лемана тех

Витрины данных на Data Lakehouse: маленькая история про большой переезд с Greenplum

Наумов Артемий
Инженер данных

Артемий Наумов

- Инженер данных в Лемана Тех
- Развиваю корпоративную платформы данных – от dwh к dlh

artemiy.naumov@lemanapro.ru

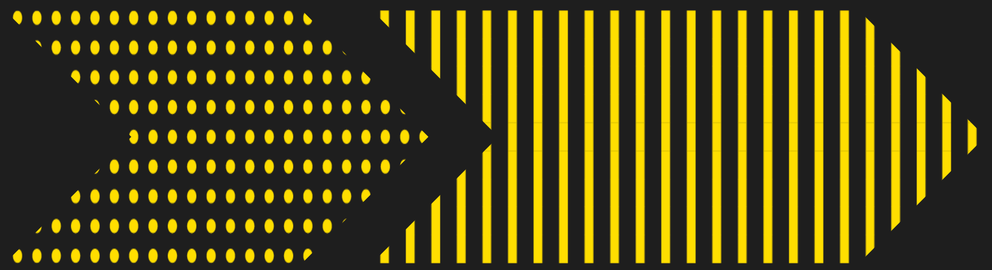
t.me/tema_nmv

План доклада

План доклада

Текущий и целевой подход к сборке витрин данных

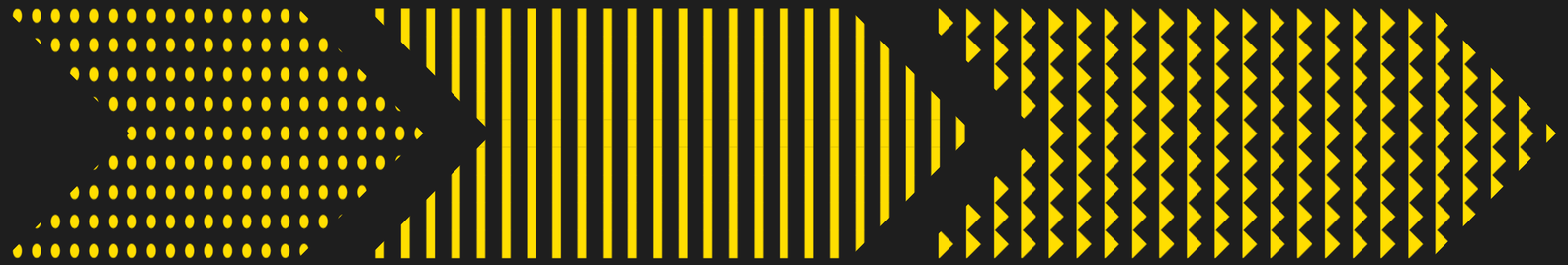
План доклада



Текущий и целевой подход к сборке витрин данных

Перенос сборки на DLH с минимальными потерями

План доклада

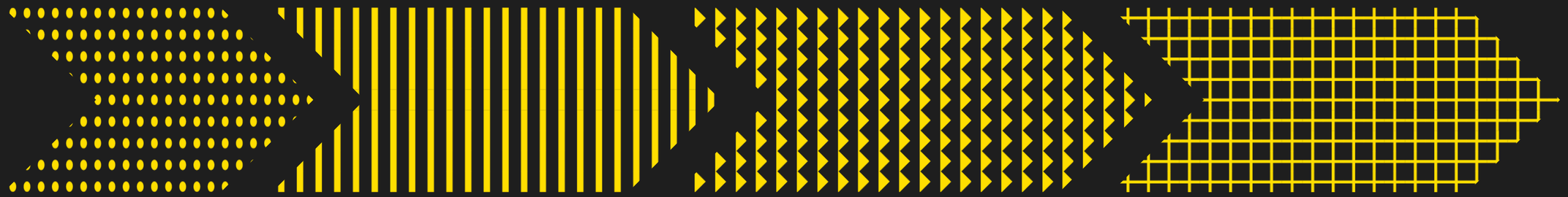


Текущий и целевой подход к сборке витрин данных

Перенос сборки на DLH с минимальными потерями

Дополнительные фичи для DLH

План доклада



Текущий и целевой подход к сборке витрин данных

Перенос сборки на DLH с минимальными потерями

Дополнительные фичи для DLH

Наш подход к организации миграции

План доклада

Текущий и целевой подход к сборке витрин данных

Перенос сборки на DLH с минимальными потерями

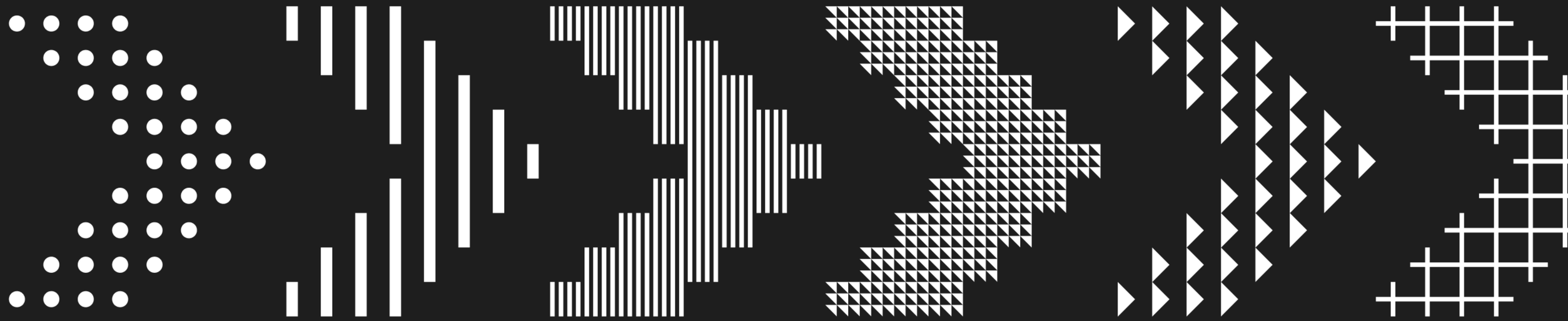
Дополнительные фичи для DLH

Наш подход к организации миграции

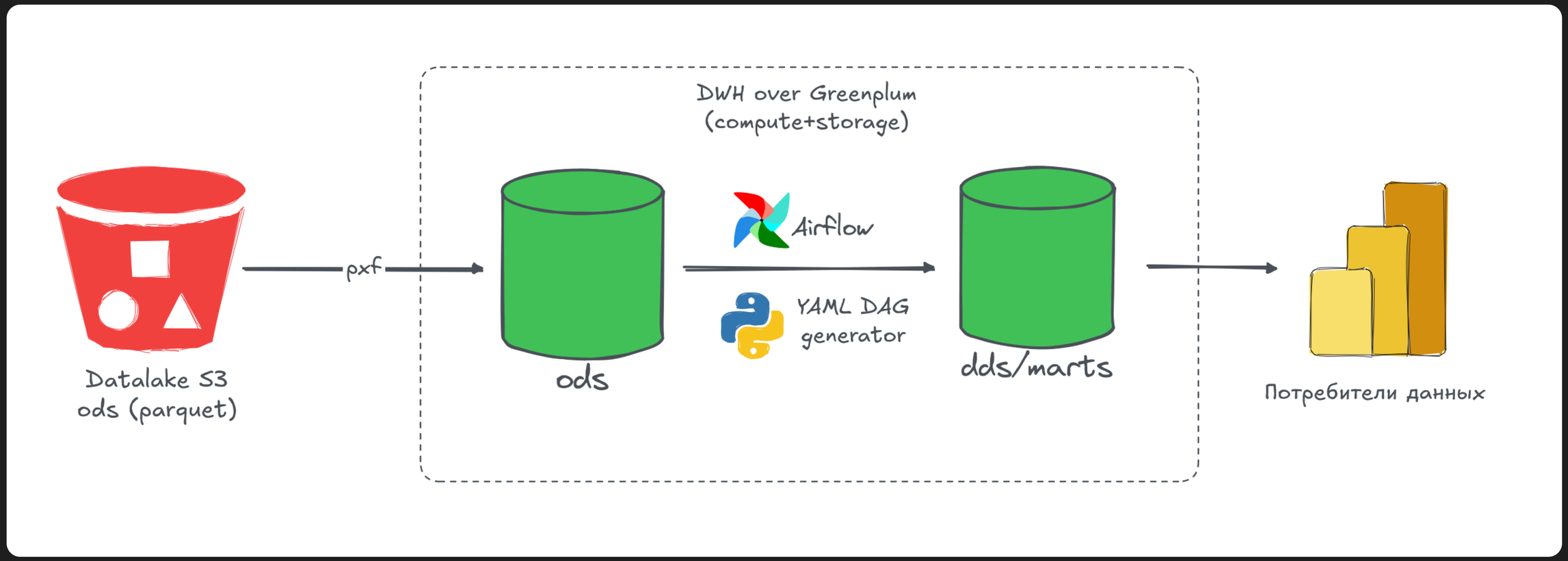
Итоги и выводы про DLH

лемана тех

Текущий и целевой подход к сборке витрин данных



Основа legacy стека – Greenplum



Преимущества и недостатки legacy подхода

Плюсы

Минусы

Преимущества и недостатки legacy подхода

Плюсы

+ OLAP архитектура, PG интерфейс для работы с данными

Минусы

Преимущества и недостатки legacy подхода

Плюсы

- \+ OLAP архитектура, PG интерфейс для работы с данными
- \+ Data Mesh – команды сами отвечают за свои данные

Минусы

Преимущества и недостатки legacy подхода

Плюсы

- \+ OLAP архитектура, PG интерфейс для работы с данными
- \+ Data Mesh – команды сами отвечают за свои данные
- \+ YAML engineering (для генерации дагов, DQ проверок и пр.)

Минусы

Преимущества и недостатки legacy подхода

Плюсы

- \+ OLAP архитектура, PG интерфейс для работы с данными
- \+ Data Mesh – команды сами отвечают за свои данные
- \+ YAML engineering (для генерации дагов, DQ проверок и пр.)

Минусы

- – Конкуренция за ресурсы, масштабирование и т.п.

Преимущества и недостатки legacy подхода

Плюсы

+ OLAP архитектура, PG интерфейс для работы с данными
+ Data Mesh – команды сами отвечают за свои данные
+ YAML engineering (для генерации дагов, DQ проверок и пр.)

Минусы

– Конкуренция за ресурсы, масштабирование и т.п.
– Сложно тестировать пользовательский код, тесты лежат в разных местах

Преимущества и недостатки legacy подхода

Плюсы

- \+ OLAP архитектура, PG интерфейс для работы с данными
- \+ Data Mesh – команды сами отвечают за свои данные
- \+ YAML engineering (для генерации дагов, DQ проверок и пр.)

Минусы

- – Конкуренция за ресурсы, масштабирование и т.п.
- – Сложно тестировать пользовательский код, тесты лежат в разных местах
- – Нет готового решения для lineage, приходится писать велосипед под себя

Преимущества и недостатки legacy подхода

Плюсы

+ OLAP архитектура, PG интерфейс для работы с данными
+ Data Mesh – команды сами отвечают за свои данные
+ YAML engineering (для генерации дагов, DQ проверок и пр.)

Минусы

– Конкуренция за ресурсы, масштабирование и т.п.
– Сложно тестировать пользовательский код, тесты лежат в разных местах
– Нет готового решения для lineage, приходится писать велосипед под себя
– Отсутствует единая модель данных, нет возможности эффективно управлять и влиять на нее

Преимущества и недостатки legacy подхода

Плюсы

+ OLAP архитектура, PG интерфейс для работы с данными
+ Data Mesh – команды сами отвечают за свои данные
+ YAML engineering (для генерации дагов, DQ проверок и пр.)

Минусы

– Конкуренция за ресурсы, масштабирование и т.п.
– Сложно тестировать пользовательский код, тесты лежат в разных местах
– Нет готового решения для lineage, приходится писать велосипед под себя
– Отсутствует единая модель данных, нет возможности эффективно управлять и влиять на нее
– Единая точка отказа. Придет ИИ-агент и все сломает

Переход к архитектуре Data Lakehouse

Наш техстек Data Lakehouse

Наш техстек Data Lakehouse

- Storage – s3 (cloud vendor) 

Наш техстек Data Lakehouse

- Storage – s3 (cloud vendor) 

- Metadata catalog – Nessie (REST) 

Наш техстек Data Lakehouse

- Storage – s3 (cloud vendor) 

- Metadata catalog – Nessie (REST) 
- Compute – Trino (масштабируемые кластеры под разные типы нагрузки) 

Наш техстек Data Lakehouse

- Storage – s3 (cloud vendor) 

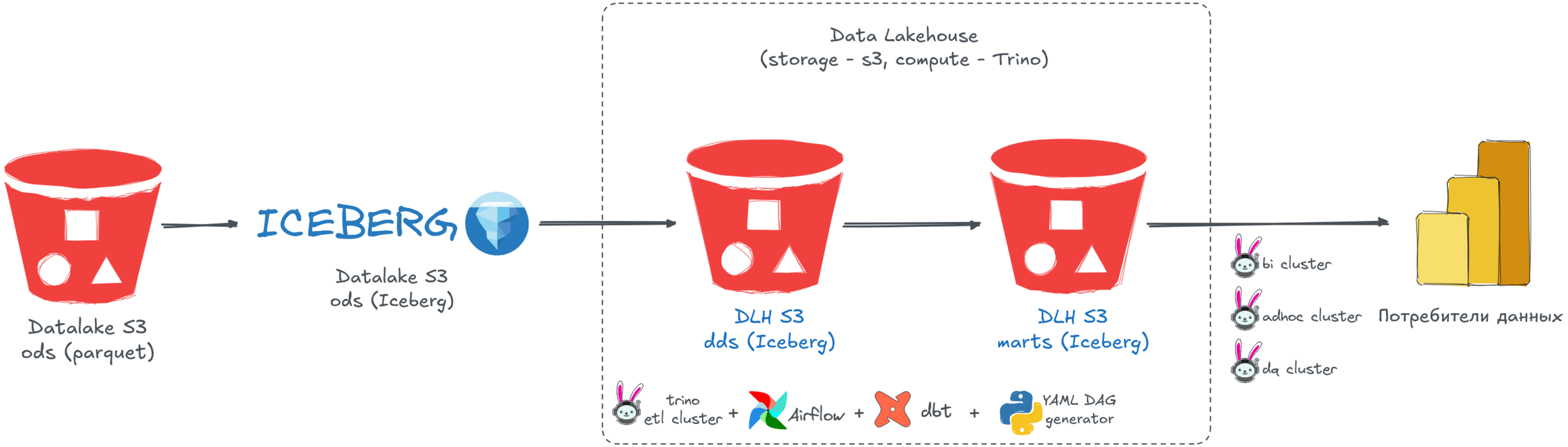
- Metadata catalog – Nessie (REST) 
- Compute – Trino (масштабируемые кластеры под разные типы нагрузки) 
- Orchestration – Airflow 2.x/3.x 

Наш техстек Data Lakehouse

- Storage – s3 (cloud vendor) 

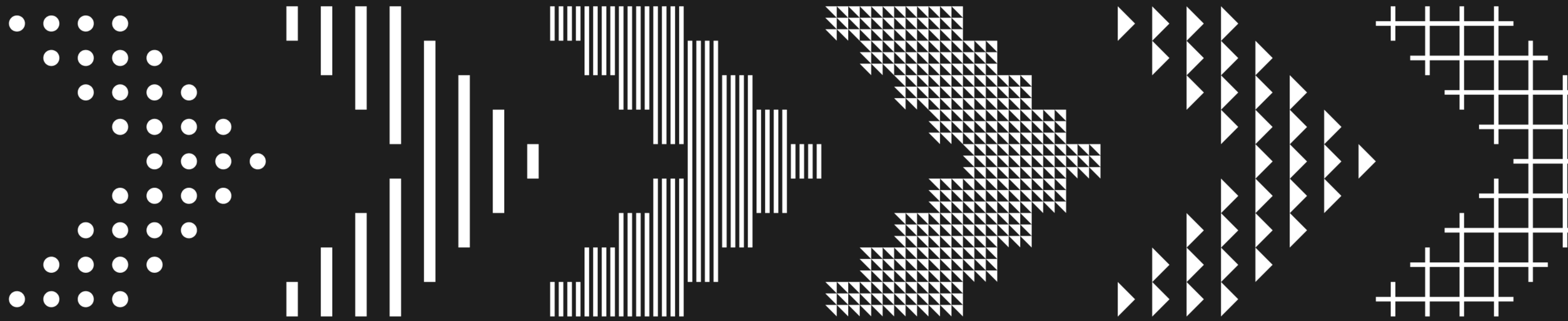
- Metadata catalog – Nessie (REST) 
- Compute – Trino (масштабируемые кластеры под разные типы нагрузки) 
- Orchestration – Airflow 2.x/3.x 
- ETL framework – dbt (dbt-trino) 

Целевой подход к сборке витрин – на DLH



лемана тех

Как перевезти пользовательскую сборку витрин в Data Lakehouse



В основе DWH проекта был монорепозиторий – оставляем такой же подход

dbt-model SQL

```sql
SELECT
    business_dt
    , visitor_id
    , visit_id
    , hit_number
    , host_name
    , page_path
    , query_string
    , product_family_id
    , product_sku
    , product_list_name
    , product_list_position
    , product_revenue
    , product_price
    , product_quantity
    , action_type
    , step
    , event_content
    , action_convarchar
    , action_name
    , action_rn_desc
    , CAST(search_phrase AS integer) AS search_phrase
FROM {{ source('dbt_test_ods', 'tst_schema_change') }}
WHERE business_dt = business_dt
```

Локальная разработка и тестирование – все в одном месте

- Код и конфиги dbt моделей

dbt-model

```sql
SELECT
    business
    , visitc
    , visit_
    , hit_nu
    , host_n
    , page_p
    , query_
    , produc
    , produc
    , produc
    , produc
    , produc
    , produc
    , produc
    , action
    , step
    , event_
    , action
    , action
    , action
    , CAST(s
FROM {{ sour
WHERE busine
```

dbt-model YAML

```yaml
version: 2

models:
  - name: dlh.dbt_test_marts.tst_schema_change_mart
    config:
      materialized: incremental
      tags: ["P0000"]
      on_schema_change: append_new_columns
      unique_key: ['business_dt', 'visit_id']
      properties:
        format: "'PARQUET'"
        format_version: "2"
    columns:
      - name: business_dt
      - name: visitor_id
      - name: visit_id
      - name: hit_number
      - name: host_name
      - name: page_path
      - name: query_string
      - name: search_phrase
      - name: product_family_id
      - name: product_sku
      - name: product_list_name
      - name: product_list_position
      - name: product_revenue
      - name: product_price
      - name: product_quantity
      - name: action_type
      - name: step
      - name: event_content
      - name: action_convarchar
      - name: action_name
      - name: action_rn_desc
```

Локальная разработка и тестирование – все в одном месте

- Код и конфиги dbt моделей

.sqlfluff TOML

```toml
[sqlfluff]

dialect = trino
templater = dbt
sql_file_exts = .sql

fix_even_unparsable = False

rules = AL01, AL02, AL03, AL04, AL08, AM02, AM04, AM06, AM07,
CP01, CP02, CP03, CP04, CP05, CV07, CV11, CV12, JJ01, LT01, LT02,
LT03, LT04, LT05, LT06, LT07, LT08, LT09, LT10, LT11, LT12, LT13,
LT14, LT15, RF02, RF03, RF04, RF05, ST03, ST05, ST07, ST09
warnings = AL01, AL02, AL03, AL04, AL08, CV07, CV12, JJ01, LT01,
LT02, LT03, LT04, LT05, LT06, LT07, LT08, LT09, LT10, LT11, LT12,
LT13, LT14, LT15, RF04, ST03, ST05, ST09

large_file_skip_byte_limit = 0

processes = -1
max_line_length = 120
output_line_length = 120

[sqlfluff:templater:dbt]
project_dir = ./
profiles_dir = ./
dbt_skip_compilation_error = False
```

Локальная разработка и тестирование – все в одном месте

- Код и конфиги dbt моделей
- Линтер для sql – sqlfluff

dbt-bouncer YAML

```yaml
custom_checks_dir: tests/dbt-bouncer

manifest_checks:
  - name: check_project_name
    project_name_pattern: "^dbt_monorepo$"
  - name: check_model_directories
    include: dlh_etl/models
    permitted_sub_directories:
      - dlh
  - name: check_model_directories
    include: dlh_etl/models/dlh
    permitted_sub_directories:
      - marts
      - dds
  - name: check_model_directories_regex
    include: dlh_etl/models/dlh/marts
    permitted_sub_directories_regex: (^.+_ods$)|(^.+_marts$)|(^.+_marts_wrk$)|(^dds$)|(^dds_wrk$)
  - name: check_model_names
    include: dlh_etl/models/dlh/marts
    model_name_pattern: ^dlh\..+\..+$
  - name: check_model_schema_name
    include: dlh_etl/models/dlh/marts
    schema_name_pattern: (^.+_ods$)|(^.+_marts$)|(^.+_marts_wrk$)|(^dds$)|(^dds_wrk$)
  - name: check_model_database_name
    include: dlh_etl/models/dlh
    database_name_pattern: (^dlh_new$)|(^development__\d{8}$)
```

Локальная разработка и тестирование – все в одном месте

- Код и конфиги dbt моделей
- Линтер для sql – sqlfluff
- Тесты для валидации структуры проекта

airflow-dag YAML

```yaml
kind: DbtDag
version: '2'
spec:
  access_control:
    - Admin
  DAG:
    daily:
      catchup: false
      email:
        - 'Artemiy.Naumov@lemanapro.ru'
      max_active_tasks: 2
      max_active_runs: 2
      owner: Artemiy Naumov
      start_date: '2026-01-01'
      retries: 1
      schedule: 0 1/12 * * *
      tags:
        - trino
        - p0000
      timezone: UTC
      profile: dlh_etlbot
      dbt_vars:
        test: test
  models:
    - dlh.dbt_test_marts.tst_schema_change_mart
    - dlh.dbt_test_marts.tst_schema_change_mart_one
    - dlh.dbt_test_marts.tst_schema_change_mart_two
```

Локальная разработка и тестирование – все в одном месте

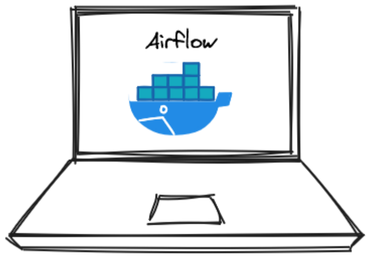
- Код и конфиги dbt моделей
- Линтер для sql – sqlfluff
- Тесты для валидации структуры проекта
- Автоматическая дагогенерация по YAML конфигам. За построение графа по списку моделей отвечает cosmos

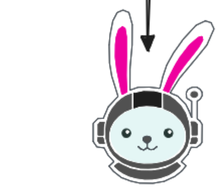
airflow-dag YAML

```yaml
kind: DbtDag
version: '2'
spec:
  access_control:
    - Admin
  DAG:
    daily:
      catchup: false
      email:
        - 'Artemiy.Naumov@lemanapro.ru'
      max_active_tasks: 2
      max_active_runs: 2
      owner: Artemiy Naumov
      start_date: '2026-01-01'
      retries: 1
      schedule: 0 1/12 * * *
      tags:
        - trino
        - p0000
      timezone: UTC
      profile: dlh_etlbot
      dbt_vars:
        test: test
  models:
    - dlh.dbt_test_marts.tst_schema_change_mart
    - dlh.dbt_test_marts.tst_schema_change_mart_one
    - dlh.dbt_test_marts.tst_schema_change_mart_two
```

Локальная разработка и тестирование – все в одном месте

- Код и конфиги dbt моделей
- Линтер для sql – sqlfluff
- Тесты для валидации структуры проекта
- Автоматическая дагогенерация по YAML конфигам. За построение графа по списку моделей отвечает cosmos



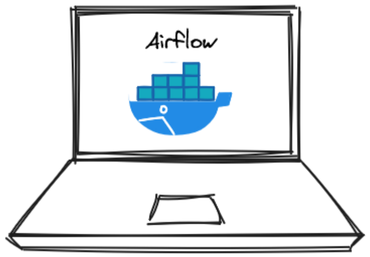


sandbox cluster

development branch

Локальная разработка

- Airflow в Docker + Trino Sandbox cluster



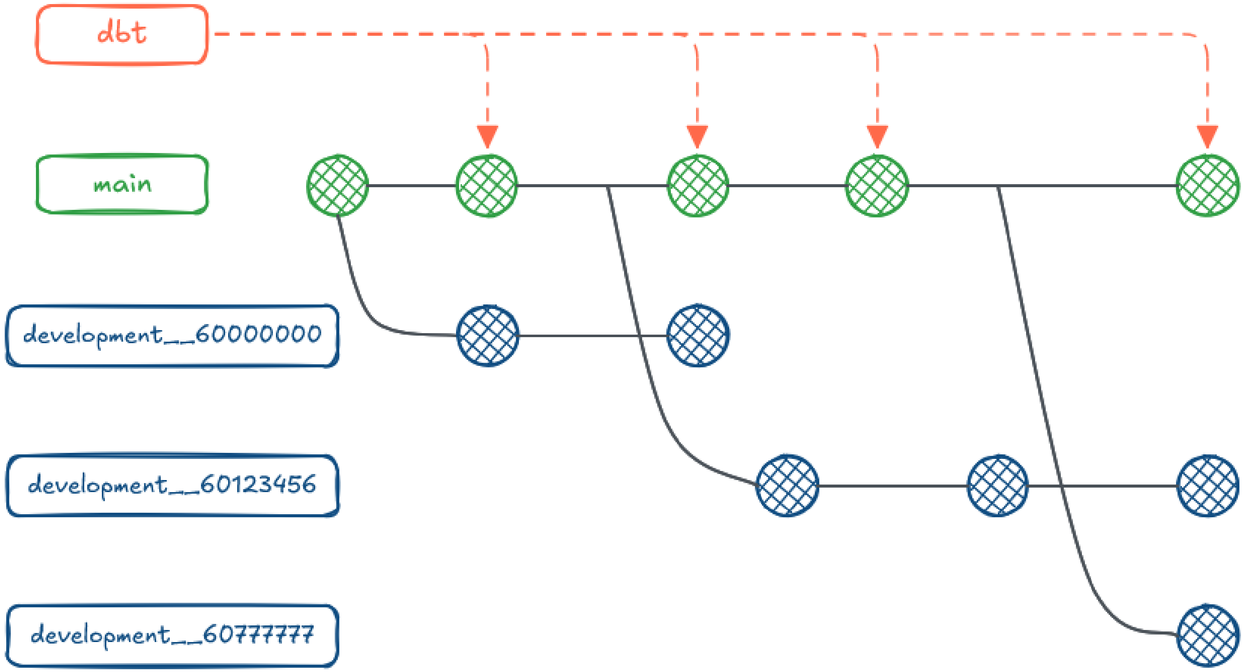
sandbox cluster

development branch

Локальная разработка

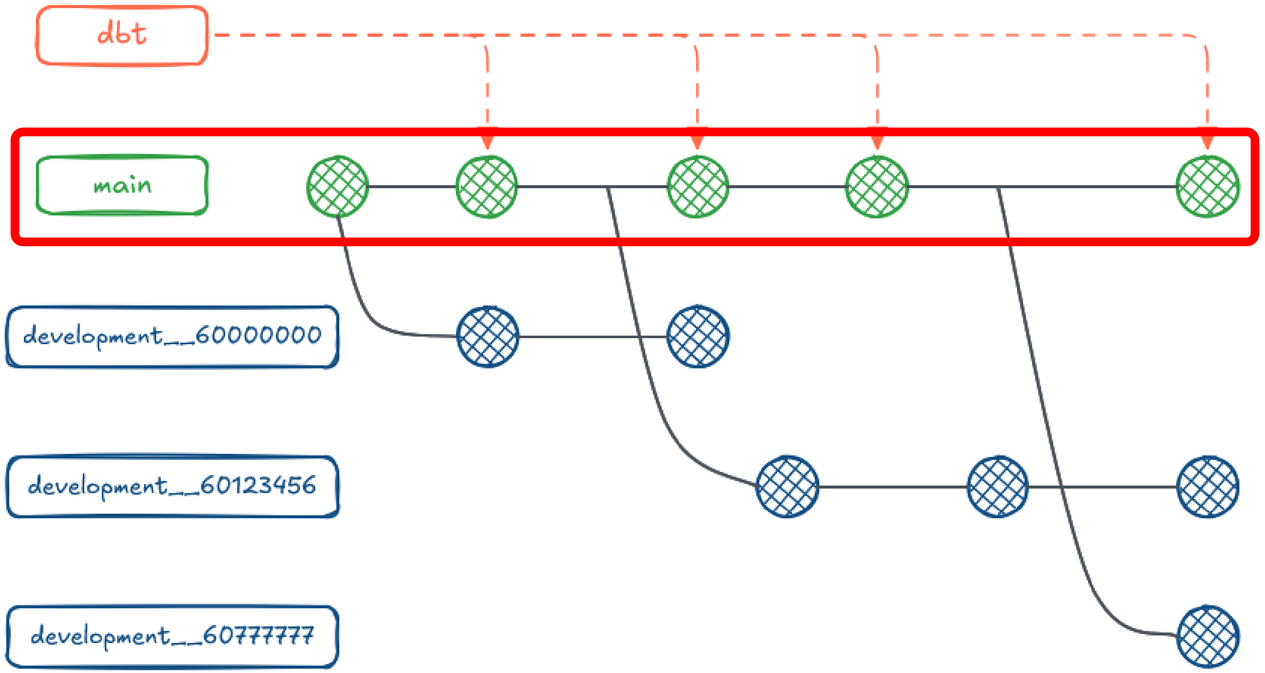
- Airflow в Docker + Trino Sandbox cluster
- Под sandbox создаем development каталог для пользователя

Бранчинг в Nessie



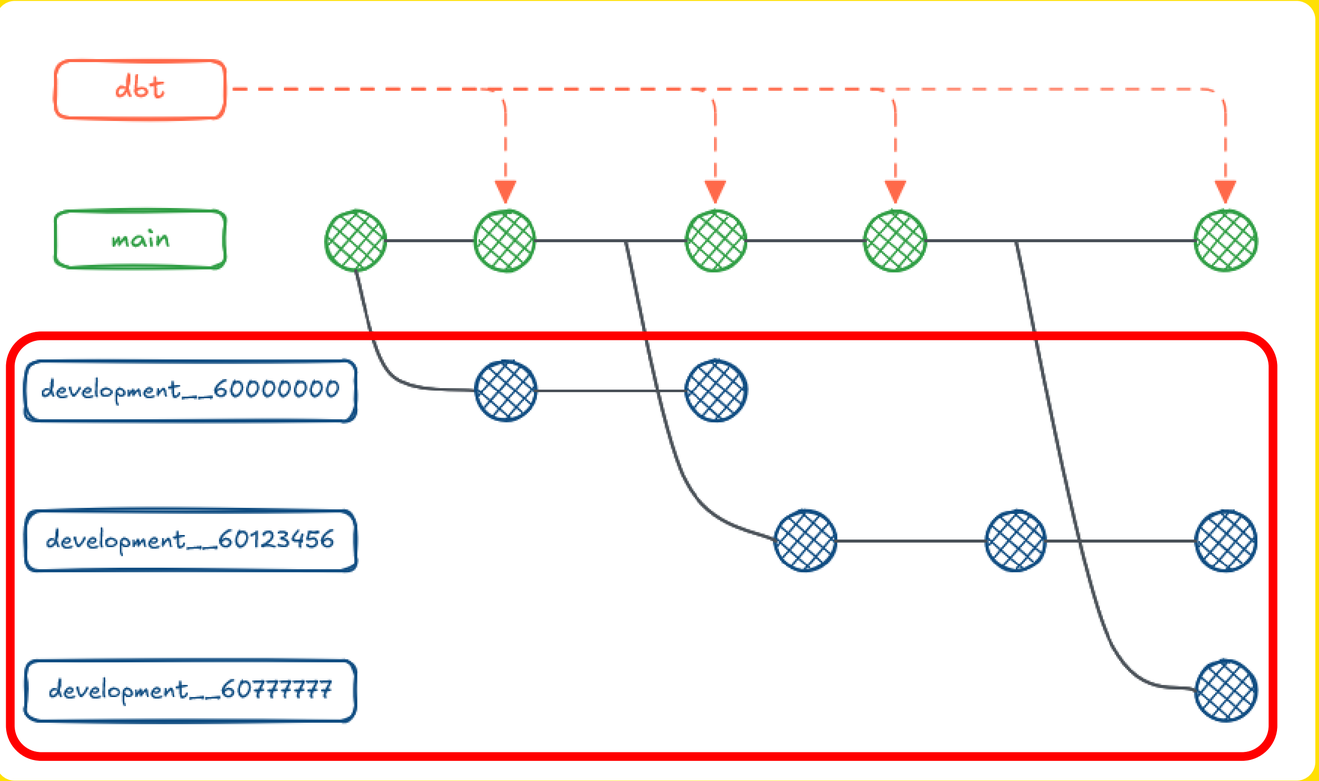
- Nessie – git-like каталог данных

Бранчинг в Nessie



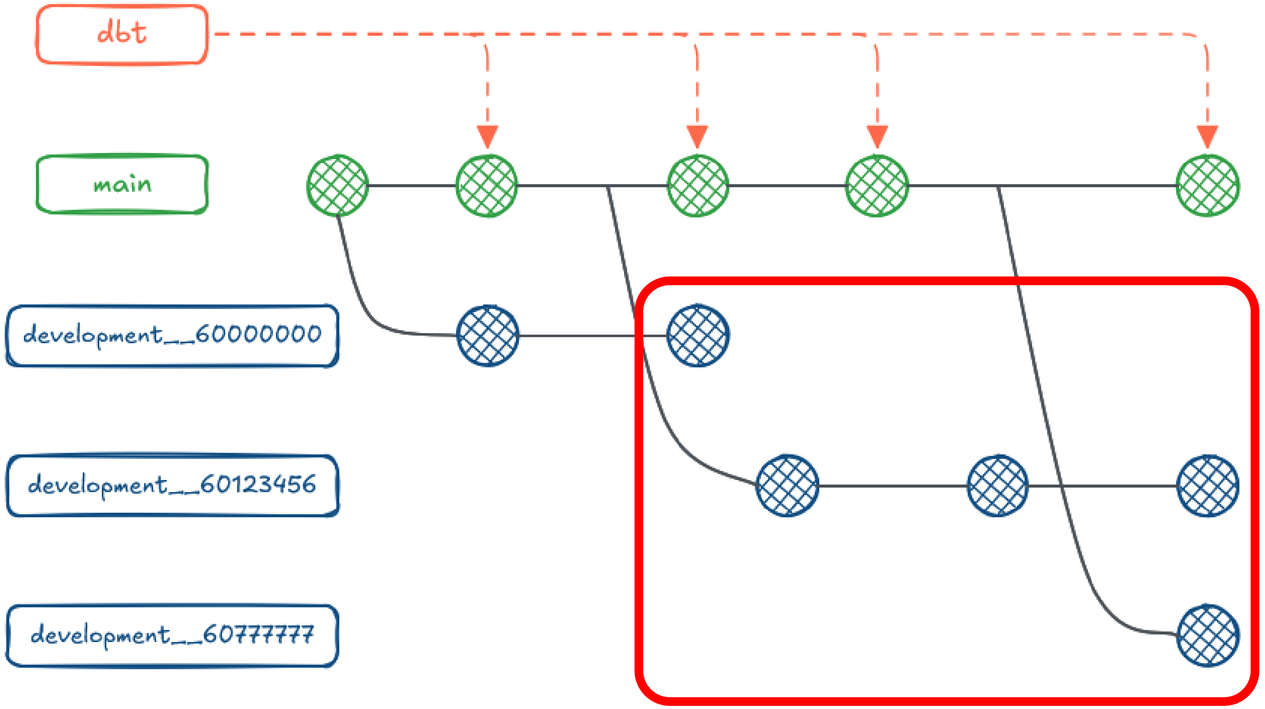
- Nessie – git-like каталог данных
- Main ветка – продовые данные

Бранчинг в Nessie



- Nessie – git-like каталог данных
- Main ветка – продовые данные
- development ветки для разработки – создаем из main, данные в них фиксируются из прода на момент создания ветки (хотелось бы так, но есть нюанс)

Бранчинг в Nessie



- Nessie – git-like каталог данных
- Main ветка – продовые данные
- development ветки для разработки – создаем из main, данные в них фиксируются из прода на момент создания ветки (хотелось бы так, но есть нюанс)
- Завершили разработку и тестирование фичи – удалили ветку

trino-dialect SQL

```sql
-- не работает в Trino:

SELECT '1'::integer;

-- работает в Trino:

SELECT CAST('1' AS INTEGER);

SELECT DATE(now());

-- hyperloglog

SELECT
    merge(hll_column) AS hll_column
FROM (
    SELECT
        CAST(hll_column_1 AS HyperLogLog) AS hll_column
    FROM your_table

    UNION ALL

    SELECT
        CAST(hll_column_2 AS HyperLogLog) AS hll_column
    FROM your_table
) t
```

Перенос SQL на диалект Trino

- Другие inline функции, нет `UNNEST()`, другая работа с `HyperLogLog`

trino-dialect SQL

```sql
-- не работает в Trino:

SELECT '1'::integer;

-- работает в Trino:

SELECT CAST('1' AS INTEGER);

SELECT DATE(now());

-- hyperloglog

SELECT
    merge(hll_column) AS hll_column
FROM (
    SELECT
        CAST(hll_column_1 AS HyperLogLog) AS hll_column
    FROM your_table

    UNION ALL

    SELECT
        CAST(hll_column_2 AS HyperLogLog) AS hll_column
    FROM your_table
) t
```

Перенос SQL на диалект Trino

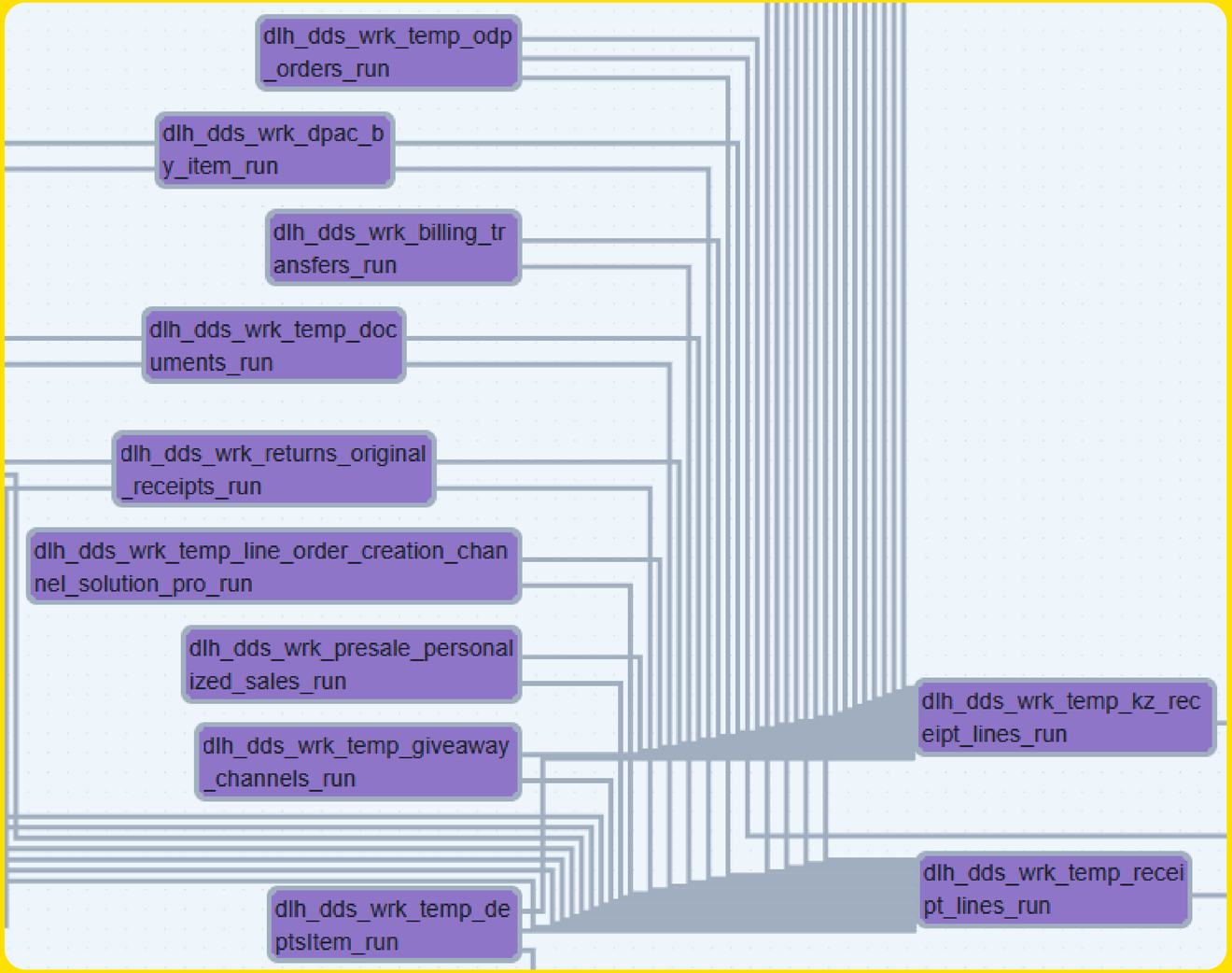
- Другие inline функции, нет `UNNEST()`, другая работа с `HyperLogLog`
- Приведение типов данных – только в явном виде через CAST, и многое другое

trino-dialect SQL

```sql
-- не работает в Trino:

SELECT '1'::integer;

-- работает в Trino:

SELECT CAST('1' AS INTEGER);

SELECT DATE(now());

-- hyperloglog

SELECT
    merge(hll_column) AS hll_column
FROM (
    SELECT
        CAST(hll_column_1 AS HyperLogLog) AS hll_column
    FROM your_table

    UNION ALL

    SELECT
        CAST(hll_column_2 AS HyperLogLog) AS hll_column
    FROM your_table
) t
```

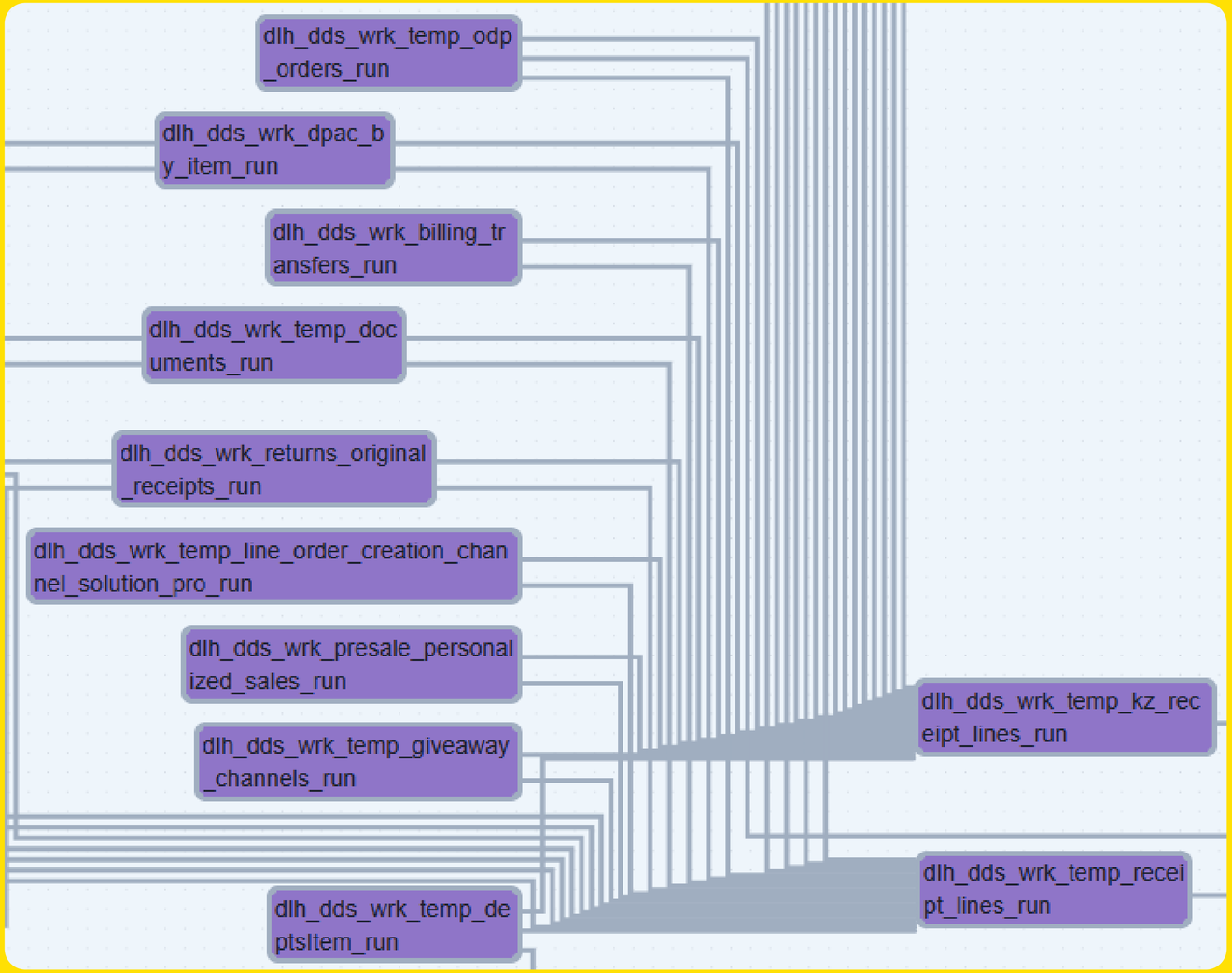
Перенос SQL на диалект Trino

- Другие inline функции, нет `UNNEST()`, другая работа с `HyperLogLog`
- Приведение типов данных – только в явном виде через CAST, и многое другое
- С подключением ии-моделей – проблем с переносом скриптов минимум



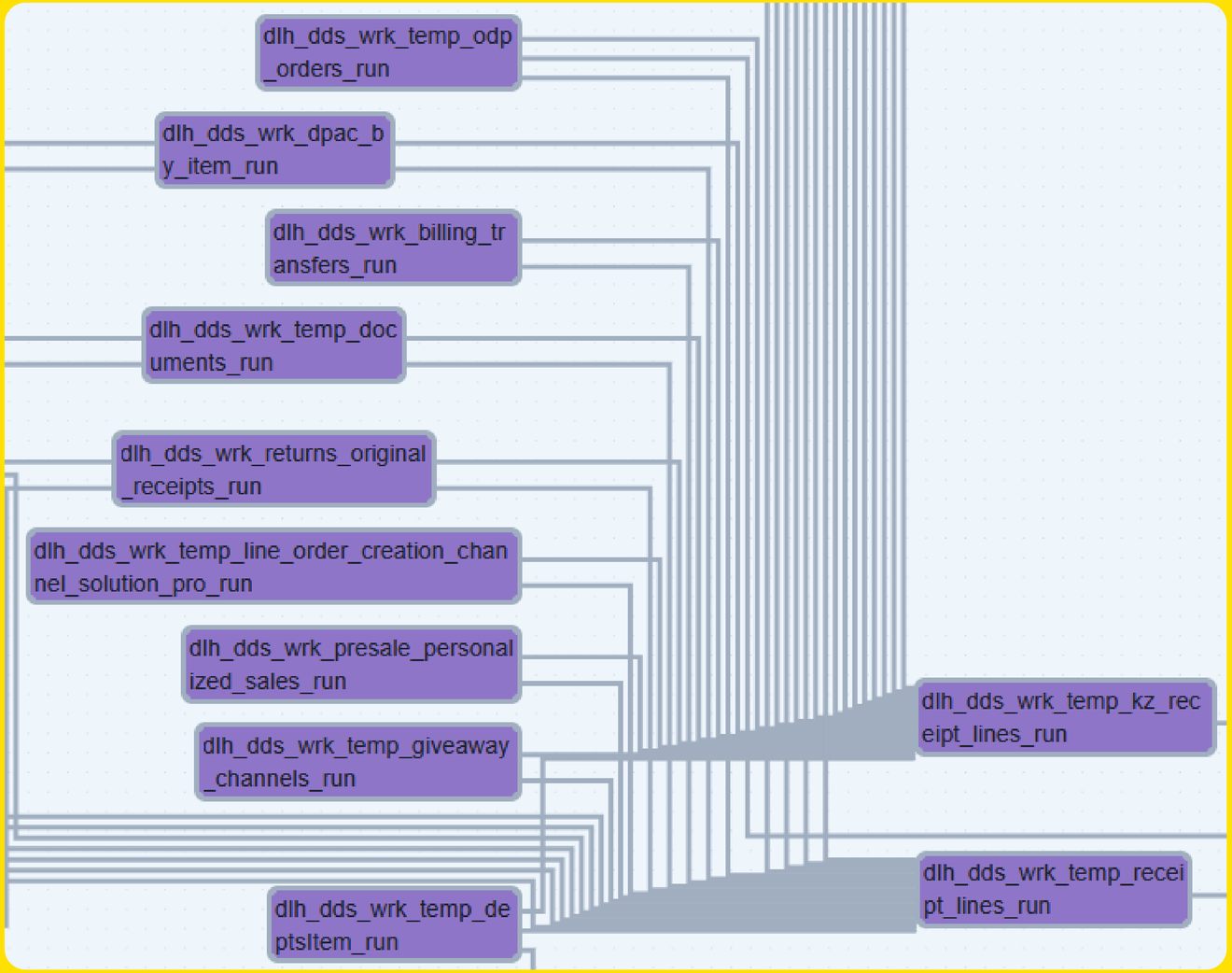
Обходим ограничения Trino

- Trino имеет лимиты на использование памяти в запросе, на размер плана запроса и пр.



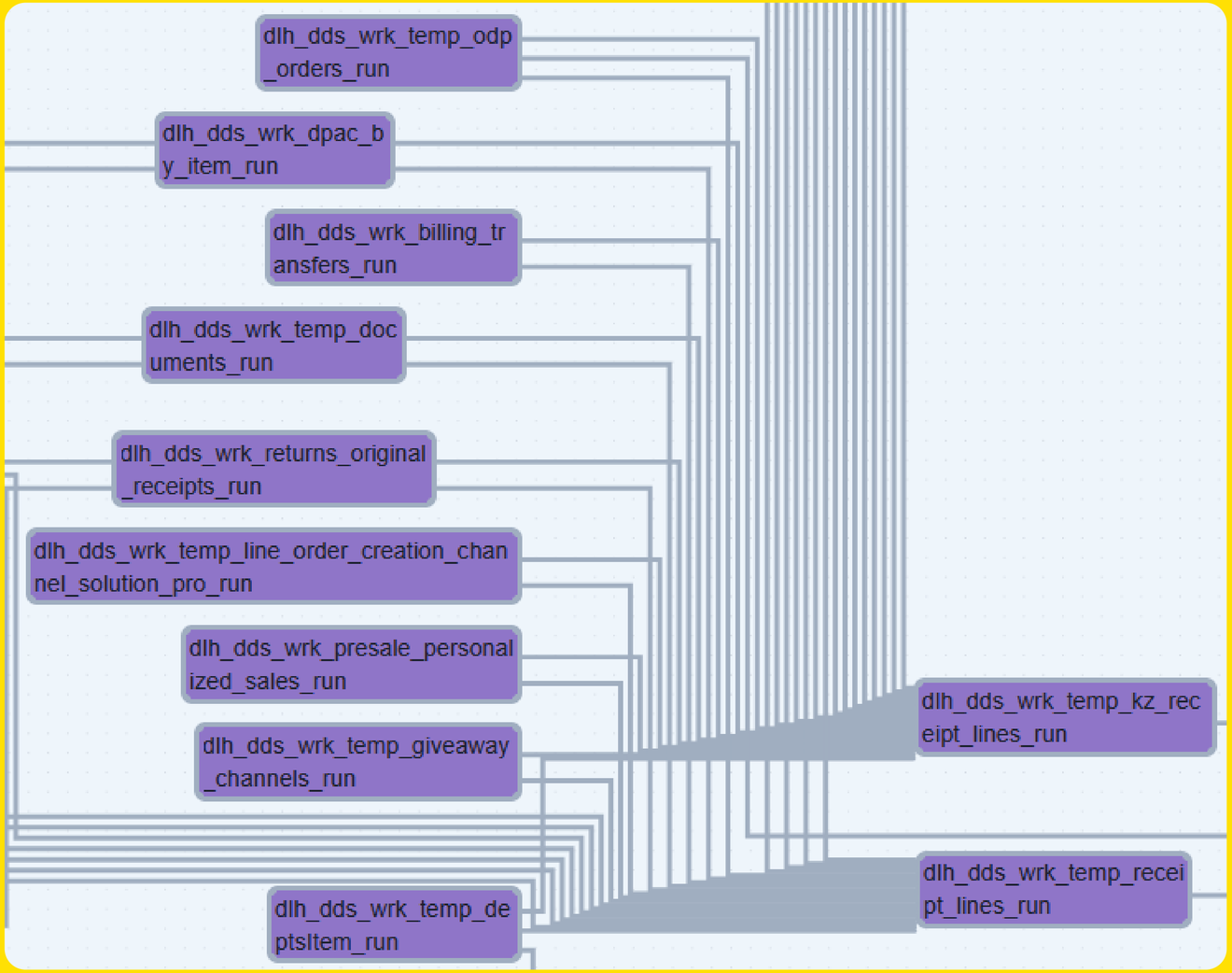
Обходим ограничения Trino

- Trino имеет лимиты на использование памяти в запросе, на размер плана запроса и пр.
- Идеальный сценарий – сборка витрины «влезает» в одну dbt модель



Обходим ограничения Trino

- Trino имеет лимиты на использование памяти в запросе, на размер плана запроса и пр.
- Идеальный сценарий – сборка витрины «влезает» в одну dbt модель
- Если нет – необходимо разбивать одну модель на несколько промежуточных с материализацией



Обходим ограничения Trino

- Trino имеет лимиты на использование памяти в запросе, на размер плана запроса и пр.
- Идеальный сценарий – сборка витрины «влезает» в одну dbt модель
- Если нет – необходимо разбивать одну модель на несколько промежуточных с материализацией
- Для промежуточных моделей – заводим `_wrk` схемы

temp_table YAML

```yaml
version: 2

models:
  - name: dlh.dbt_test_marts.tst_temp_table
    config:
      materialized: temp_table
      tags: ["P0000"]
      unique_key: id
      properties:
        format: "'PARQUET'"
        format_version: "2"
    columns:
      - name: id
        data_type: integer
      - name: created_dttm
        data_type: timestamp(6) with time zone
```

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun

temp_table YAML

```yaml
version: 2

models:
  - name: dlh.dbt_test_marts.tst_temp_table
    config:
      materialized: temp_table
      tags: ["P0000"]
      unique_key: id
      properties:
        format: "'PARQUET'"
        format_version: "2"
    columns:
      - name: id
        data_type: integer
      - name: created_dttm
        data_type: timestamp(6) with time zone
```

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun

temp_table SQL

```sql
create table "dlh_new"."dbt_test_marts"."tst_temp_table__bb648a94abe8b713639bf11380c1f8f1"

  WITH (format = 'PARQUET',
format_version = 2)
  as (
    SELECT
  CAST(RANDOM(1, 100) AS INTEGER) AS id
  , CURRENT_TIMESTAMP(6) AS created_dttm
  )
```

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun

temp_table SQL

```sql
create table "dlh_new"."dbt_test_marts"."tst_temp_table__bb648a94abe8b713639bf11380c1f8f1"

  WITH (format = 'PARQUET',
format_version = 2)
  as (
    SELECT
  CAST(RANDOM(1, 100) AS INTEGER) AS id
  , CURRENT_TIMESTAMP(6) AS created_dttm
  )
```

dbt-trino/macros/materializations/incremental.sql SQL

```sql
delete from {{ target }}
where exists (
    select 1
    from {{ source }}
    where
    {% for key in unique_key %}
        {{ target }}.{{ key }} = {{ source }}.{{ key }}
        {{ "and " if not loop.last }}
    {% endfor %}
    )
    {% if incremental_predicates %}
        {% for predicate in incremental_predicates %}
            and {{ predicate }}
        {% endfor %}
    {% endif %}
```

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun
- `delete_insert` – для инкрементальной загрузки данных с удалением без фильтра по ключевым полям

dbt-trino/macros/materializations/incremental.sql SQL

```
delete from {{ target }}
where exists (
    select 1
    from {{ source }}
    where
    {% for key in unique_key %}
        {{ target }}.{{ key }} = {{ source }}.{{ key }}
        {{ "and " if not loop.last }}
    {% endfor %}
    )
    {% if incremental_predicates %}
        {% for predicate in incremental_predicates %}
            and {{ predicate }}
        {% endfor %}
    {% endif %}
```

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun
- `delete_insert` – для инкрементальной загрузки данных с удалением без фильтра по ключевым полям

delete_insert YAML

```yaml
version: 2

models:
  - name: dlh.dbt_test_marts.tst_delete_insert_materialization_model
    config:
      tags: ["P0000"]
      materialized: delete_insert
      unique_key: id
      on_schema_change: sync_all_columns
      delete_predicates:
        - "date_column = DATE('{{ var('ds') }}')"
      properties:
        format: "'PARQUET'"
        format_version: "2"
        partitioning:
          - 'date_column'
    columns:
      - name: id
        data_type: integer
      - name: name
        data_type: varchar
      - name: date_column
        data_type: date
      - name: created_dttm
        data_type: timestamp(6)
```

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun
- `delete_insert` – для инкрементальной загрузки данных с удалением без фильтра по ключевым полям

delete_insert YAML

```yaml
version: 2

models:
  - name: dlh.dbt_test_marts.tst_delete_insert_materialization_model
    config:
      tags: ["P0000"]
      materialized: delete_insert
      unique_key: id
      on_schema_change: sync_all_columns
      delete_predicates:
        - "date_column = DATE('{{ var('ds') }}')"
      properties:
        format: "'PARQUET'"
        format_version: "2"
        partitioning:
          - 'date_column'
    columns:
      - name: id
        data_type: integer
      - name: name
        data_type: varchar
      - name: date_column
        data_type: date
      - name: created_dttm
        data_type: timestamp(6)
```

Кастомные материализации dbt

- `temp_table` – эмуляция временных таблиц. Удаление всех временных таблиц – в конце каждого DagRun
- `delete_insert` – для инкрементальной загрузки данных с удалением без фильтра по ключевым полям

parametrized-model SQL

```sql
{% set p_load_date = var('ds') %}

{% set ru_default_vat_rate_bef_26 = 0.20 %}
{% set ru_default_vat_rate_for_26 = 0.22 %}
{% set kz_default_vat_rate_bef_26 = 0.12 %}
{% set kz_default_vat_rate_for_26 = 0.16 %}

{% if var('incremental_update_flg', false) %}
    {% set start_date_interval = var("incremental_update_days_interval", 6) %}
{% else %}
    {% set start_date_interval = 0 %}
{% endif %}

WITH
    dates AS (
        SELECT
            cost_date
        FROM UNNEST(
            SEQUENCE(
                DATE('{{ p_load_date }}') - INTERVAL '{{ start_date_interval }}' DAY,
                DATE('{{ p_load_date }}'),
                INTERVAL '1' DAY
            )
        ) AS t(cost_date)
    )

SELECT
    ...
```

Параметризация моделей

- Зачастую требуется параметризация SQL кода (по аналогии с функциями Greenplum)

parametrized-model SQL

```sql
{% set p_load_date = var('ds') %}

{% set ru_default_vat_rate_bef_26 = 0.20 %}
{% set ru_default_vat_rate_for_26 = 0.22 %}
{% set kz_default_vat_rate_bef_26 = 0.12 %}
{% set kz_default_vat_rate_for_26 = 0.16 %}

{% if var('incremental_update_flg', false) %}
    {% set start_date_interval = var("incremental_update_days_interval", 6) %}
{% else %}
    {% set start_date_interval = 0 %}
{% endif %}

WITH
    dates AS (
        SELECT
            cost_date
        FROM UNNEST(
            SEQUENCE(
                DATE('{{ p_load_date }}') - INTERVAL '{{ start_date_interval }}' DAY,
                DATE('{{ p_load_date }}'),
                INTERVAL '1' DAY
            )
        ) AS t(cost_date)
    )

SELECT
    ...
```

Параметризация моделей

- Зачастую требуется параметризация SQL кода (по аналогии с функциями Greenplum)
- Решение – используем возможности jinja шаблонизации, выражения `set`, `if-then-else`, `for-loop` и др.

parametrized-model SQL

```sql
{% set p_load_date = var('ds') %}

{% set ru_default_vat_rate_bef_26 = 0.20 %}
{% set ru_default_vat_rate_for_26 = 0.22 %}
{% set kz_default_vat_rate_bef_26 = 0.12 %}
{% set kz_default_vat_rate_for_26 = 0.16 %}

{% if var('incremental_update_flg', false) %}
    {% set start_date_interval = var("incremental_update_days_interval", 6) %}
{% else %}
    {% set start_date_interval = 0 %}
{% endif %}

WITH
    dates AS (
        SELECT
            cost_date
        FROM UNNEST(
            SEQUENCE(
                DATE('{{ p_load_date }}') - INTERVAL '{{ start_date_interval }}' DAY,
                DATE('{{ p_load_date }}'),
                INTERVAL '1' DAY
            )
        ) AS t(cost_date)
    )

SELECT
    ...
```

Параметризация моделей

- Зачастую требуется параметризация SQL кода (по аналогии с функциями Greenplum)
- Решение – используем возможности jinja шаблонизации, выражения `set`, `if-then-else`, `for-loop` и др.

parametrized-model SQL

```sql
{% set p_load_date = var('ds') %}

{% set ru_default_vat_rate_bef_26 = 0.20 %}
{% set ru_default_vat_rate_for_26 = 0.22 %}
{% set kz_default_vat_rate_bef_26 = 0.12 %}
{% set kz_default_vat_rate_for_26 = 0.16 %}

{% if var('incremental_update_flg', false) %}
    {% set start_date_interval = var("incremental_update_days_interval", 6) %}
{% else %}
    {% set start_date_interval = 0 %}
{% endif %}

WITH
    dates AS (
        SELECT
            cost_date
        FROM UNNEST(
            SEQUENCE(
                DATE('{{ p_load_date }}') - INTERVAL '{{ start_date_interval }}' DAY,
                DATE('{{ p_load_date }}'),
                INTERVAL '1' DAY
            )
        ) AS t(cost_date)
    )

SELECT
    ...
```

Параметризация моделей

- Зачастую требуется параметризация SQL кода (по аналогии с функциями Greenplum)
- Решение – используем возможности jinja шаблонизации, выражения `set`, `if-then-else`, `for-loop` и др.

dbt-vars YAML

```yaml
        schedule: 0 20 * * MON,TUE,THU,SAT
        tags:
          - performance
          - trino
          - p0000
        timezone: Europe/Moscow
        profile: dlh_dds_etlbot
        dbt_vars:
          incremental_update_flg: true
          incremental_update_days_interval: 6
```

dbt-vars SQL

```sql
SELECT *
FROM {{ ref('dlh.dbt_test_marts.test_mart') }}
WHERE
    opened_date
    {% if var('incremental_update_flg', false) %}
        BETWEEN DATE('{{ p_load_date }}') - INTERVAL '{{ var("incremental_update_days_interval", 6) }}' DAY AND DATE('{{ p_load_date }}')
    {% else %}
        = DATE('{{ p_load_date }}')
    {% endif %}
```

dbt variables

- Для задания своих dbt variables в даге – параметр `dbt_vars` в YAML

dbt-vars YAML

```yaml
schedule: 0 20 * * MON,TUE,THU,SAT
tags:
  - performance
  - trino
  - p0000
timezone: Europe/Moscow
profile: dlh_dds_etlbot
dbt_vars:
  incremental_update_flg: true
  incremental_update_days_interval: 6
```

dbt-vars SQL

```sql
SELECT *
FROM {{ ref('dlh.dbt_test_marts.test_mart') }}
WHERE
    opened_date
    {% if var('incremental_update_flg', false) %}
        BETWEEN DATE('{{ p_load_date }}') - INTERVAL '{{ var("incremental_update_days_interval", 6) }}' DAY AND DATE('{{ p_load_date }}')
    {% else %}
        = DATE('{{ p_load_date }}')
    {% endif %}
```

dbt variables

- Для задания своих dbt variables в даге – параметр `dbt_vars` в YAML

dbt-vars YAML

```yaml
schedule: 0 20 * * MON,TUE,THU,SAT
tags:
  - performance
  - trino
  - p0000
timezone: Europe/Moscow
profile: dlh_dds_etlbot
dbt_vars:
  incremental_update_flg: true
  incremental_update_days_interval: 6
```

dbt-vars SQL

```sql
SELECT *
FROM {{ ref('dlh.dbt_test_marts.test_mart') }}
WHERE
    opened_date
    {% if var('incremental_update_flg', false) %}
        BETWEEN DATE('{{ p_load_date }}') - INTERVAL '{{ var("incremental_update_days_interval", 6) }}' DAY AND DATE('{{ p_load_date }}')
    {% else %}
        = DATE('{{ p_load_date }}')
    {% endif %}
```

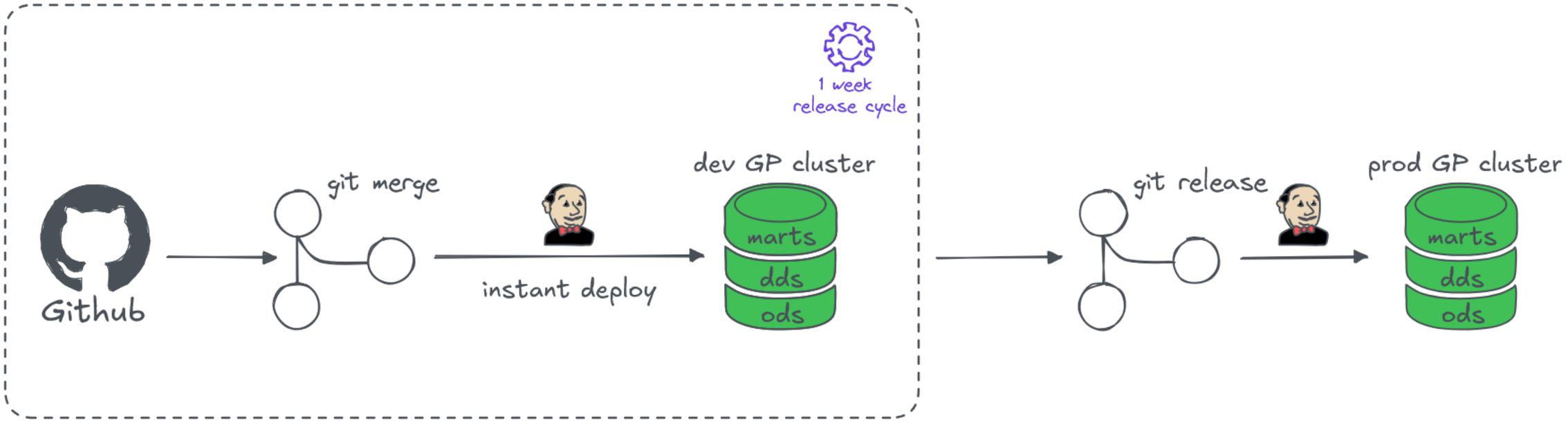
dbt variables

- Для задания своих dbt variables в даге – параметр `dbt_vars` в YAML
- По умолчанию прокидываем в variables основные переменные Airflow

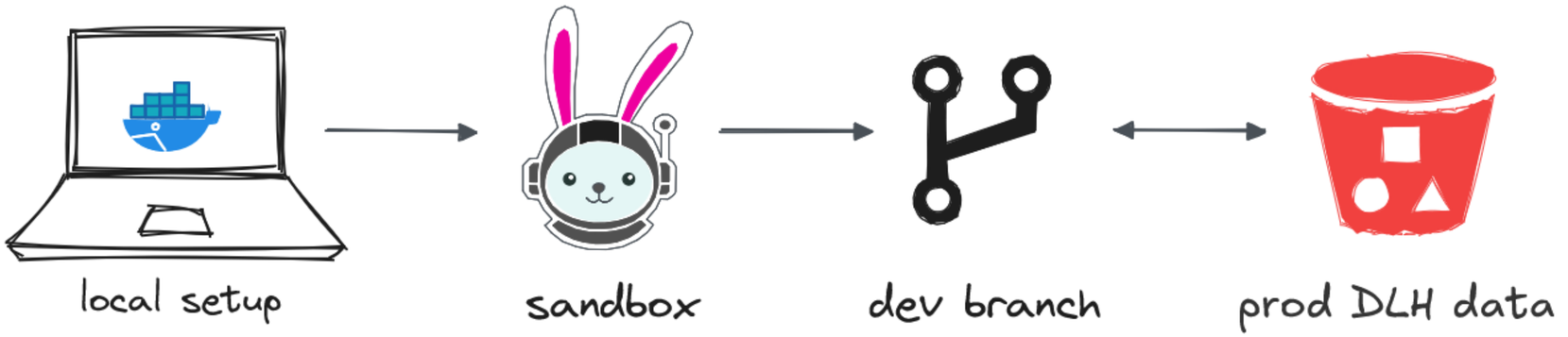
- `execution_date` (равно `ds`)
- `ds`
- `ts`
- `data_interval_start`
- `data_interval_end`
- `prev_data_interval_start_success`
- `prev_data_interval_end_success`
- `prev_start_date_success`
- `prev_end_date_success`
- `dagrun_dag_id` (равно `dag_run.dag_id`)
- `dagrun_run_id` (равно `dag_run.run_id`)

Моментальный деплой на прод окружение – без релизных циклов

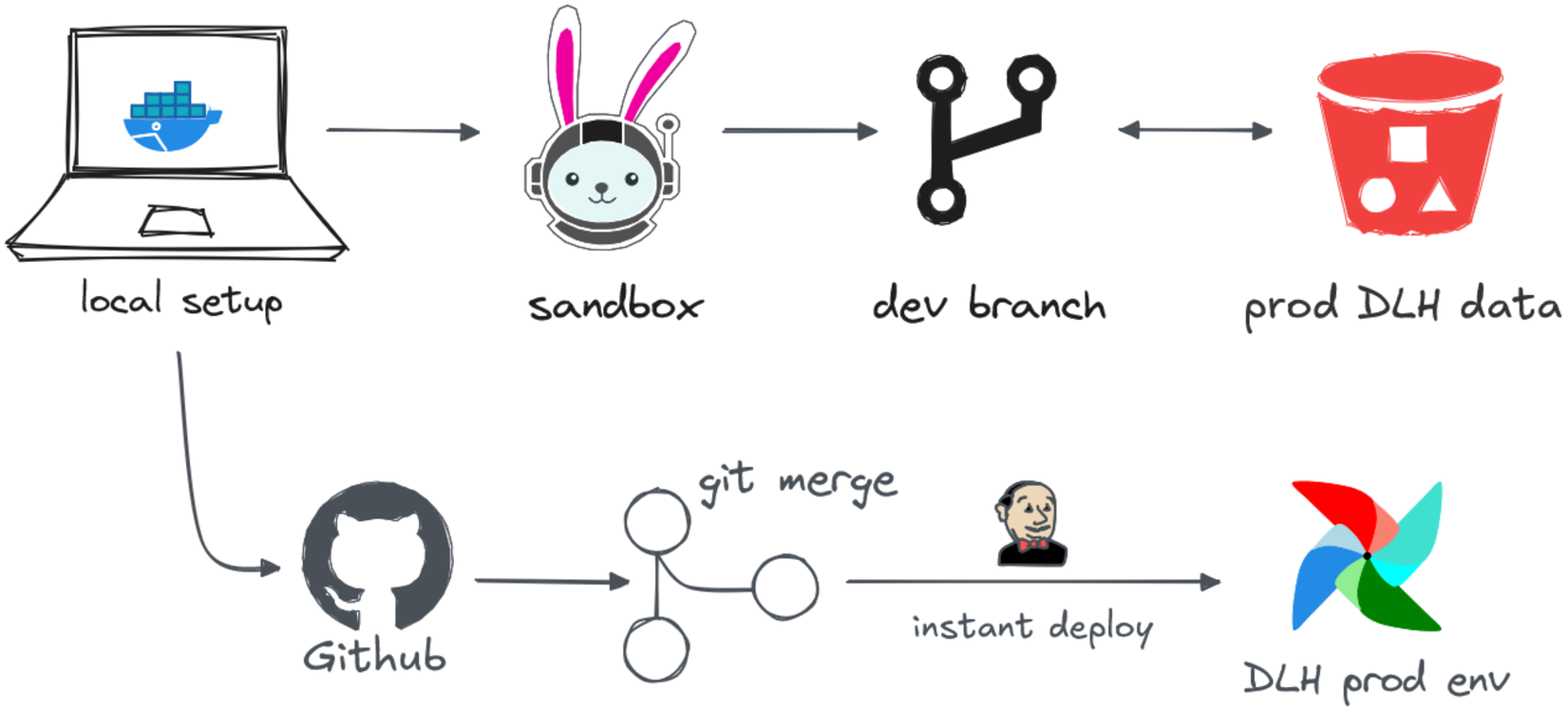
dwh – недельный цикл



dlh – сразу в production

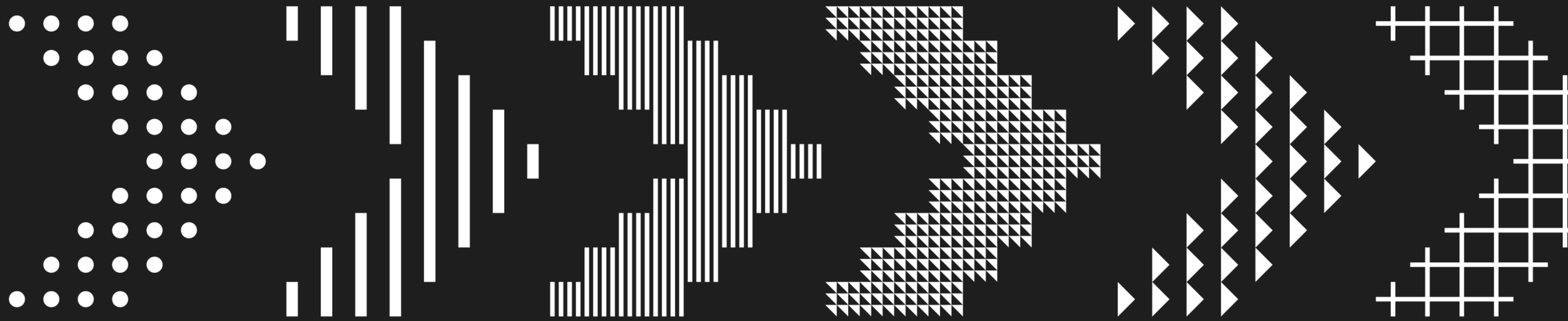


dlh – сразу в production



лемана тех

Дополнительные фичи для DLH, без которых все не полетит



Ролевая модель Data Lakehouse

Ролевая модель – ключевой элемент DLH для доступа к работе с данными

Ролевая модель – ключевой элемент DLH для доступа к работе с данными

- Чтение данных пользователями – только через представления

Ролевая модель – ключевой элемент DLH для доступа к работе с данными

- Чтение данных пользователями – только через представления
- Одна общая публичная роль – `dataread-public`

Ролевая модель – ключевой элемент DLH для доступа к работе с данными

- Чтение данных пользователями – только через представления
- Одна общая публичная роль – `dataread-public`
- `-sensitive` и `-confidential` роли для разграничения доступа к витринам

Ролевая модель – ключевой элемент DLH для доступа к работе с данными

- Чтение данных пользователями – только через представления
- Одна общая публичная роль – `dataread-public`
- `-sensitive` и `-confidential` роли для разграничения доступа к витринам
- `-etl` и `-sandbox` роли на схемы – для etlbots и локальной разработки

Ролевая модель – ключевой элемент DLH для доступа к работе с данными

- Чтение данных пользователями – только через представления
- Одна общая публичная роль – `dataread-public`
- `-sensitive` и `-confidential` роли для разграничения доступа к витринам
- `-etl` и `-sandbox` роли на схемы – для etlbots и локальной разработки
- Пользователи запрашивают доступы через внутренний IAM сервис. Согласование – на владельцах данных и ИБ

Ролевая модель – на уровне Trino

Ролевая модель – на уровне Trino

- По маппингу пользователей, групп Trino и объектов (представлений) формируем конфигурационные файлы для Trino ACL и монтируем их в поды

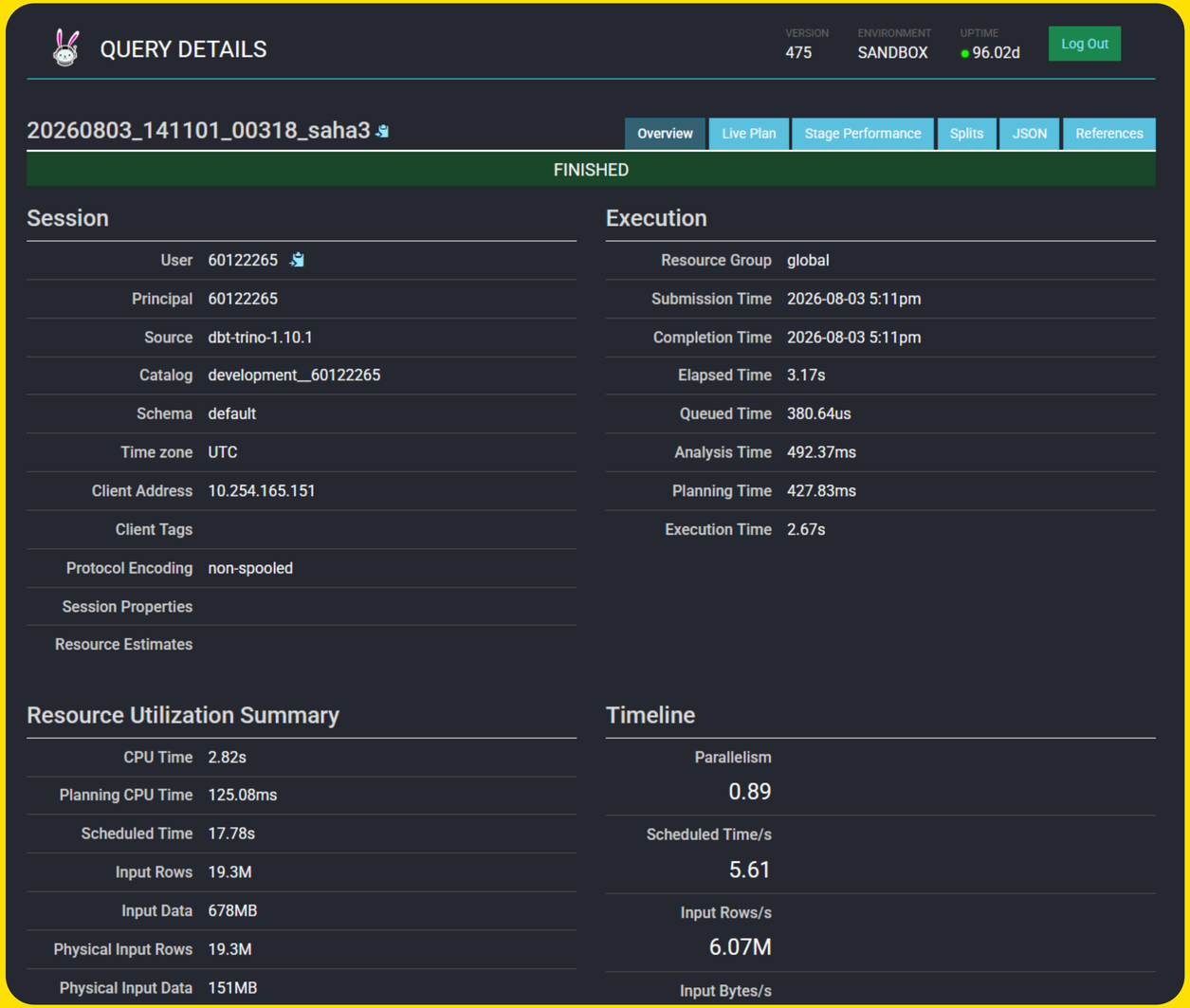
Ролевая модель – на уровне Trino

- По маппингу пользователей, групп Trino и объектов (представлений) формируем конфигурационные файлы для Trino ACL и монтируем их в поды
- Можем сформировать единую сквозную ролевую модель и подкинуть ее на любой новый кластер

Ролевая модель – на уровне Trino

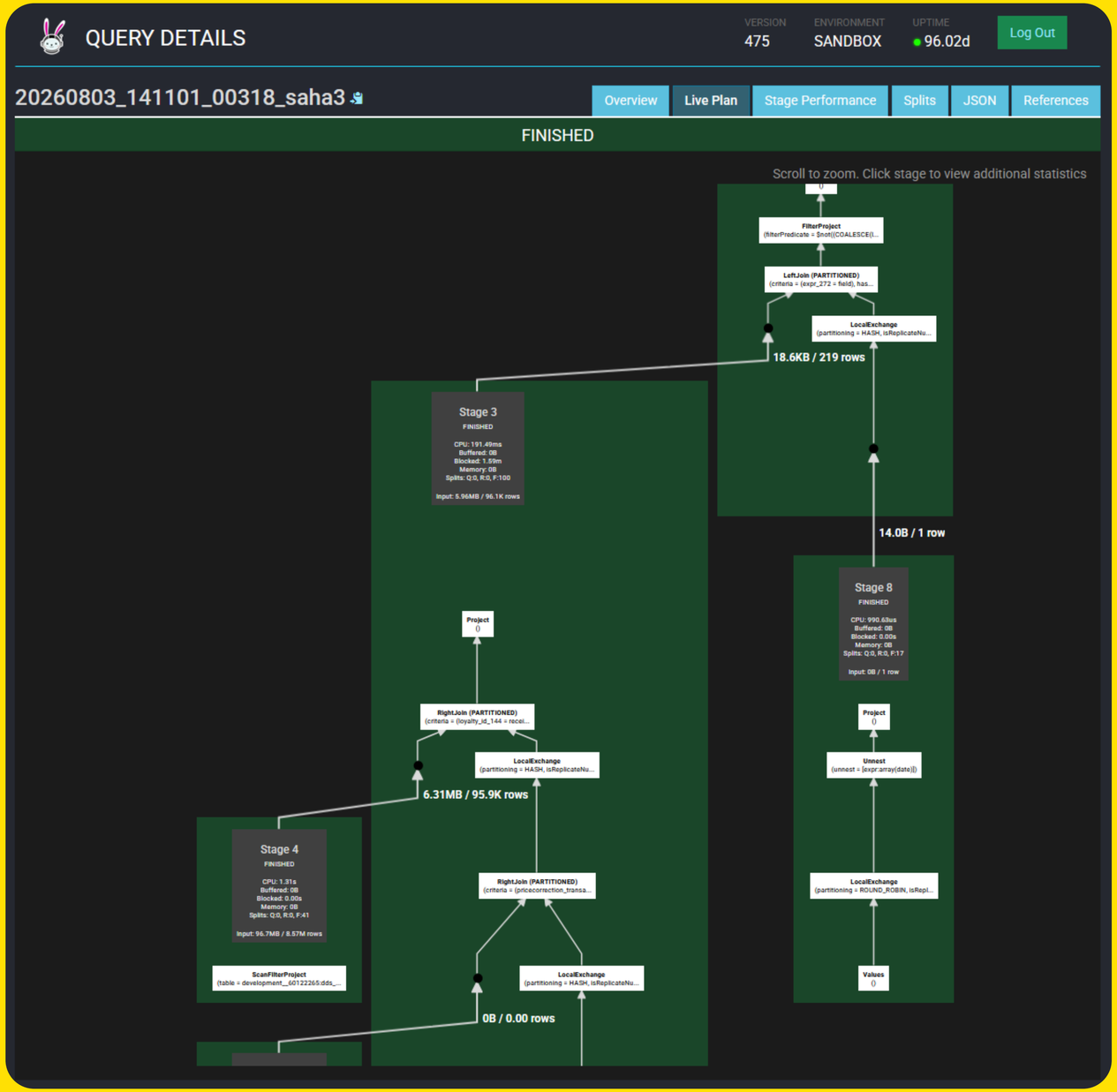
- По маппингу пользователей, групп Trino и объектов (представлений) формируем конфигурационные файлы для Trino ACL и монтируем их в поды
- Можем сформировать единую сквозную ролевую модель и подкинуть ее на любой новый кластер
- Ролевая модель на уровне каталога и других движков – зона для роста

Мониторинг и оптимизация



Мониторинг и оптимизация запросов

- Базовый мониторинг запросов – в Trino UI



Мониторинг и оптимизация запросов

- Базовый мониторинг запросов – в Trino UI

```
INFO - 12:17:15  SQL status: CREATE TABLE (1 rows) in 1.335 seconds
INFO - 12:17:15  Externally logging main query id (for debug purposes): 20260817_121714_53524_f5pkj
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: ROLLBACK
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: Close
INFO - 12:17:15  Sending event: {'category': 'dbt', 'action': 'run_model', 'label': 'da012543-b7d4-4718-9d5b-62
INFO - 12:17:15  1 of 1 OK created sql temp_table model dbt_test_marts.tst_temp_table_materialization_1  [CREAT
INFO - 12:17:15  Finished running node model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1
INFO - 12:17:15  On master: COMMIT
```

Мониторинг и оптимизация запросов

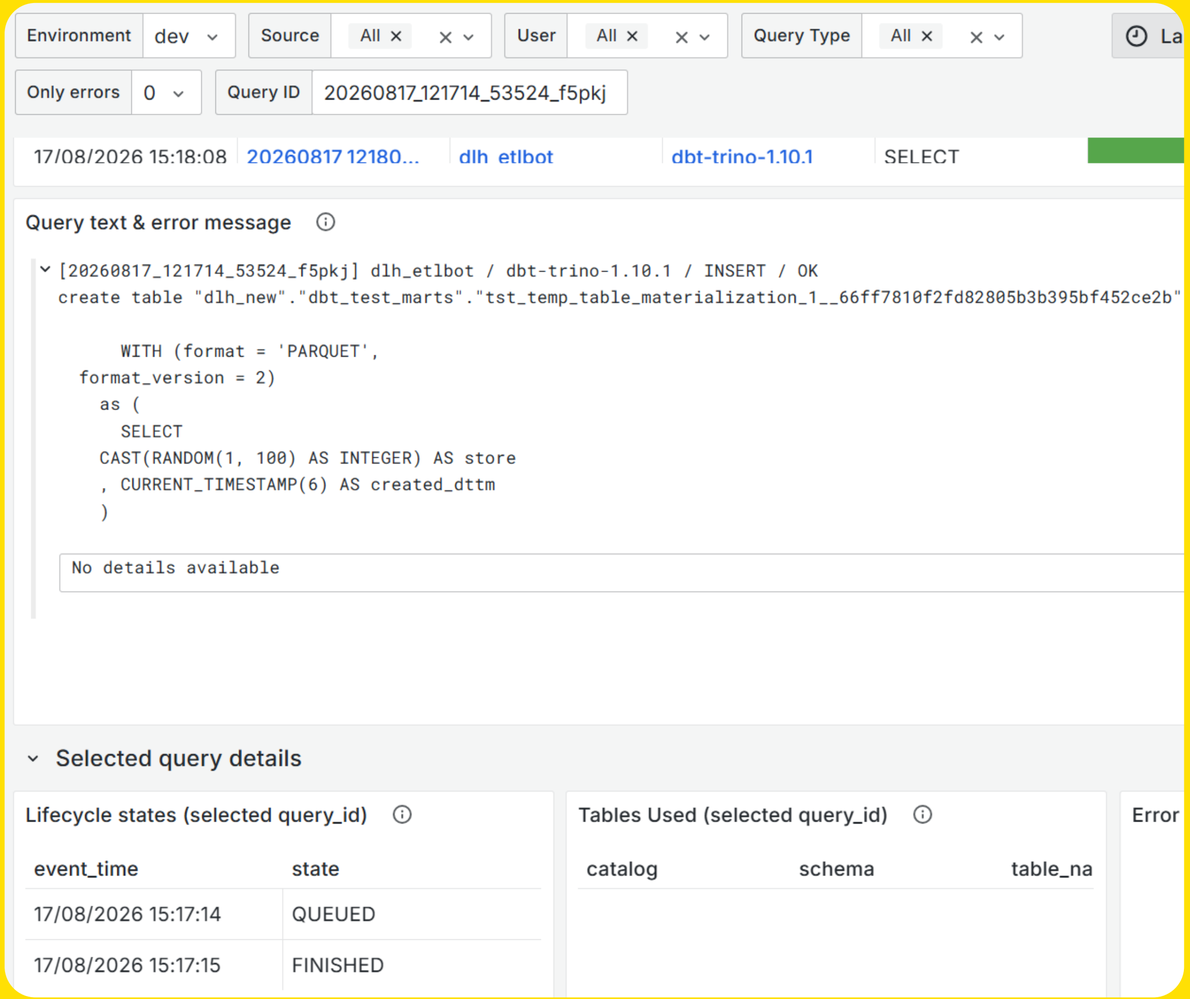
- Базовый мониторинг запросов – в Trino UI
- Дополнительно логируем `query_id` в airflow task, далее находим логи в Grafana

```
INFO - 12:17:15  SQL status: CREATE TABLE (1 rows) in 1.335 seconds
INFO - 12:17:15  Externally logging main query id (for debug purposes): 20260817_121714_53524_f5pkj
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: ROLLBACK
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: Close
INFO - 12:17:15  Sending event: {'category': 'dbt', 'action': 'run_model', 'label': 'da012543-b7d4-4718-9d5b-62
INFO - 12:17:15  1 of 1 OK created sql temp_table model dbt_test_marts.tst_temp_table_materialization_1  [CREAT
INFO - 12:17:15  Finished running node model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1
INFO - 12:17:15  On master: COMMIT
```

Мониторинг и оптимизация запросов

- Базовый мониторинг запросов – в Trino UI
- Дополнительно логируем `query_id` в airflow task, далее находим логи в Grafana

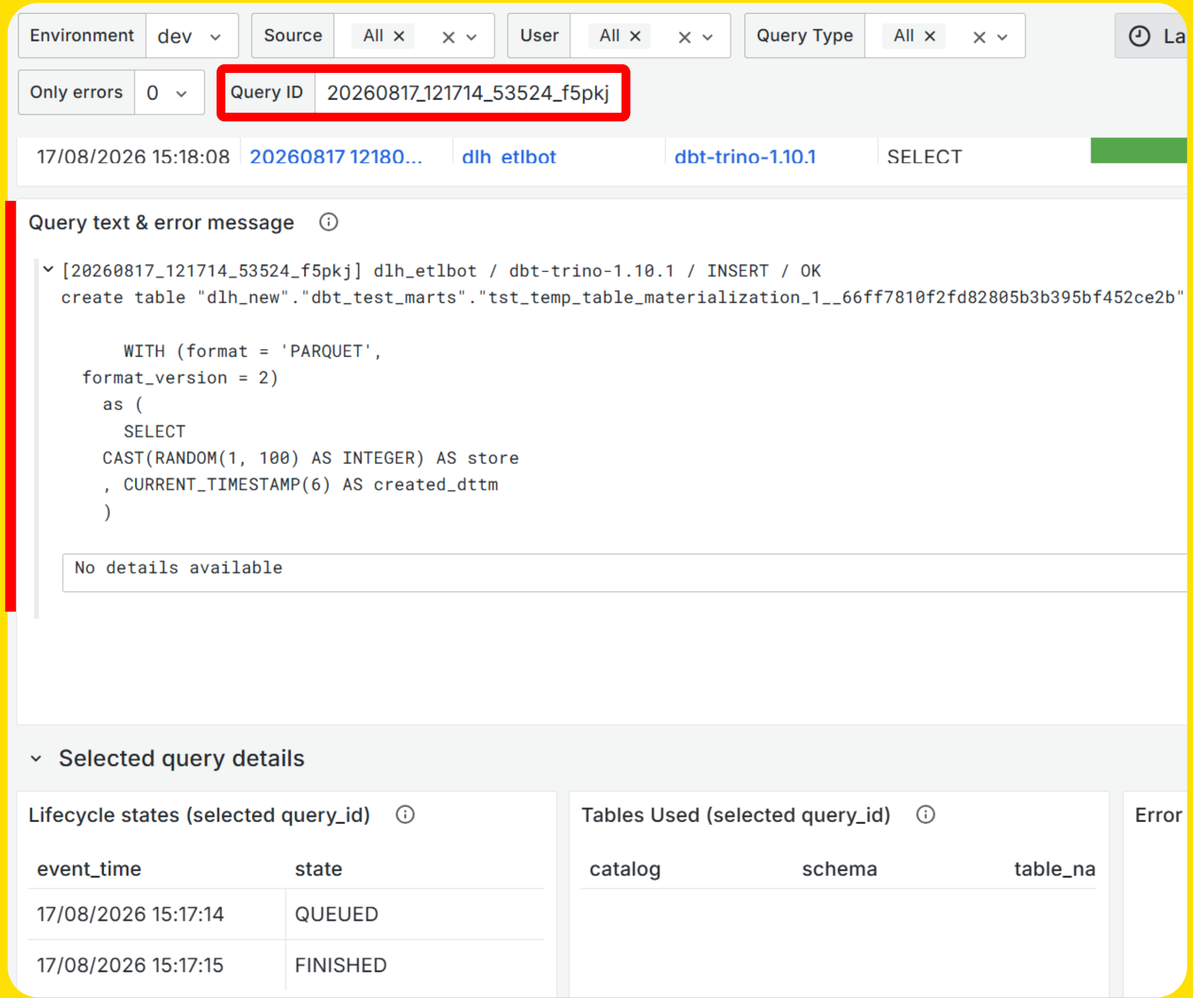
```
INFO - 12:17:15  SQL status: CREATE TABLE (1 rows) in 1.335 seconds
INFO - 12:17:15  Externally logging main query id (for debug purposes): 20260817_121714_53524_f5pkj
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: ROLLBACK
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: Close
INFO - 12:17:15  Sending event: {'category': 'dbt', 'action': 'run_model', 'label': 'da012543-b7d4-4718-9d5b-62
INFO - 12:17:15  1 of 1 OK created sql temp_table model dbt_test_marts.tst_temp_table_materialization_1  [CREAT
INFO - 12:17:15  Finished running node model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1
INFO - 12:17:15  On master: COMMIT
```



Мониторинг и оптимизация запросов

- Базовый мониторинг запросов – в Trino UI
- Дополнительно логируем `query_id` в airflow task, далее находим логи в Grafana

```
INFO - 12:17:15  SQL status: CREATE TABLE (1 rows) in 1.335 seconds
INFO - 12:17:15  Externally logging main query id (for debug purposes): 20260817_121714_53524_f5pkj
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: ROLLBACK
INFO - 12:17:15  On model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1: Close
INFO - 12:17:15  Sending event: {'category': 'dbt', 'action': 'run_model', 'label': 'da012543-b7d4-4718-9d5b-62
INFO - 12:17:15  1 of 1 OK created sql temp_table model dbt_test_marts.tst_temp_table_materialization_1  [CREAT
INFO - 12:17:15  Finished running node model.dbt_monorepo.dlh.dbt_test_marts.tst_temp_table_materialization_1
INFO - 12:17:15  On master: COMMIT
```

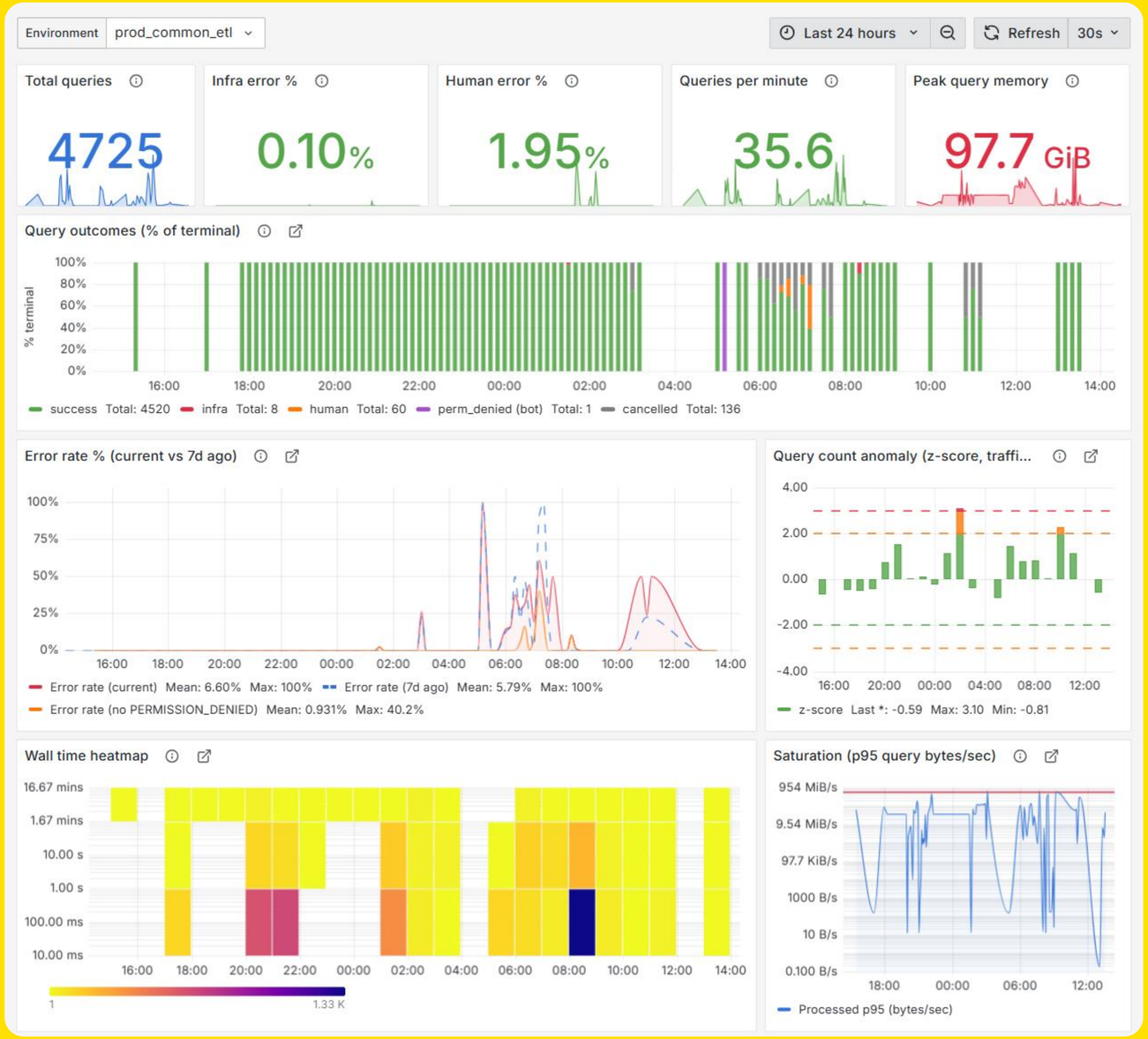


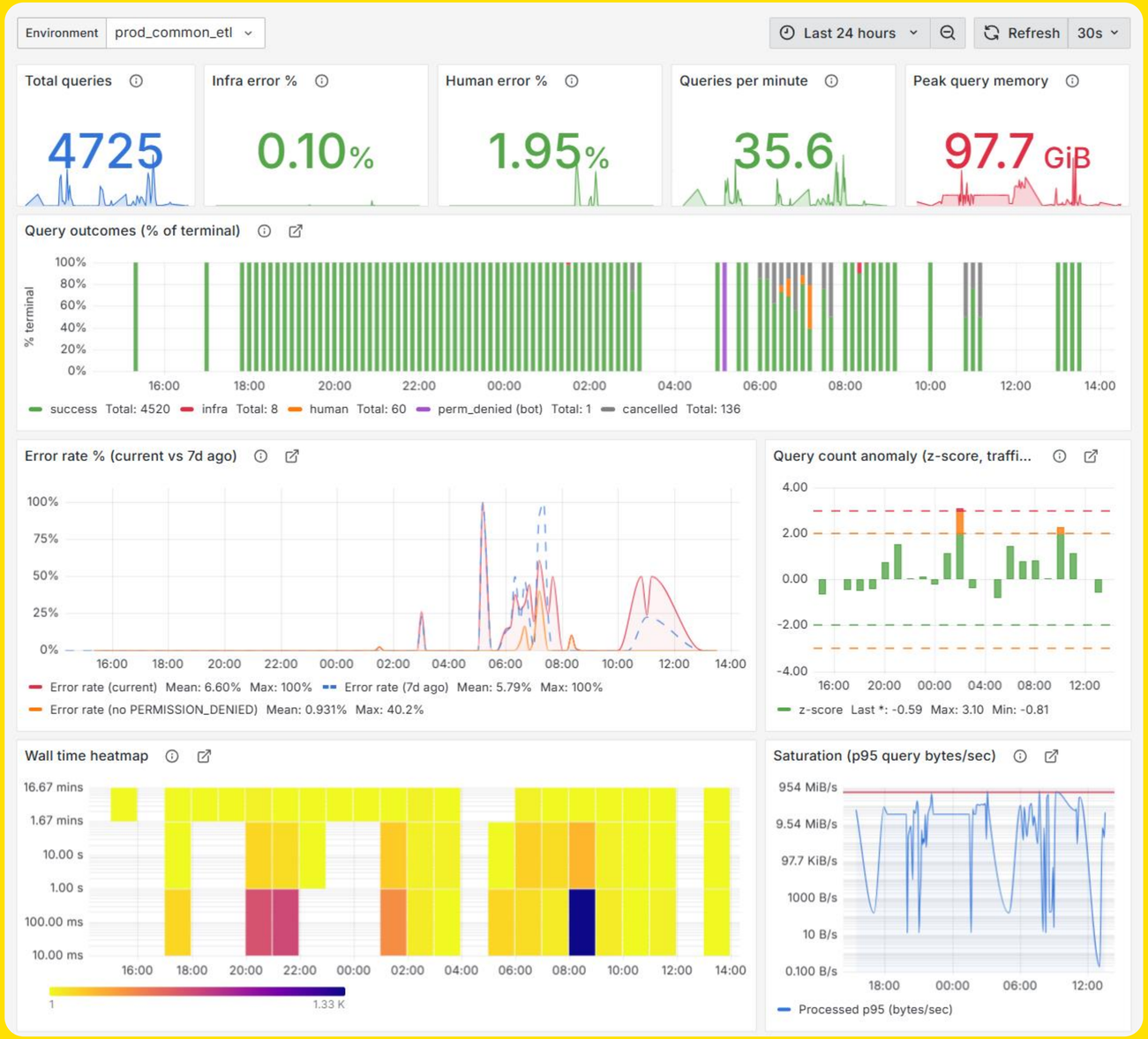
Мониторинг и оптимизация запросов

- Базовый мониторинг запросов – в Trino UI
- Дополнительно логируем `query_id` в airflow task, далее находим логи в Grafana

Мониторинг Trino

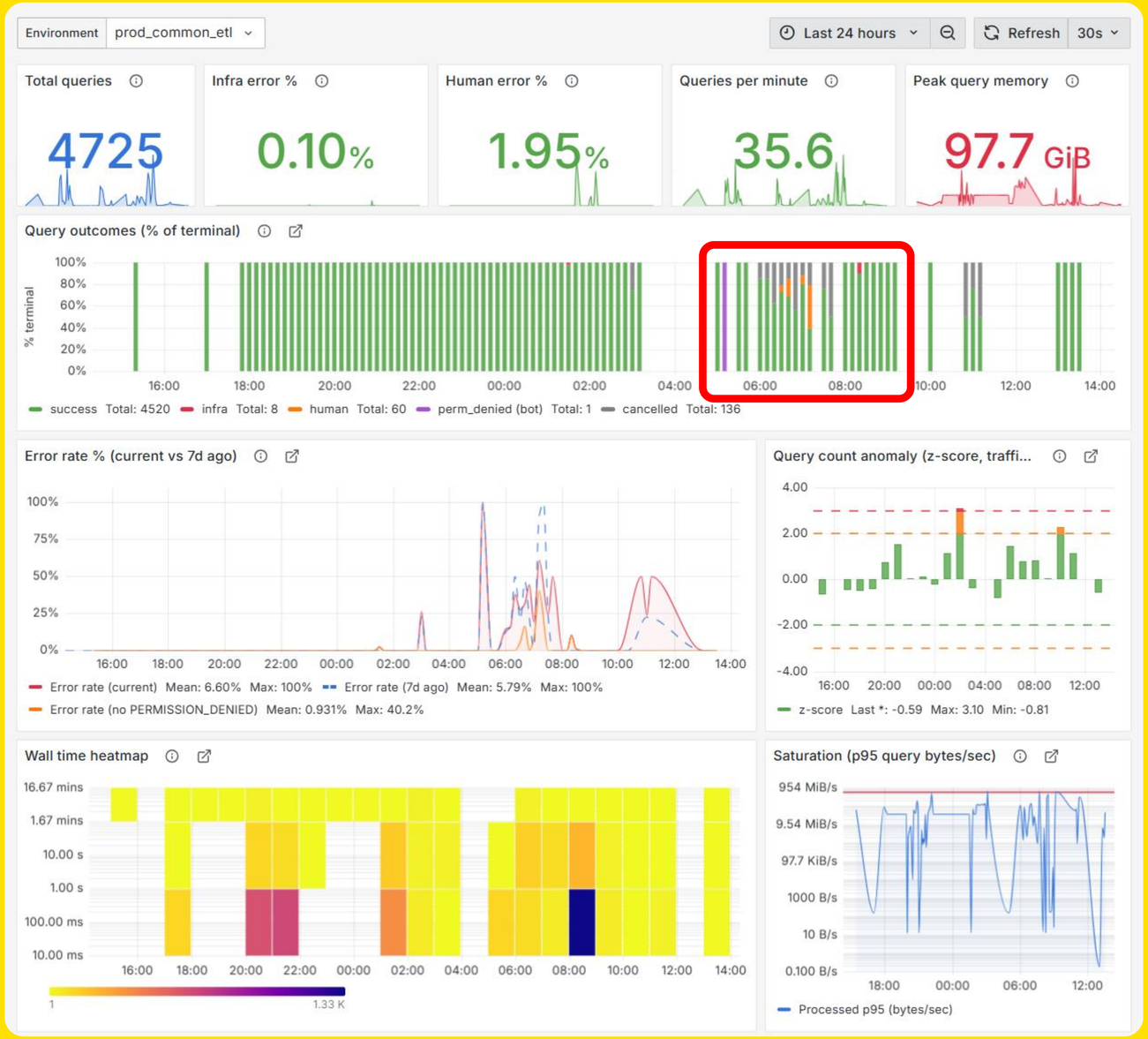
- Дополнительные дашборды со статистикой по каждому кластеру





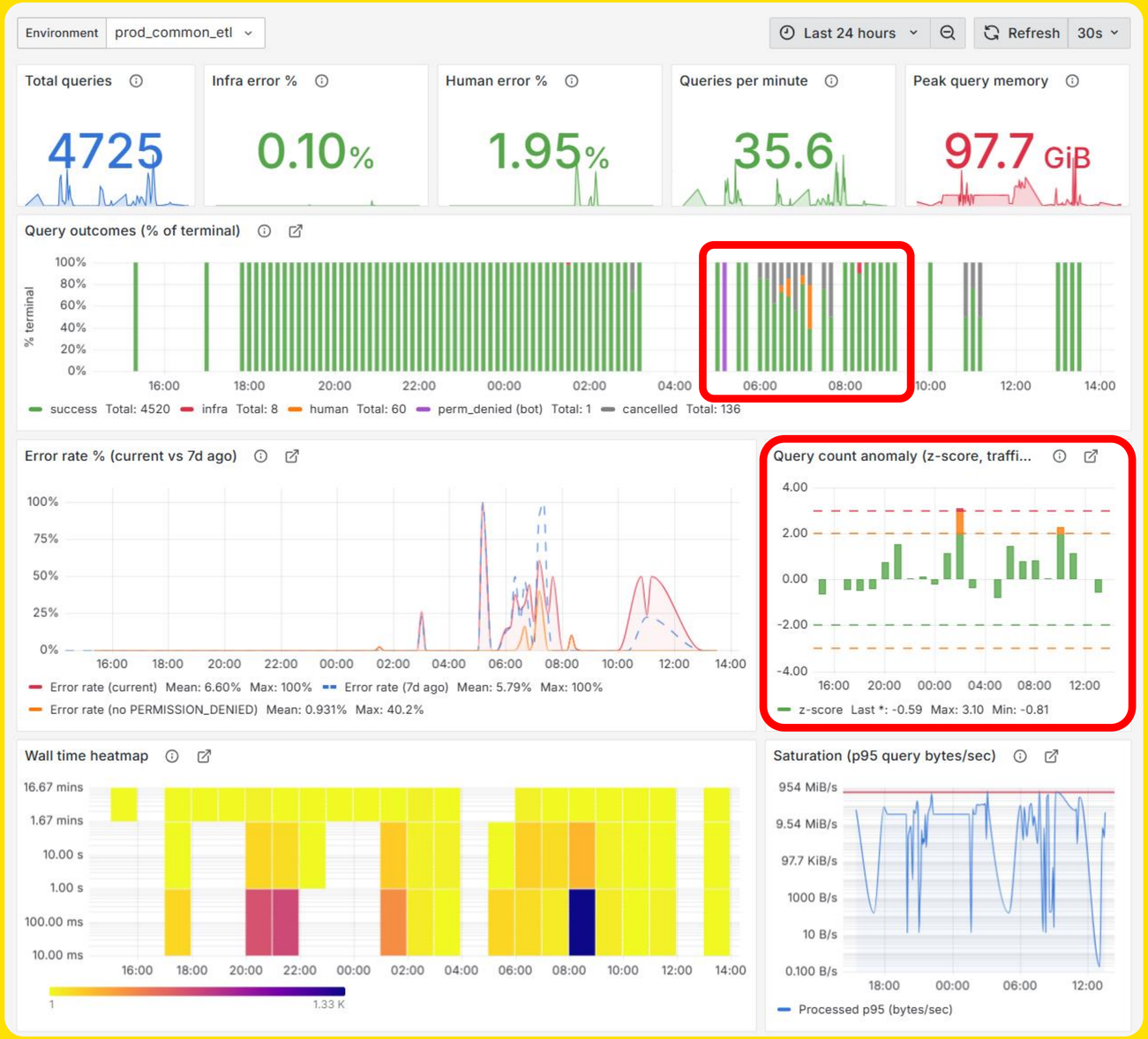
Мониторинг Trino

- Дополнительные дашборды со статистикой по каждому кластеру
- Видим аномалии по нагрузке, проблемных пользователей и объектов



Мониторинг Trino

- Дополнительные дашборды со статистикой по каждому кластеру
- Видим аномалии по нагрузке, проблемных пользователей и объектов



Мониторинг Trino

- Дополнительные дашборды со статистикой по каждому кластеру
- Видим аномалии по нагрузке, проблемных пользователей и объектов

Garbage collector для DLH

Сборка мусора – то, что приходится делать самостоятельно

Сборка мусора – то, что приходится делать самостоятельно

- Базовое обслуживание – через nessie gc

Сборка мусора – то, что приходится делать самостоятельно

- Базовое обслуживание – через nessie gc
- При удалении таблиц (DROP TABLE) – файлы данных/метаданных не удаляются с s3

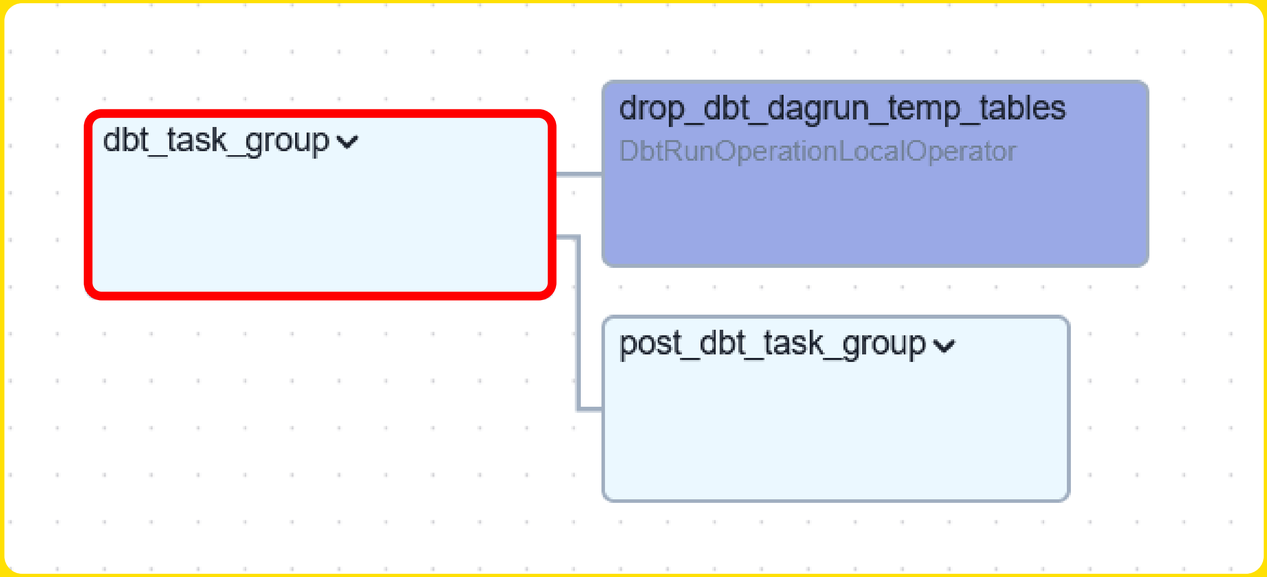
Сборка мусора – то, что приходится делать самостоятельно

- Базовое обслуживание – через nessie gc
- При удалении таблиц (DROP TABLE) – файлы данных/метаданных не удаляются с s3
- Определяем orphaned data files для удаленных таблиц – на них нет ссылок в каталоге – и выставляем Expiration rules, бакеты очищаются асинхронно

Сборка мусора – то, что приходится делать самостоятельно

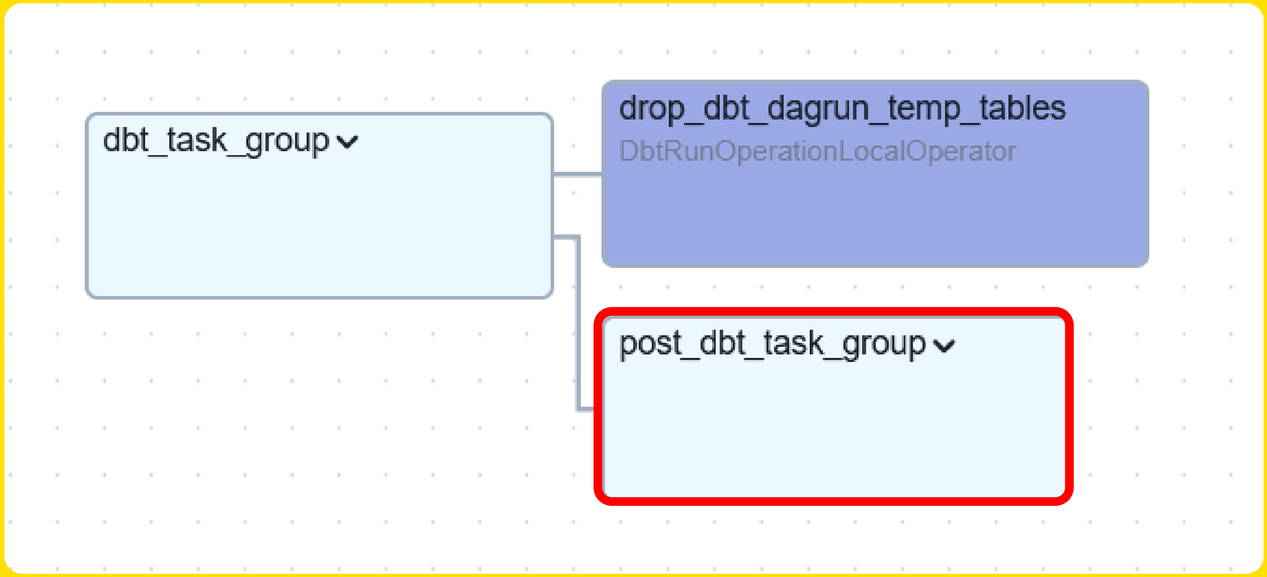
- Базовое обслуживание – через nessie gc
- При удалении таблиц (DROP TABLE) – файлы данных/метаданных не удаляются с s3
- Определяем orphaned data files для удаленных таблиц – на них нет ссылок в каталоге – и выставляем Expiration rules, бакеты очищаются асинхронно
- gc для DLH – отдельный сервис, не влияет на рантайм сборки витрин

Структура Airflow DAG

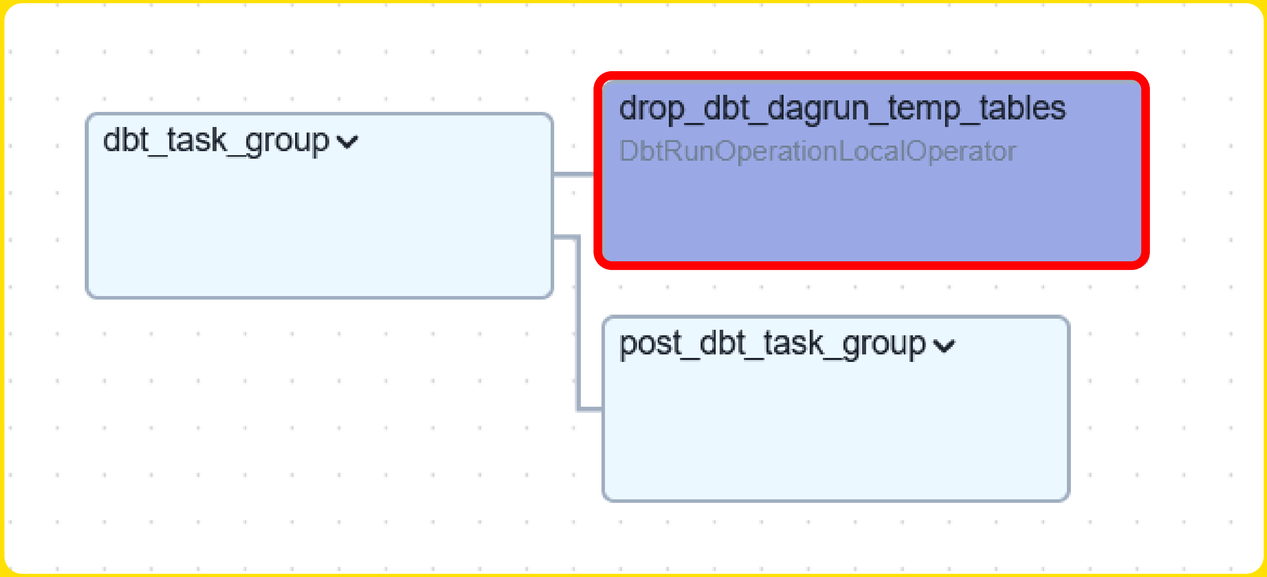


- `dbt_task_group` – исполнение моделей. Одна модель – одна таска

Структура Airflow DAG



- `dbt_task_group` – исполнение моделей. Одна модель – одна таска
- `post_dbt_task_group` – обслуживание каталога и s3



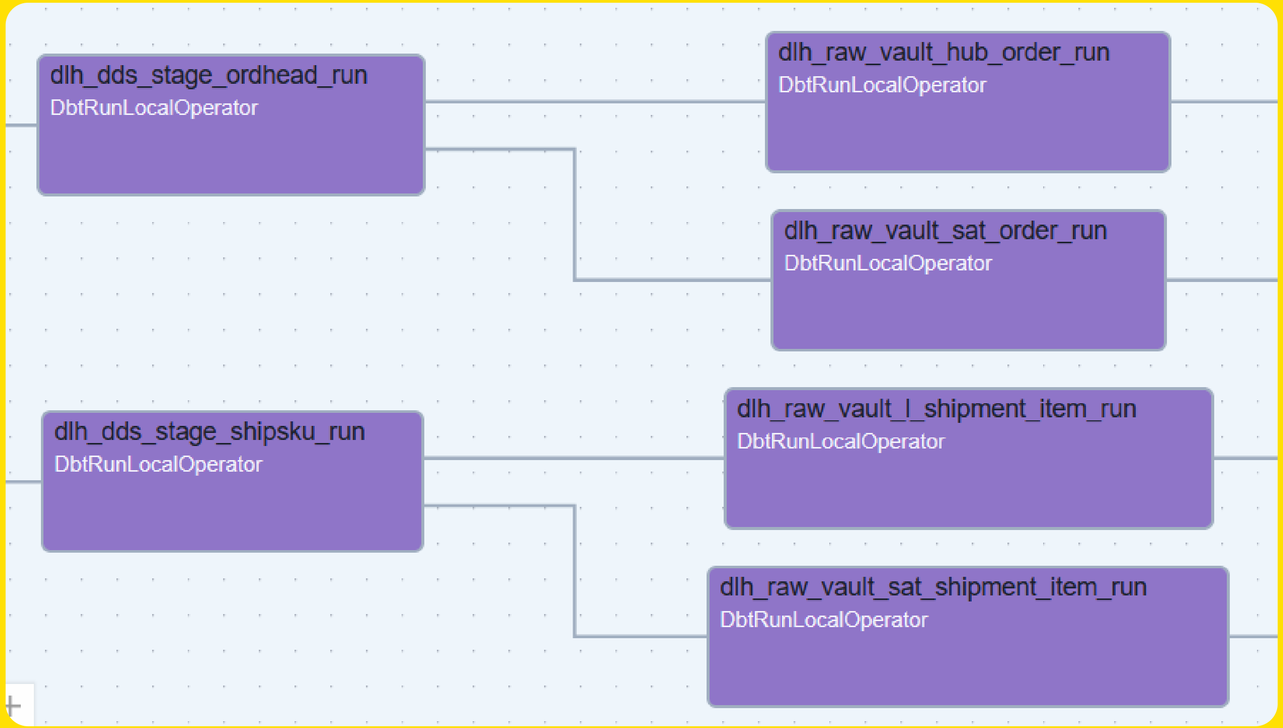
Структура Airflow DAG

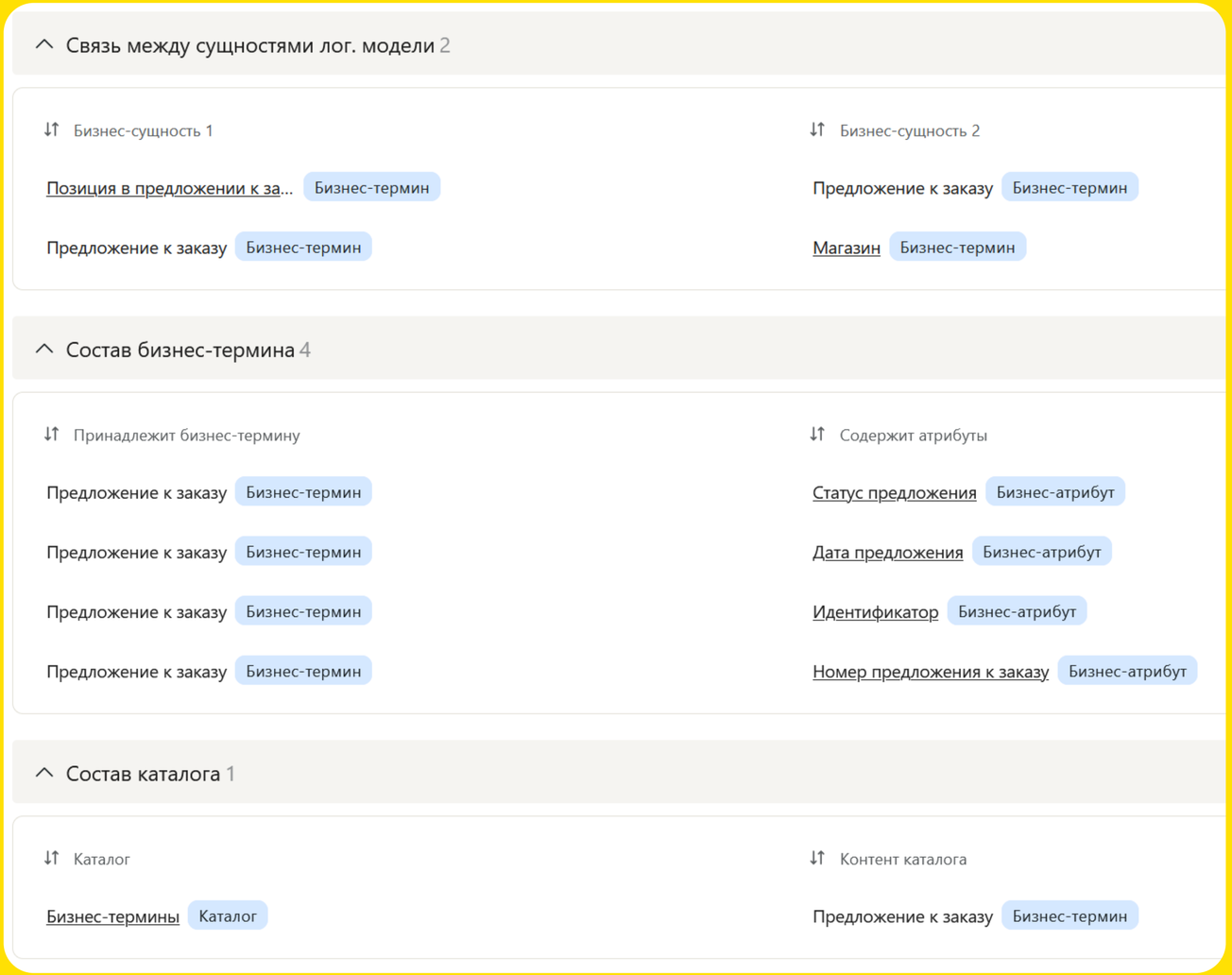
- `dbt_task_group` – исполнение моделей. Одна модель – одна таска
- `post_dbt_task_group` – обслуживание каталога и s3
- `drop_dbt_dagrun_temp_tables` – удаление временных таблиц

Детальный слой в Data Lakehouse

dds слой в DLH

- Модель данных – data vault 2.0 на фреймворке `automate-dv`



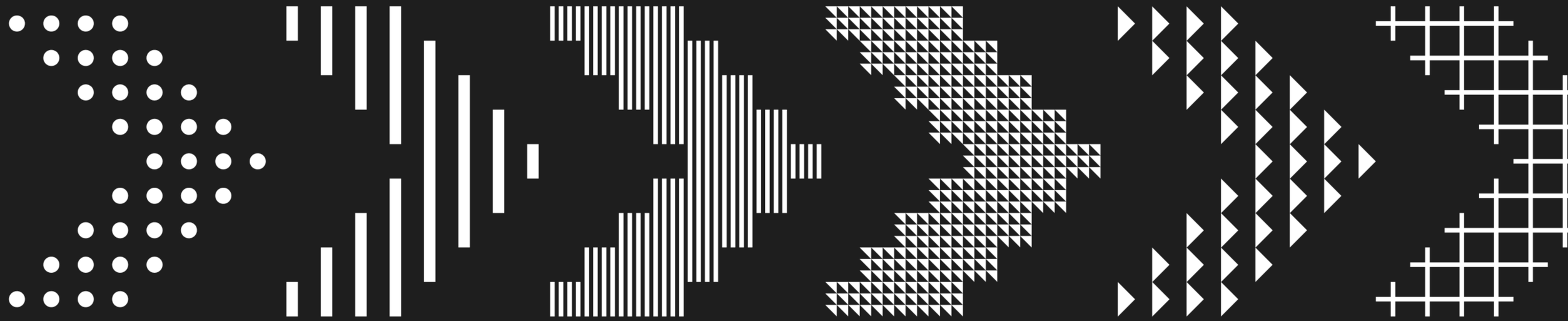


dds слой в DLH

- Модель данных – data vault 2.0 на фреймворке `automate-dv`
- Создаем объекты-хабы для бизнес-сущностей из дата-каталога

лемана тех

Организация процесса миграции команд и витрин



Приоритет миграции – витрины для ключевых отчетов

План миграции сборок витрин

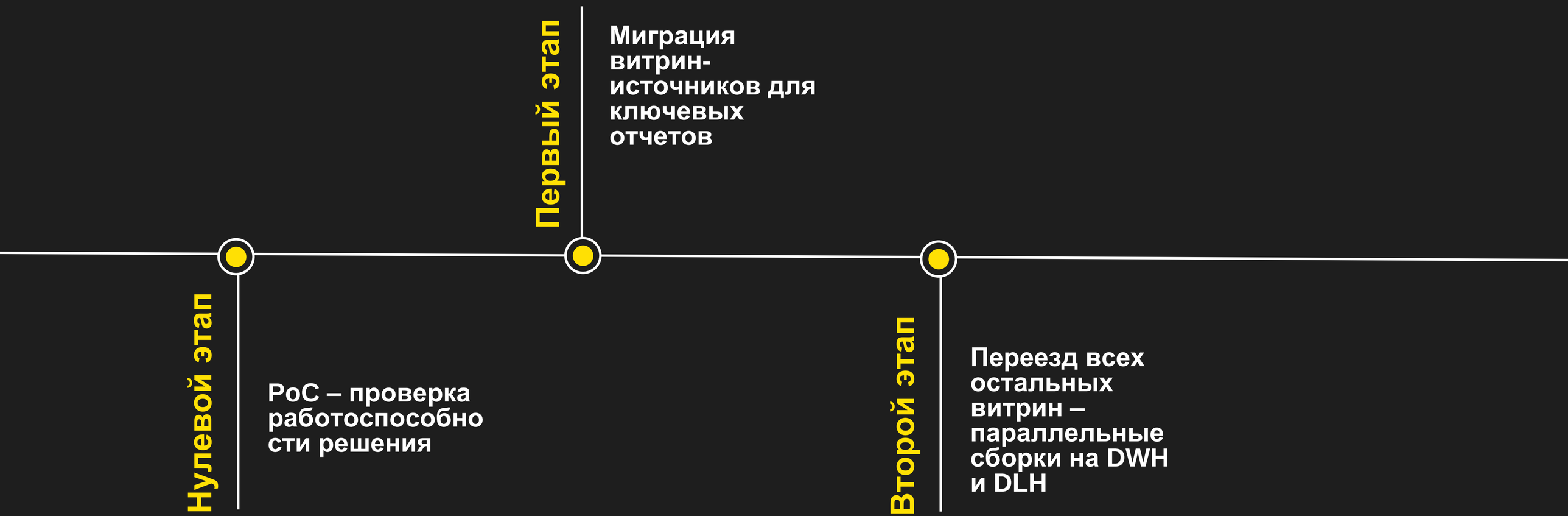
План миграции сборок витрин

Нулевой этап

PoC – проверка работоспособности решения

План миграции сборок витрин

План миграции сборок витрин



План миграции сборок витрин

Нулевой этап

PoC – проверка работоспособности решения

Первый этап

Миграция витрин-источников для ключевых отчетов

Второй этап

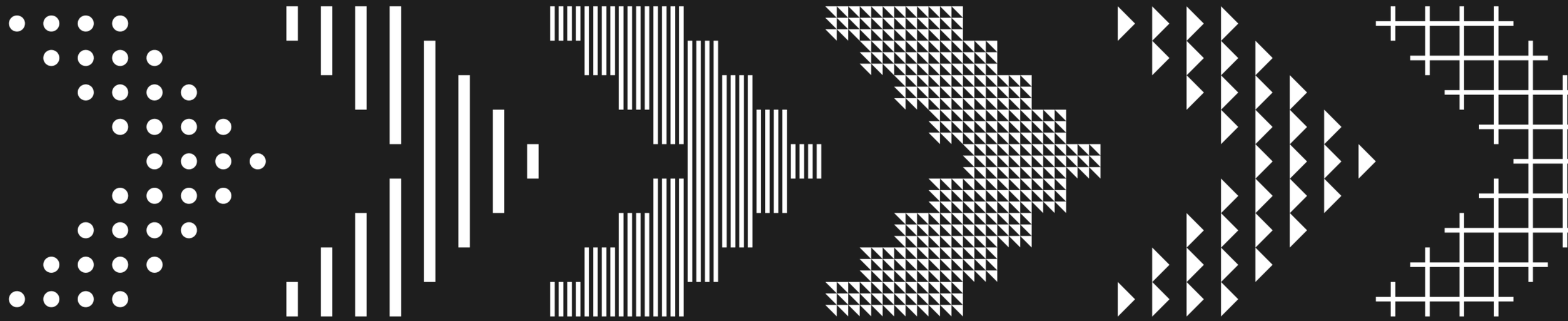
Переезд всех остальных витрин – параллельные сборки на DWH и DLH

Третий этап

Сборки marts слоя остаются только на DLH. Экспорт готовых витрин в DWH для специфичных сценариев

лемана тех

Итоги и выводы про Data Lakehouse



Немного цифр про наш Data Lakehouse

40 / 290

Интеграций источников перенесено в Data Lakehouse

5 ТБ

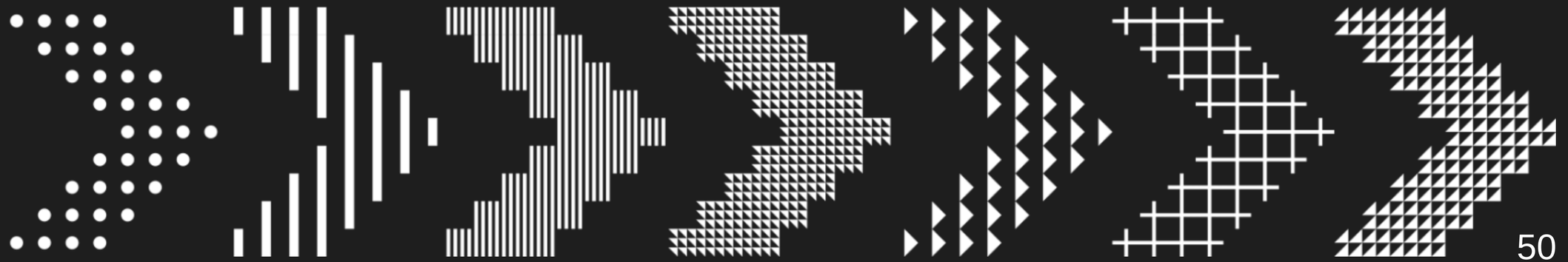
Текущий объем DLH (ods + marts)

7%

На 7% дольше выполняется сборка витрин для критической отчетности по сравнению с dwh

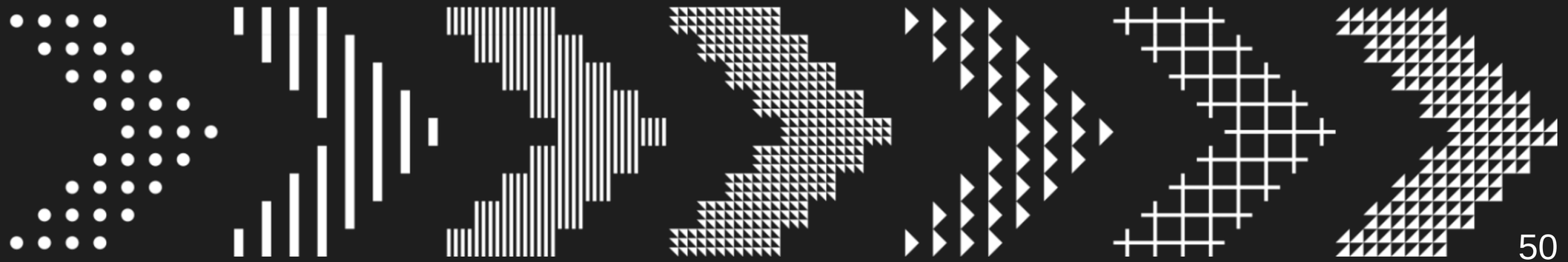
Вопросов пока остается больше, чем ответов

Итоги и выводы



Итоги и выводы

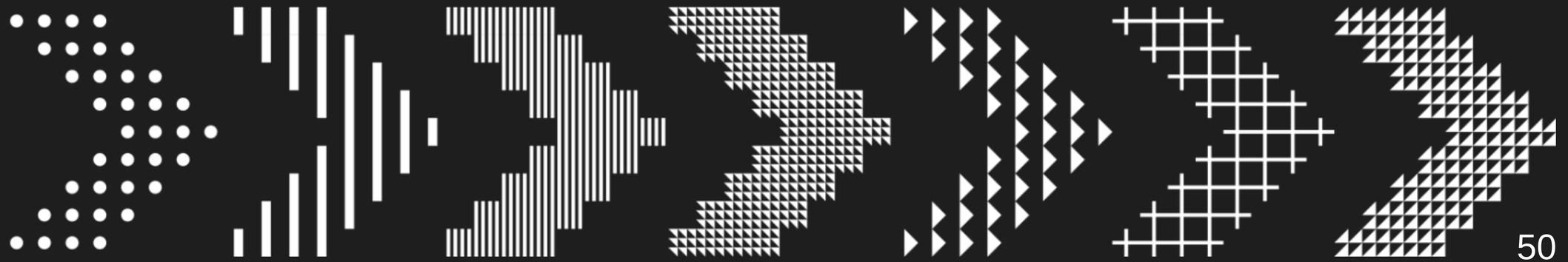
Мы убедились на практике в эффективности подхода с разделением compute/storage, при этом общее время сборки отдельных витрин данных может быть больше, чем в dwh => это должно быть ожидаемым результатом при сравнении dwh и dlh



Итоги и выводы

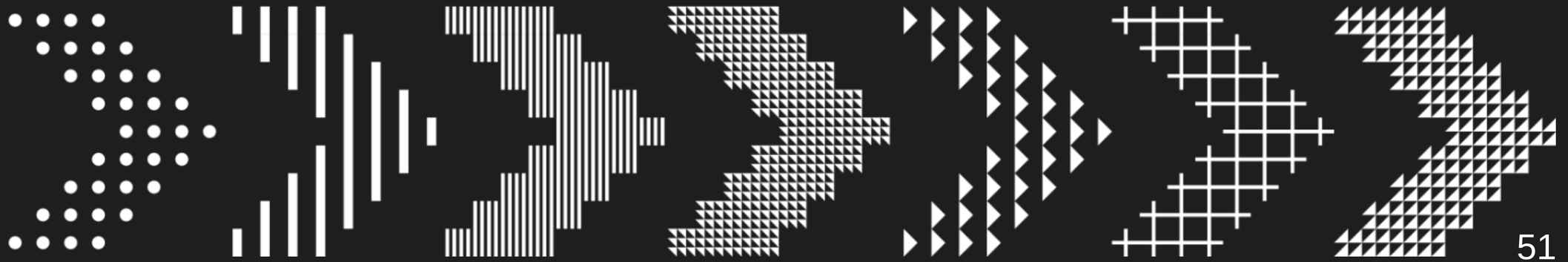
Мы убедились на практике в эффективности подхода с разделением compute/storage, при этом общее время сборки отдельных витрин данных может быть больше, чем в dwh => это должно быть ожидаемым результатом при сравнении dwh и dlh

Мы сохранили привычный сценарий сборки витрин и SQL интерфейс для доступа к данным => можно выбрать альтернативный Trino-like движок для своего DLH, но оставить пользователям такой же универсальный способ работы с данными



Итоги и выводы

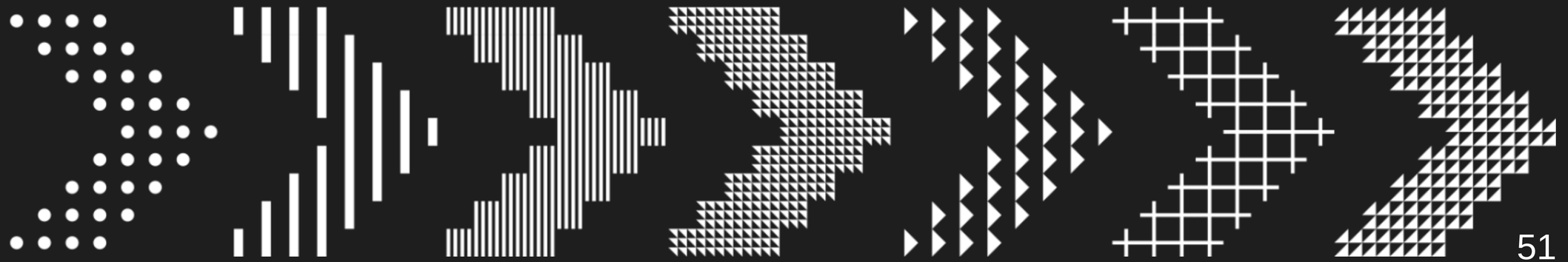
Миграция на DLH не может быть быстрой в т.ч. из-за того, что компоненты DLH не будут идеально работать из коробки и иметь весь необходимый функционал => придется допиливать их под себя или выбирать альтернативные инструменты



Итоги и выводы

Миграция на DLH не может быть быстрой в т.ч. из-за того, что компоненты DLH не будут идеально работать из коробки и иметь весь необходимый функционал => придется допиливать их под себя или выбирать альтернативные инструменты

С увеличением количества пользователей DLH мы ожидаем, что известные уже сейчас узкие места дадут о себе знать: очереди и шедулинг в Airflow, ресурсы и очереди Trino, эффективность Data Vault на dds слое и многое другое => будем конфигурировать кластеры под разные типы нагрузки



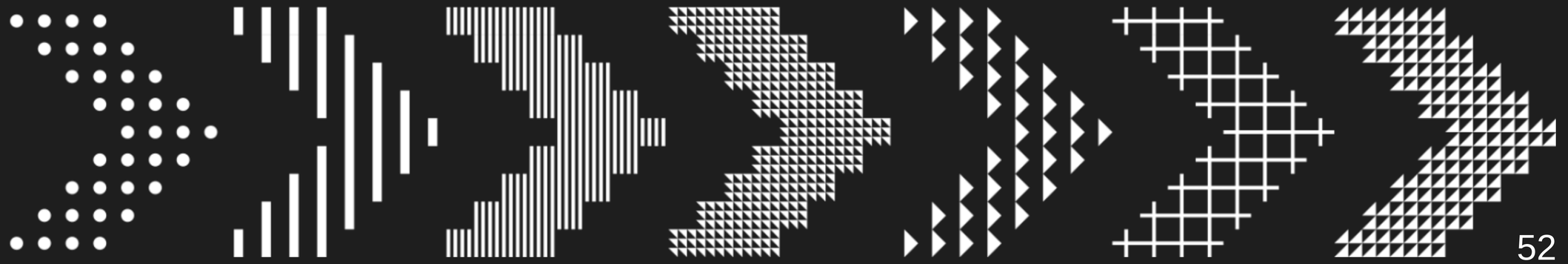
Итоги и выводы

На текущий момент мы не можем полностью уйти с DWH стека, приходится поддерживать зоопарк технологий => это не однозначный недостаток, а работа с рисками – своеобразный High Availability для витрин данных

Итоги и выводы

На текущий момент мы не можем полностью уйти с DWH стека, приходится поддерживать зоопарк технологий => это не однозначный недостаток, а работа с рисками – своеобразный High Availability для витрин данных

Несмотря на стабильную работу инструментов, составляющих DLH стек, мы рассматриваем возможность миграции на другие сервисы или их версии (REST каталог, self-hosted s3, Airflow 3x)



лемана тех

Артемий Наумов

artemiy.naumov@lemanapro.ru

t.me/tema_nmv