

SMARTDATA 2026

Контракты данных

Когда схема становится договором

Никита Борзунов

Uzum · Data Platform

События Kafka – в таблицы для аналитики

На входе



JSON-события

Вложенная структура
Поля и типы источника



Нужны аналитику



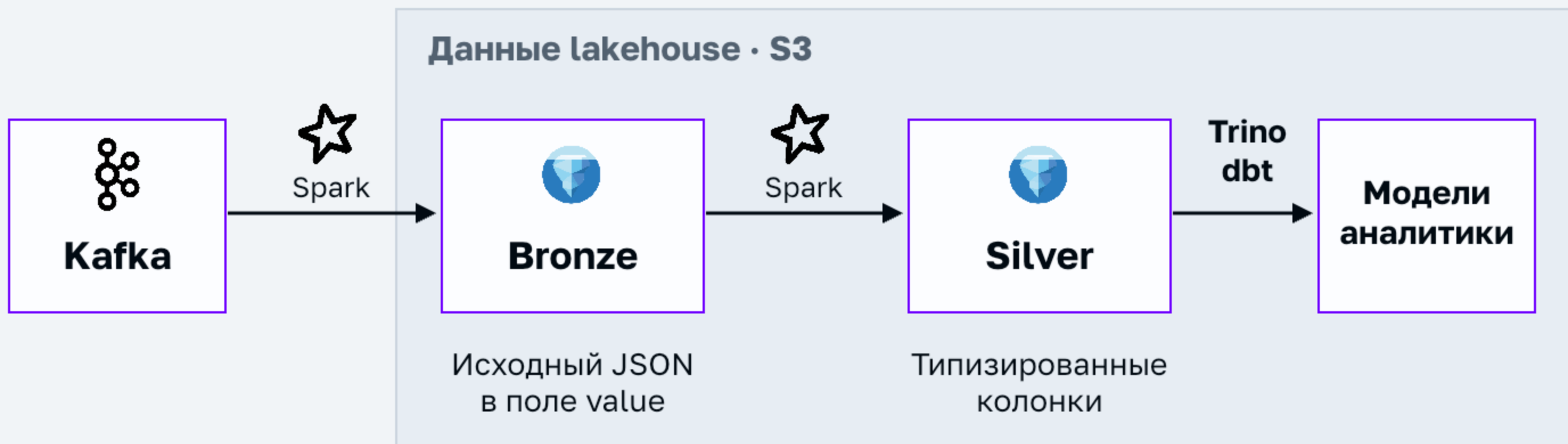
Таблицы Silver

Понятные колонки
Описание и проверки

Доставить · подготовить · описать · проверить

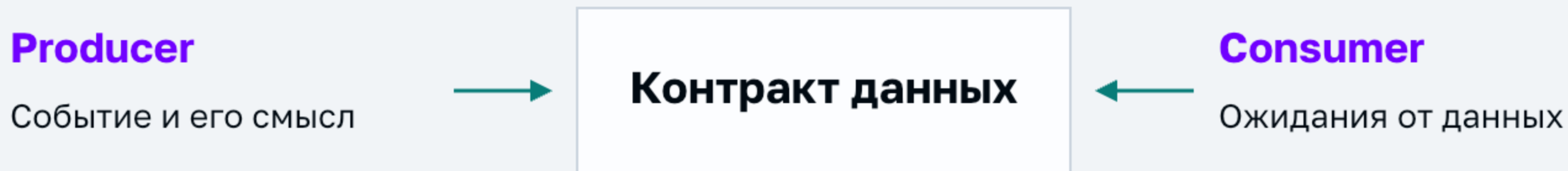
Архитектура загрузки событий

Сервис публикует JSON-события в Kafka



Что нужно согласовать перед загрузкой?

Договорённости фиксируем в контракте



Кто уже использует контракты данных?

Ручной процесс после согласования контракта

Дальше каждый технический шаг команды выполняли вручную



Согласования между командами и ожидание спринтов

Ручные действия заменяет генератор

Ручной процесс

Что готовили

Kafka topic и ACL

Схемы и стримеры

Развёртывание

Описание и правила DQ

Кто

Инфраструктура

DE

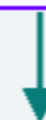
DE / DevOps

DE / DA

По контракту

CI · автогенерация

Код и конфигурации
Метаданные



DE

Review · approve · merge

Дни или недели

Несколько часов

Оценка команды для типового сценария с согласованным контрактом

Два этапа наполнения контракта

01 Producer

Что и откуда поступает

Сервис-источник и Kafka topic
Формат, поля, типы и ключи
Владелец со стороны backend

02 Платформа и аналитика

Как загружаем и используем

Смысл и сценарий использования
Стратегия загрузки в Bronze/Silver
DQ, freshness, ownership

Платформа и аналитика дополняют техническую часть контракта

Контракт: событие, загрузка и обязательства

Событие и структура

```
x-event-description: >
  Отправка и доставка push-уведомления
x-event-topic:
  cdp-push-notifications-sent.v1
x-event-cluster: common
type: object
properties:
  event_type: {type: string}
  event_date:
    type: string
    format: date-time
  event_data:
    type: object
    properties:
      campaign_id: {type: integer}
```

Стратегия загрузки

```
x-data-platform:
  bronze_storage: iceberg
  silver_write_mode: append
  starting_offsets: earliest
  max_offsets_per_trigger: 100000
  max_files_per_trigger: 50
  trigger_interval: "1 minute"
  expand_nested_json: true
```

Обязательства: целевая схема

```
ownership: {producer: crm_backend,
             consumer: Marketing Analytics}
sla: {freshness: 5m}
data_quality:
  required: [event_date, event_type]
  event_data.campaign_id: {min: 1}
```

Автоматизация ручных операций

Решения команды

- Смысл события и требования
- Параметры загрузки
- Проверка и согласование

Работа генератора

- Схемы, DDL и стримеры
- Конфигурации развёртывания
- Описание объектов в каталоге

Сквозной процесс

Запуск генерации



Review и развёртывание



Данные в Silver

Три этапа интеграции по контракту

01**02****03**

Согласовать контракт

YAML + схема события
CI и LLM review
Owners + DE → merge

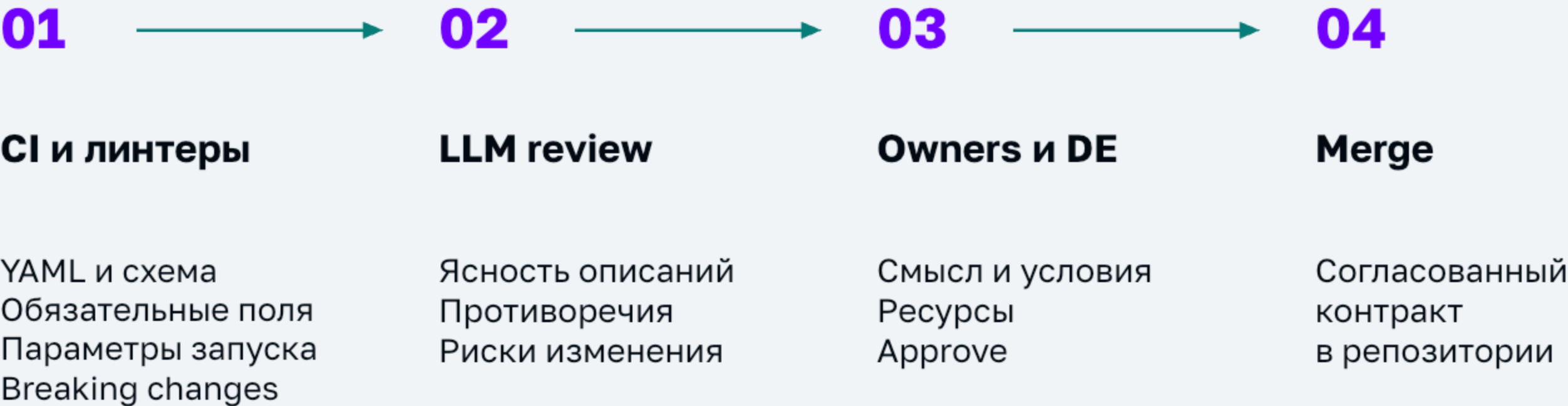
Подготовить Kafka

Topic и права доступа
Конфигурация Terraform
Review → apply

Создать интеграцию

AVSC	Схема события
Python	Код стримеров
Helm	Ресурсы и запуск
ArgoCD	Развёртывание

01 Проверка PR с контрактом



02 Создание Kafka topic

Согласованный контракт

```
topic: cdp-push-notifications-sent.v1  
cluster: common  
partitions / replication  
retention / producer ACL
```

Инфраструктурная конфигурация

→ **Topic и права доступа
в Terraform / Terragrunt**

PR и review



Merge



Drone apply



Kafka topic



01

02

03 ПЛАТФОРМА

✓ Контракт

Kafka

Слои

AVSC

Bronze

Silver

DQ

Запуск

OMD

03 Генерация платформенной интеграции

3.2 Генерация AVSC

Торіс + контракт

Схема события для генератора

3.3–3.8 Артефакты интеграции

AVSC + параметры

Python, Helm, DQ и маршрутизация

3.9 Публикация и запуск

Публикация + review DE

Push / PR → ArgoCD → Spark

3.10 Каталог данных

Отдельная ветка CI

Описание и lineage в OMD

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

Слои

AVSC

Bronze

Silver

DQ

Запуск

OMD

3.1 Слои хранения в интеграции

Генерируем Bronze и Silver

Gold создаёт DA



Bronze

Iceberg

Исходный JSON
в поле value
Kafka metadata



Silver

Iceberg

Типизированные поля
Target и DLQ



Gold

Iceberg

Витрины
и аналитические ETL

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

Слои

AVSC

Bronze

Silver

DQ

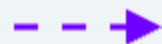
Запуск

OMD

3.2 Генерация Avro-схемы

YAML-контракт

```
event_data:  
  type: object  
  properties:  
    campaign_id:  
      type: integer  
    message_id:  
      type: integer
```



AVSC: вложенный record

```
{  
  "name": "event_data",  
  "type": "record",  
  "fields": [  
    {"name": "campaign_id",  
      "type": ["null", "int"]},  
    {"name": "message_id",  
      "type": ["null", "int"]}]  
}
```

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

✓ Слои

AVSC

Bronze

Silver

DQ

Запуск

OMD

3.3 Генерация загрузки Bronze

Вход: AVSC + параметры контракта

AVSC описывает JSON в value. Техническую схему Bronze задаёт шаблон.

Шаблон Bronze

```
key, value      STRING
partition, offset STRING
kafka_timestamp STRING
updated_at      STRING
date            DATE
```



DDL внутри Python-стримера



При запуске job создаёт Iceberg в S3 и регистрирует таблицу в Hive Metastore

На выходе

pyspark-etl

Python + Helm values

PR → infra-argocd

ArgoCD Application для Bronze

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

✓ Слои

✓ AVSC

Bronze

Silver

DQ

Запуск

OMD

3.4 Преобразование полей AVSC

Поля в AVSC

```
event_type: string
```

Прямое сопоставление типа

```
event_data: record  
  campaign_id: int  
  message_id: int
```

Колонки Silver

```
event_type STRING
```

```
event_data_campaign_id INT  
event_data_message_id INT
```

expand_nested_json: true

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

✓ Слои

✓ AVSC

✓ Bronze

Silver

DQ

Запуск

OMD

3.5 Генерация схемы Silver и DLQ

AVSC + настройки контракта

```
event_data.campaign_id: int  
expand_nested_json: true
```



parse_avsc

StructType · SELECT · DDL
в сгенерированном Python

При запуске Spark job



Исполнит DDL



Silver · Iceberg в S3



target

DLQ

Hive Metastore: имена таблиц и указатели на метаданные

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

✓ Слои

✓ AVSC

✓ Bronze

Silver

DQ

Запуск

OMD

3.6 Технические проверки перед Silver

Целевой контур · внутри Spark job Bronze → Silver

Обязательность

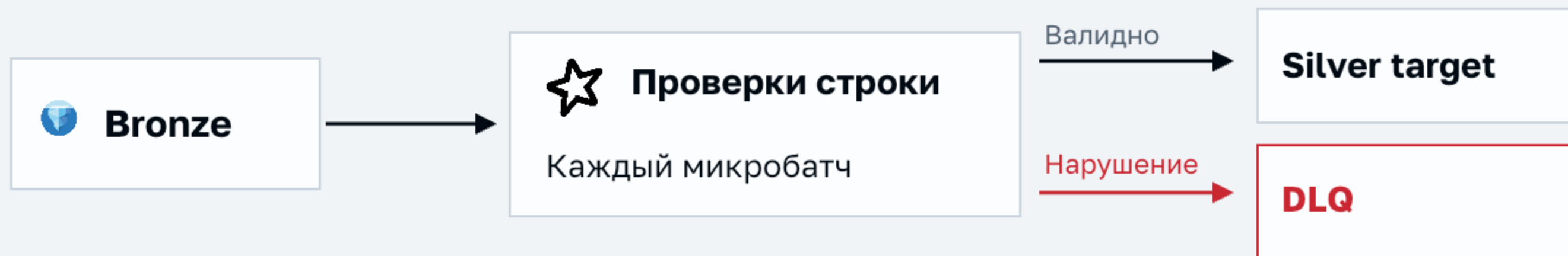
event_type IS NOT NULL

Допустимые значения

channel: push | email

Диапазон

campaign_id > 0



На выходе

Python: DQ + DDL target / DLQ

Helm values в pyspark-etl + PR ArgoCD Application в infra-argocd

01 02 03 ПЛАТФОРМА

✓ Контракт	✓ Kafka	✓ Слои	✓ AVSC	✓ Bronze	✓ Silver	DQ	Запуск	OMD
------------	---------	--------	--------	----------	----------	----	--------	-----

3.7 Бизнесовые проверки после Silver

Целевой контур · SQL и расписание в контракте

Silver

Загруженные данные



Планировщик

Каждые 4 часа



dbt singular test

SQL бизнесового правила

Проверяем окно $[T - 4 \text{ ч}, T)$

По бизнесовой дате события · например, лимит событий на клиента

Нарушение → alert → owner + DE → runbook

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

✓ Слои

✓ AVSC

✓ Bronze

✓ Silver

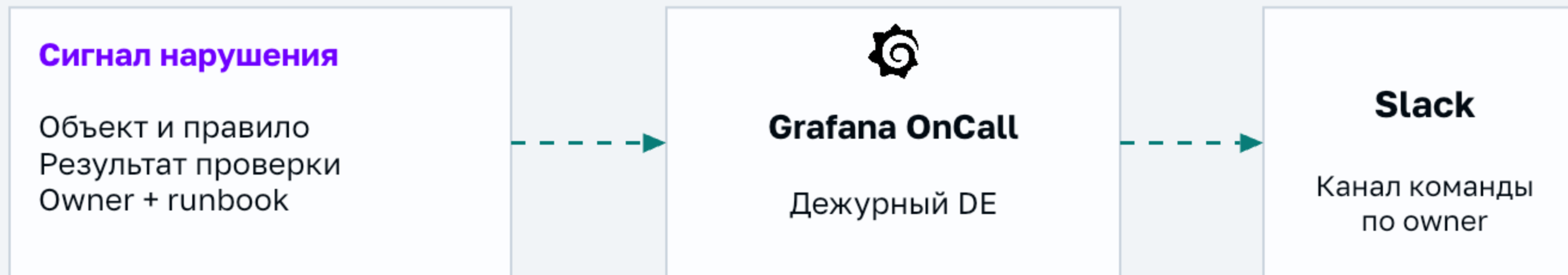
DQ

Запуск

OMD

3.8 Маршрут нарушения качества

Целевая схема маршрутизации



Runbook

DE и DA пишут инструкции вручную

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

✓ Слои

✓ AVSC

✓ Bronze

✓ Silver

DQ

Запуск

OMD

3.9 Публикация и запуск приложений

Публикация из CI

`pyspark-etl`

Python и Helm values: push

`infra-argocd`

ArgoCD Application: PR

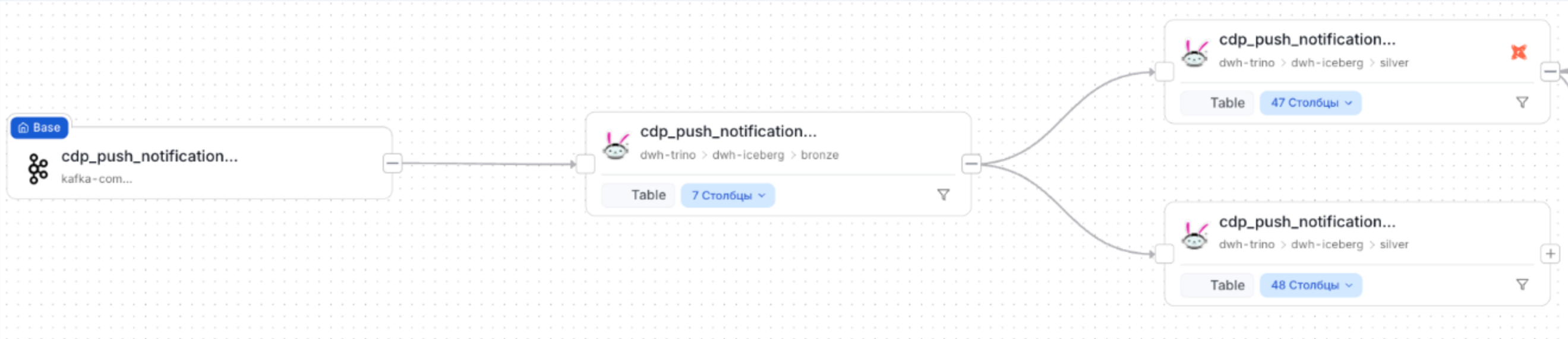
DE проверяет PR после успешной публикации кода



01 02 03 ПЛАТФОРМА

✓ Контракт	✓ Kafka	✓ Слои	✓ AVSC	✓ Bronze	✓ Silver	✓ DQ	Запуск	OMD
------------	---------	--------	--------	----------	----------	------	--------	-----

3.10 Описание и lineage в OpenMetadata



Описание и ответственность

Домены Marketing	Владельцы DE Team	Критичность Tier2	Период хранения Бессрочно	Сертификация --	Тип Regular
--	---	---	---	---------------------------------------	-----------------------------------

События отправки и доставки push-уведомлений

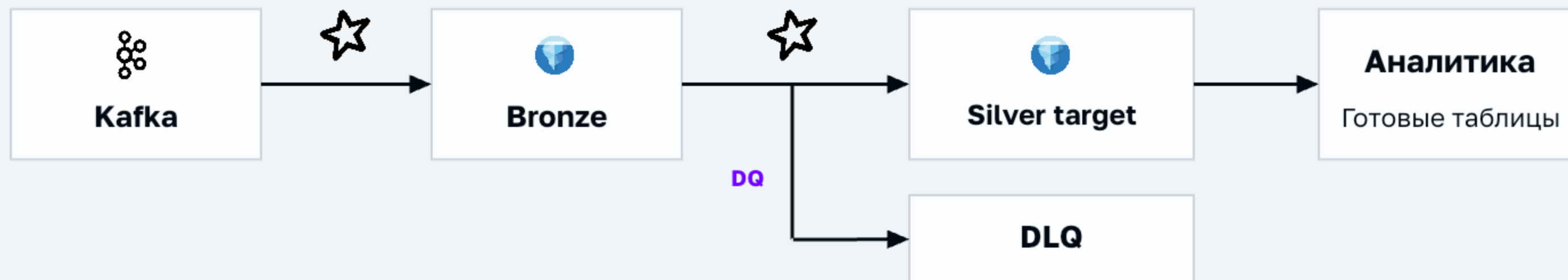
01	02	03 ПЛАТФОРМА						
✓ Контракт	✓ Kafka	✓ Слои	✓ AVSC	✓ Bronze	✓ Silver	✓ DQ	✓ Запуск	OMD

Полная схема интеграции

Контракт

-----> Генерация в CI

-----> Публикация и развёртывание



OpenMetadata

Описание и lineage

DQ и алерты: целевой контур

Grafana OnCall / Slack / runbook

01

02

03 ПЛАТФОРМА

✓ Контракт

✓ Kafka

✓ Слои

✓ AVSC

✓ Bronze

✓ Silver

✓ DQ

✓ Запуск

✓ OMD

Что остаётся за DE

Конфигурация

Ресурсы, режим записи и параметры чтения

Диагностика

Нагрузка, lag и freshness после изменений

История и миграции

Backfill, проверка полноты, перевод потребителей

Runbooks вместе с DA

Реакция на DQ: диагностика, данные, повтор

Как команды переходили на новый процесс

Продюсер · backend-команда

Новый топик начинается с технической схемы события

Поля, типы и формат сообщения
LLM skill помогает подготовить YAML

Потребитель · data-команда

Потребность в данных становится дополнением к тому же YAML

Смысл, DQ и SLA согласуют owners
DE принимает изменения
и контролирует доставку в Silver

Как помогали перейти

MVP на нужном топике



Демо и воркшоп



Первый запуск с поддержкой

Проблемы на этапе внедрения

Сохраняли структуру источника

```
event_data: {  
  campaign_id: 123  
}
```

Вложенность оставалась
Разбор ложился на аналитика

Аналитик ожидал готовые поля

→

```
event_data_campaign_id INT
```

Плоские типизированные колонки
Можно сразу писать SQL

Новый кейс расширял общий генератор

Новые случаи развивали общий генератор

Старый топик

НАБЛЮДЕНИЕ

Имя расходится
с контрактом

ИЗМЕНЕНИЕ

Сверка имени
в CI

РЕЗУЛЬТАТ

Ошибка видна
до генерации

Пустой источник

НАБЛЮДЕНИЕ

Producer ещё
не публикует события

ИЗМЕНЕНИЕ

Контроль наполнения
и alert

РЕЗУЛЬТАТ

Отличаем пустой topic
от сбоя доставки

Новая вложенность

НАБЛЮДЕНИЕ

Аналитику нужны
отдельные колонки

ИЗМЕНЕНИЕ

Общий mapping
в генераторе

РЕЗУЛЬТАТ

Следующая команда
получает готовые поля

Какие изменения считаем breaking

Правила проверки в генераторе

Удаление или переименование

Старое поле исчезает

Смена типа или формата

Меняется схема чтения

Новое required-поле

Старые события могут его не содержать

Старая схема + новая схема

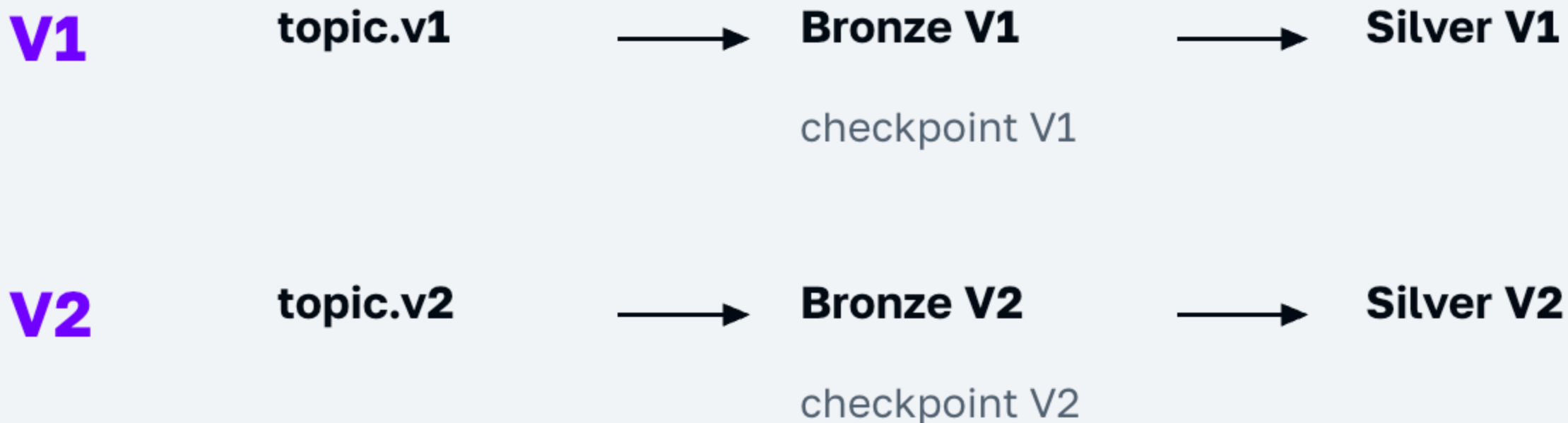


Ошибка проверки

Обязательное сравнение версий в CI нужно закрепить

Для V2 нужна отдельная интеграция

Сценарий миграции



Раздельные таблицы, S3-пути, SparkApplication и checkpoints

DE переводит всех потребителей на V2

Сценарий миграции

01	V2 рядом с V1	Раздельные потоки, таблицы и checkpoints
02	История V1 в V2	Преобразование и граница с текущим потоком
03	Сверка данных	Полнота, ключи, DQ и свежесть
04	Все потребители на V2	Проверка чтений и зависимых процессов
05	Остановка V1	Только после полного перехода потребителей
06	Удаление артефактов V1	После сверки истории и окончания окна отката

Наблюдение подсказывало, что менять



Grafana

Поток и ресурсы
Kafka lag, Spark CPU / RAM



Superset

Contracts и Trino
Автогенерация и чтение таблиц

Наблюдение

Параметры контракта

Проверка эффекта

Память драйвера почти у лимита

 Silver · LMS worker_action_event · driver

RAM **99,3%**
Лимит: 1,4 GiB

CPU **7,2%**
Лимит: 1 ядро

Пики в одном окне
% от лимита ресурса



Проверяем память и размер микробатча

DE меняет техническую конфигурацию

Технический блок YAML

```
x-data-platform:  
  max_offsets_per_trigger: 200000  
  trigger_interval: "1 minute"  
  executor:  
    instances: 4  
    memory: 4G
```

Дальше работает тот же CI

Merge DE и повторный запуск
Генерация Python / Helm
DE принимает deployment PR
ArgoCD применяет изменения



DE проверяет эффект

Ресурсы Spark, lag и freshness
Источника и Silver

Два сценария подключения топики

Новые топики

Технический контракт обязателен в стандартном процессе

Генерация и проверки CI

Существующие топики

Создаём контракт вручную и сверяем его с событиями

Помогаем с первым подключением

Когда данные нужны платформе

Дополняем бизнесовый блок и запускаем генерацию артефактов загрузки

Что именно измеряем

Охват контрактами

Топики с YAML / все топики в охвате

Автогенерация

Топики со стримерами в репозитории

Работа потока

Состояние загрузки и отставание

Использование

Команды, аккаунты и чтения Silver

Развиваем связь этих метрик через контракт и owner

Охват контрактами и автогенерацией

662 топики в охвате · 212 с контрактом

145 / 212

Техническая схема с полями

191 / 212

Описание события

46

Артефакты загрузки

Топики со сгенерированной загрузкой



Расширяем подход на вход и выход платформы

ПЛАНЫ

Контракт + CI

Общие схема, ownership, проверки и генерация

Входящие интеграции

CDC / API



Платформа данных

Исходящие интеграции

Платформа



Kafka / S3



Другие сервисы

Единый стандарт для каждой интеграции



Все объекты в OMD

Схема, описание, ownership и lineage



DQ и мониторинг для каждого объекта

Проверки источника и результата
Lag и freshness с согласованными порогами
Owner, Slack-канал и runbook для нарушений

Главный результат — новый способ работы

Самое сложное — изменить привычки команд

Опыт Kafka-событий — основа для внедрения
контрактов в другие сценарии интеграции

Вопросы

Контракты данных

Когда схема становится договором

Никита Борзунов

Uzum · Data Platform