #1

010 -> 2 -> LEN

Поле #1, LEN wire type

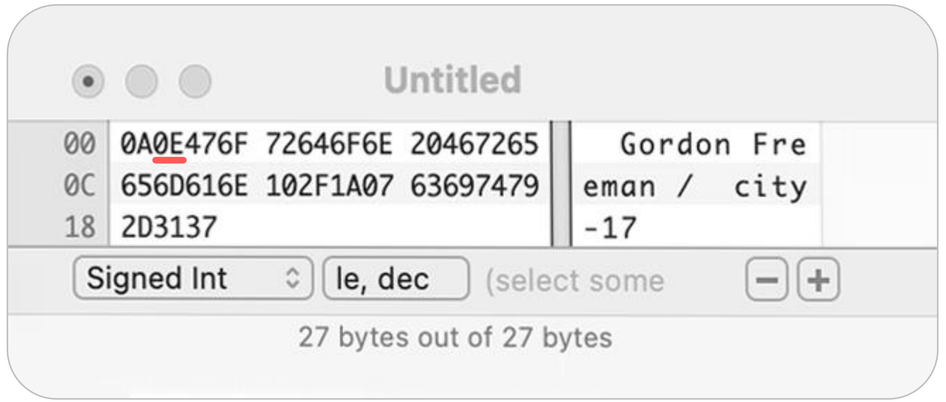


7.2 Форматы — Protobuf

0x0E -> 14(dec)

varint-encoded

длина строки = 14



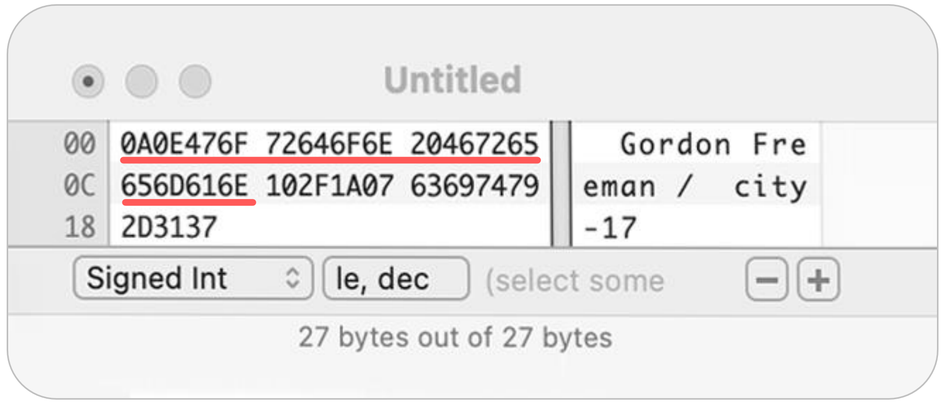
7.2 Форматы — Protobuf

Поле #1, LEN wire type

длина строки = 14

значение строки:
Gordon Freeman

TLV – Tag-Length Value



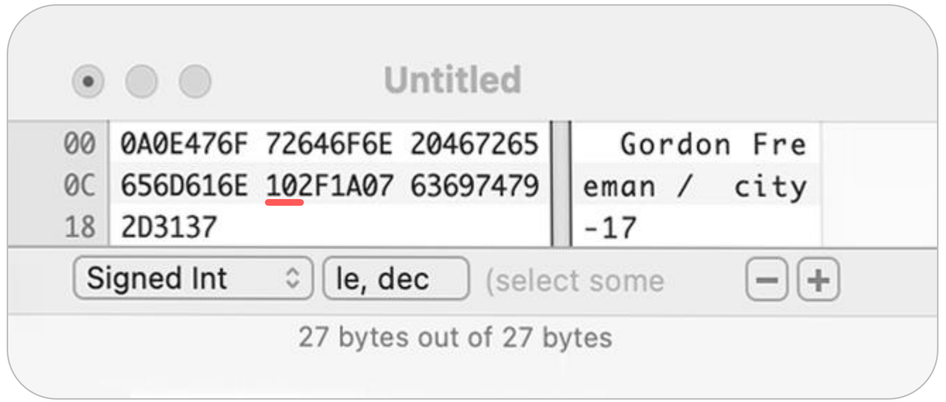
7.2 Форматы — Protobuf

0x10 -> 0001.0000

0001.0000 -> 00010.000

00010 - Поле #2

000 -> 2 -> VARINT



Поле #2, VARINT wire type

7.2 Форматы — Protobuf

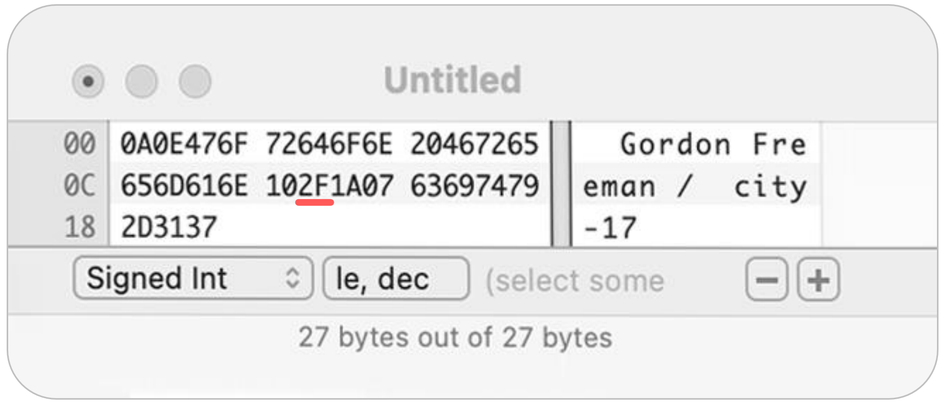
0x2F -> 0010.1111

0010.1111 -> 0.0101111

0 – terminal byte

0101111 -> 47(dec)

Значение поля: 47(dec)



7.2 Форматы — Protobuf

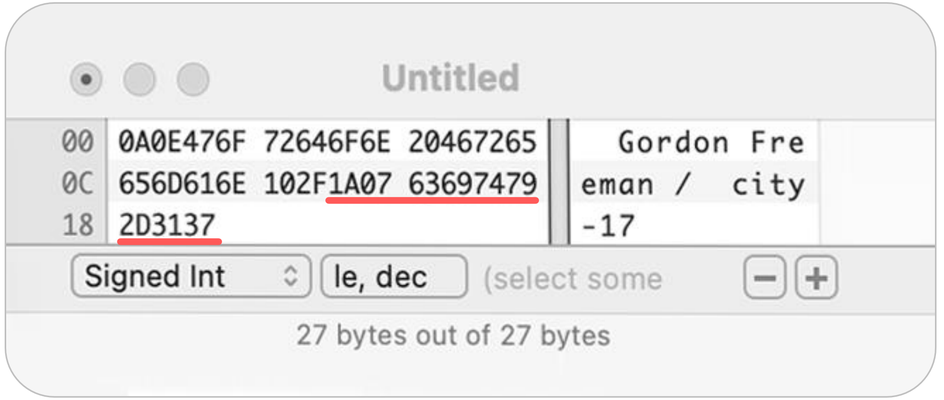
0x1A -> 0001.1010

0001.1010 -> 00011.010

00011 - поле #3

010 – значение LEN wire type

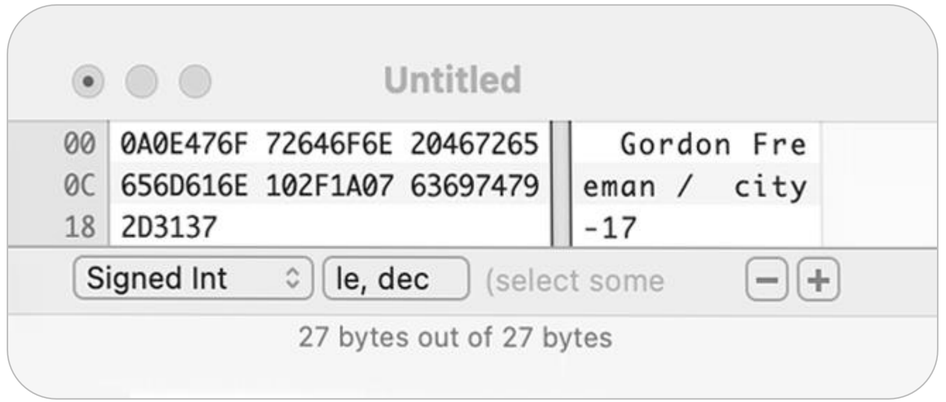
Значение: "city-17"



7.2 Форматы — AVRO

В сообщении закодировано:

- Строка «Gordon Freeman» длиной 14 байт
- Число «47»
- Строка «city-17» длиной 7 байт



Чего в сообщении не закодировано:

- Какому номеру поля соответствует закодированное значение
- Типы данных полей

Что ещё:

- В сообщении не закодированы имена полей
- Вложенные структуры кодируются как LEN wire type, и содержимое декодируется рекурсивно

Compatibility

7.2 Форматы — Protobuf

- 12 rules "proto best practices"
- Например:

  Не меняйте номера существующим полям

  Используйте reserved когда удаляете поля

  Целочисленные типы можно менять между собой, например: int32->int64

  …
- Цель — чтобы новый мессадж, сериализованный по новой схеме, мог быть успешно десериализован старым кодом

Compatibility #1 — doc
clck.ru/3D3bcM

7.2 Форматы — Protobuf

Compatibility #2 — confluent schema registry
clck.ru/3D3qUw

Тут перечислены все типы изменений (25 штук)

```
public class Difference {
  public enum Type {
    PACKAGE_CHANGED, MESSAGE_ADDED,
    MESSAGE_REMOVED, MESSAGE_MOVED,
    ENUM_ADDED, ENUM_REMOVED, ENUM_CONST_ADDED,
    ENUM_CONST_CHANGED,
    ENUM_CONST_REMOVED, FIELD_ADDED,
    FIELD_REMOVED,FIELD_NAME_CHANGED,
    FIELD_KIND_CHANGED,
    FIELD_SCALAR_KIND_CHANGED,
    FIELD_NAMED_TYPE_CHANGED,
    FIELD_NUMERIC_LABEL_CHANGED,
    FIELD_STRING_OR_BYTES_LABEL_CHANGED,
    REQUIRED_FIELD_ADDED, REQUIRED_FIELD_REMOVED,
    ONEOF_ADDED, ONEOF_REMOVED,
    ONEOF_FIELD_ADDED, ONEOF_FIELD_REMOVED,
    MULTIPLE_FIELDS_MOVED_TO_ONEOF,
    FIELD_MOVED_TO_EXISTING_ONEOF
  }
```

7.2 Форматы — Protobuf

Compatibility #2 — confluent schema registry
clck.ru/3D3qfC

Тут из этих 25 перечислили COMPATIBLE_CHANGES (14 штук)

```
static {
   Set<Type> changes = new HashSet<>();

   changes.add(MESSAGE_ADDED);
   changes.add(MESSAGE_MOVED);
   changes.add(ENUM_ADDED);
   changes.add(ENUM_REMOVED);
   changes.add(ENUM_CONST_ADDED);
   changes.add(ENUM_CONST_CHANGED);
   changes.add(ENUM_CONST_REMOVED);
   changes.add(FIELD_ADDED);
   changes.add(FIELD_REMOVED);
   changes.add(FIELD_NAME_CHANGED);
   changes.add(FIELD_STRING_OR_BYTES_LABEL_CHANGED);
   changes.add(ONEOF_ADDED);
   changes.add(ONEOF_REMOVED);
   changes.add(ONEOF_FIELD_ADDED);

   COMPATIBLE_CHANGES = Collections.
   unmodifiableSet(changes);
 }
```

7.2 Форматы — Protobuf

Статья про проблемы с oneof
clck.ru/3D3qyo

Compatibility #2 — confluent schema registry. 11 incompatible changes

MESSAGE_REMOVED

PACKAGE_CHANGED

FIELD_KIND_CHANGED (scalar <-> message <-> map)

FIELD_SCALAR_KIND_CHANGED (number <-> string)

FIELD_NAMED_TYPE_CHANGED (переименование существующих мессаджей)

FIELD_NUMERIC_LABEL_CHANGED (optional <-> repeated <-> required)

REQUIRED_FIELD_ADDED

REQUIRED_FIELD_REMOVED

ONEOF_FIELD_REMOVED

MULTIPLE_FIELDS_MOVED_TO_ONEOF

FIELD_MOVED_TO_EXISTING_ONEOF

7.2 Форматы — Protobuf

Compatibility #3 — karapace
clck.ru/3D3rNi

Тут перечислены все типы изменений (26 штук)

```
class Modification(Enum):
    PACKAGE_ALTER = auto()
    SYNTAX_ALTER = auto()
    MESSAGE_ADD = auto()
    MESSAGE_DROP = auto()
    MESSAGE_MOVE = auto()
    ENUM_CONSTANT_ADD = auto()
    ENUM_CONSTANT_ALTER = auto()
    ENUM_CONSTANT_DROP = auto()
    ENUM_ADD = auto()
    ENUM_DROP = auto()
    TYPE_ALTER = auto()
    FIELD_ADD = auto()
    FIELD_DROP = auto()
    FIELD_MOVE = auto()
    FIELD_LABEL_ALTER = auto()
    FIELD_NAME_ALTER = auto()
    FIELD_KIND_ALTER = auto()
    FIELD_TYPE_ALTER = auto()
    FIELD_TAG_ALTER = auto()
    ONE_OF_ADD = auto()
    ONE_OF_DROP = auto()
    ONE_OF_MOVE = auto()
    ONE_OF_FIELD_ADD = auto()
    ONE_OF_FIELD_DROP = auto()
    ONE_OF_FIELD_MOVE = auto()
    FEW_FIELDS_CONVERTED_TO_ONE_OF = auto()
```

7.2 Форматы — Protobuf

Compatibility #3 — karapace
clck.ru/3D3rNi

Тут из этих 26 перечислили INCOMPATIBLE_CHANGES (8 штук)

```
def is_compatible(self) -> bool:
    return self not in [
        self.MESSAGE_MOVE,
        self.MESSAGE_DROP,
        self.FIELD_LABEL_ALTER,
        self.FIELD_KIND_ALTER,
        self.FIELD_TYPE_ALTER,
        self.FIELD_TAG_ALTER,
        self.ONE_OF_FIELD_DROP,
        self.FEW_FIELDS_CONVERTED
        _TO_ONE_OF,
    ]
```

7.2 Форматы — Protobuf

Compatibility #3 — karapace

- MESSAGE_MOVE
- MESSAGE_DROP
- FIELD_LABEL_ALTER
- FIELD_KIND_ALTER
- FIELD_TYPE_ALTER
- FIELD_TAG_ALTER
- ONE_OF_FIELD_DROP
- FEW_FIELDS_CONVERTED_TO_ONE_OF

7.2 Форматы — Protobuf

Compatibility #4 — redpanda
clck.ru/3D3s3Q

```
bool check_compatible(...) {
  for (int i = 0; i < writer->field_count();
  ++i) {
    if (r && !check_compatible(r, writer
    >field(i))) {
      return false;
    }
```

```
bool check_compatible(...) {
  switch (writer->type()) {
  ...
  case pb::FieldDescriptor::
  Type::TYPE_SFIXED64:
    return check_compatible(
      get_encoding(reader->type()), get_encoding(writer->type()));
    }
```

```
bool check_compatible(encoding reader, encoding writer) {
  return reader == writer && reader !=
  encoding::struct_;
}
```

7.2 Форматы — Protobuf

Compatibility #5 — protolock (это утилита проверки изменения proto-схем) clck.ru/3D3sFV

- NoUsingReservedFields — попытка использовать reserved field number
- NoRemovingReservedFields — попытка удалить reserved field number
- NoRemovingFieldsWithoutReserve — попытка удалить поле без reserved
- NoChangingFieldIDs — попытка полю изменить номер
- NoChangingFieldTypes — попытка полю изменить тип
- NoChangingFieldNames — попытка изменить имя поля
- NoRemovingRPCs — попытка удалить функцию внутри gRPC сервиса
- NoChangingRPCSignature — попытка изменить сигнатуру gRPC функции
- NoMovingExistingFieldsIntoOrOutOfOneof — попытка переместить существующее поле в oneof

7.2 Форматы — Protobuf

Compatibility #6 — apicurio
clck.ru/3D3sSJ

Правила из protolock, но NoMovingExistingFields IntoOrOutOfOneof заменено checkRequiredFields

```
public List<ProtobufDifference> findDifferences() {
    List<ProtobufDifference> totalIssues = new ArrayList<>();
    totalIssues.addAll(checkNoUsingReservedFields());
    totalIssues.addAll(checkNoRemoving ReservedFields());
    totalIssues.addAll(checkNoRemoving FieldsWithoutReserve());
    totalIssues.addAll(checkNoChangingFieldIDs());
    totalIssues.addAll(checkNoChangingFieldTypes());
    totalIssues.addAll(checkNoChangingFieldNames());
    totalIssues.addAll(checkNoRemovingServiceRPCs());
    totalIssues.addAll(checkNoChangingRPCSignature());
    if (Syntax.PROTO_2.equals(fileBefore.getSyntax())) {
        totalIssues.addAll(checkRequiredFields());
    }
    return totalIssues;
}
```

7.2 Форматы — Protobuf

Compatibility #7 — buf

https://github.com/bufbuild/buf/blob/main/private/bufpkg/bufcheck/bufbreaking/internal/bufbreakingv1/vars.go#L24 — 66 rules

https://github.com/bufbuild/buf/blob/main/private/bufpkg/bufcheck/bufbreaking/internal/bufbreakingv2/vars.go#L24 — 65 rules

Тут правила делятся на 4 группы:

- FILE — то, что ломает сборку per-file
- PACKAGE — то, что ломает сборку per-package
- WIRE_JSON — то, что ломает protojson
- WIRE — то, что ломает binary encoding

7.2 Форматы — Protobuf

Compatibility #8 — stencil

https://github.com/raystack/stencil/blob/main/formats/protobuf/compatibility.go#L12 — 16 правил

- messageDelete
- nonInclusivereservedRange
- nonInclusiceReservedNames
- fieldDelete
- fieldDeleteWithoutReservedNumber
- fieldDeleteWithoutReservedName
- fieldNameChange
- fieldLabelchange
- fieldKindChange
- fieldTypeChange
- enumDelete
- enumValueDelete
- enumValueDeleteWithoutReservedNumber
- enumValueDeleteWithoutReservedName
- enumValueNumberChange
- syntaxChange

7.2 Форматы — Protobuf

Каждая имплементация проверок обратной совместимости для protobuf — это собственная трактовка понятия «совместимость», и полезно знать какие проверки реализует ваш инструмент,
и все ли вам необходимые проверки в нём есть

7.2 Форматы — Protobuf

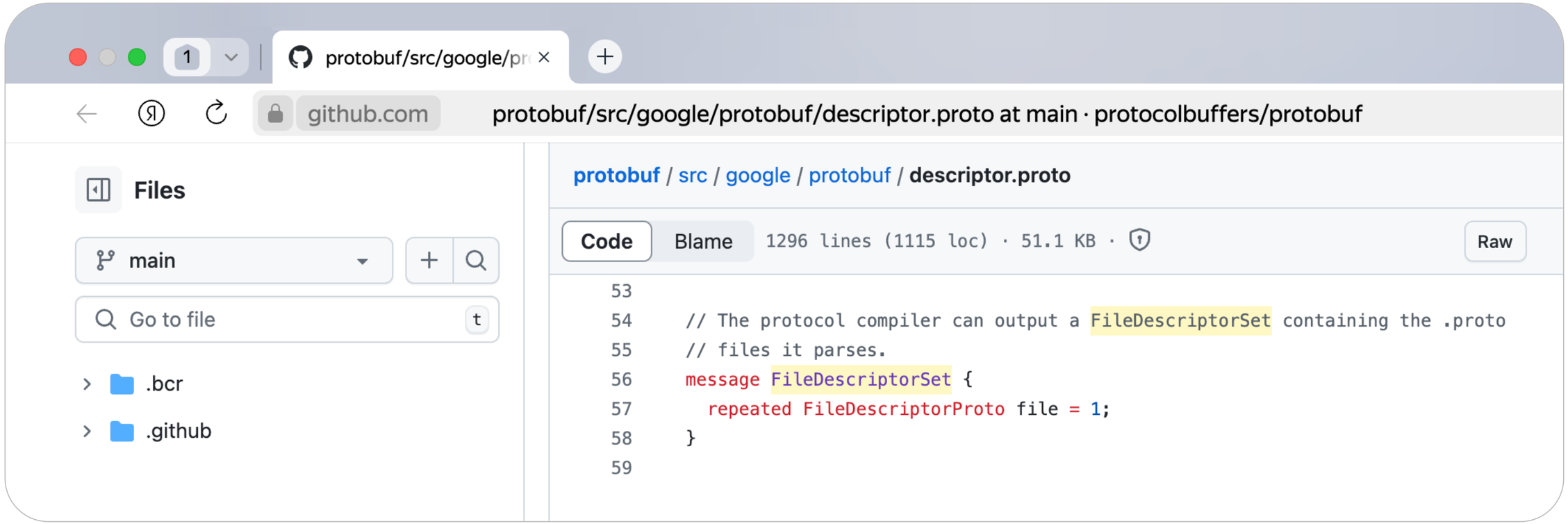
desc

Desc-файл (aka FDS — File Descriptor Set) — это такой сериализованный proto message, в котором содержится распаршенная информация из .proto-файлов.

Обычно там содержится всё дерево зависимостей proto-файлов, что делает desc-файл самодостаточным описанием сериализованного сообщения.

github.com/protocolbuffers/protobuf/blob/main/src/google/protobuf/descriptor.proto

7.2 Форматы — Protobuf



github.com/protocolbuffers/protobuf/blob/main/src/google/protobuf/descriptor.proto

7.2 Форматы — Protobuf

Удобные инструменты для траблшутинга:

- protoscope — github.com/protocolbuffers/protoscope
- kaitai struct —kaitai.io/

7.2 Форматы — Protobuf

	Confluent SR API	Storage	AVRO	Protobuf (queues)	Protobuf (API)	Configurable protobuf	Linting
Confluent	⊕	Kafka-only	⊕	⊕			
Karapace	⊕	Kafka-only	⊕	⊕			
Redpanda	⊕	Kafka-only	⊕	⊕			
Apicurio	⊕	Kafka, PostgreSQL, Microsoft SQL Server	⊕	⊕	⊕		
Buf	⊕	Proprietary commercial SaaS		⊕	⊕	⊕	⊕
Stencil		PostgreSQL	⊕				

7.2 Форматы — JSON Schema

7.3 Форматы — JSON Schema

JSON Schema — способ описать **валидацию** JSON-документов

7.3 Форматы — JSON Schema

JSON Schema — способ описать **валидацию** JSON-документов

JSON Schema — сама по себе является JSON-документом

7.3 Форматы — JSON Schema

JSON Schema — способ описать **валидацию** JSON-документов

JSON Schema — сама по себе является JSON-документом

эволюция схемы – не описана в формате

7.3 Форматы — JSON Schema

JSON Schema — способ описать **валидацию** JSON-документов

JSON Schema — сама по себе является JSON-документом

эволюция схемы – не описана в формате

JSON Schema — настолько большой космолёт,
что невольно задумываешься,
а не тьюринг-ли он полный —
но нет, и на том спасибо

7.3 Форматы — JSON Schema

Для самостоятельного изучения:

- grammar-based & rule-based schema languages
- content model – open, closed, partially

clck.ru/3D3rBh

7.3 Форматы — JSON Schema

Также в JSON Schema есть:

- Несколько диалектов
- Pattern properties
- String: minLength, maxLength
- Pattern — для регулярок
- Format — для времени
- Куча типов данных, в т.ч.: даты, ip, uuid, uri, email, …
- Json-pointer (указатель на другую схему)
- Диапазоны: minimum/maximum/exclusiveMinimum/…
- Required
- MinProperties/maxProperties
- Массивы — minItems/maxItems/uniqueItems
- Кортежи
- MinContains/maxContains
- AllOf/anyOf/oneOf/not
- DependentRequired
- DependenciesSchemas
- If-then-else
- $schema, $id, $anchor, $ref, $def
- …

7.3 Форматы — JSON Schema — impl

Confluent

clck.ru/3D3ovb — 113 изменений

clck.ru/3D3p2t — 57 из 113 — compatible

56 из 113 — incompatible

Apicurio

clck.ru/3D3pFU — 170 изменений

81 из 170 — compatible

89 из 170 — incompatible

Karapace

clck.ru/3D3pia

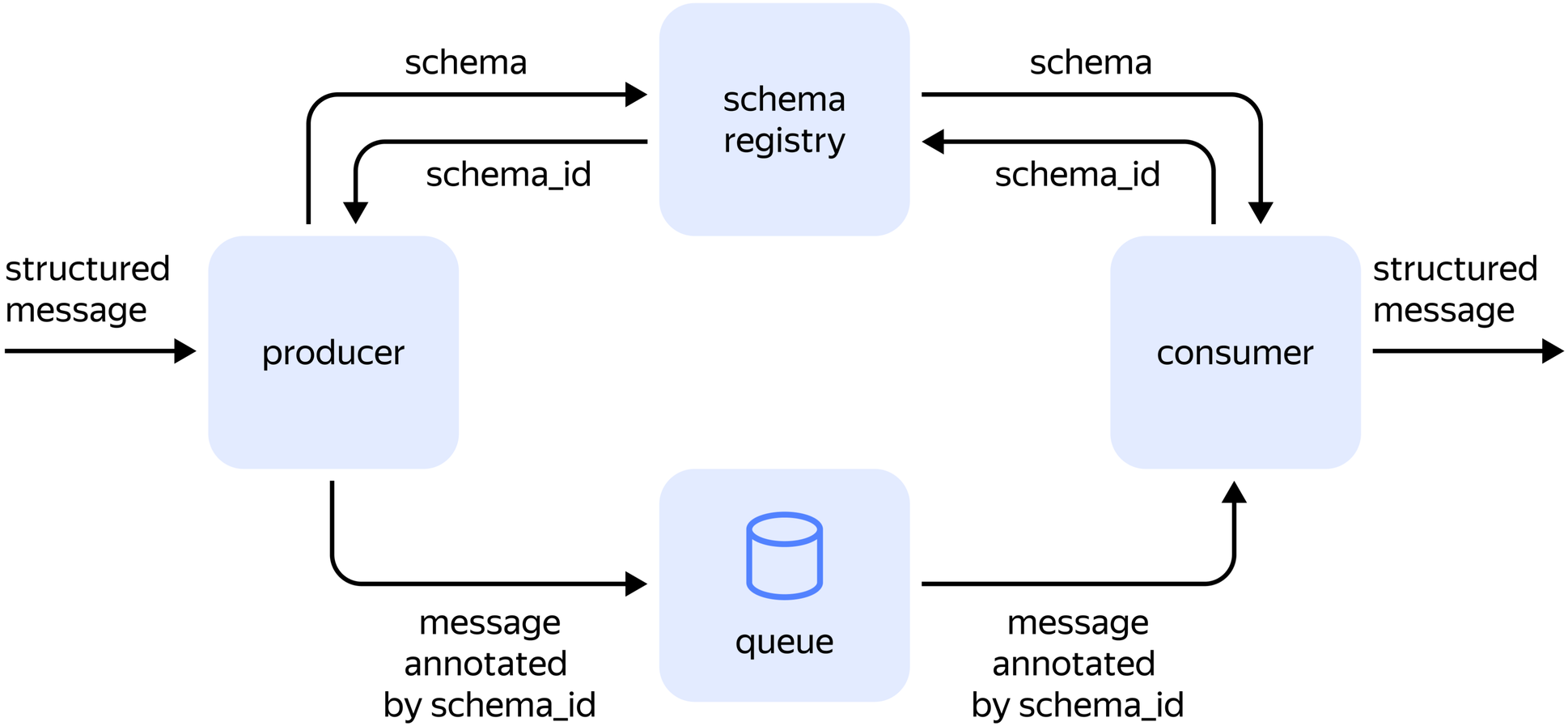
52 — incompatible

7.3 Форматы — JSON Schema

	Confluent SR API	Storage	AVRO	Protobuf (queues)	Protobuf (API)	Configurable protobuf	Linting	Json Schema
Confluent	⊕	Kafka-only	⊕	⊕				⊕
Karapace	⊕	Kafka-only	⊕	⊕				⊕
Redpanda	⊕	Kafka-only	⊕	⊕				⊕
Apicurio	⊕	Kafka, PostgreSQL, Microsoft SQL Server	⊕	⊕	⊕			⊕
Buf	⊕	Proprietary commercial SaaS		⊕	⊕	⊕	⊕	
Stencil		PostgreSQL	⊕					

8. Что использовать в очередях — Avro, Protobuf или JSON?

8.1 Что использовать в очередях — Avro, Protobuf или JSON? Логический уровень



8.1 Что использовать в очередях — Avro, Protobuf или JSON? Логический уровень

На логическом уровне нет разницы, какой формат был использован на транспорте

8.2 Что использовать в очередях — Avro, Protobuf или JSON? Размер данных

JSON чуть тяжелее чем protobuf & AVRO

Protobuf чуть-чуть тяжелее AVRO, но вряд ли это будет решающим фактором

AVRO поддерживает кодеки компрессии

8.3 Что использовать в очередях — Avro, Protobuf или JSON? Производительность

Protobuf обычно по бенчмаркам производительнее AVRO (обычно десятки процентов), но вряд ли это будет решающим фактором

8.4 Что использовать в очередях — Avro, Protobuf или JSON? Типы данных

В AVRO несколько богаче система типов — например, в protobuf & JSON нет uuid & decimal, но вряд ли это будет решающим фактором

8.5 Что использовать в очередях — Avro, Protobuf или JSON? Default values

Default values:

Есть в AVRO и JSON Schema	Есть в proto2	Нет в proto3

8.6 Что использовать в очередях — Avro, Protobuf или JSON? Рекомендация confluent

Any format, be it XML, JSON, or ASN.1, provided it is used consistently across the board, is better than a mishmash of ad hoc choices.

confluent.io/blog/event-streaming-platform-2/

In our own use, and in working with a few dozen companies, we have found Apache Avro to be easily the most successful format for streaming data

8.7 Что использовать в очередях — Avro, Protobuf или JSON? — Общие соображения

JSON human-readable, зачастую это удобно

Если в проекте используется только один формат — тащить другой выглядит излишне

Если у кого-то в команде есть опыт работы с каким-то одним форматом — логично выбрать его

Если в вашем проекте есть ограничения на форматы — очевидно это будет главным фактором выбора

8.8 Что использовать в очередях — Avro, Protobuf или JSON? Вывод

Что же выбрать:

- Однозначного ответа нет
- На логическом уровне разницы нет
- По размеру чуть лучше protobuf/AVRO чем JSON
- AVRO поддерживает кодеки компрессии
- По производительности protobuf чуть лучше AVRO
- По типам данных AVRO побогаче
- Default values: есть в AVRO/JSON Schema/proto2
- Confluent рекомендует AVRO
- JSON — human-readable
- Важнейшим фактором будет сам проект

9. О своей разработке

«Дзен и искусство ухода за мотоциклом»

9. О своей разработке

Функциональные требования нашей Schema Registry:

- Покрыть оба кейса:
  - Очереди сообщений
  - Разработка API
- Принести пользу и внутри компании, и в облаке
- Интегрироваться с другими нашими облачными инструментами
- Очень желательно быть совместимым с confluent schema registry API
- Наличие UI

9. О своей разработке

Для соответствия функциональным требованиям, нужно всего лишь:

- Иметь возможность гибко настраивать понятие compatibility для protobuf
- Поддерживать несколько форматов

  Как минимум: protobuf, AVRO и JSON schema
- Предоставлять desc-API, с дедупликацией на стороне стораджа
- Предоставлять изоляцию пользователей
- Иметь техническую возможность быть managed-решением
- Иметь лицензионную возможность быть частью облака
- Иметь контроль над кодовой базой

9. О своей разработке

Получили сервис:

- Написанный на golang
- С классической архитектурой (K8s, Postgres, ALB)
- Managed-решение
- Multi-tenant решение с двухуровневой изоляцией пользователей
- Один tenant — решение, совместимое с confluent schema registry API
- API одного тенанта расширено для дополнительного desc-API
- Единая подсистема хранения — и для confluent API и для desc-API
- Поддержка форматов: AVRO, JSON Schema, protobuf
- Для protobuf реализован механизм политик сравнения
- UI

10. Дополнительные материалы

10. Дополнительные материалы

	protobuf.dev/
	protobuf.com/docs/introduction
Confluent schema registry doc	docs.confluent.io/platform/current/schema-registry/index.html
Karapace — quick start	karapace.io/quickstart
yokota.blog	yokota.blog/2021/08/26/understanding-protobuf-compatibility yokota.blog/2021/03/29/understanding-json-schema-compatibility
Эволюция схемы данных. Носим данные из реляционной СУБД в Hadoop	clck.ru/3D3aDX
Protobuf и buf: блеск, нищета и импортозамещение	habr.com/ru/companies/oleg-bunin/articles/816631
You need two schemas to deserialize an Avro message… but which two?	dalelane.co.uk/blog/?p=5031
Schema Registry с Protobuf в Kafka — зачем оно надо?	habr.com/ru/companies/lenta_utkonos_tech/articles/715298
Protobuffers Are Wrong	reasonablypolymorphic.com/blog/protos-are-wrong
Amazon EventBridge Schema Registry: Discover, Create, and Manage OpenAPI Schemas for Event-Driven Architectures	diversedaily.com/amazon-eventbridge-schema-registry-discover-create-and-manage-openapi-schemas-for-event-driven-architectures

11. Итоги

11. Итоги - таблица

	Confluent SR API	Storage	AVRO	Protobuf (queues)	Protobuf (API)	Configurable protobuf	Linting	Json Schema
Confluent	⊕	Kafka-only	⊕	⊕				⊕
Karapace	⊕	Kafka-only	⊕	⊕				⊕
Redpanda	⊕	Kafka-only	⊕	⊕				⊕
Apicurio	⊕	Kafka, PostgreSQL, Microsoft SQL Server	⊕	⊕	⊕			⊕
Buf	⊕	Proprietary commercial SaaS		⊕	⊕	⊕	⊕	
Stencil		PostgreSQL	⊕					

11.2 Итоги

- Схематизация — это must have
- Нужно использовать Schema Registry — с ним жизнь станет проще
- Мы посмотрели на JSON/AVRO/Protobuf

  Внутреннее устройство

  Тонкости эволюции

  Что бывает, когда эволюция схемы — часть дизайна, и когда нет

  Исходник

  Что из этого выбрать

- Мы посмотрели что есть на рынке, узнали что и как выбрать
- Разобрали множество неочевидных нюансов
- Обсудили пример, когда стоит написать своё решение

Спасибо за внимание!

Тимофей Брунько
Разработчик Yandex Cloud
timmyb32r@gmail.com

t.me/timmyb32r