Кто мы?

Карандасов Евгений

Старший QA Auto Engineer

Фролова Екатерина

Старший Data Engineer

Работа в хранилище

Глазами других...

Суть доклада

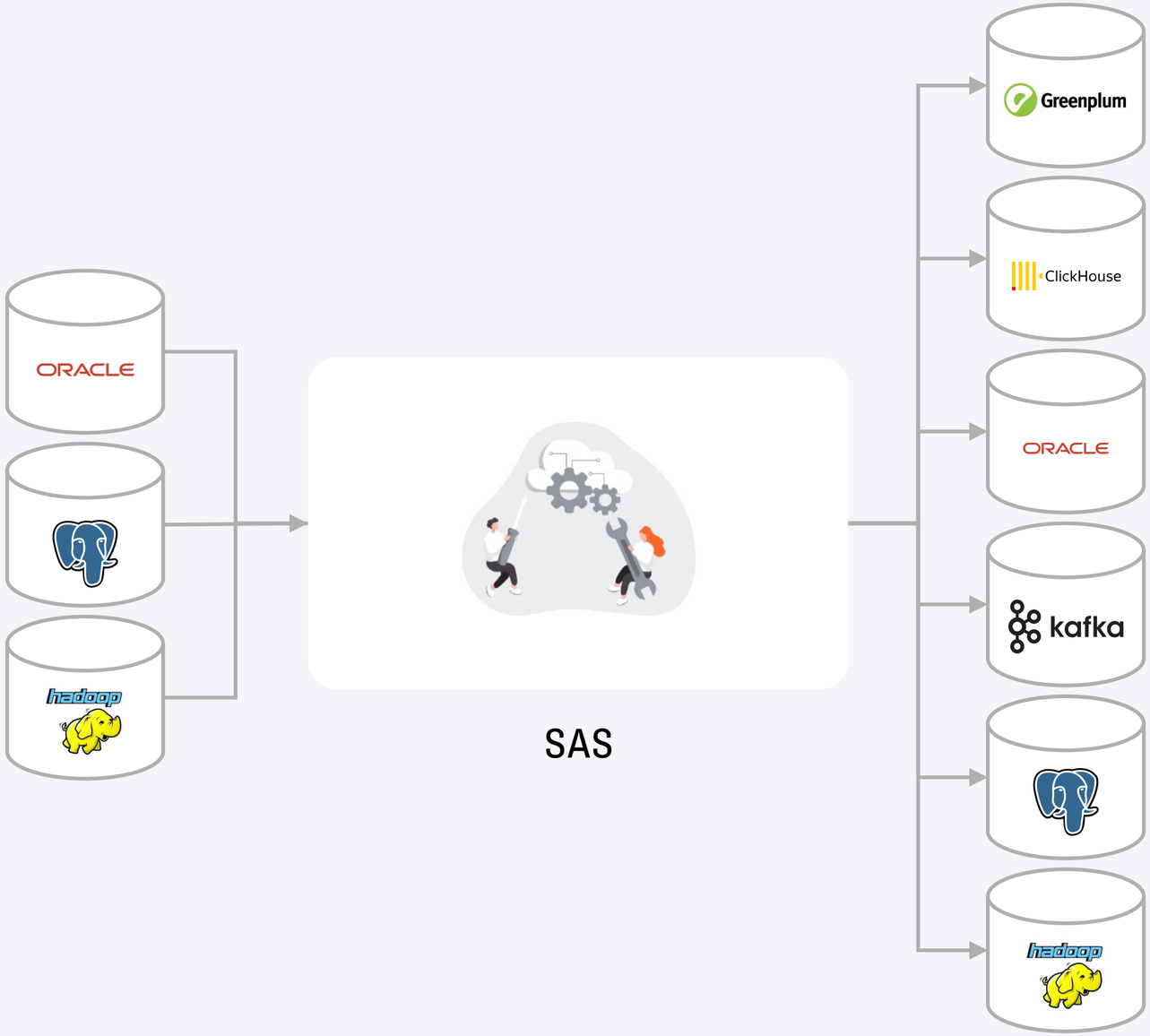
Как всего одна задача может повлиять на пересмотр устоявшегося взаимодействия DEV и QA внутри команды?

ТИНЬКОФФ

Как мы тестировали выгрузку из Greenplum в Kafka

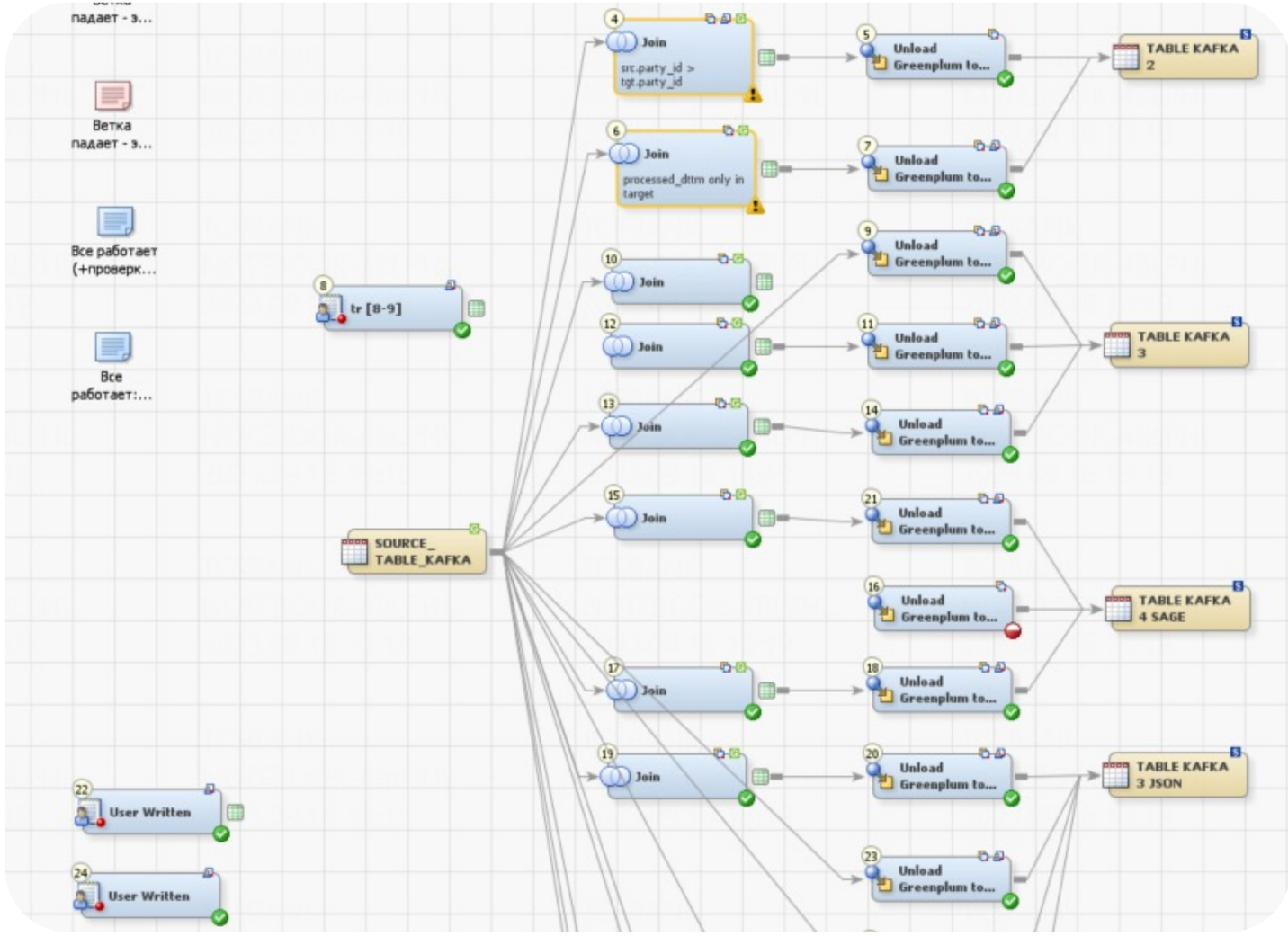
Наша работа

Как выглядели процессы?



Наша работа

Одна ветка = один тест



SAS – legacy инструмент

Проблемы

Процессов становилось все больше, а инструмент не тянул это

Проблемы

- Процессов становилось все больше, а инструмент не тянул это
- Текущий инструмент не развивался

Проблемы

- Процессов становилось все больше, а инструмент не тянул это
- Текущий инструмент не развивался
- Низкая производительность при большом количестве пользователей

Проблемы

- Процессов становилось все больше, а инструмент не тянул это
- Текущий инструмент не развивался
- Низкая производительность при большом количестве пользователей
- Сложность разработки процессов (SQL прототип -> ETL процесс)

Проблемы

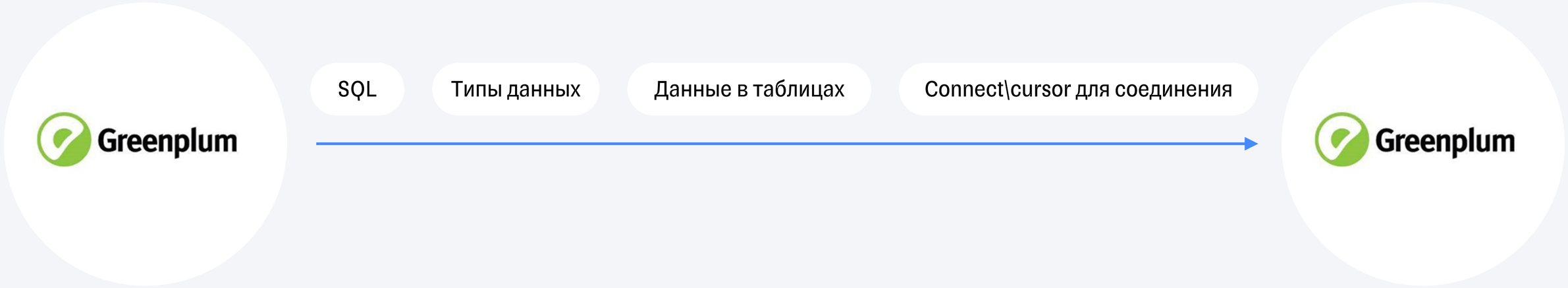
- Процессов становилось все больше, а инструмент не тянул это
- Текущий инструмент не развивался
- Низкая производительность при большом количестве пользователей
- Сложность разработки процессов (SQL прототип -> ETL процесс)
- Сложность тестирования процессов (нужны специальные ETL процессы)

Пришли санкции...

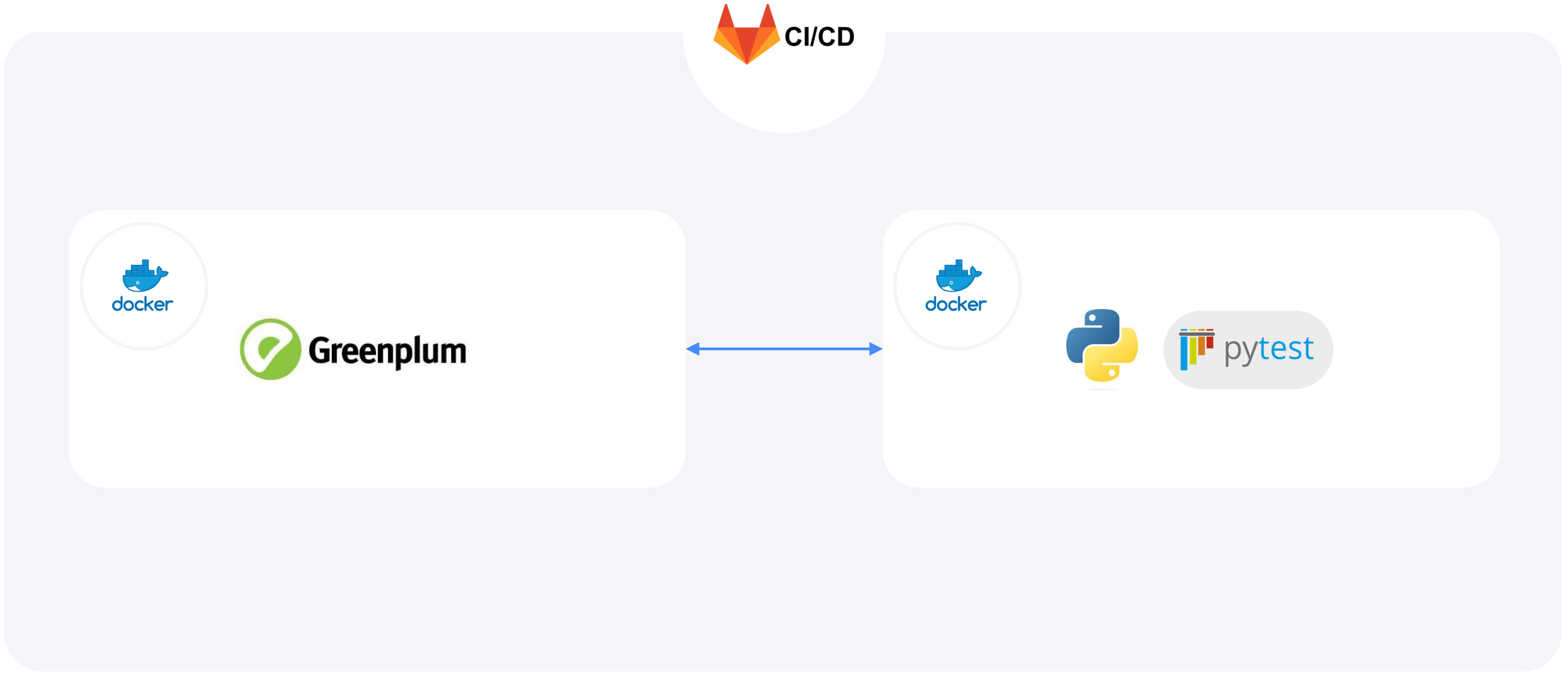
Надо ускорять
переход на TEDI

Greenplum To Greenplum

Все внутри одной базы данных



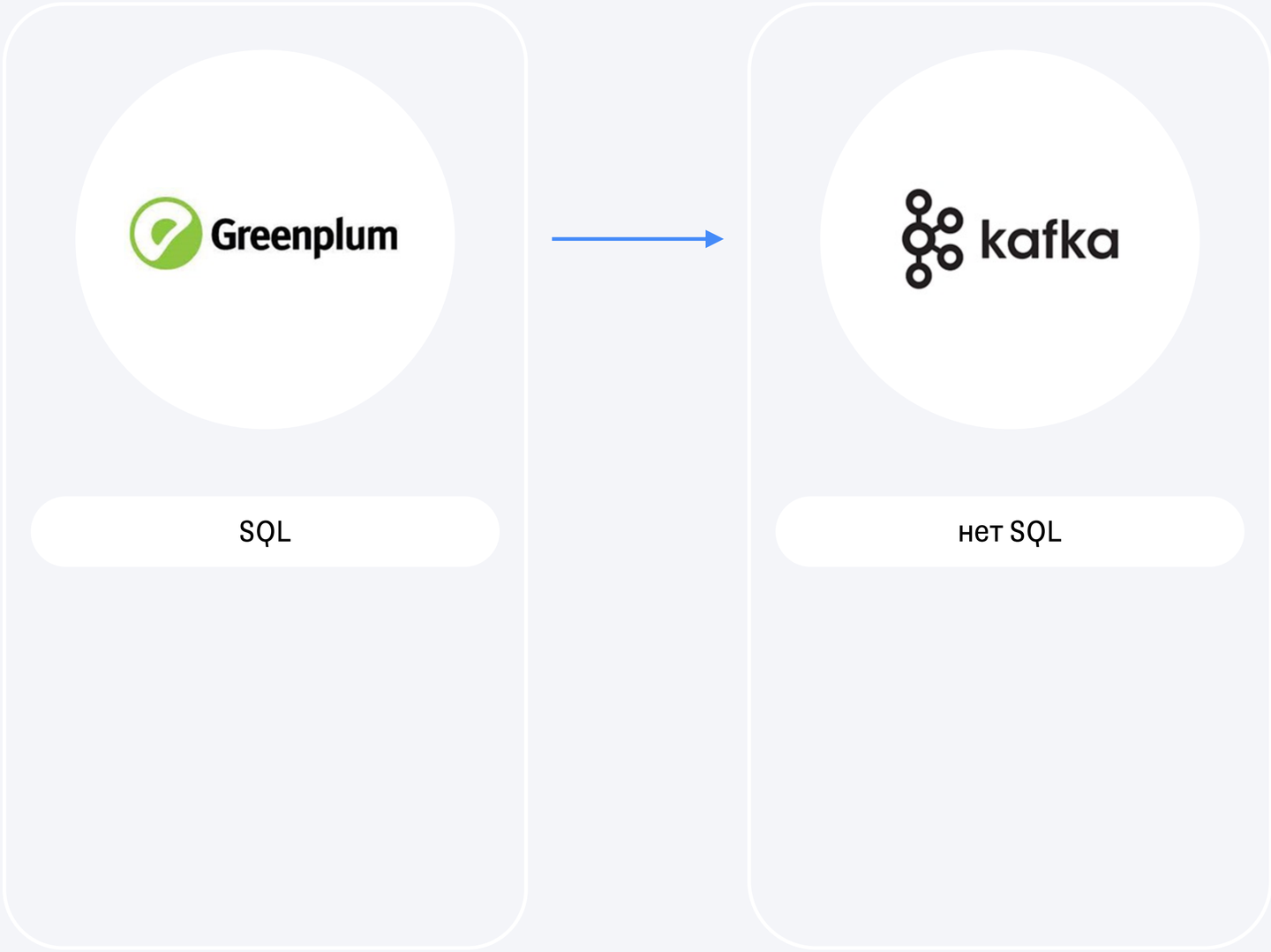
Как выглядел наш CI



Настала пора грузить не только в Greenplum

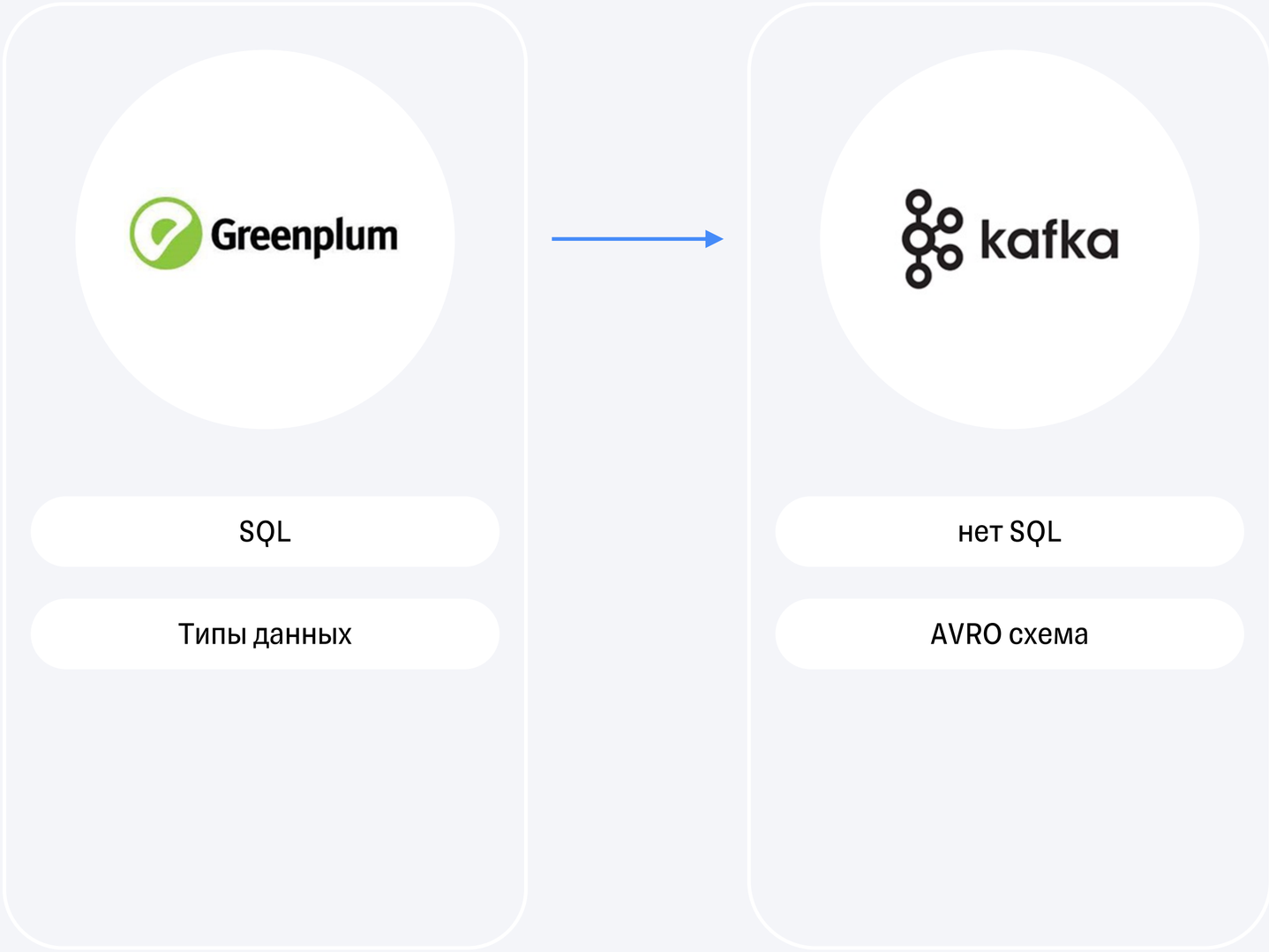
Greenplum To Kafka

Появилась новая система



Greenplum To Kafka

Появилась новая система



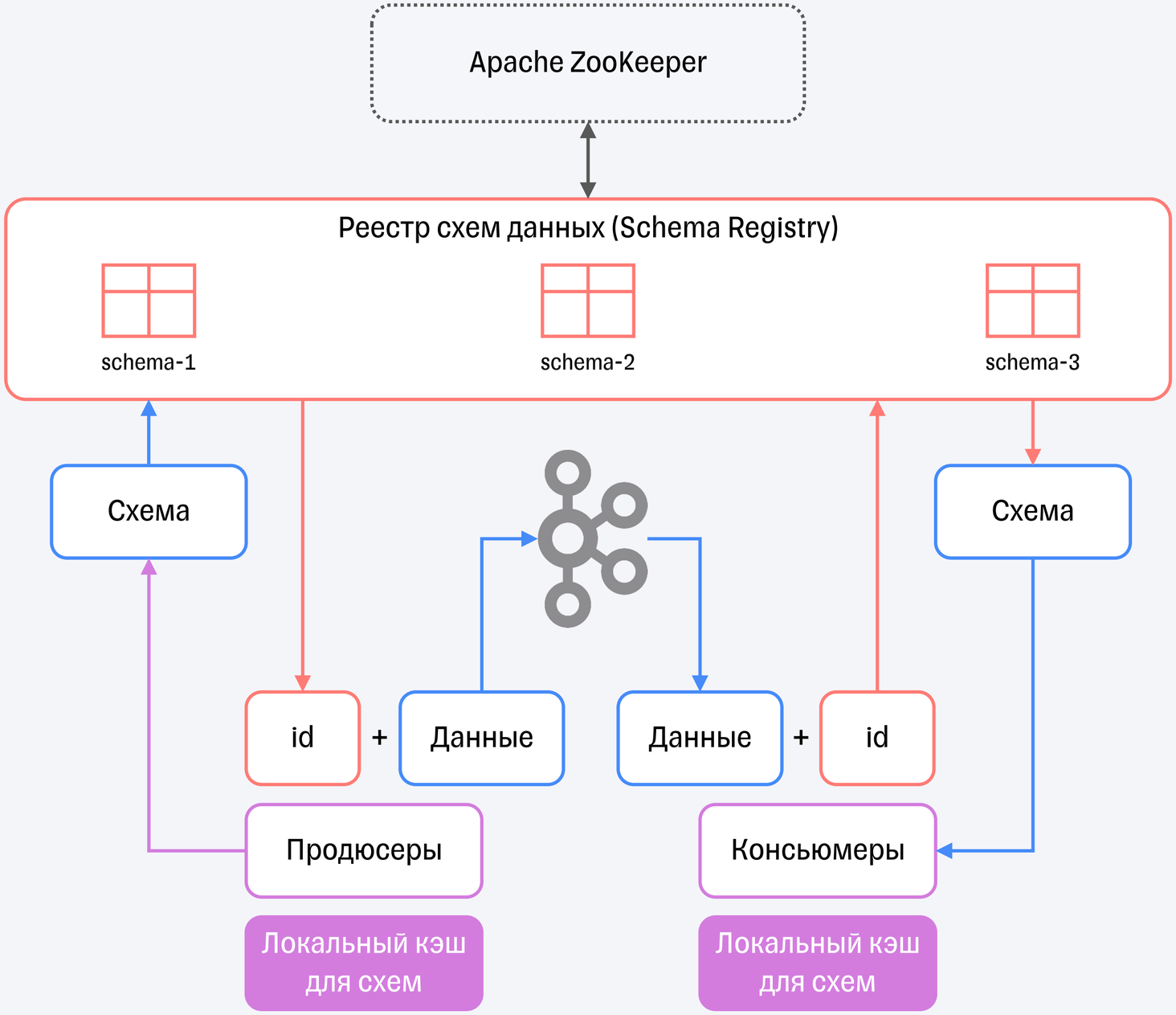
Greenplum To Kafka

Появилась новая система

Greenplum To Kafka

Появилась новая система

Как выглядит Kafka



Наши ограничения

SAS = TEDI

с точки зрения пользователя

Как мы видели интеграцию kafka?

- Писать в kafka, развернутую в тестовой инфраструктуре

Как мы видели интеграцию kafka?

- Писать в kafka, развернутую в тестовой инфраструктуре
- Использовать готовые библиотеки (Testcontainers)

Как мы видели интеграцию kafka?

- Писать в kafka, развернутую в тестовой инфраструктуре
- Использовать готовые библиотеки (Testcontainers)
- Подменить Producer на самописный и писать сообщения в файлы

Вариант 1

Использовать общую kafka

Плюсы

- Из инфраструктуры - нужен только свой топик
- Из настроек - только задать AVRO схему\схемы

Минусы

- Общая инфраструктура
- Нет изолированности

Вариант 2

Готовые библиотеки

Testcontainers

Плюсы

- Готовое решение
- Не нужно поднимать все зависимости kafka

Минусы

- Все равно использование docker
- Были проблемы с интеграцией

Вариант 3

Подменить Producer

Плюсы

- Не нужно настраивать реальную kafka
- Проверка результата = чтение из файла

Минусы

- Невозможно гарантировать правильность работы
- 100% успешные тесты не показатель

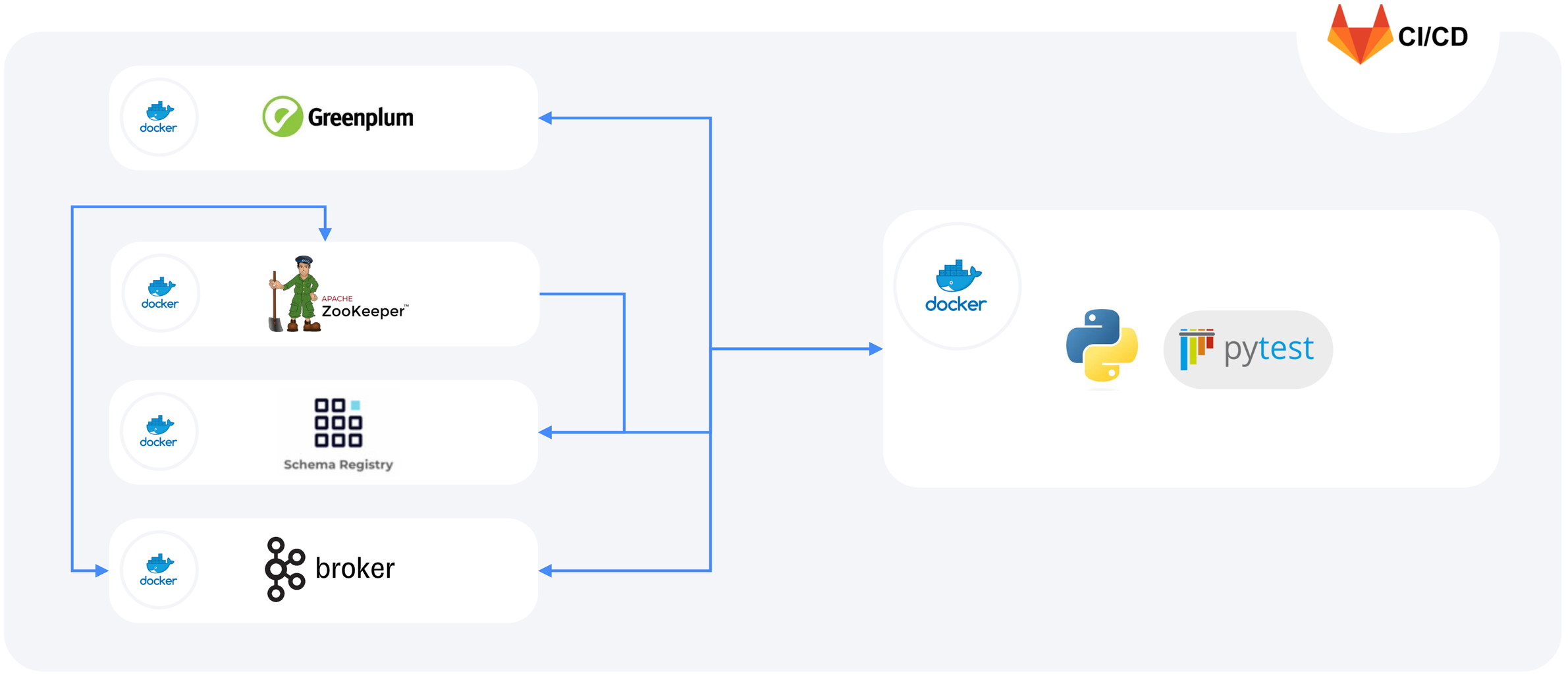
Какой вариант подходит нам?

А что выберете вы?

ТИНЬКОФФ

Мы выбрали качество данных → добавление kafka в CI

После интеграции Kafka CI стал выглядеть так



ТИНЬКОФФ

Пример YAML конфигурации CI

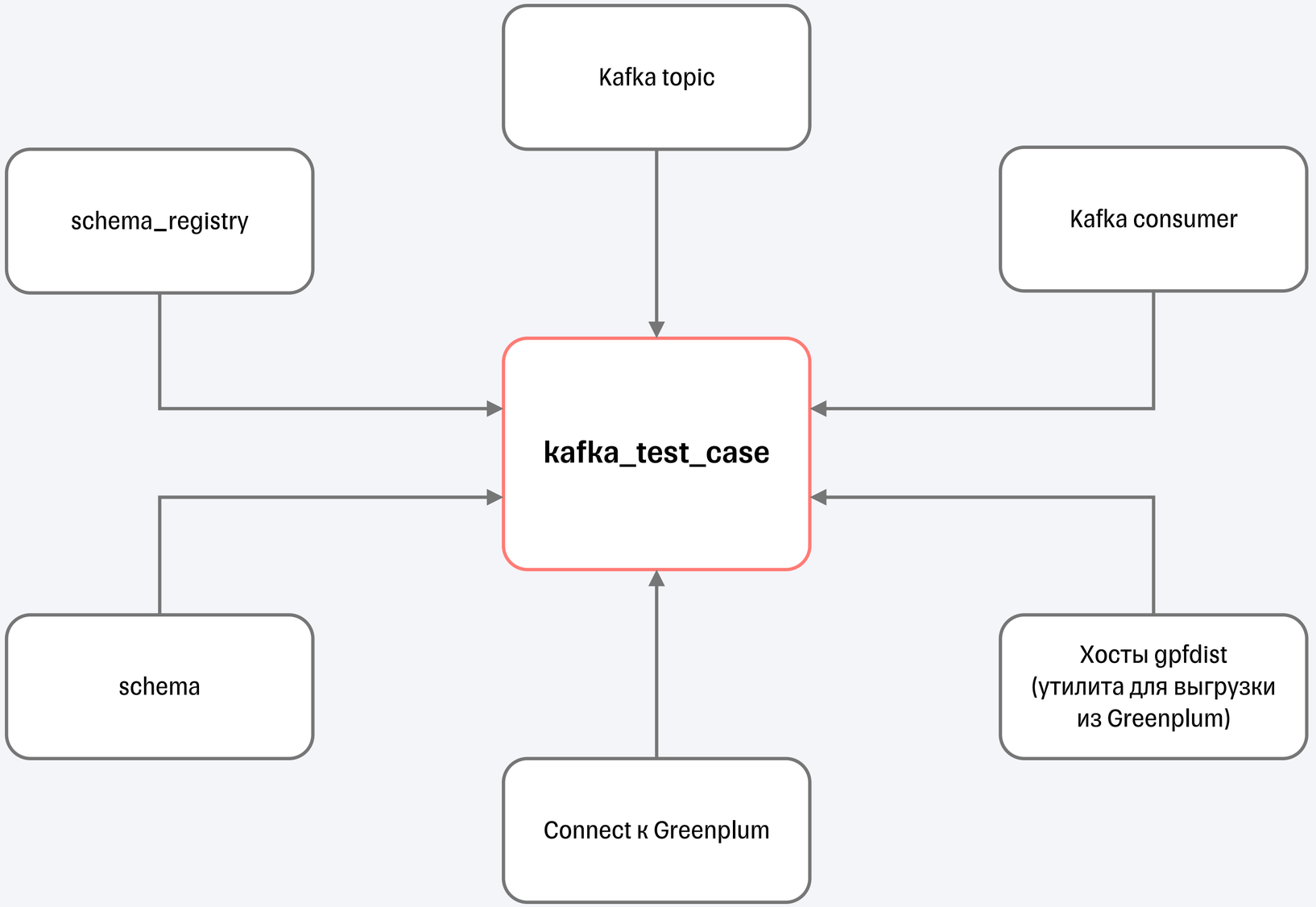
docker-compose.kafka_loader.yaml

```yaml
version: '3.6'

services:
    greenplum:
        extends:
            file: docker/greenplum/docker-compose.greenplum.yml
            service: greenplum
    zookeeper:
        extends:
            file:  docker/zookeeper/docker-compose.zookeeper.yml
           service: zookeeper
    broker1:
        extends:
            file: docker/kafka/docker-compose.kafka.yml
            service: broker1
        depends_on:
            zookeeper:
        condition: service_healthy
    schemaregistry:
        extends:
            file: docker/schemaregistry/docker-compose.schemaregistry.yml
            service: schemaregistry
      hostname: schemaregistry
        depends_on:
            zookeeper:
                condition: service_healthy
```

docker-compose.kafka.yaml

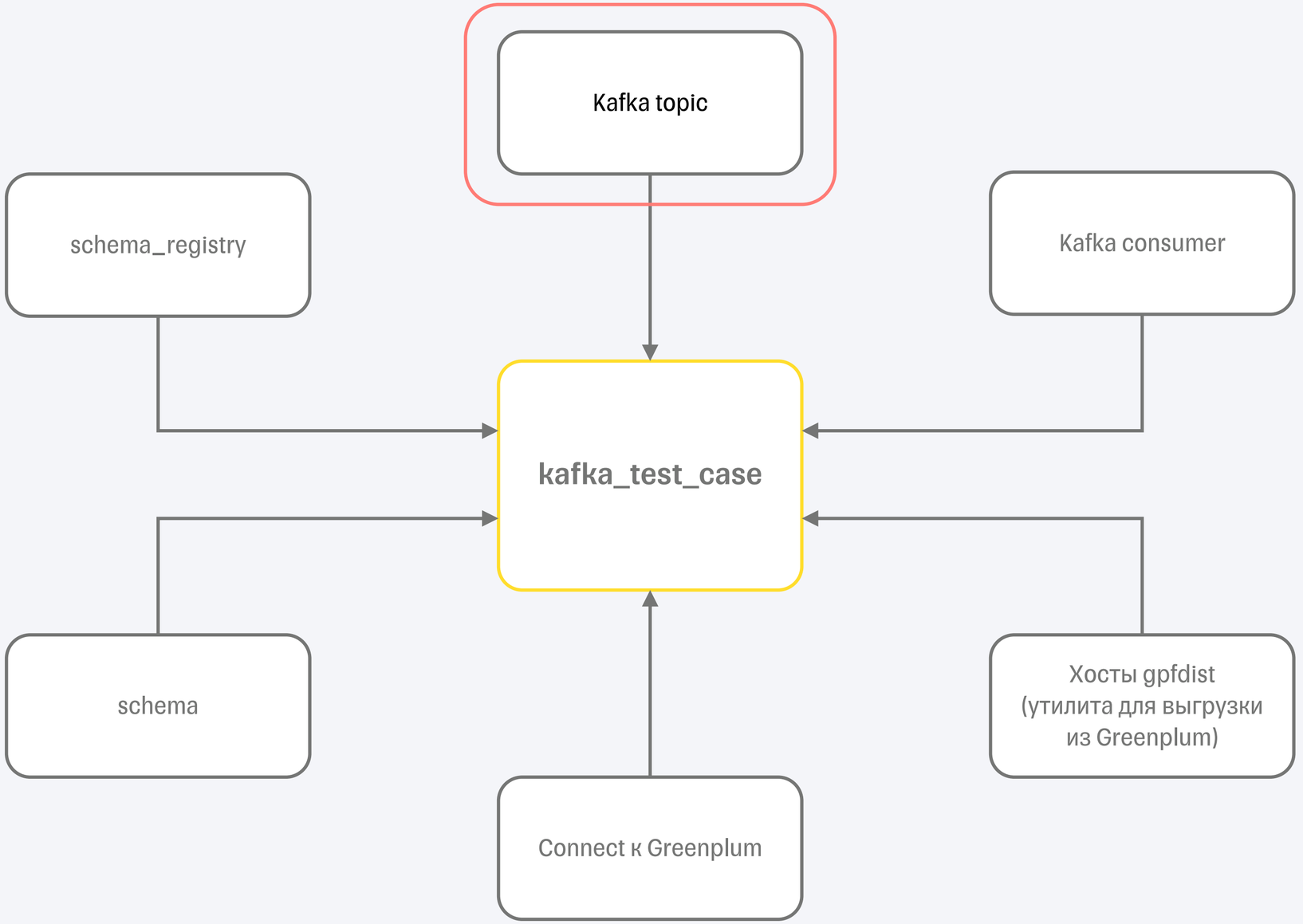
```yaml
version: '3.6'

services:
    broker1:
        image: <repository>/confluentinc/cp-kafka:5.5.3
        hostname: broker1
        container_name: broker1
        ports:
            - "9092:9092"
            - "29092:29092"
        restart: unless-stopped
        volumes:
            - './init-kafka-broker.sh:/etc/confluent/docker/init-kafka-broker.sh:ro'
            - './kafka_server_jaas.conf:/kafka_server_jaas.conf:ro'
        command: /bin/bash /etc/confluent/docker/init-kafka-broker.sh
        env_file: ../../.env
        healthcheck:
            test: "CMD-SHELL", "kafka-broker-api-versions --bootstrap-server=broker1:29092"]
            interval: 30s
            timeout: 10s
            retries: 5
        environment:
            - KAFKA_LISTENERS=PLAINTEXT://broker1:29092,SASL_PLAINTEXT://:9092
            - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://broker1:29092,SASL_PLAINTEXT://broker1:9092
            - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
            - ZOOKEEPER_SASL_ENABLED=false
```

Наборы фикстур



И вообще-то еще некоторый набор дополнительных фикстур, специфичных для загрузчиков DWH ...

Наборы фикстур



SCOPE = ?

Выбор scope для фикстуры топика

Неожиданные, но важные для нас нюансы

function

Время работы каждого теста непредсказуемо растягивалось

- Требуется время на создание топика для каждого нового теста
- Плюс время на подписку консюмером на топик
- Какой бы таймаут не устанавливали на чтение сообщения, не всегда находилось, что читать из-за задержек создания и подключения

Выбор scope для фикстуры топика

Неожиданные, но важные для нас нюансы

function

Время работы каждого теста непредсказуемо растягивалось

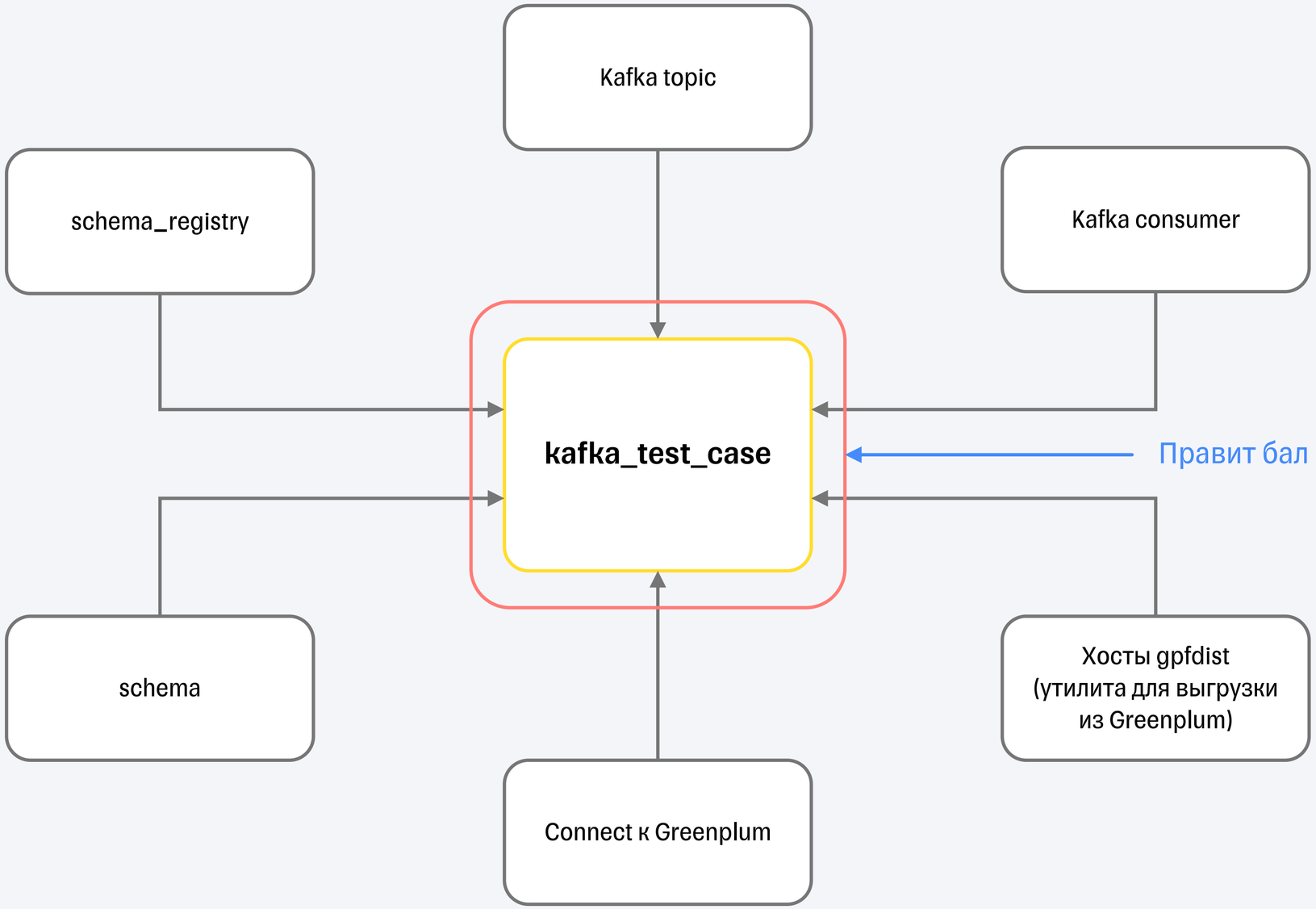
- Требуется время на создание топика для каждого нового теста
- Плюс время на подписку консюмером на топик
- Какой бы таймаут не устанавливали на чтение сообщения, не всегда находилось, что читать из-за задержек создания и подключения

session

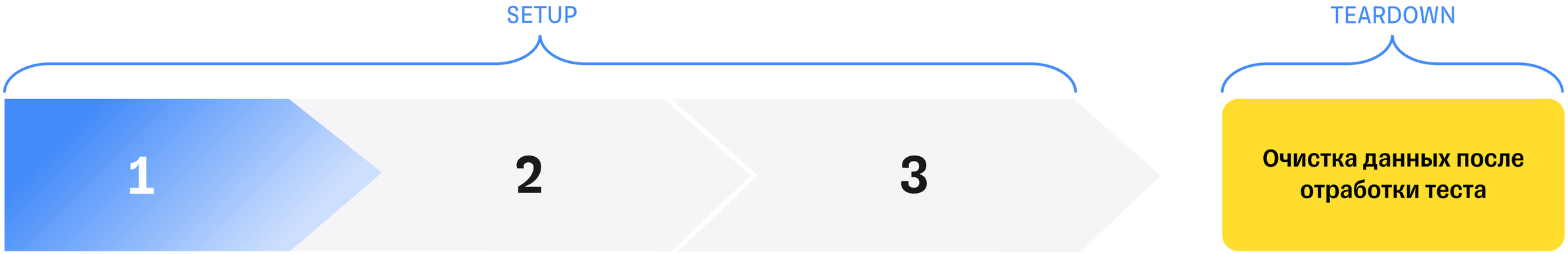
При параллельном запуске на каждую сессию сразу создается топик и подписывается группа консюмеров

- Не тратим время на регистрацию и подписку
- Помечаем тест с помощью его id в заголовках топика
- При проверочном чтении однозначно по id теста вычитываем только свои сообщения, без задержек

Наборы фикстур



Фикстура тест-кейса



Prepare инстанс

Специальный подготовительный класс в котором подготавливается окружение и реализованы проверки

Параметры запуска

Подготовка всех необходимых параметров загрузчика для конкретного тест-кейса

Формирование AVRO схемы

Формирование тест-кейса

ddl.yaml

dml.yaml → Prepare инстанс → Фикстура тест-кейса

test_cases.yaml

Тестовые данные

- Все в YAML

ddl.yaml

dml.yaml

test_cases.yaml

Тестовые данные

- Все в YAML
- На каждую тестируемую систему 3 файла

ddl.yaml

dml.yaml

test_cases.yaml

Тестовые данные

- Все в YAML
- На каждую тестируемую систему 3 файла
- В случае с kafka еще понадобилась папка scheme с avro-схемами

ddl.yaml

dml.yaml

test_cases.yaml

ddl.yaml

Содержит **описания структуры** следующих объектов:

- Источник
- Фактический результат
- Ожидаемый результат

```
"src_base": &src_base
    description: описание источника
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

ddl.yaml

Описание, для чего нужен этот объект

```yaml
"src_base": &src_base
    description: описание источника
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

ddl.yaml

Имя схемы, где будет создан объект

```yaml
"src_base": &src_base
    description: описание источника
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

ddl.yaml

Имя таблицы

```yaml
"src_base": &src_base
    description: описание источника
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

ddl.yaml

Названия столбцов
с указанием их типов

```
"src_base": &src_base
    description: описание источника
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

ddl.yaml

Ключ распределения таблицы

```yaml
"src_base": &src_base
    description: описание источника
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

ddl.yaml

```yaml
"src_base": &src_base
    description: описание источника
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

Флаг, является ли объект вьюхой

Тестовые данные

- Все в YAML
- На каждую тестируемую систему 3 файла
- В случае с kafka еще понадобилась папка schemes с avro-схемами

ddl.yaml

dml.yaml

test_cases.yaml

dml.yaml: источник

Содержит **SQL скрипты наполнения** следующих объектов:

- Источник

```
"src->long_row": >
  INSERT INTO {table} VALUES
    ('TST_1' , repeat('a', 40000),  1, '2017-01-01 00:00:00.000 +0300',
array['1', '2', '3', '4' ,'5',' '], '2017-01-01');
```

dml.yaml: ожидаемый результат

```
"tgt_er->json_long":
    key_fields_count: 0
    titles: ['key_1', 'fld_str', 'fld_int', 'fld_dttm', 'fld_str_arr', 'fld_dt’]
    messages:
        - ['TST_1', '{str_32768}', 1, '2017-01-01 00:00:00', ['1', '2', '3', '4'
,'5',' '], '2017-01-01']
```

dml.yaml: ожидаемый результат

Название объекта

```
"tgt_er->json_long":
    key_fields_count: 0
    titles: ['key_1', 'fld_str', 'fld_int', 'fld_dttm', 'fld_str_arr', 'fld_dt']
    messages:
        - ['TST_1', '{str_32768}', 1, '2017-01-01 00:00:00', ['1', '2', '3', '4'
,'5',' '], '2017-01-01']
```

dml.yaml: ожидаемый результат

Количество ключей

```
"tgt_er->json_long":
    key_fields_count: 0
    titles: ['key_1', 'fld_str', 'fld_int', 'fld_dttm', 'fld_str_arr', 'fld_dt']
    messages:
        - ['TST_1', '{str_32768}', 1, '2017-01-01 00:00:00', ['1', '2', '3', '4'
,'5',' '], '2017-01-01']
```

dml.yaml: ожидаемый результат

Заголовки
для формирования JSON

```
"tgt_er->json_long":
    key_fields_count: 0
    titles: ['key_1', 'fld_str', 'fld_int', 'fld_dttm', 'fld_str_arr', 'fld_dt']
    messages:
        - ['TST_1', '{str_32768}', 1, '2017-01-01 00:00:00', ['1', '2', '3', '4'
,'5',' '], '2017-01-01']
```

dml.yaml: ожидаемый результат

Список сообщений, которые мы ожидаем прочитать из топика

```
"tgt_er->json_long":
    key_fields_count: 0
    titles: ['key_1', 'fld_str', 'fld_int', 'fld_dttm', 'fld_str_arr', 'fld_dt']
    messages:
        - ['TST_1', '{str_32768}', 1, '2017-01-01 00:00:00', ['1', '2', '3', '4'
,'5',' '], '2017-01-01']
```

Тестовые данные

- Все в YAML
- На каждую тестируемую систему 3 файла
- В случае с kafka еще понадобилась папка scheme с avro-схемами

ddl.yaml

dml.yaml

test_cases.yaml

test.yaml: тест-кейсы

Содержит описания тест-кейсов, включая:

- DDL объектов

test.yaml: тест-кейсы

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов

test.yaml: тест-кейсы

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов
- Параметры запуска

test.yaml: тест-кейсы

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов
- Параметры запуска
- Исключение и текст ошибки

test.yaml: тест-кейсы

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов
- Параметры запуска
- Исключение и текст ошибки
- Описание feature и story

test.yaml: тест-кейсы

Ссылки на DML объекты

```yaml
"avro + src_is_empty":
    description: "src пустой + грузим в AVRO"
    dml_in: [  ]
    dml_out_er: tgt_er->src_is_empty
    mark: [ external ]
    avro_params:
        schema_version: "1"
        <<: *avro_params
    allure_feature: "Kafka"
    allure_story: ["avro"]
    <<: *smoke_avro_test
```

test.yaml: тест-кейсы

Флаг использования теста в TEDI

```
"avro + src_is_empty":
    description: "src пустой + грузим в AVRO"
    dml_in: [  ]
    dml_out_er: tgt_er->src_is_empty
    mark: [ external ]
    avro_params:
        schema_version: "1"
        <<: *avro_params
    allure_feature: "Kafka"
    allure_story: ["avro"]
    <<: *smoke_avro_test
```

test.yaml: тест-кейсы

Параметры AVRO схемы

```
"avro + src_is_empty":
    description: "src пустой + грузим в AVRO"
    dml_in: [  ]
    dml_out_er: tgt_er->src_is_empty
    mark: [ external ]
    avro_params:
      schema_version: "1"
        <<: *avro_params
    allure_feature: "Kafka"
    allure_story: ["avro"]
    <<: *smoke_avro_test
```

test.yaml: тест-кейсы

Параметры для Allure

```
"avro + src_is_empty":
    description: "src пустой + грузим в AVRO"
    dml_in: [  ]
    dml_out_er: tgt_er->src_is_empty
    mark: [ external ]
    avro_params:
        schema_version: "1"
        <<: *avro_params
    allure_feature: "Kafka"
    allure_story: ["avro"]
    <<: *smoke_avro_test
```

test.yaml: тест-кейсы

Ссылки на другие объекты

```yaml
"avro + src_is_empty":
    description: "src пустой + грузим в AVRO"
    dml_in: [  ]
    dml_out_er: tgt_er->src_is_empty
    mark: [ external ]
    avro_params:
        schema_version: "1"
        <<: *avro_params
    allure_feature: "Kafka"
    allure_story: ["avro"]
    <<: *smoke_avro_test
```

test.yaml: тест-кейсы

```
"json + src_is_empty":
    description: "src пустой + грузим в JSON"
    dml_in: [ ]
    dml_out_er: tgt_er->src_is_empty
    mark: [ external ]
    allure_feature: "Kafka“
    allure_story: ["json"]
    <<: *smoke_json_test
```

```
"avro + src_is_empty":
    description: "src пустой + грузим в AVRO"
    dml_in: [ ]
    dml_out_er: tgt_er->src_is_empty
    mark: [ external ]
    avro_params:
        schema_version: "1"
        <<: *avro_params
    allure_feature: "Kafka"
    allure_story: ["avro"]
    <<: *smoke_avro_test
```

AVRO схемы

- **schema_info.yaml** – описание всех используемых AVRO схем

schema_info.yaml

schema_1.json

schema_N.json

AVRO схемы

- **schema_info.yaml** – описание всех используемых AVRO схем
- **schema_1.json** – отдельная AVRO схема

schema_info.yaml

schema_1.json

schema_N.json

avro схемы: schema_info.yaml

Ключ схемы

```
schemes:
    all_types:
        schema_name: 'topic_all_types-value-{}',
        description_versions:
            1:
                file:       'all_types_schema.json'
                description: "В источнике все типы полей“
            2:
                file:       'all_types_schema_latest.json'
                description: "Для теста latest версии схемы по умолчанию"
```

avro схемы: schema_info.yaml

Имя схемы
в Schema Registry

```
schemes:
    all_types:
        schema_name: 'topic_all_types-value-{}',
        description_versions:
            1:
                file:       'all_types_schema.json'
                description: "В источнике все типы полей“
            2:
                file:       'all_types_schema_latest.json'
                description: "Для теста latest версии схемы по умолчанию"
```

avro схемы: schema_info.yaml

Номера версий данной схемы

```
schemes:
    all_types:
        schema_name: 'topic_all_types-value-{}’,
        description_versions:
            1:
                file:       'all_types_schema.json’
                description: "В источнике все типы полей“
            2:
                file:       'all_types_schema_latest.json’
                description: "Для теста latest версии схемы по умолчанию"
```

avro схемы: schema_info.yaml

Имена файлов со схемами

```
schemes:
    all_types:
        schema_name: 'topic_all_types-value-{}',
        description_versions:
            1:
                file:       'all_types_schema.json'
                description: "В источнике все типы полей“
            2:
                file:       'all_types_schema_latest.json'
                description: "Для теста latest версии схемы по умолчанию"
```

AVRO схемы

- **schema_info.yaml –** описание всех используемых AVRO схем
- **schema_1.json –** отдельная AVRO схема

schema_info.yaml

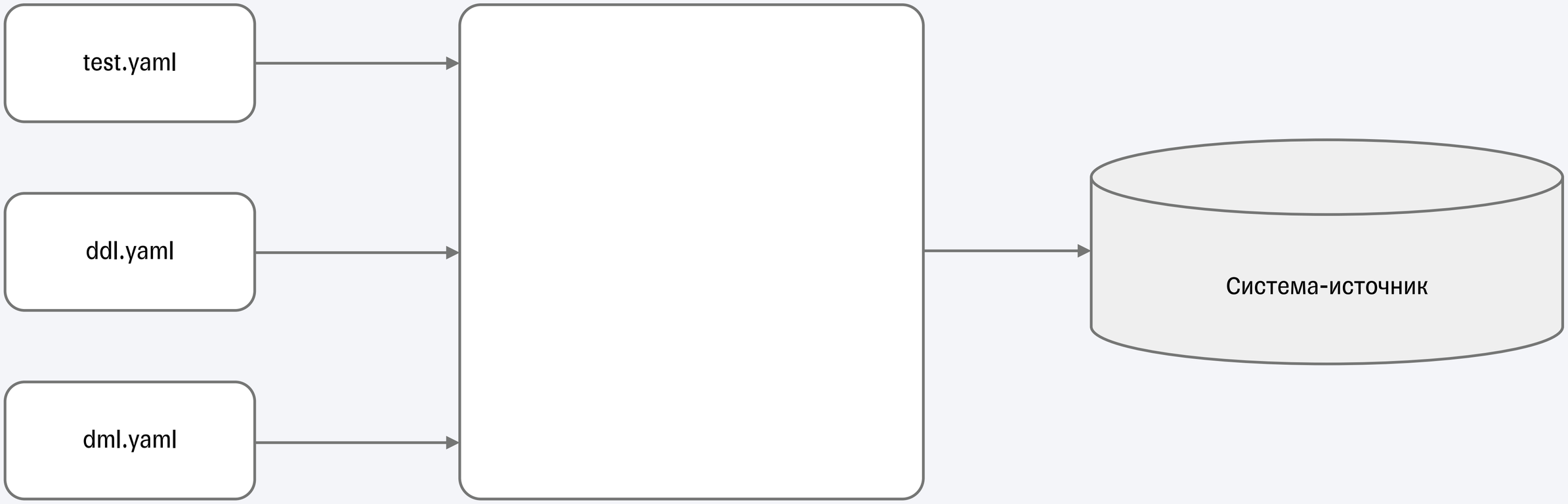
schema_1...N.json

AVRO схемы

- В файле **schema_info.yaml** задано описание всех используемых AVRO схем со ссылками на файлы JSON
- Написан обработчик этого файла, который для каждого отдельного кейса использует только нужную AVRO схему
- В каждом отдельном JSON файле задано полное описание AVRO схемы

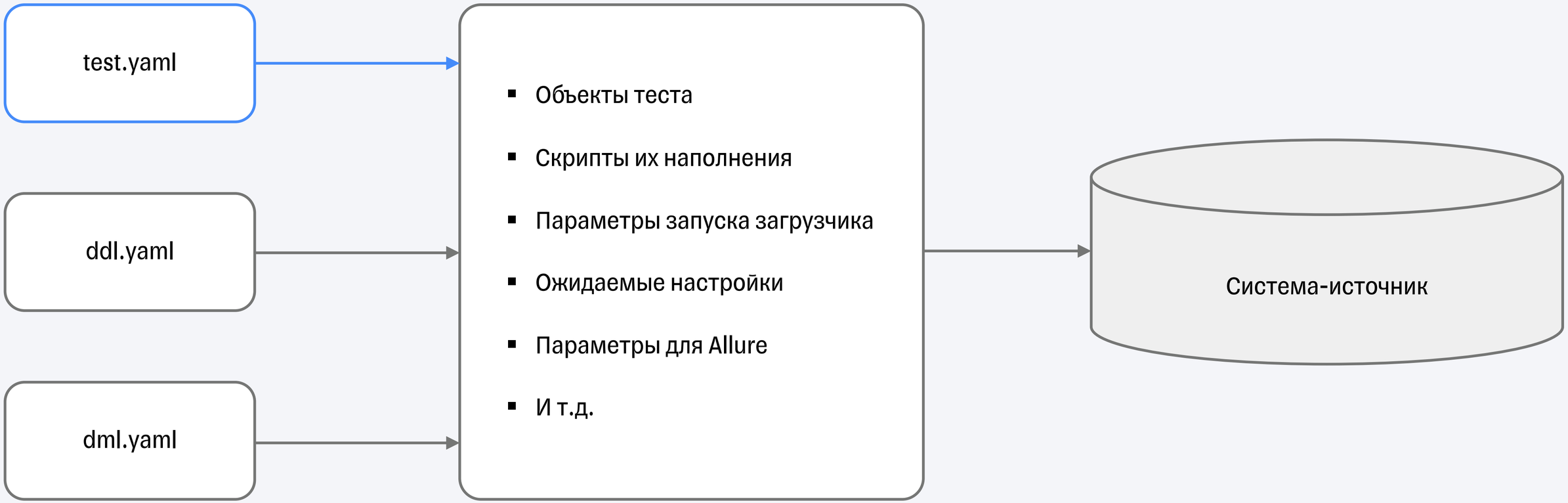
```
{
    "type": "record",
    "name": "TestKafkaLoaderBase",
    "namespace": “project.example",
    "fields": [
        {
            "name":"key_1",
            "type": ["null","string"] ,
            "default":null
        },
        {
            "name":"fld_str",
            "type": ["null","string"] ,
            "default":null
        },
        {
            "name":"fld_int",
            "type":"int",
            "default":null
        },
        {
            "name":"fld_dttm",
            "type":"string",
            "logicalType":"date"
        }
    ]
}
```

Prepare класс



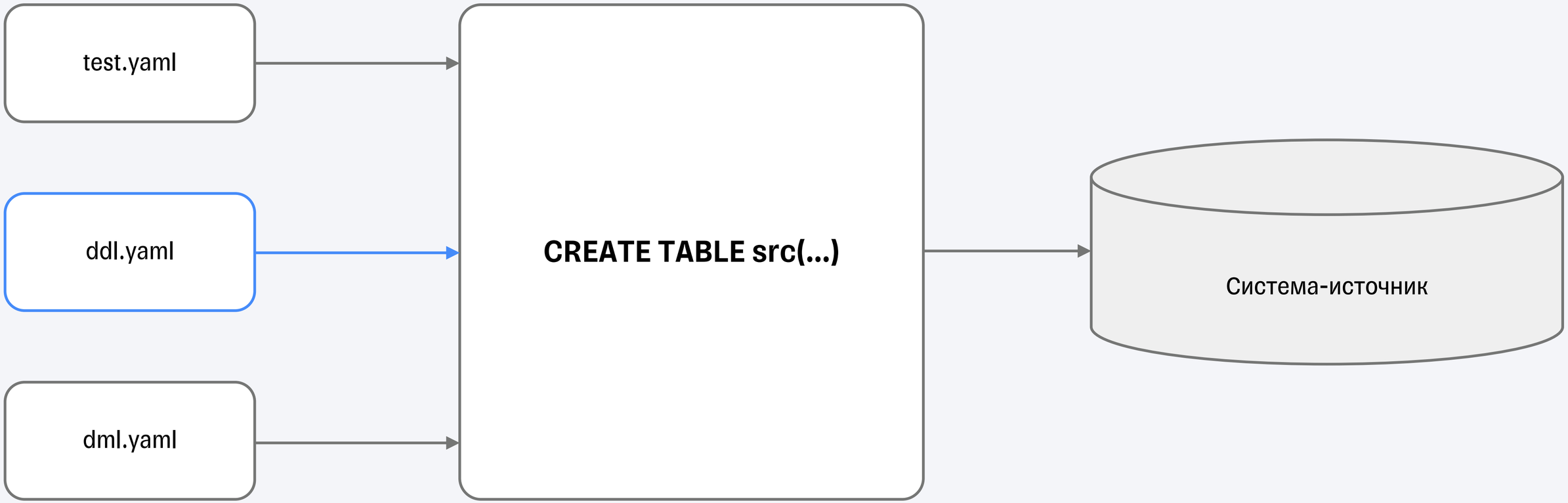
Prepare класс

Сбор информации о тесте



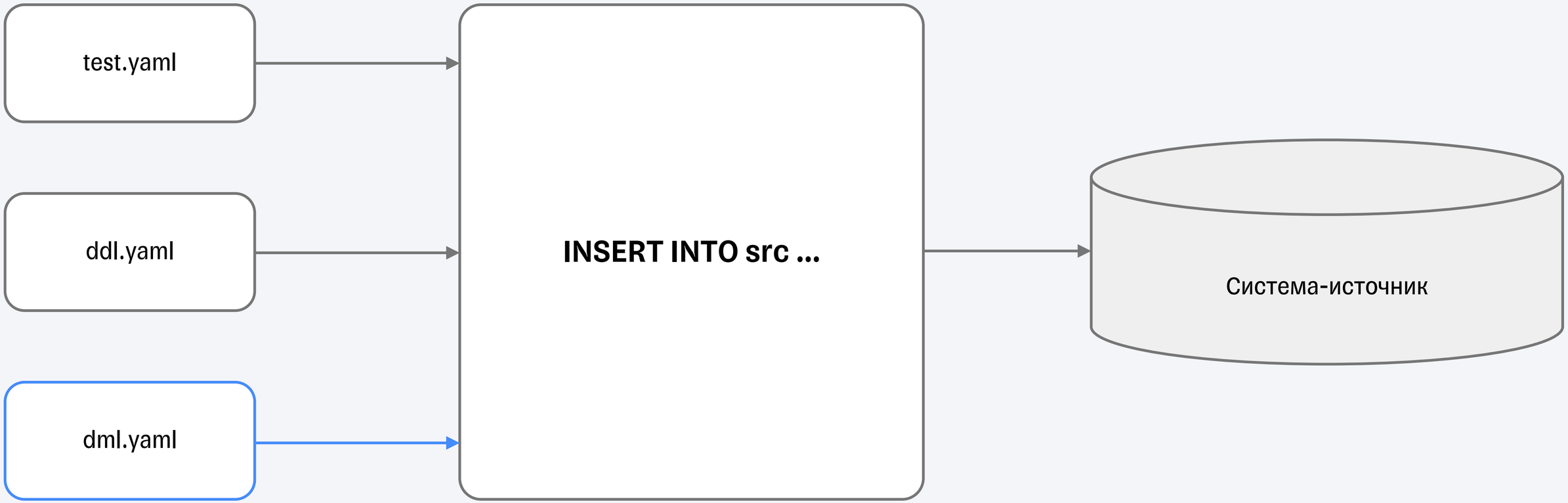
Prepare класс

Создание таблиц



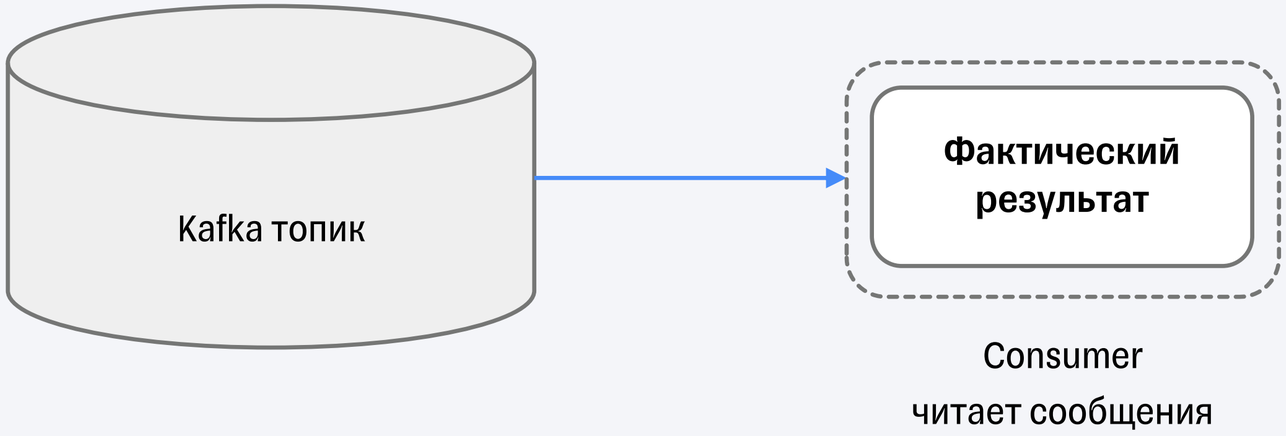
Prepare класс

Заполнение таблиц

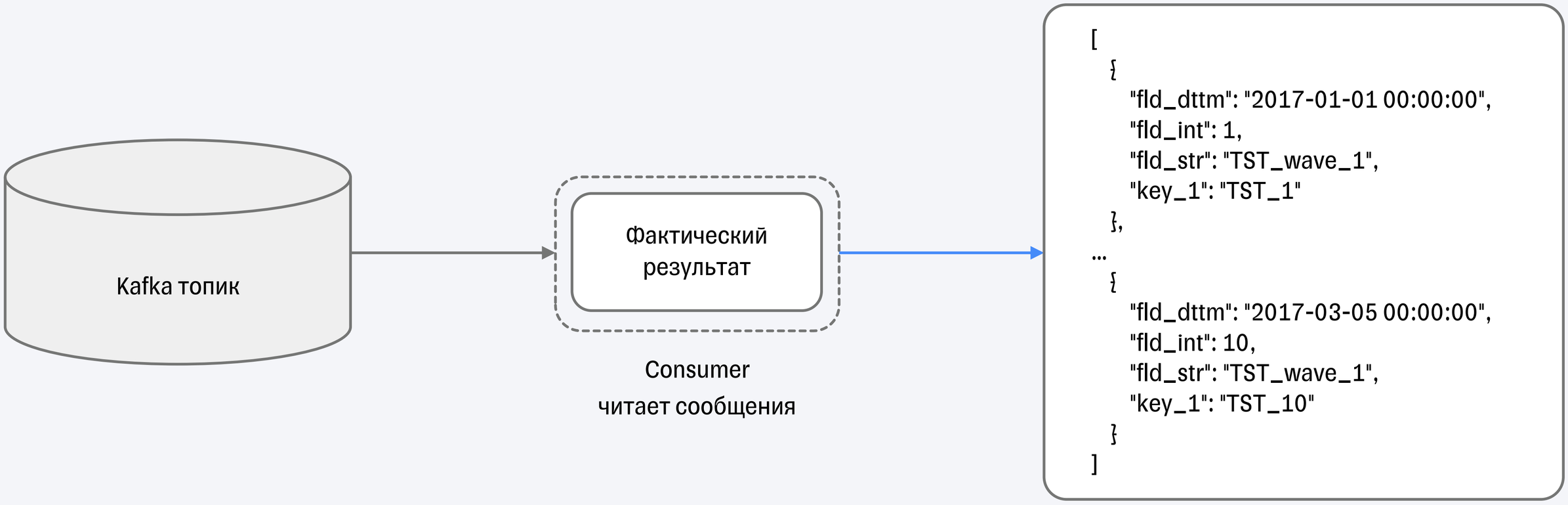


Prepare класс

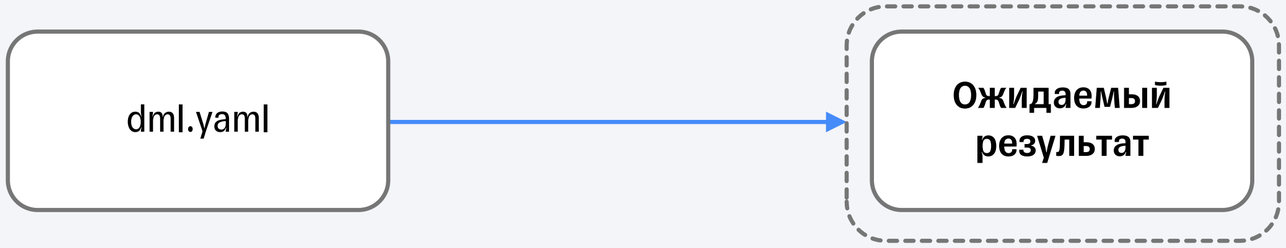
Получение фактического результата



Получение фактического результата



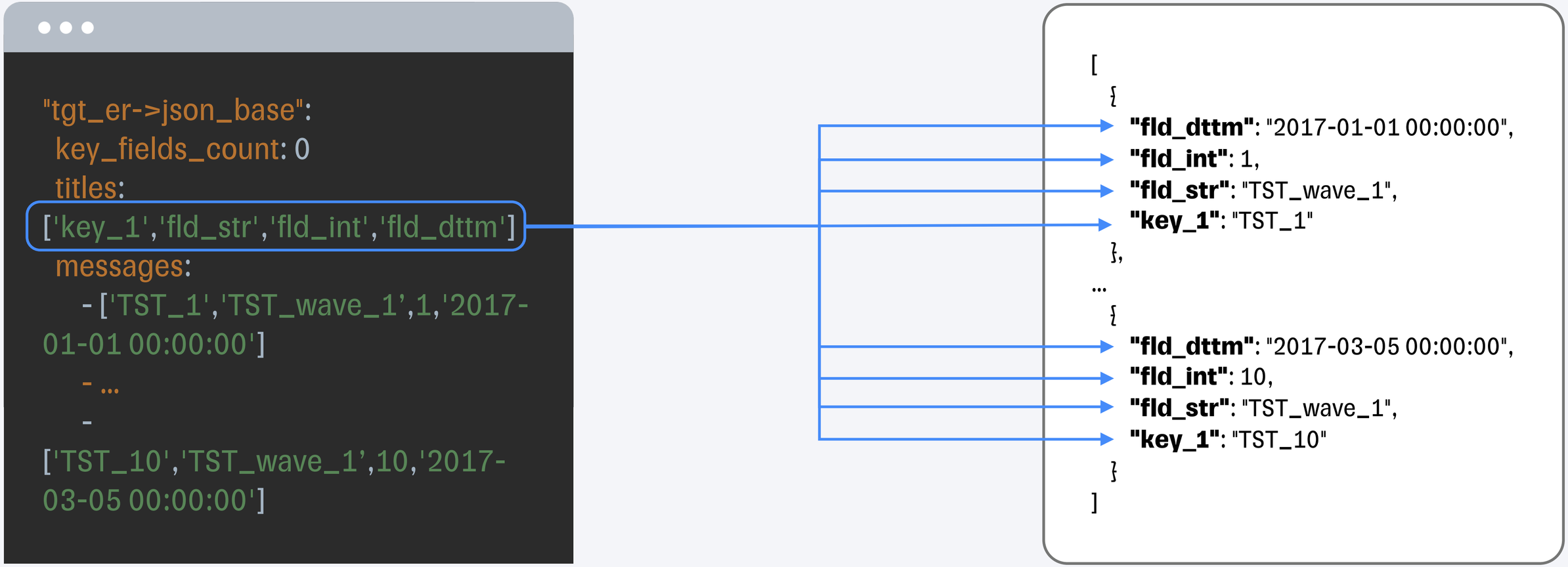
Генерация ожидаемого результата



Генерируется объект
на основе YAML

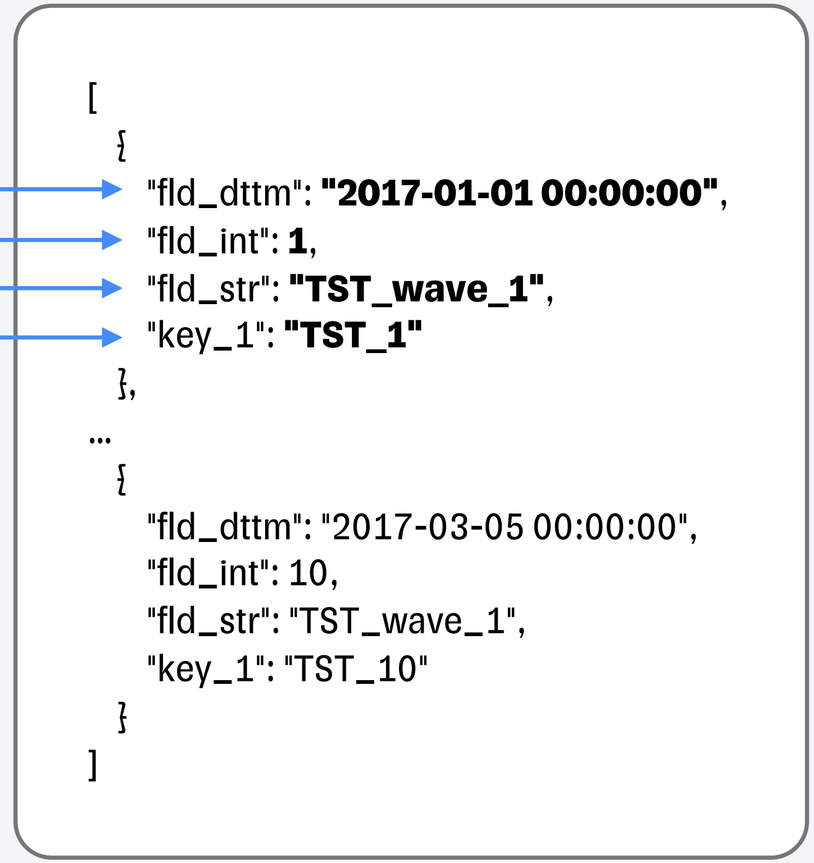
```
"tgt_er->json_base":
 key_fields_count: 0
 titles:
['key_1','fld_str','fld_int','fld_dttm']
 messages:
    - ['TST_1','TST_wave_1',1,'2017-
01-01 00:00:00']
    - ...
    -
['TST_10','TST_wave_1',10,'2017-
03-05 00:00:00']
```

Генерация ожидаемого результата



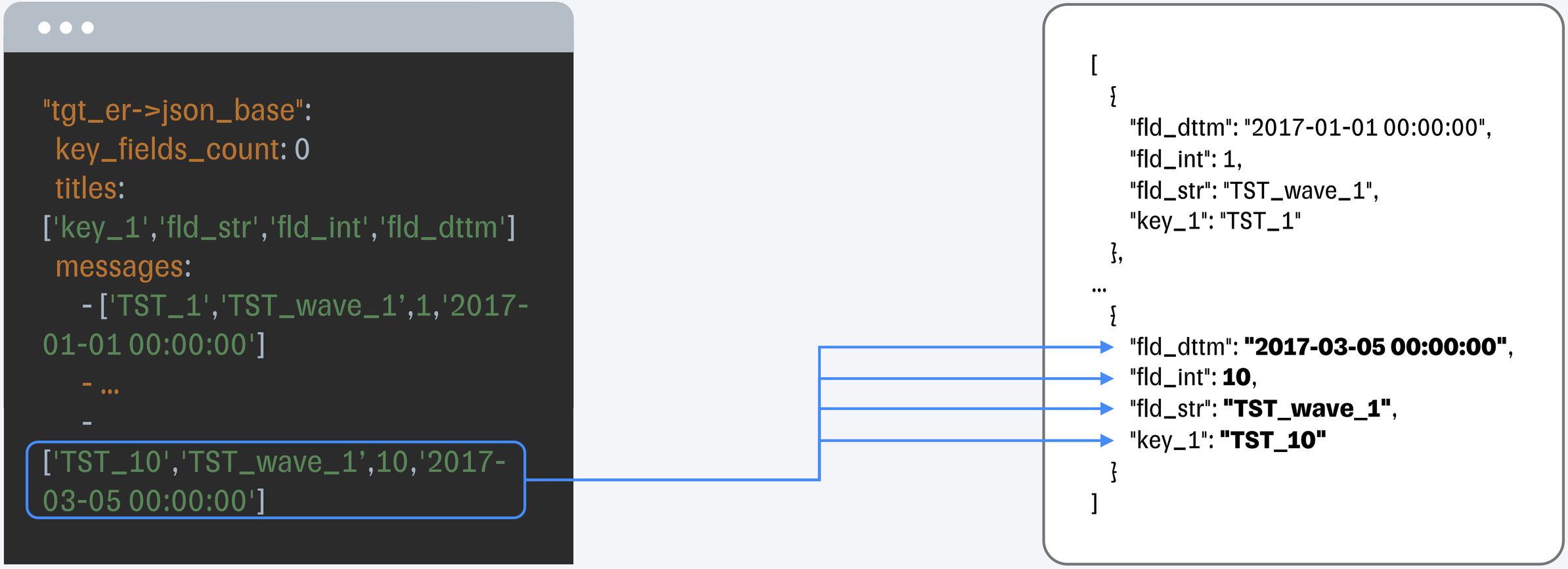
Генерация ожидаемого результата

```
"tgt_er->json_base":
 key_fields_count: 0
 titles:
['key_1','fld_str','fld_int','fld_dttm']
 messages:
    - ['TST_1','TST_wave_1’,1,'2017-
01-01 00:00:00']
    - ...
    -
['TST_10','TST_wave_1’,10,'2017-
03-05 00:00:00']
```

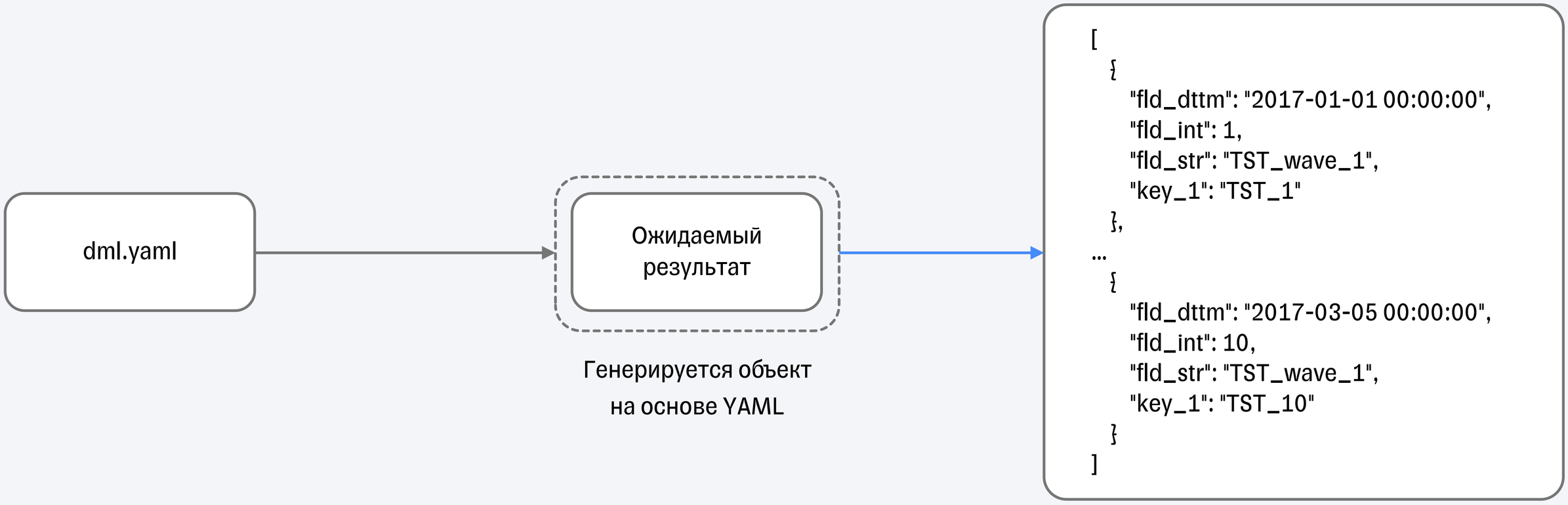


Генерация ожидаемого результата

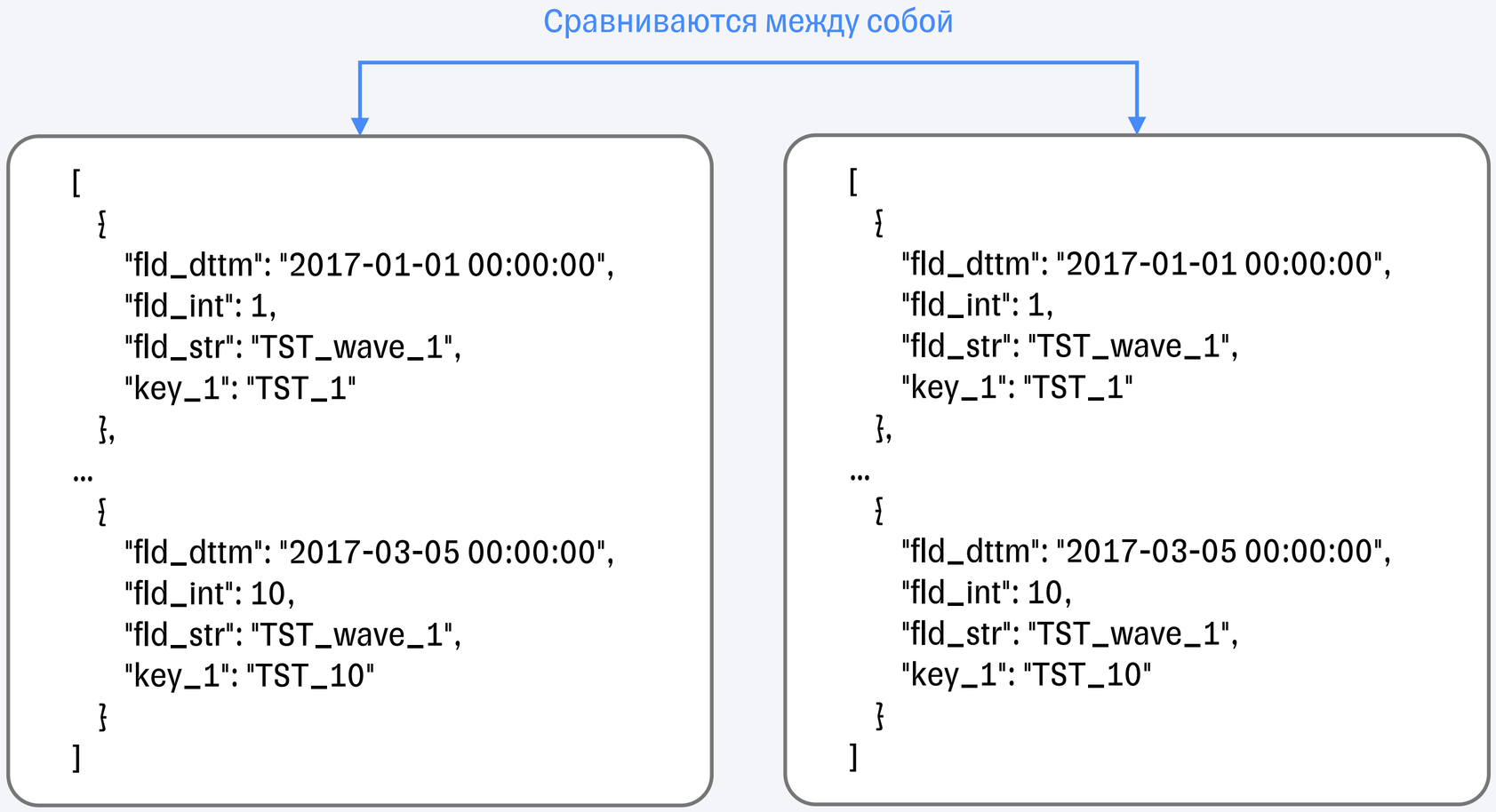
```
"tgt_er->json_base":
 key_fields_count: 0
 titles:
['key_1','fld_str','fld_int','fld_dttm']
 messages:
    - ['TST_1','TST_wave_1',1,'2017-
01-01 00:00:00']
    - ...
    -
['TST_10','TST_wave_1',10,'2017-
03-05 00:00:00']
```



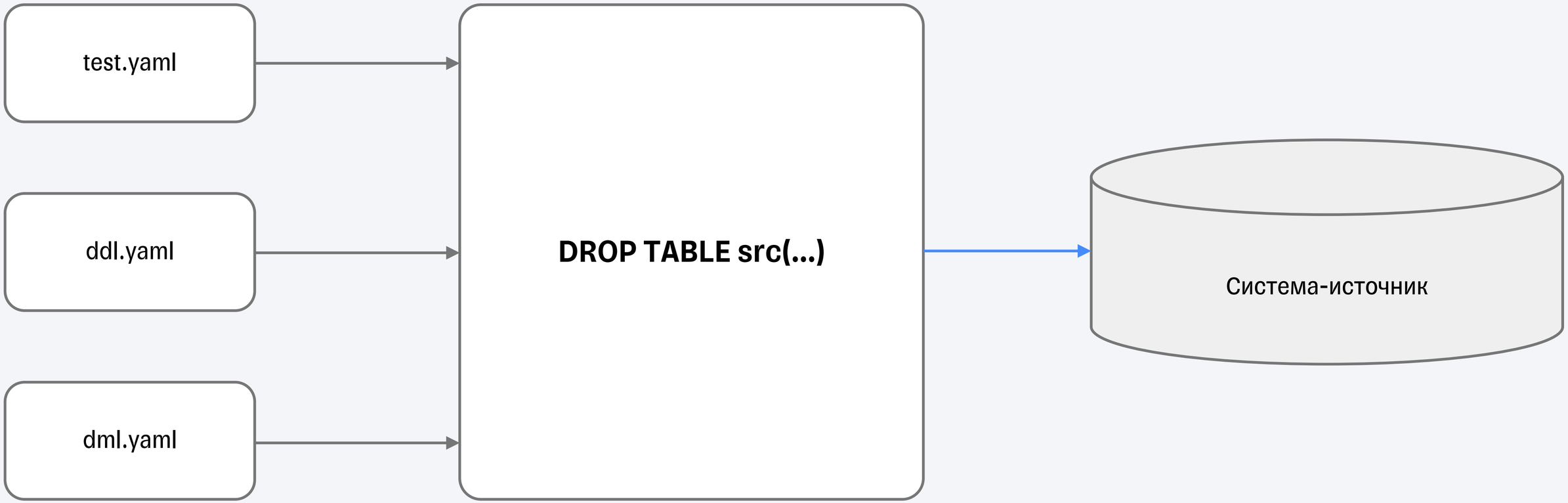
Генерация ожидаемого результата



Проверка результата



Очистка окружения

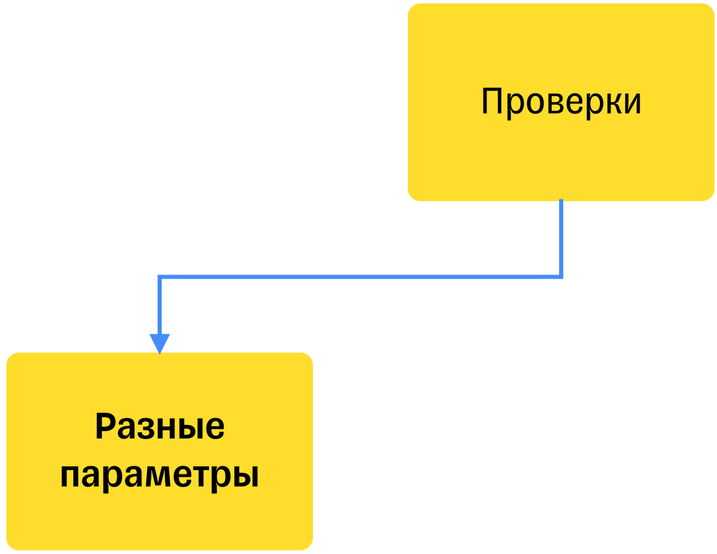


Prepare класс

Как готовились кейсы?

- Надо было перенести все текущие проверки из тестового SAS джоба «как есть»
- Мы сформировали MindMap с описанием необходимых тестов

Как готовились кейсы?



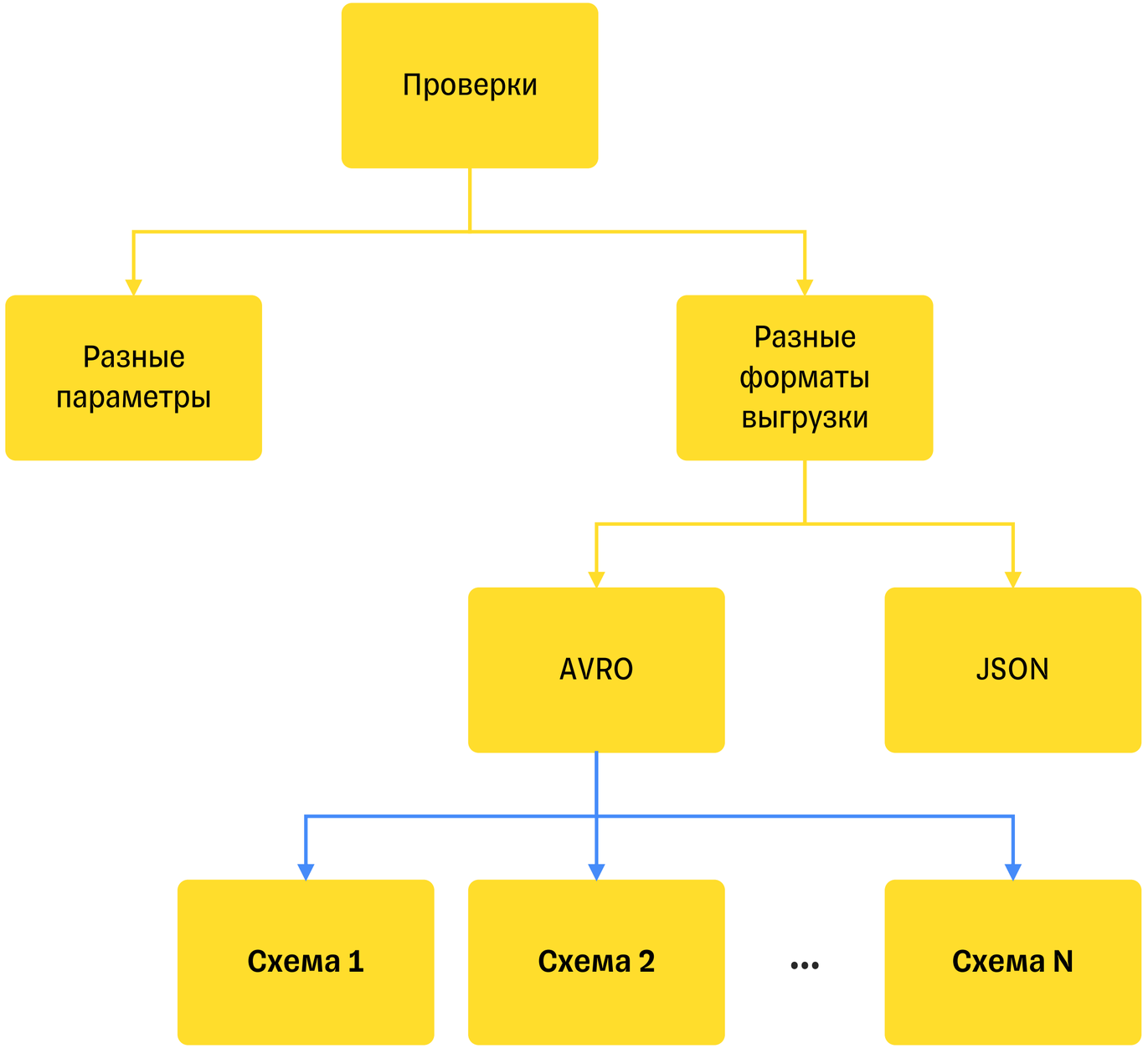
- Надо было перенести все текущие проверки из тестового SAS джоба «как есть»
- Мы сформировали MindMap с описанием необходимых тестов

Как готовились кейсы?

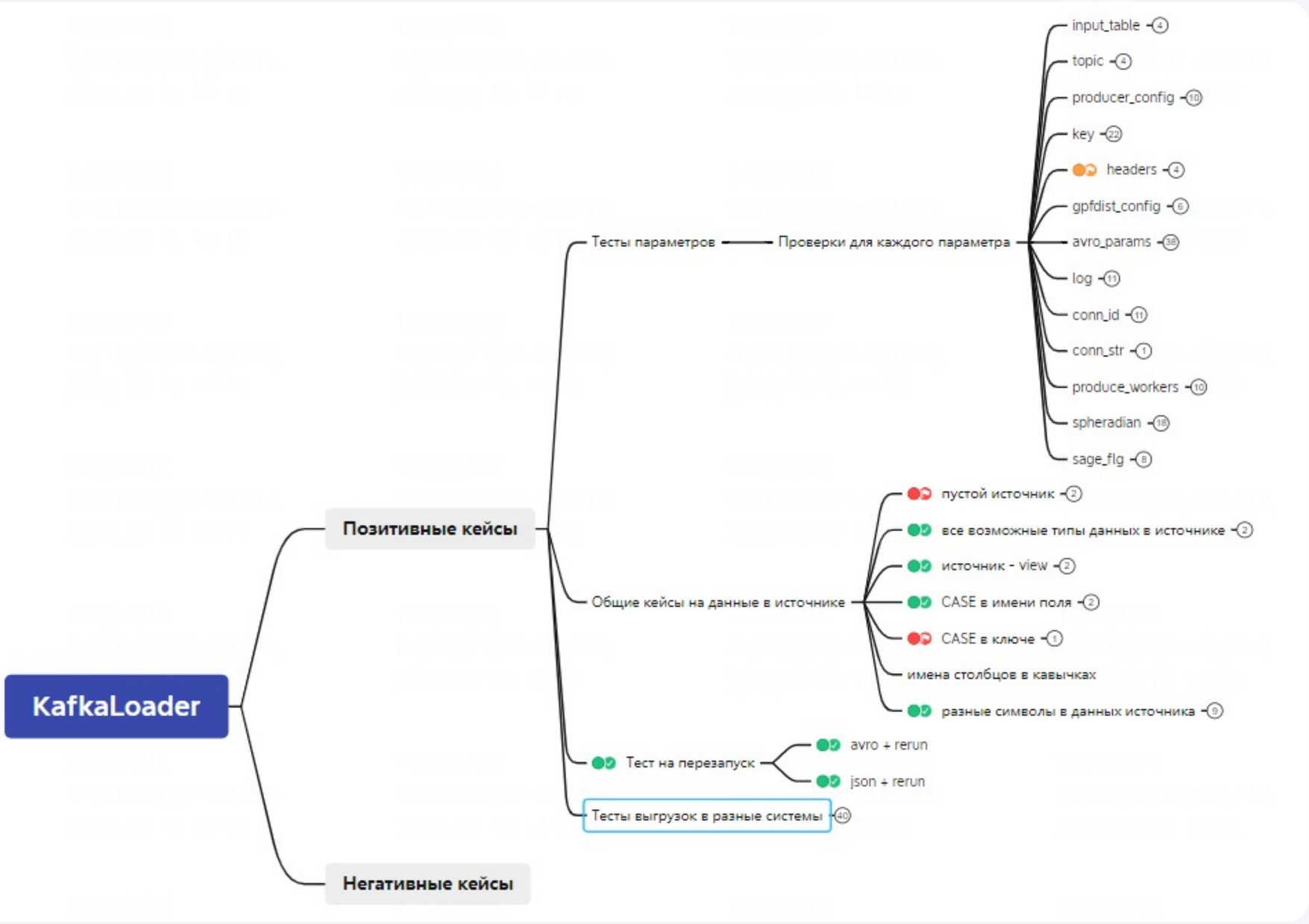
- Надо было перенести все текущие проверки из тестового SAS джоба «как есть»
- Мы сформировали MindMap с описанием необходимых тестов

Как готовились кейсы?

- Надо было перенести все текущие проверки из тестового SAS джоба «как есть»
- Мы сформировали MindMap с описанием необходимых тестов



Набор кейсов



Итого

До

Тестирование

- Сложно добавлять тесты
- Тесты пишут только QA
- Нет TDD
- Только функциональные тесты

После

Итого

До

Тестирование

- Сложно добавлять тесты
- Тесты пишут только QA
- Нет TDD
- Только функциональные тесты

После

Тестирование

- Простое добавление тестов
- Тесты пишут и разработчики
- Есть TDD
- Функциональные + unit

Итого

До

Тестирование

- Сложно добавлять тесты
- Тесты пишут только QA
- Нет TDD
- Только функциональные тесты

CI

- Только Greenplum

После

Тестирование

- Простое добавление тестов
- Тесты пишут и разработчики
- Есть TDD
- Функциональные + unit

Итого

До

Тестирование

- Сложно добавлять тесты
- Тесты пишут только QA
- Нет TDD
- Только функциональные тесты

CI

- Только Greenplum

После

Тестирование

- Простое добавление тестов
- Тесты пишут и разработчики
- Есть TDD
- Функциональные + unit

CI

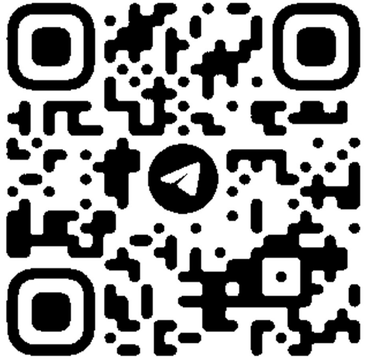
- Greenplum
- Kafka + Zookeeper + Schema Registry
- Oracle
- Clickhouse
- Postgres

Обещанные ссылки

Доклад М.Иванова
на SmartData

Файлы конфигурации Kafka
в docker

Карандасов
Евгений



Фролова
Екатерина

ТИНЬКОФФ

Спасибо!