



# dbt

Ядро современной платформы данных

Ермаков Евгений,  
Head of Data Platform





Выпускник МГТУ им Н.Э.Баумана  
к.т.н. в области СУБД

Более 10 лет опыта работы с системами  
обработки данных в различных сферах:  
финансы, e-commerce, IT

## Евгений Ермаков

Head of Data Platform at Toloka

 [jkermakov@toloka.ai](mailto:jkermakov@toloka.ai)

# Три кита и четыре столпа дата платформы

Что?



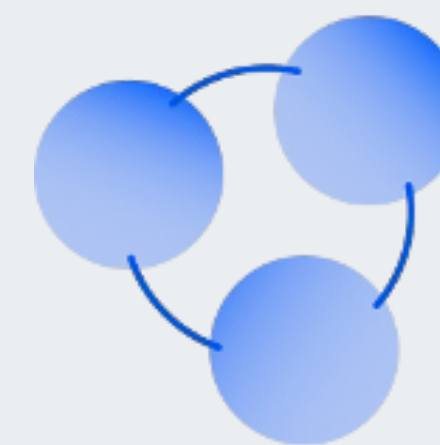
Архитектура  
=  
Data Lakehouse

Почему?



Процессы  
=  
Data mesh

Как?



Технологии  
=  
Modern Data Stack



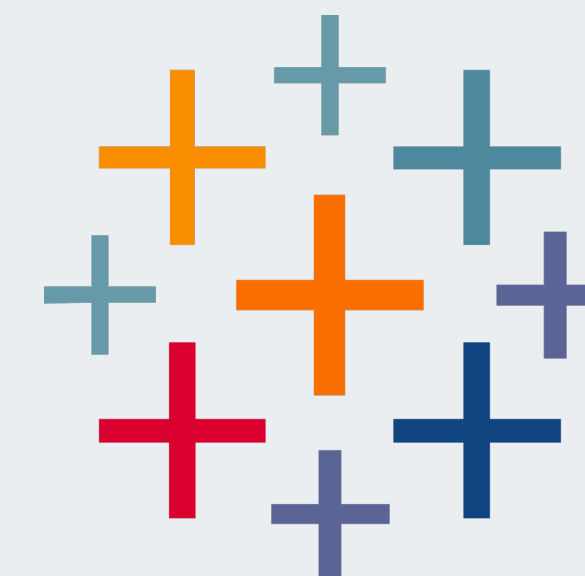
databricks



dbt



airflow



tableau

# Три кита и четыре столпа дата платформы

Что?



Архитектура

=

Data Lakehouse

Почему?

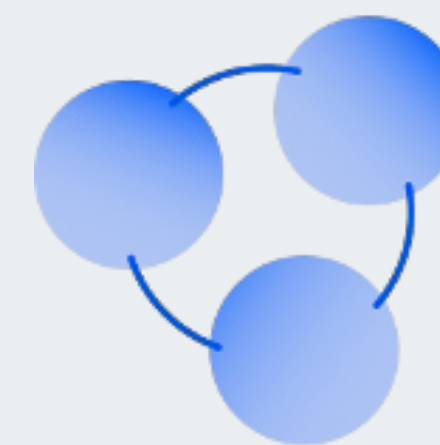


Процессы

=

Data mesh

Как?



Технологии

=

Modern Data Stack



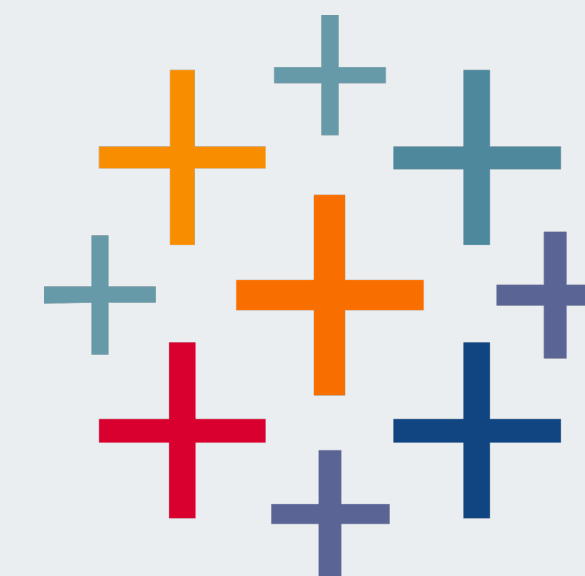
databricks



dbt

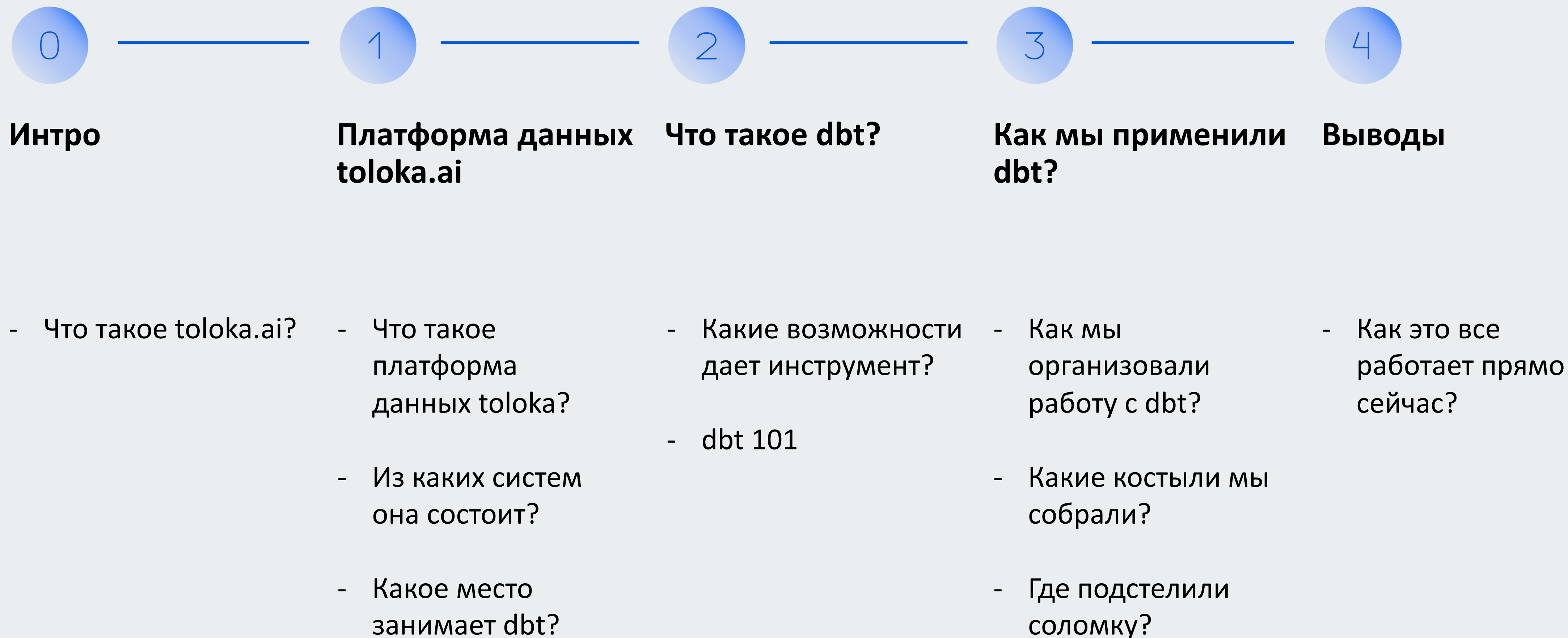


airflow



tableau

# План доклада

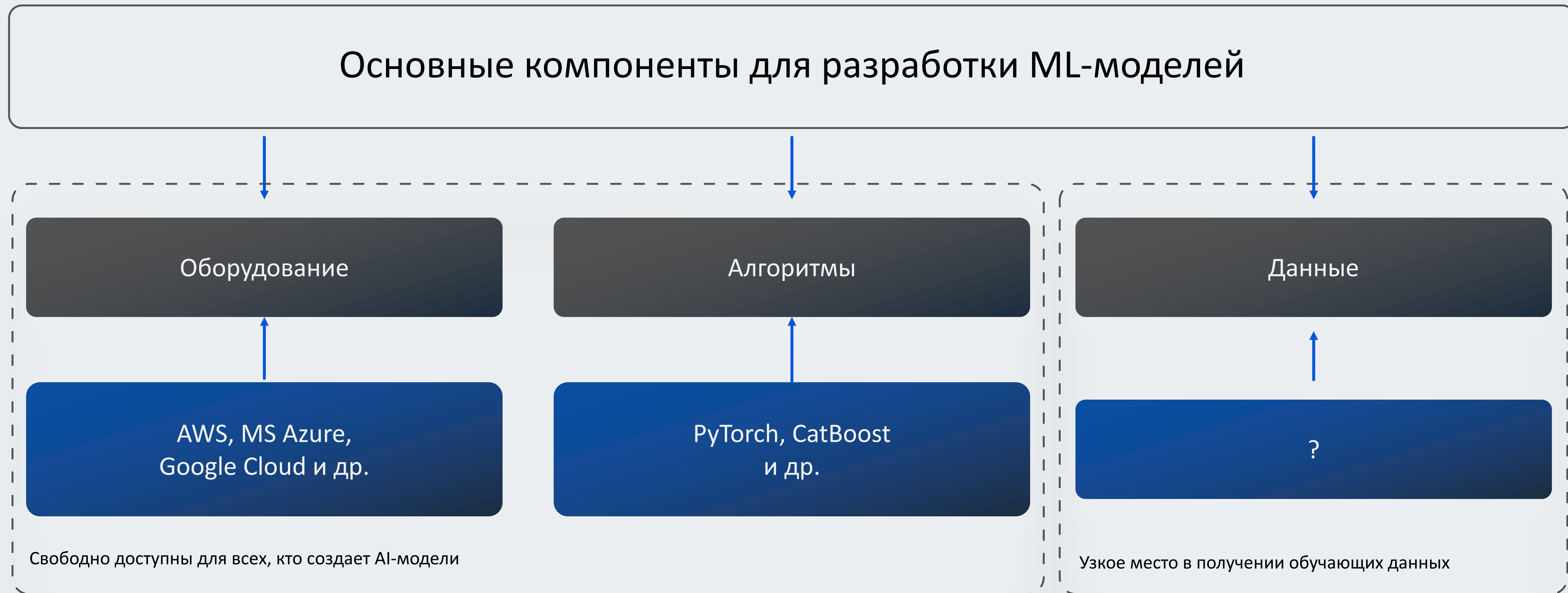


0. Интро

# Что такое toloka.ai?

# Датацентричный подход к ML

Каждый имеет доступ к оборудованию и алгоритмам.  
Данные — ключ к получению конкурентного преимущества.



# Обучающие данные – ядро ML

В среднем 80% времени\*, затрачиваемого на AI-разработку, уходит на обработку обучающих данных

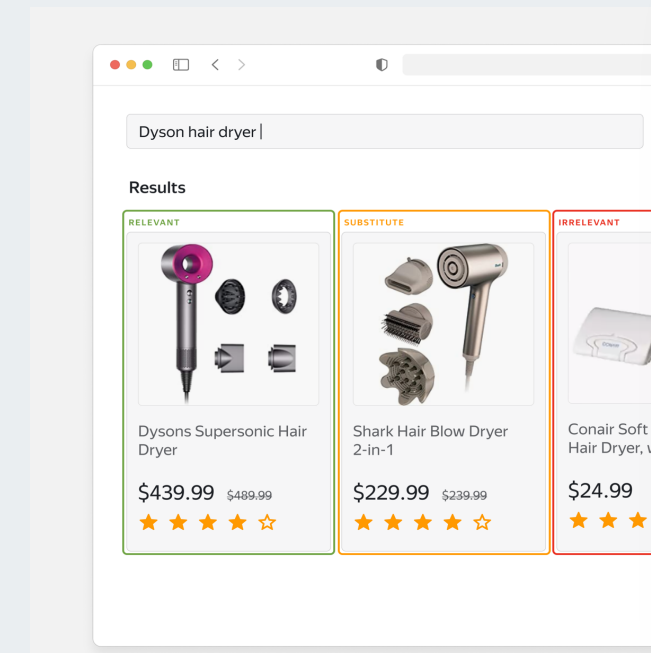
Точность моделей зависит от качества данных



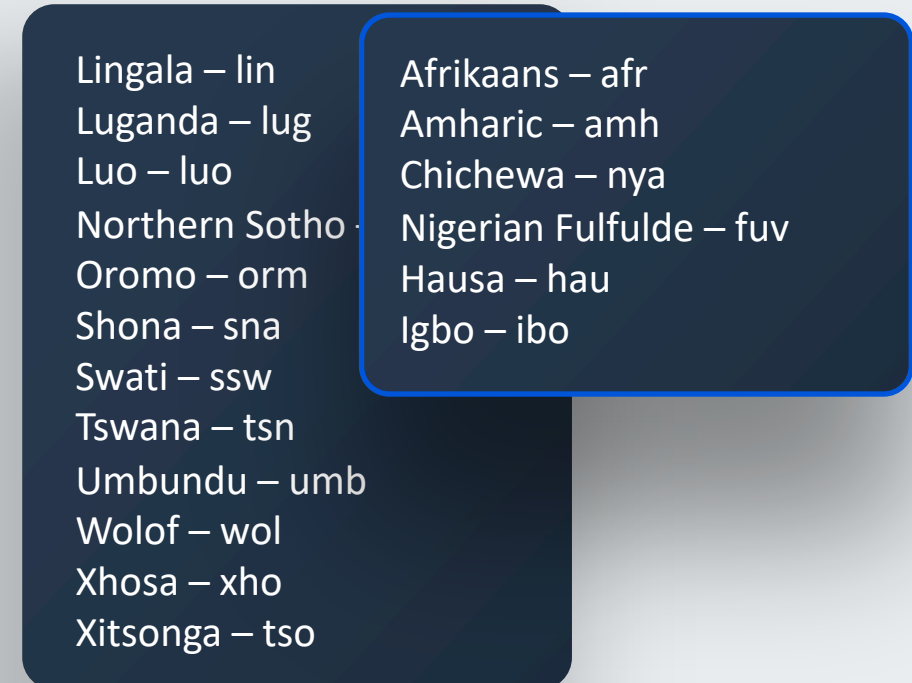
Беспилотные автомобили



Речевые технологии

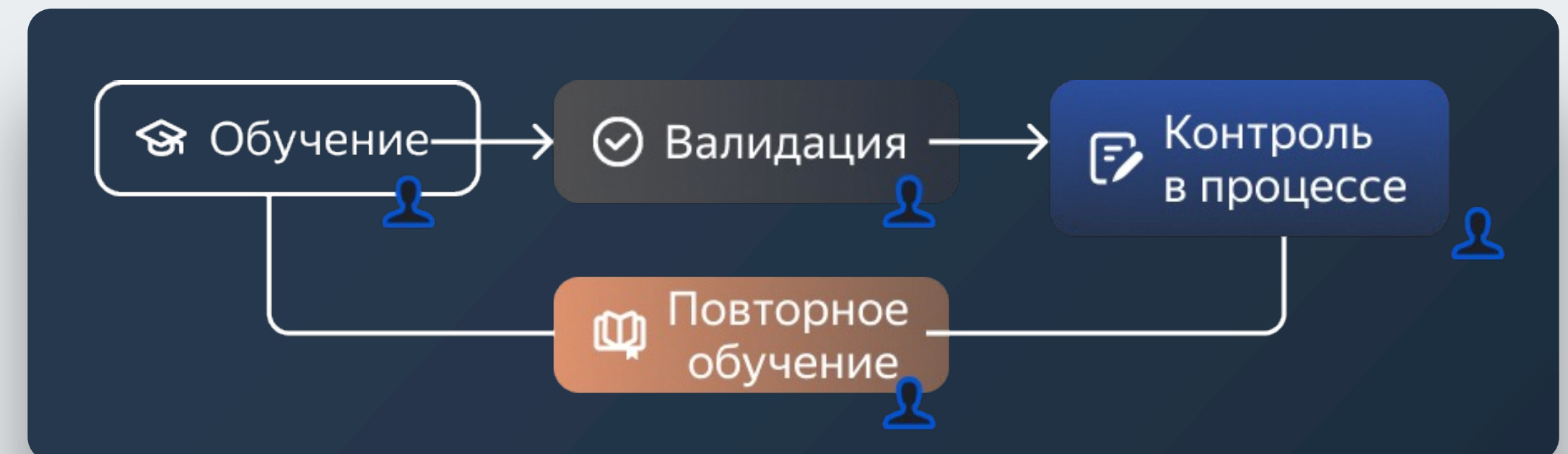


Технологии поиска



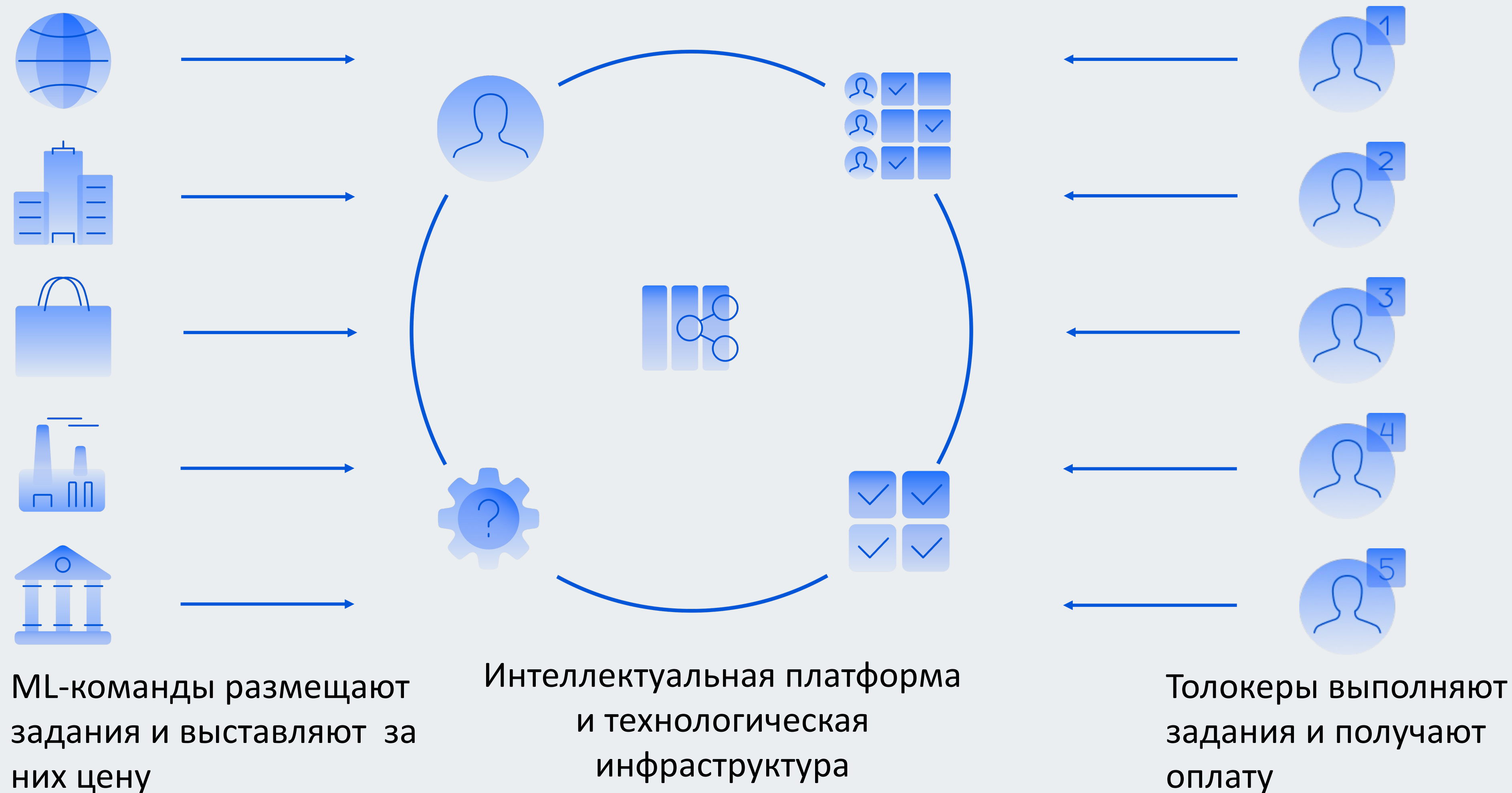
Машинный перевод

Потребность обработанных вручную больших массивов данных возникает на каждом этапе жизненного цикла ML-разработки и внедрения





# Toloka platform



1. Платформа данных

# Из чего состоит toloka data platform?

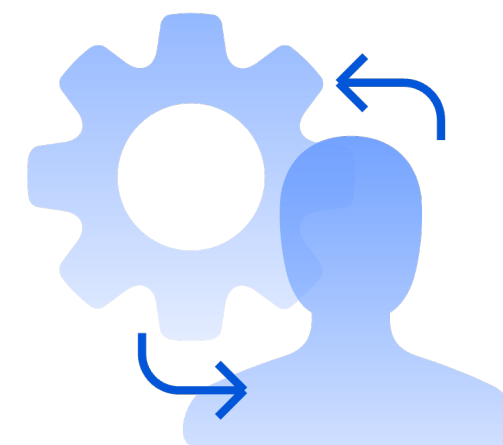
# Характеристики платформы



Data / Daily delta  
**500 Tb / 270 Gb**



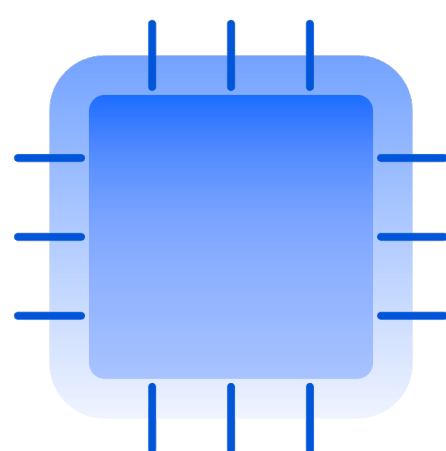
Data models  
**over 350**



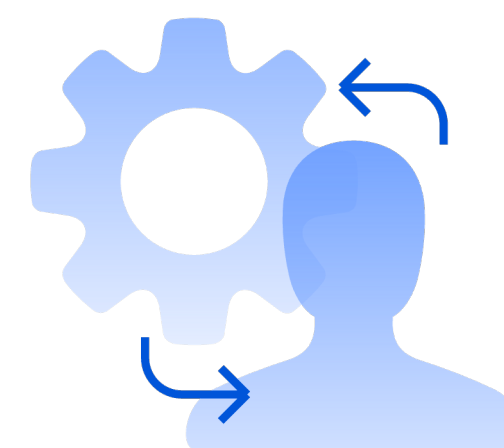
Internal Users  
**>100**



Airflow dags  
**over 100**



Tasks per day  
**over 5000**



Data Domains  
**42**

# Good ol' Yandex stack

## Ingestion

Event Streaming

 Logbroker

Connectors / EL

 Logfeller

## Data Governance

Data Quality

## Transformation

Streaming Processing

Hurma 

Batch Processing

 YT

## Storage

 YT  MDS

## Orchestration

 Nirvana

 Reactor

## Consumption

Querying

 YQL

 ClickHouse over YT

Business Intelligence

 DataLens


Analytical tools

 YQL

 jupyter

Juggler 

Monitoring

 Solomon

# Три кита дата платформы

Что?



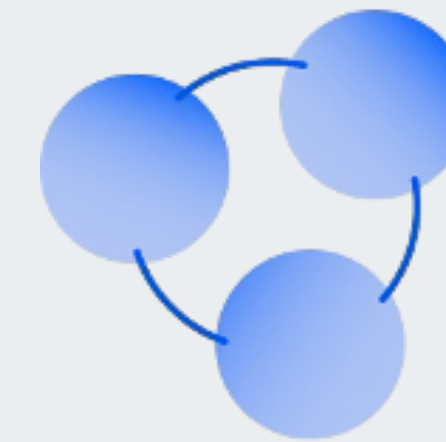
Архитектура  
=  
Data Lakehouse

Почему?



Процессы  
=  
Data mesh

Как?



Технологии  
=  
Modern Data Stack

# Четыре столпа дата платформы

Что?



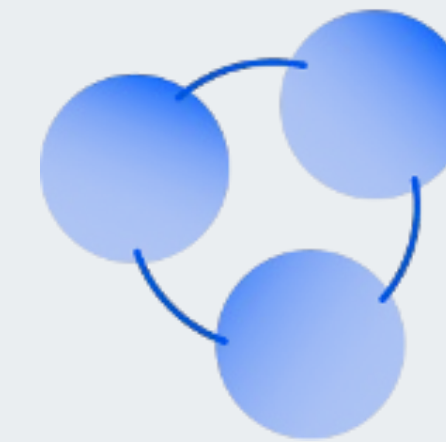
Архитектура  
=  
Data Lakehouse

Почему?



Процессы  
=  
Data mesh

Как?



Технологии  
=  
Modern Data Stack



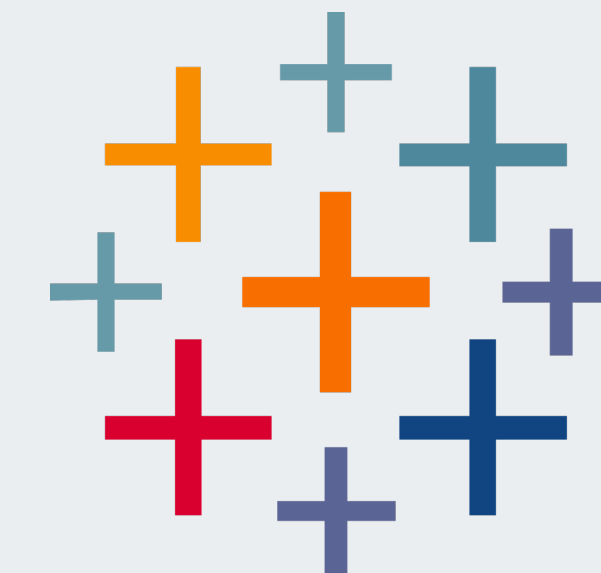
databricks



dbt



airflow



tableau

# Toloka Data Platform

## Ingestion

### Event Streaming



### Connectors / EL



## Data Governance

### Data Quality



## Transformation

### Streaming Processing



### Batch Processing



## Storage

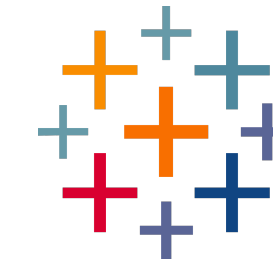


## Consumption

### Quering



### Business Intelligence



### Analytical tools



## Orchestration



### Monitoring



# Dbt at Toloka Data Platform

## Ingestion

Event Streaming



Connectors / EL



## Data Governance

Data Quality



## Transformation

Streaming Processing



Batch Processing



## Storage

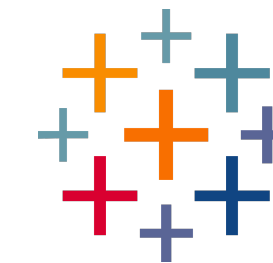


## Consumption

Quering



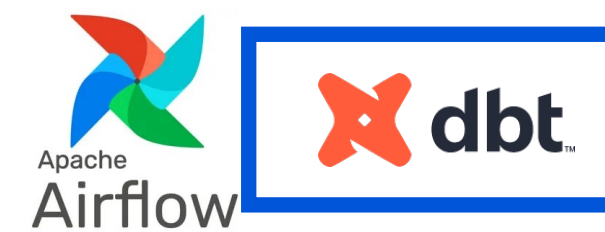
Business Intelligence



Analytical tools



## Orchestration



Data Catalog \ Data Observability



Monitoring





# Transformation

Удобный механизм работы для разных специалистов

- › Аналитики: SQL-only или low-code
- › Разработчики: императивный код
- › DE: поддержка миграций и data lineage



- › Python\Scala Spark
- › Все возможности императивного кода

Developers

Data engineers



- › SQL + pyspark
- › Поддержка миграций
- › Соккрытие сложности вставки данных

Data engineers

Analysts

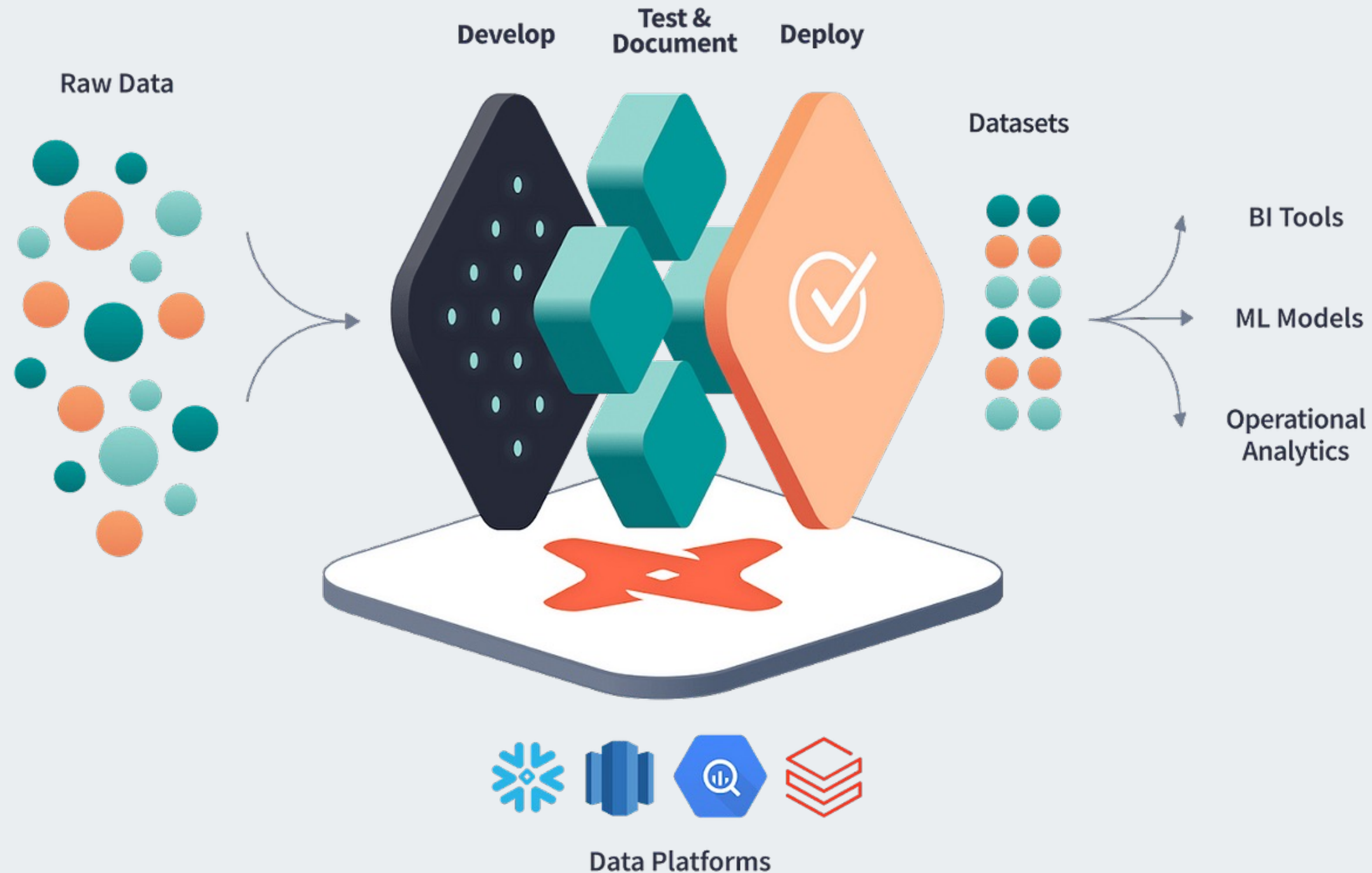
2. dbt

# Что такое dbt?

# Data Transformation Tool

DBT (Data Building Tool) — это инструмент командной строки, который позволяет аналитикам и инженерам данных преобразовывать данные в своих хранилищах, используя операторы выбора.

DBT отвечает за T (трансформацию) в ETL-процессе, но не выполняет операции извлечения (E) и загрузки (L).

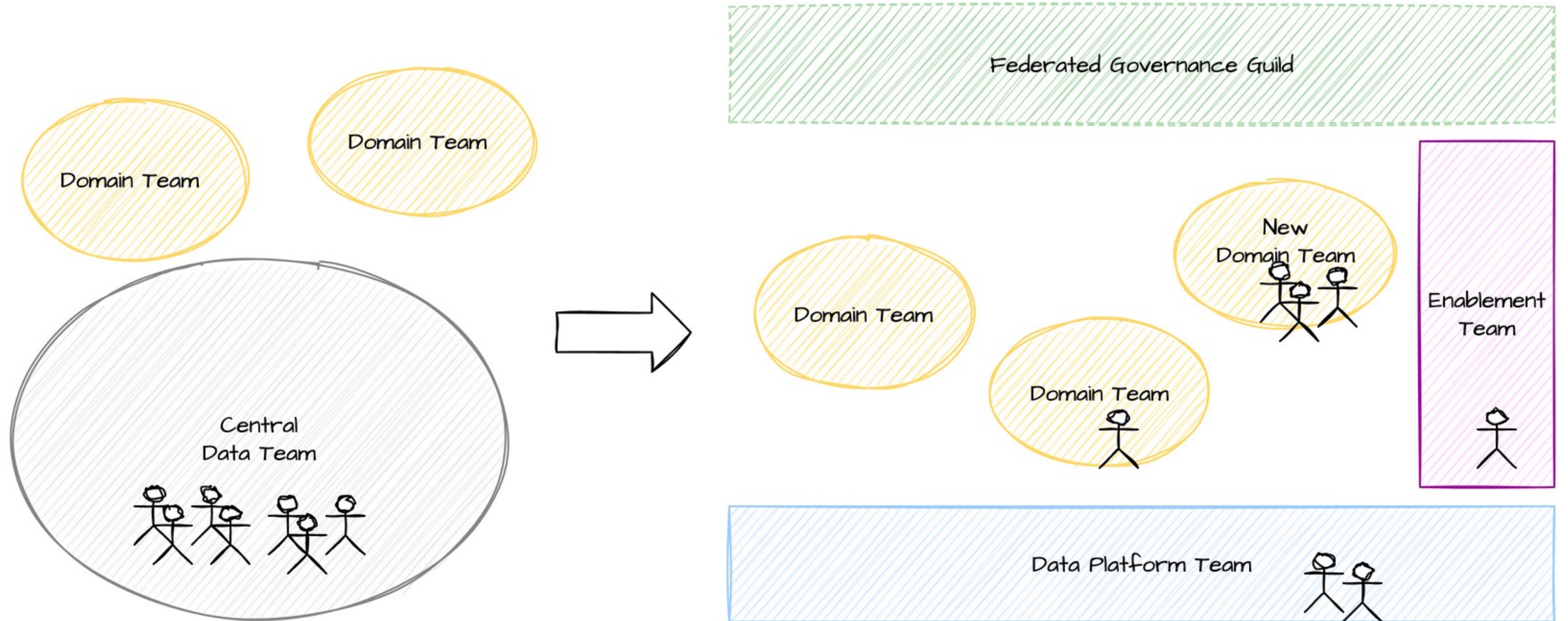


# Analytics Engineer

Data Engineer	Analytics Engineer	Data Analyst
<ul style="list-style-type: none"><li>• Build custom data integrations</li><li>• Manage overall pipeline orchestration</li><li>• Develop &amp; deploy machine learning endpoints</li><li>• Build and maintain the data platform</li><li>• Data warehouse performance optimizations</li></ul>	<ul style="list-style-type: none"><li>• Provide clean, transformed data ready for analysis</li><li>• Apply software engineering best practices to analytics code (ex: version control, testing, continuous integration)</li><li>• Maintain data documentation &amp; definitions</li><li>• Train business users on how to use data visualization tools</li></ul>	<ul style="list-style-type: none"><li>• Deep insights work (ex: why did churn spike last month? what are the best acquisition channels?)</li><li>• Work with business users to understand data requirements</li><li>• Build critical dashboards</li><li>• Forecasting</li></ul>

# Data mesh

## Data Team's Journey



[datamesh-architecture.com](https://www.datamesh-architecture.com)

**Из чего состоит dbt?**

# dbt model

Основной “кирпичик” dbt-проекта.

Состоит из двух файлов: .sql/.py с описанием логики и .yml с описанием конфигурации.

Особенности:

- › select запрос на формирование объекта
- › jinja шаблонизация
- › другие модели используются через макрос ref() или source()

```
with customers as (  
    select *  
    from {{ ref('stg_eltool_customers') }}  
)  
state as (  
    select *  
    from {{ ref('stg_eltool_state') }}  
)  
select c.customer_id,  
       c.zipcode,  
       c.city,  
       c.state_code,  
       s.state_name,  
       c.datetime_created,  
       c.datetime_updated,  
       c.dbt_valid_from::TIMESTAMP as valid_from,  
       CASE  
           WHEN c.dbt_valid_to IS NULL THEN '9999-12-31'::TIMESTAMP  
           ELSE c.dbt_valid_to::TIMESTAMP  
       END as valid_to  
from customers c  
     join state s on c.state_code = s.state_code
```

# dbt model

Основной “кирпичик” dbt-проекта.

Состоит из двух файлов: .sql/.py с описанием логики и .yml с описанием конфигурации.

Особенности:

- › Возвращаем dataframe с рассчитанными данными
- › Все возможности pyspark
- › другие модели используются через специальные команды `ref()` или `source()`

```
import pyspark.sql.functions as F

def model(dbt, session):
    dbt.config(materialized = "incremental")
    df = dbt.ref("upstream_table")

    if dbt.is_incremental:

        # only new rows compared to max in current table
        max_from_this = f"select max(updated_at) from {dbt.this}"
        df = df.filter(df.updated_at >= session.sql(max_from_this).collect()[0][0])

        # or only rows from the past 3 days
        df = df.filter(df.updated_at >= F.date_add(F.current_timestamp(), F.lit(-3)))

    ...

    return df
```



# dbt materialization

Материализация – стратегия, по которой dbt выполнит sql\py запрос в СУБД.

Виды материализаций:

- › table (sql\py)
- › view (sql)
- › incremental (sql\py)
  - › append
  - › merge
  - › insert\_overwrite
  - › delete+insert
- › ephemeral (sql)
- › materialized view (sql)

models/events/stg\_event\_log.sql

```
{{ config(materialized='table', sort='timestamp', dist='user_id') }}  
  
select *  
from ...
```

models/my\_model.sql

```
{{  
  config(  
    materialized='incremental',  
    unique_key='date_day',  
    incremental_strategy='delete+insert',  
    ...  
  )  
}}  
  
select ...
```

# Настройки проекта

dbt\_project.yml – центральный файл с настройками проекта

- › пути к моделям
- › общие настройки моделей
- › профиль по-умолчанию

profiles.yml – файл с настройками подключения

- › Может быть неограниченное количество
- › Обычно зависит от окружения (prod\test\dev) и от инстанса СУБД















```
<> dbt_project.yml
```

```
# Example dbt_project.yml file
name: 'jaffle_shop'
profile: 'jaffle_shop'
...
```

```
<> ~/.dbt/profiles.yml
```

```
# example profiles.yml file
jaffle_shop:
  target: dev
  outputs:
    dev:
      type: postgres
      host: localhost
      user: alice
      password: <password>
      port: 5432
      dbname: jaffle_shop
      schema: dbt_alice
      threads: 4
```

# Поддерживаемые платформы

 <b>AlloyDB</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.1	 <b>BigQuery</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.4	 <b>Databricks</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.2	 <b>Dremio*</b> ✗ <a href="#">Install using the CLI</a> pypi package 1.5.0
 <b>Postgres</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.1	 <b>Redshift</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.1	 <b>Snowflake</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.2	 <b>Spark</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.0
 <b>Starburst/Trino</b> ✗ <a href="#">Set up in dbt Cloud</a> ✗ <a href="#">Install using the CLI</a> pypi package 1.6.1	 <b>Fabric Synapse*</b> ✗ <a href="#">Install using the CLI</a> pypi package 1.4.0rc3  <a href="#">Verification in progress</a>	 <b>Azure Synapse*</b> ✗ <a href="#">Install using the CLI</a> pypi package 1.3.2  <a href="#">Verification in progress</a>	 <b>Teradata*</b> ✗ <a href="#">Install using the CLI</a> pypi package 1.5.4

**Как запустить dbt?**

# Команды dbt

## › dbt build

Собирает все зависимости между моделями и компилирует sql\py запросы. Результат пишется в manifest.json

```
$ dbt build
Running with dbt=0.21.0-b2
Found 1 model, 4 tests, 1 snapshot, 1 analysis, 341 macros, 0 operations, 1 seed file, 2 sources, 2 exposures

18:49:43 | Concurrency: 1 threads (target='dev')
18:49:43 |
18:49:43 | 1 of 7 START seed file dbt_jcohen.my_seed..... [RUN]
18:49:43 | 1 of 7 OK loaded seed file dbt_jcohen.my_seed..... [INSERT 2 in 0.09s]
18:49:43 | 2 of 7 START view model dbt_jcohen.my_model..... [RUN]
18:49:43 | 2 of 7 OK created view model dbt_jcohen.my_model..... [CREATE VIEW in 0.12s]
18:49:43 | 3 of 7 START test not_null_my_seed_id..... [RUN]
18:49:43 | 3 of 7 PASS not_null_my_seed_id..... [PASS in 0.05s]
18:49:43 | 4 of 7 START test unique_my_seed_id..... [RUN]
18:49:43 | 4 of 7 PASS unique_my_seed_id..... [PASS in 0.03s]
18:49:43 | 5 of 7 START snapshot snapshots.my_snapshot..... [RUN]
18:49:43 | 5 of 7 OK snapshotted snapshots.my_snapshot..... [INSERT 0 5 in 0.27s]
18:49:43 | 6 of 7 START test not_null_my_model_id..... [RUN]
18:49:43 | 6 of 7 PASS not_null_my_model_id..... [PASS in 0.03s]
18:49:43 | 7 of 7 START test unique_my_model_id..... [RUN]
18:49:43 | 7 of 7 PASS unique_my_model_id..... [PASS in 0.02s]
18:49:43 |
18:49:43 | Finished running 1 seed, 1 view model, 4 tests, 1 snapshot in 1.01s.

Completed successfully

Done. PASS=7 WARN=0 ERROR=0 SKIP=0 TOTAL=7
```

# Команды dbt

› dbt run

Запускает скомпилированный sql\py запрос в зависимости от переданных параметров.

Запуск модели по имени\тегу

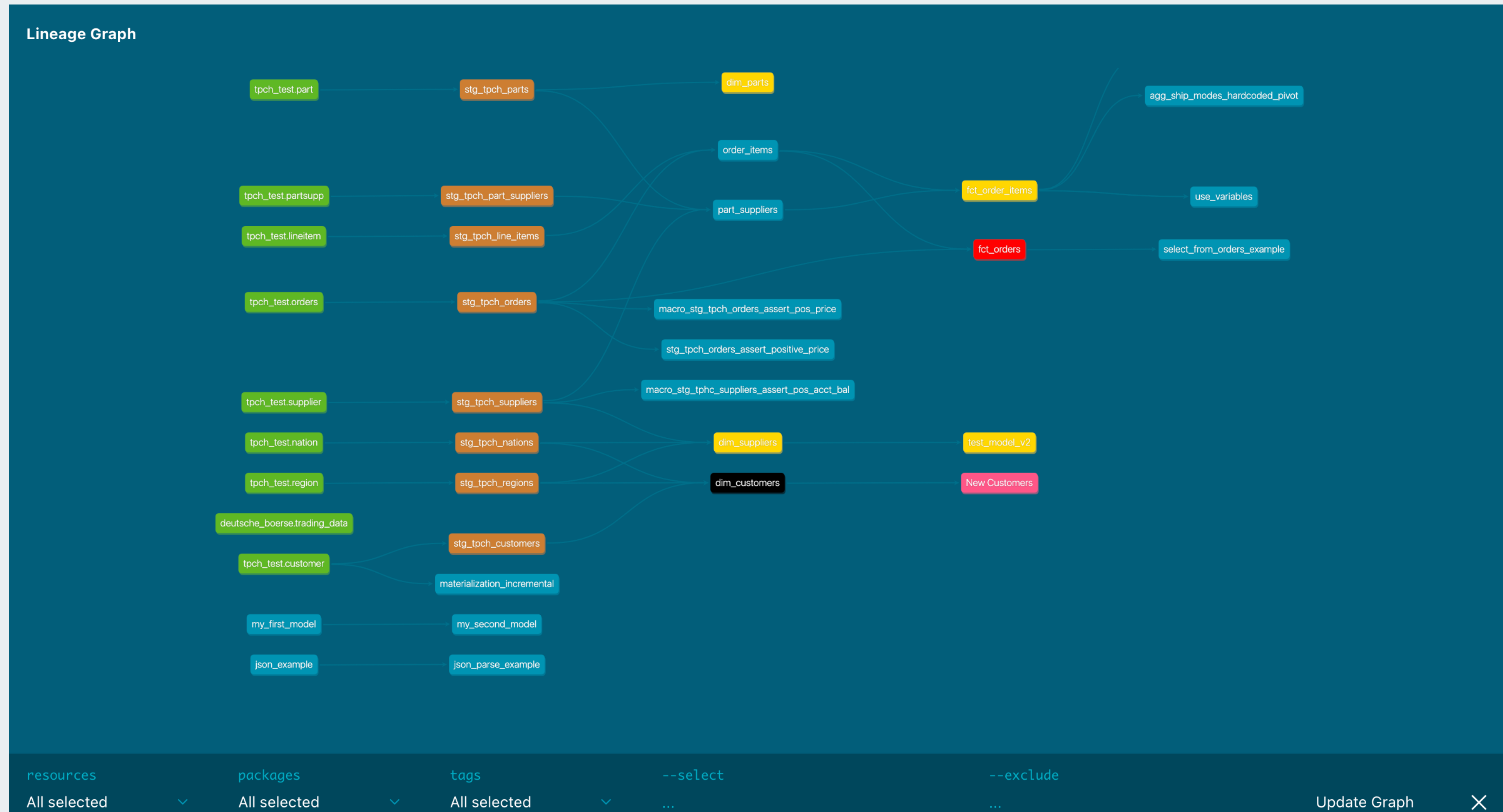
```
$ dbt run --select my_dbt_project_name # runs all models in your project
$ dbt run --select my_dbt_model # runs a specific model
$ dbt run --select path.to.my.models # runs all models in a specific directory
$ dbt run --select my_package.some_model # run a specific model in a specific package
$ dbt run --select tag:nightly # run models with the "nightly" tag
$ dbt run --select path/to/models # run models contained in path/to/models
$ dbt run --select path/to/my_model.sql # run a specific model by its path
```

Запуск модели по графу

```
$ dbt run --select my_model+ # select my_model and all children
$ dbt run --select +my_model # select my_model and all parents
$ dbt run --select +my_model+ # select my_model, and all of its parents and children
```

# Документация dbt

- › dbt docs generate
- › dbt docs serve



# Документация dbt

- › dbt docs generate
- › dbt docs serve

The screenshot displays the dbt web interface. On the left, a navigation pane shows a tree view of projects and sources. The 'tpch' project is expanded, showing the 'ods' folder and the 'lineitem' model. The main content area shows the 'Code' tab for the 'tpch.ods.lineitem' model, which is incremental. The code is a SQL query with Jinja templating for configuration and filtering.

```
1 {{ config(
2   materialized='incremental',
3   file_format='delta',
4   unique_key='lineitem_sk',
5   incremental_strategy='merge'
6 ) }}
7
8 select {{ dbt_utils.surrogate_key(['L_ORDERKEY','L_LINENUMBER']) }} as lineitem_sk
9   , L_ORDERKEY as orderkey
10  , L_PARTKEY as partkey
11  , L_SUPPKEY as suppkey
12  , L_LINENUMBER as linenum
13  , L_QUANTITY as quantity
14  , L_EXTENDEDPRICE as expended_price
15  , L_DISCOUNT as discount
16  , L_TAX as tax
17  , L_RETURNFLAG as return_flg
18  , L_LINestatus as linestatus
19  , L_SHIPDATE as ship_dt
20  , L_COMMITDATE as commit_dt
21  , L_RECEIPTDATE as receipt_dt
22  , L_SHIPINSTRUCT as shipinstruct
23  , L_SHIPMODE as shipmode
24  , L_COMMENT as comment
25  {{ tech_fields() }}
26 from {{ source('tpch', 'lineitem') }}
27 where 1=1
28 {% if is_incremental() %}
29   and L_SHIPDATE between
30     {{ dbt_utils.dateadd(datepart='year', interval=-30, from_date_or_timestamp=""~var("start_dttm")~"") }}
```



# dbt test

- › Описываются в yml файле модели
- › Применяются как на поле, так и на всю таблицу
- › Запускаются командой dbt test

```
version: 2

models:
  - name: fact_interaction
    description: '{{ doc("fact_interaction_description") }}'

    columns:
      - name: sk_interaction
        description: Surrogate key unique identifier for the case
        tests:
          - unique
          - not_null
      - name: bk_source_driver
        description: Business key from the source
        tests:
          - unique
          - not_null
      - name: count_interactions
        description: Number of interactions with the customer
        tests:
          - not_null
      - name: fk_interaction_detail
        description: Foreign key pointing to interaction detail dimension
        tests:
          - not_null
          - unique
          - relationships:
              to: source('sid', 'dim_interaction_detail')
              field: sk_interaction_detail
      - name: fk_interaction_text
        description: Foreign key pointing to interaction text dimension
        tests:
          - not_null
          - unique
          - relationships: # Referential integrity test
              to: source('sid', 'dim_interaction_text')
              field: sk_interaction_detail
      - name: performance_band
        description: Performance classified as Ungraded or Very Good or Good or OK or Bad or Very Bad
        tests:
          - accepted_values:
              values: ['Ungraded', 'Very Good', 'Good', 'OK', 'Bad', 'Very Bad']
```

# Суцности dbt

- [snapshots](#) A way to capture the state of your mutable tables so you can refer to it later.
- [seeds](#) CSV files with static data that you can load into your data platform with dbt.
- [tests](#) SQL queries that you can write to test the models and resources in your project.
- [macros](#) Blocks of code that you can reuse multiple times.
- [docs](#) Docs for your project that you can build.
- [sources](#) A way to name and describe the data loaded into your warehouse by your Extract and Load tools.
- [exposures](#) A way to define and describe a downstream use of your project.
- [metrics](#) A way for you to define metrics for your project.
- [groups](#) Groups enable collaborative node organization in restricted collections.
- [analysis](#) A way to organize analytical SQL queries in your project such as the general ledger from your QuickBooks.

# dbt

## Сильные стороны

- › Низкий порог входа
- › Соккрытие сложности вставки и миграции данных
- › Документация и lineage «из коробки»
- › DQ\тестирование данных в виде ожиданий от данных

## Слабые стороны

- › Jinja макросы 😞
- › Местами сложная кастомизируемость (заданная структура проекта, одно имя модели на все проекты и т.п.)
- › Слабая поддержка ru-моделей (появилась в версии 1.4)

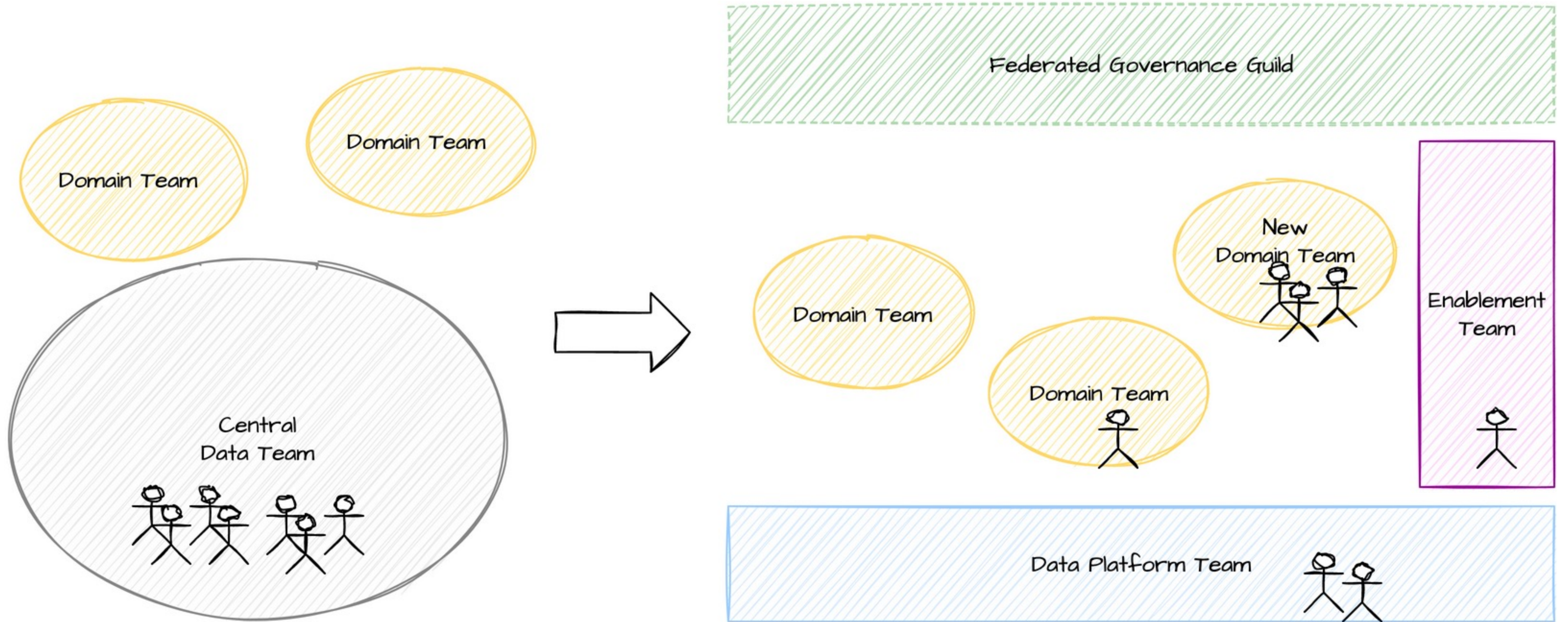
### 3. Внедрение dbt

# Как мы использовали dbt?

**Вопрос: как организовать проект (с учетом Data Mesh) ?**

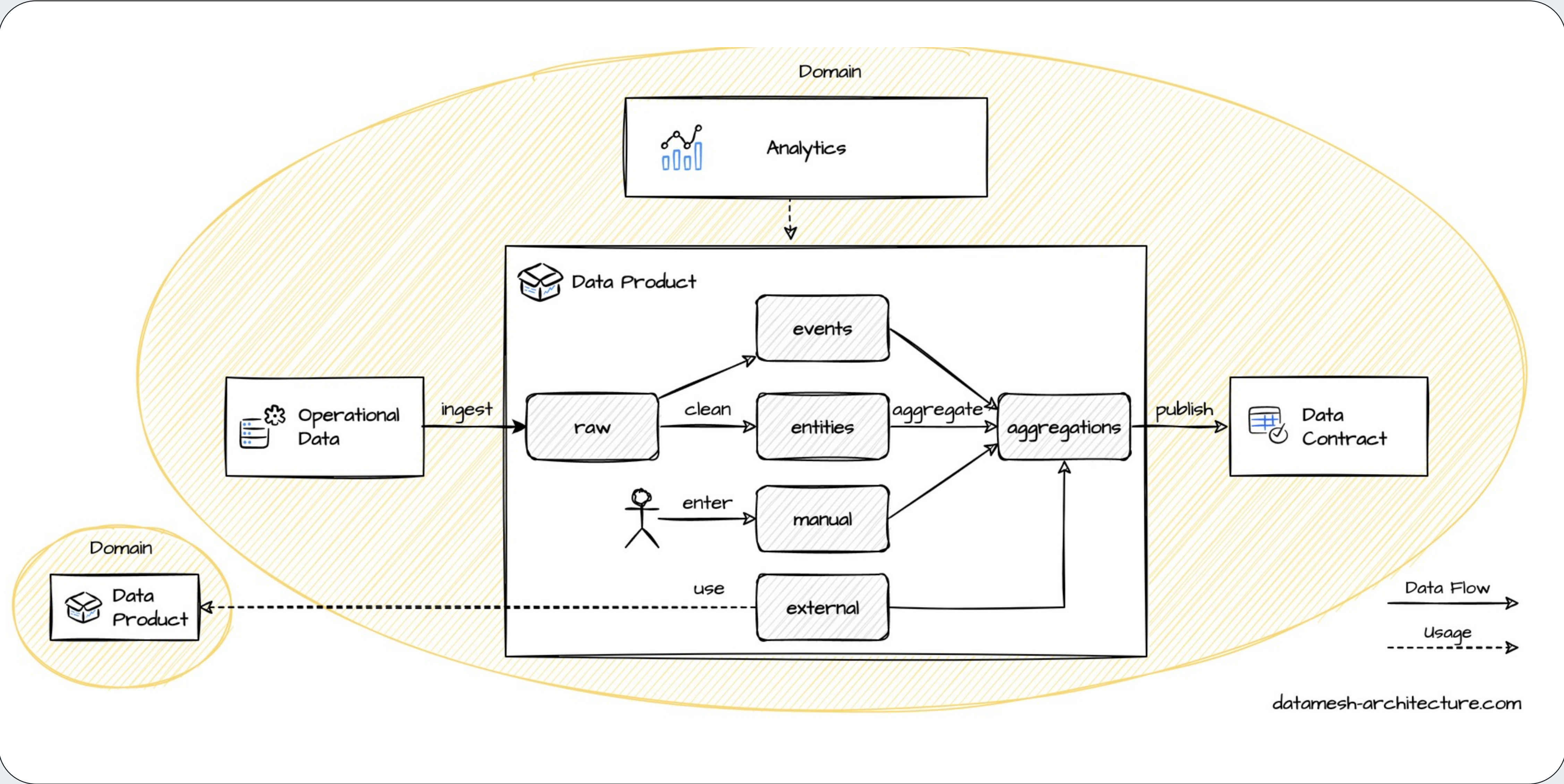
# Data mesh

## Data Team's Journey



[datamesh-architecture.com](https://datamesh-architecture.com)

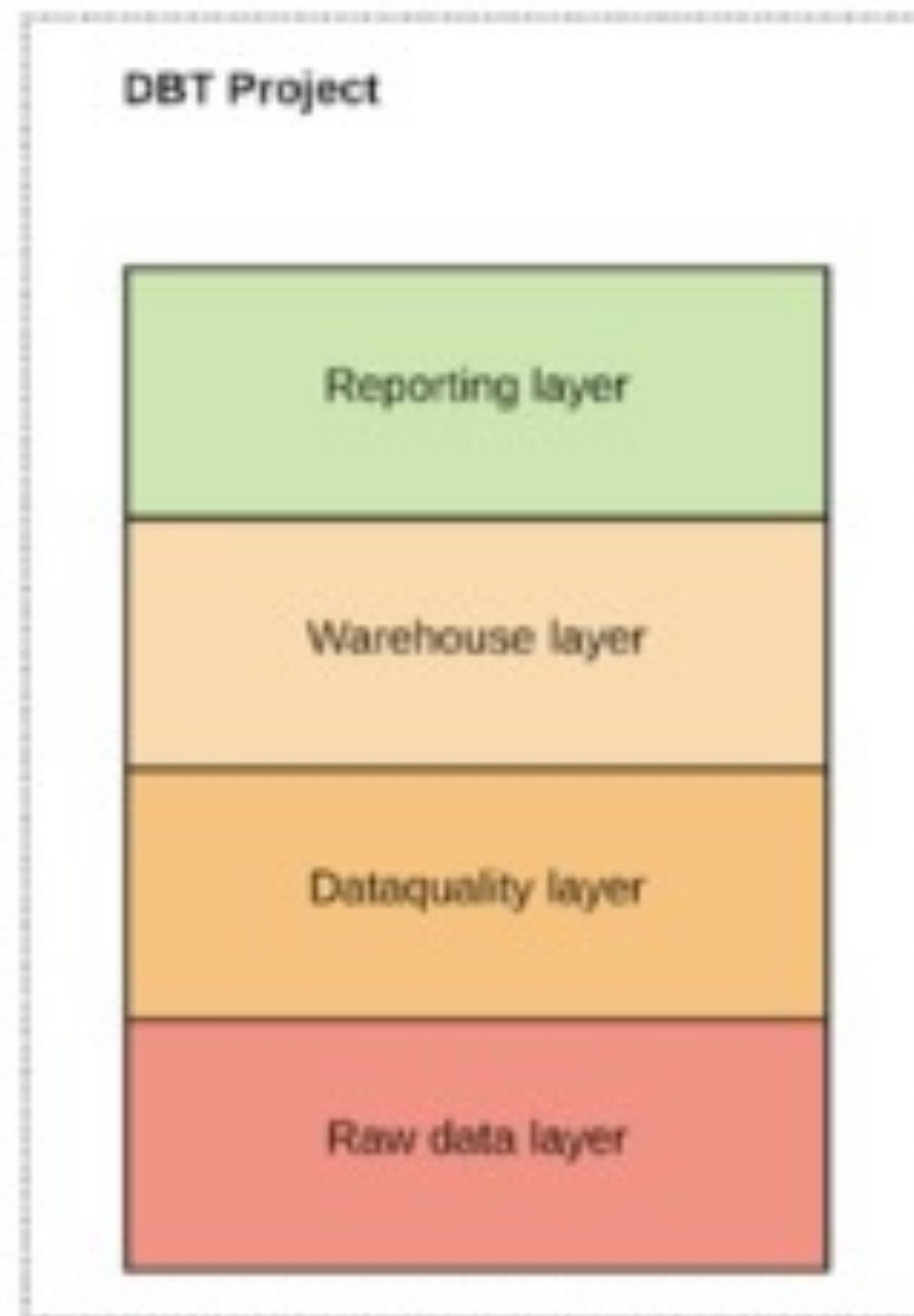
# dbt+airflow



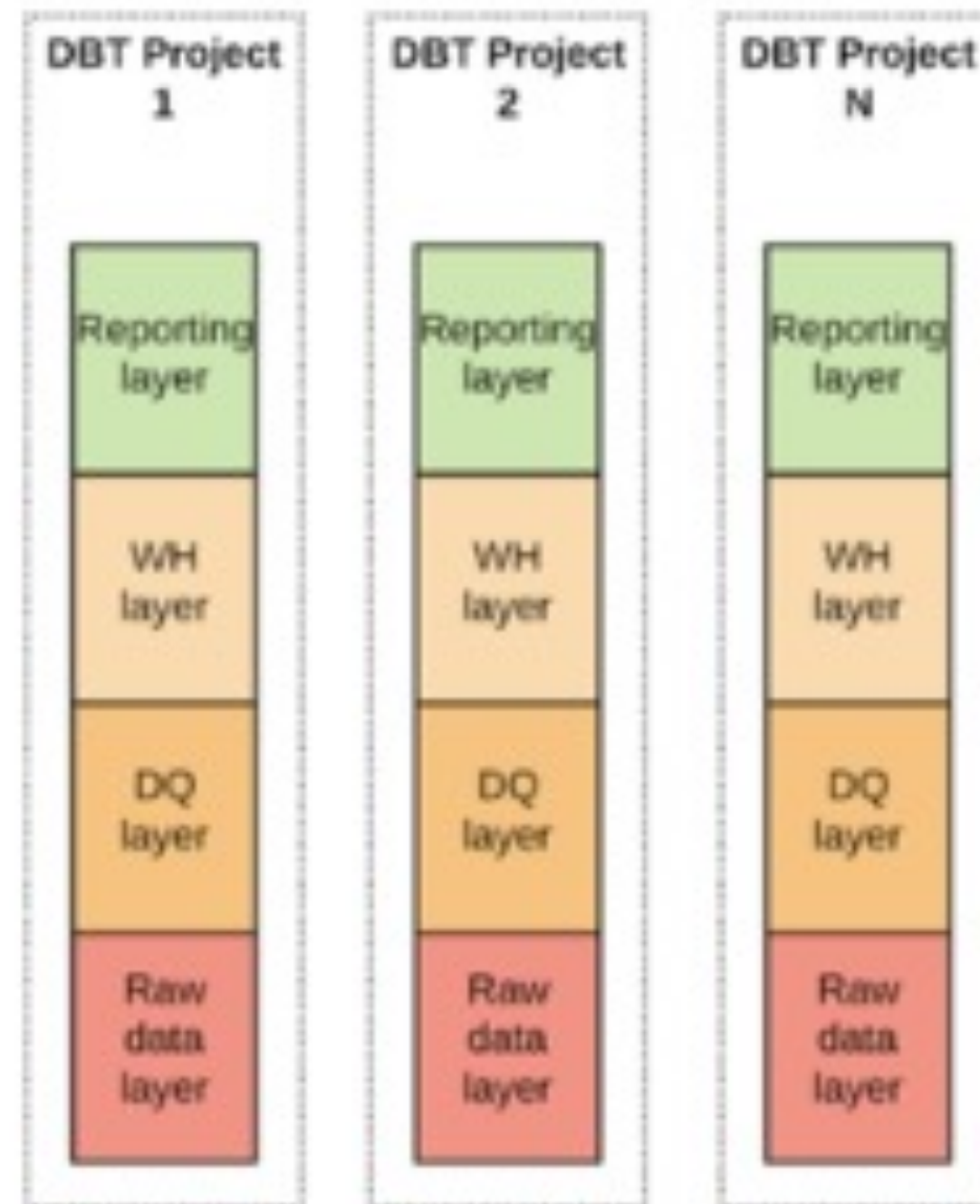
datamesh-architecture.com

# Четыре варианта организации проекта

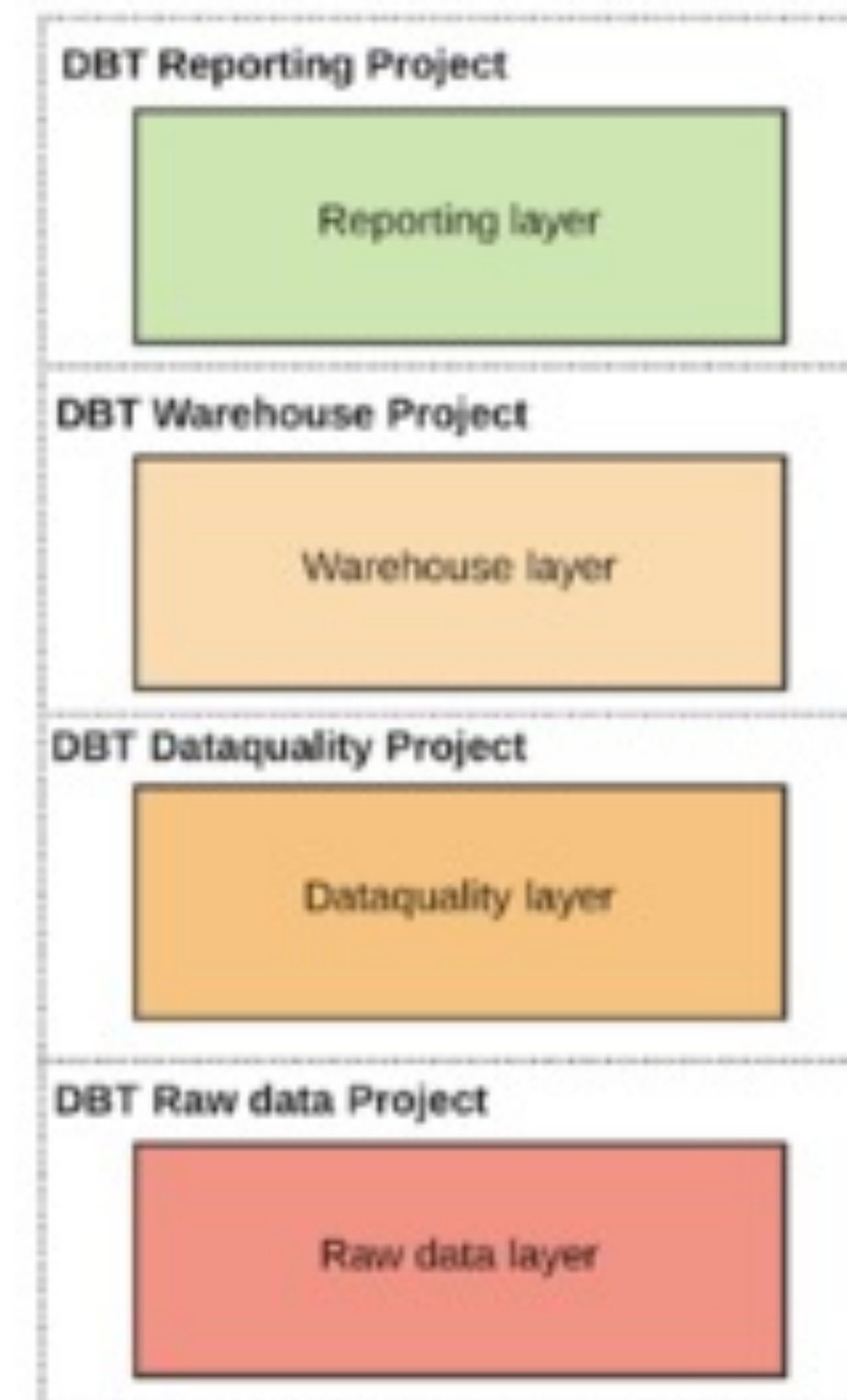
## Монолитная



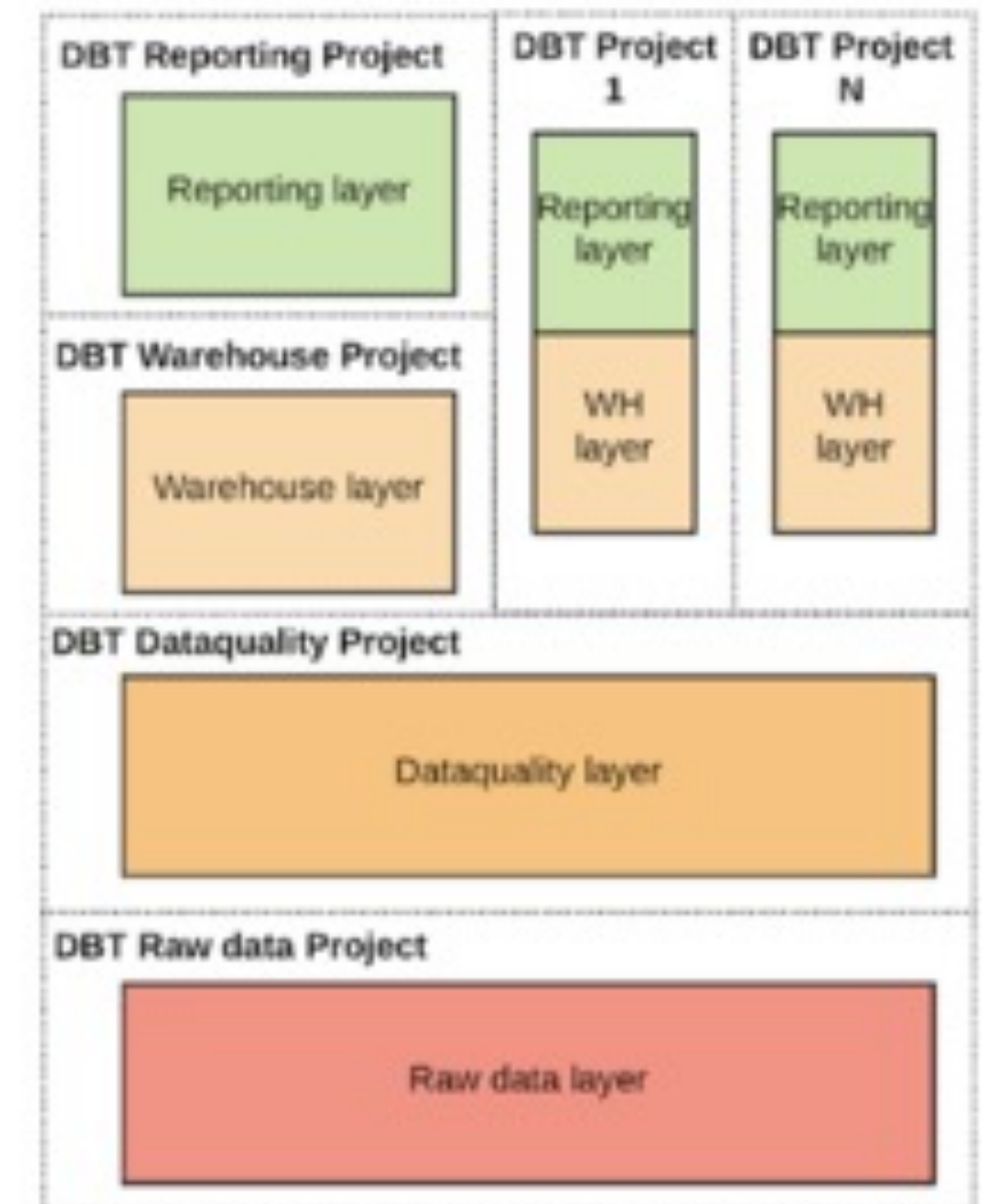
## Микросервисная



## Послойная



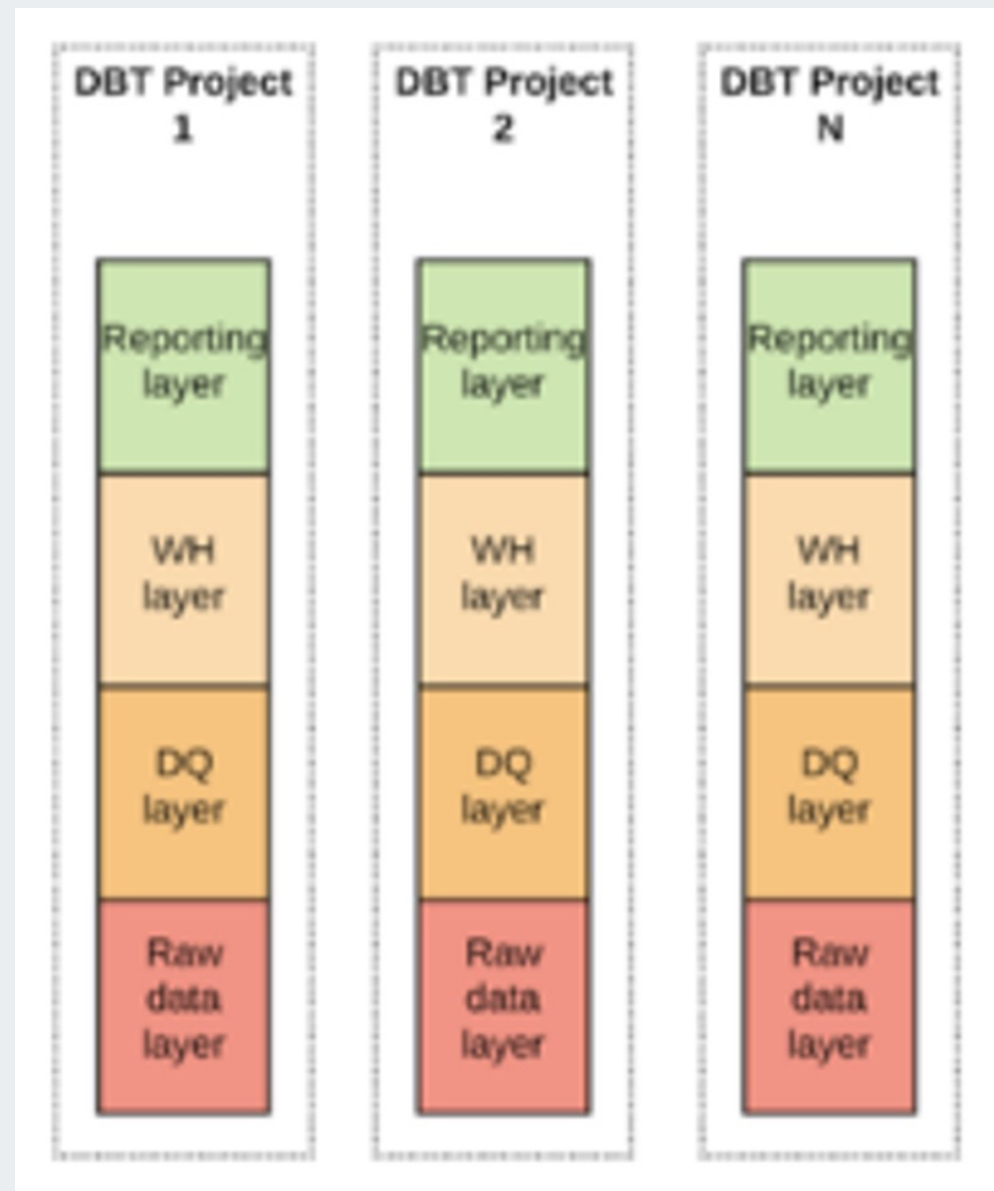
## Смешанная



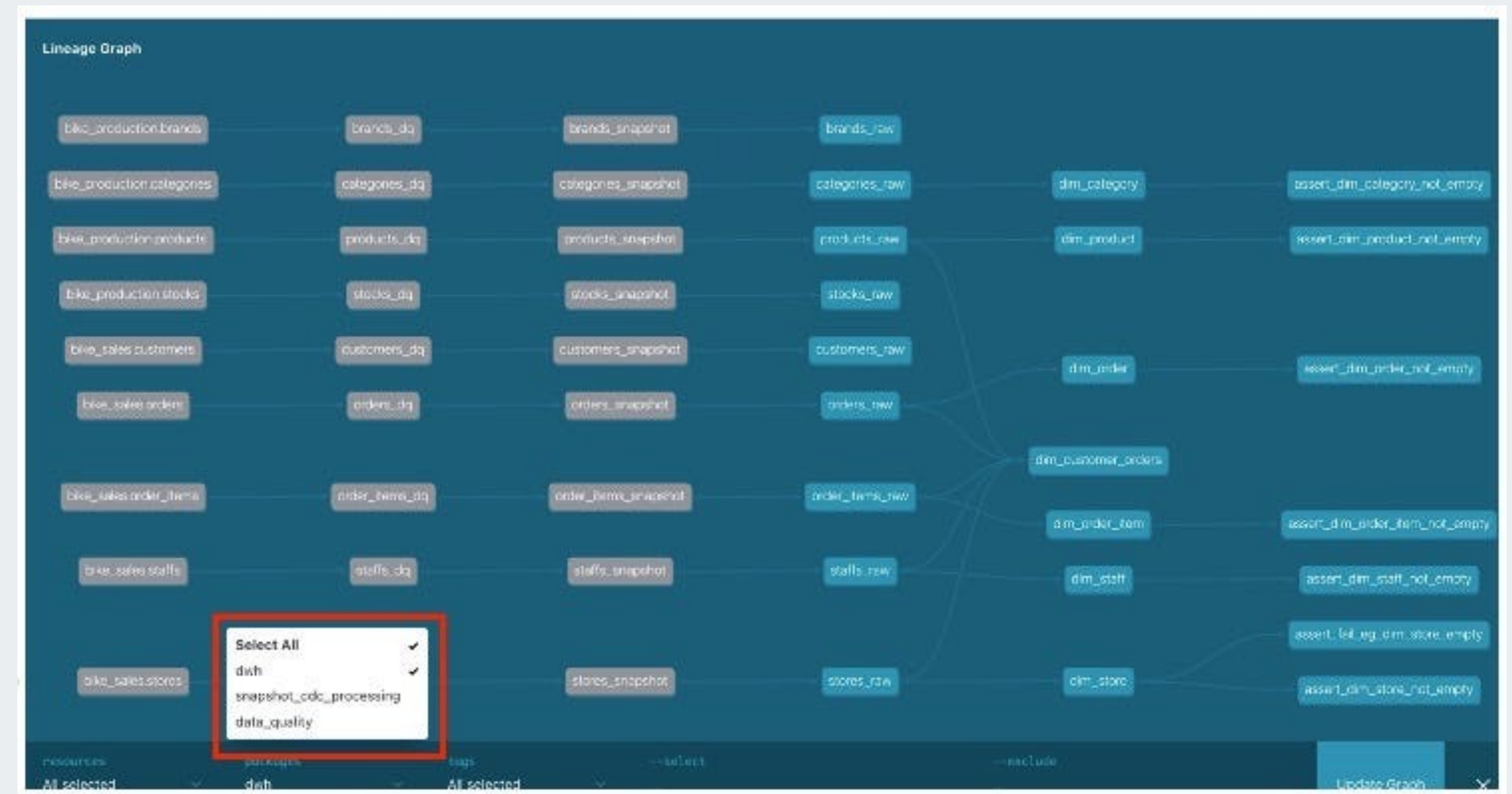


# Data Mesh – микросервисная архитектура DWH

## Микросервисная организация проекта

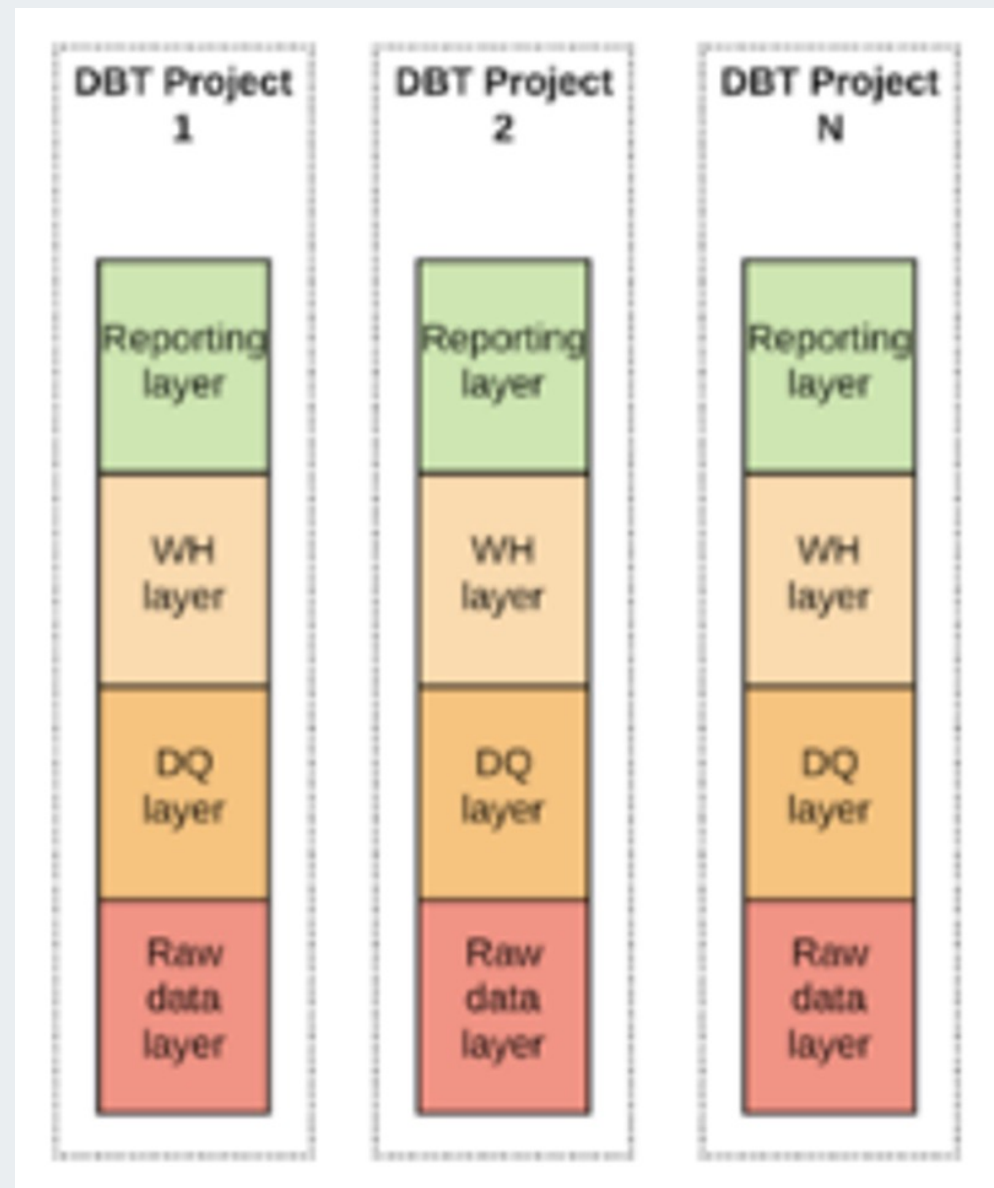


## «Фейковый» проект для общей документации

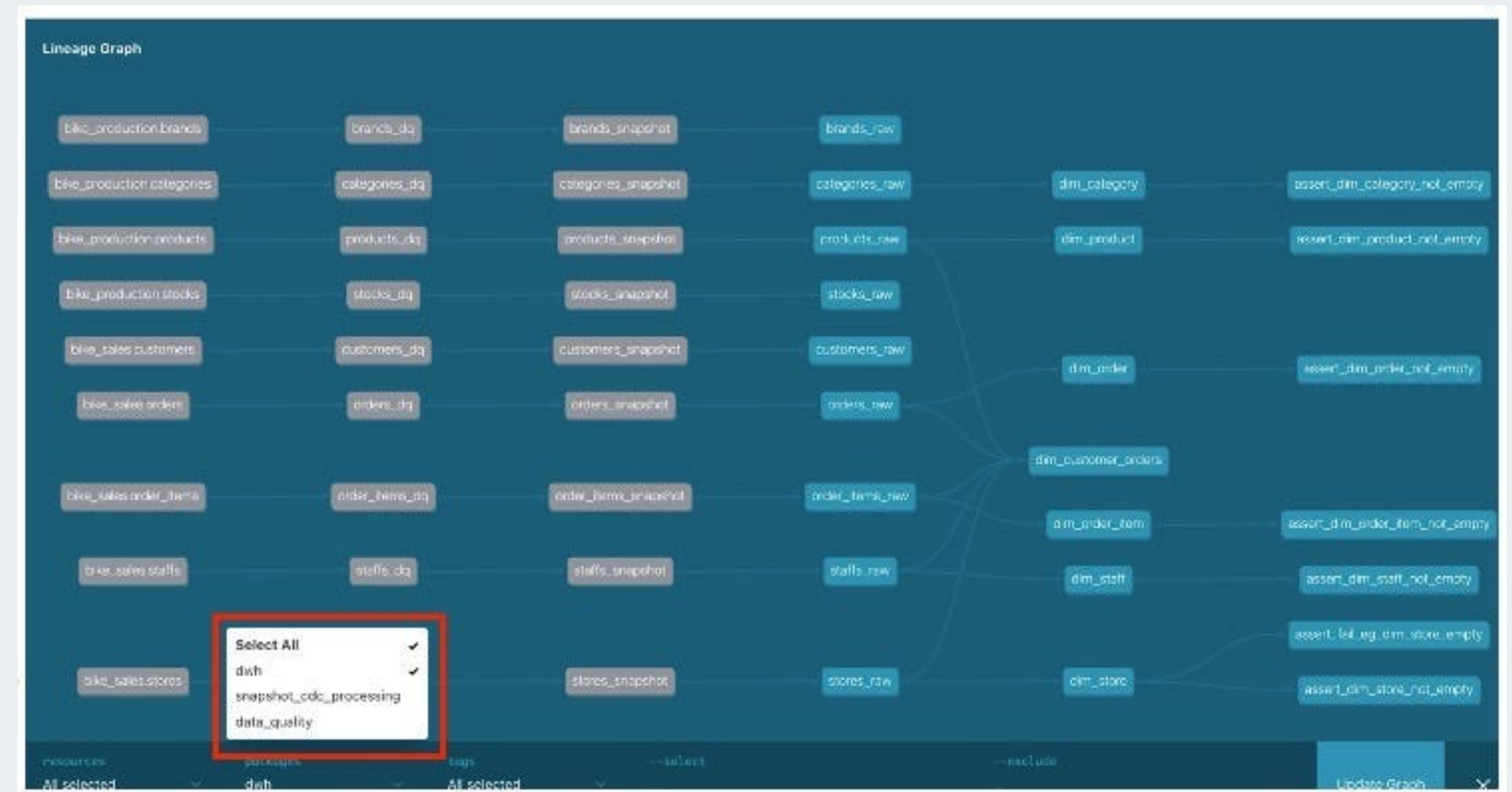


# Data Mesh – микросервисная архитектура DWH

## Микросервисная организация проекта



## «Фейковый» проект для общей документации



Да, но нет.

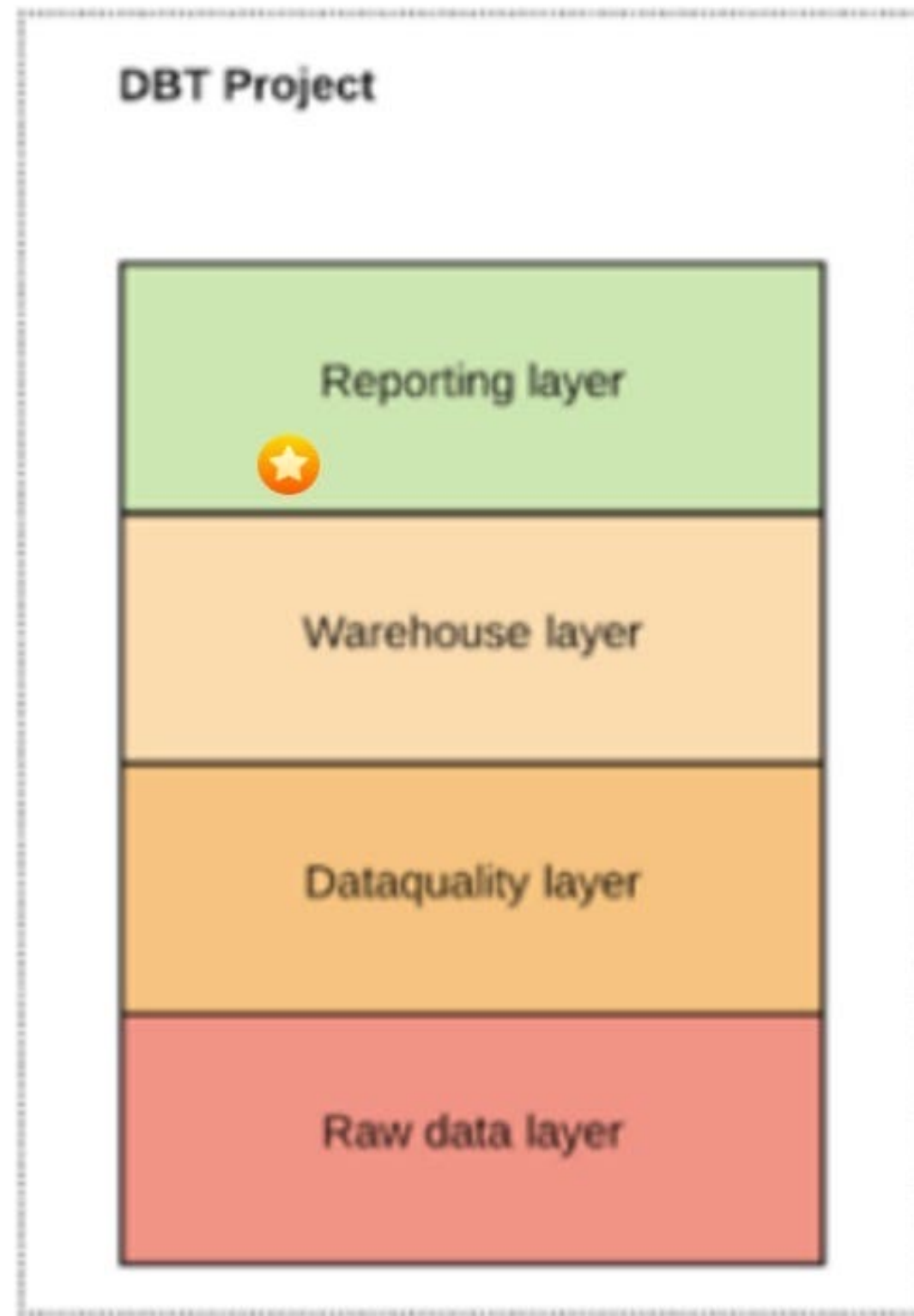
› Мы не можем называть одинаково модели в разных проектах (починили с версии 1.6)

<https://docs.getdbt.com/faqs/models/unique-model-names>

› Проекты\домены не могут импортировать друг друга, проблема циклических зависимостей

# Подоменная организация


## Монолитная



## File/folder structure of the (default) 'Monolithic' dbt project structure

```
<dbt project name>
├── analysis
├── data
├── macros
├── models
│   ├── raw_data_layer
│   ├── data_quality_layer
│   ├── warehouse_layer
│   └── reporting_layer
├── profiles
├── snapshots
├── tests
└── dbt_project.yml
```

## Домен данных на пути к модели



```
<dbt project name>
├── dbt_project.yml
├── data
├── macros
├── models
│   ├── data_domain
│   └── data_domain
│       ├── data_layer
│       │   ├── model.yml
│       │   └── model.sql
│       └── data_layer
├── snapshots
└── tests
```

# Плюсы\минусы решения

## Сильные стороны

- › Общая документация и переиспользование данных
- › Единые контролируемые стандарты
- › Единый граф расчета всей организации
- › Codeowners доменов средствами GH

## Слабые стороны

- › Конфликты в общем dbt\_projects.yml
- › Невозможность закрыть часть кода (например, антифрод)
- › Единовременный релизный процесс на все домены

**Вопрос: как разделить prod\dev enviroment ?**

# Production environment

Объект формируется по пути <domain\_name>.<layer\_name>.<model\_name>

The screenshot shows the 'Data Explorer' interface for a 'westeurope' environment. The left sidebar lists various data catalogs, with 'svc\_toloka\_backend' highlighted. The main panel displays details for this catalog, including its owner ('tk-team-data-platform-sre') and a list of six schemas. The schemas are listed in a table with columns for Name, Created at, and Owner.

Name	Created at	Owner
information_schema	2023-02-01 03:20:57	System user
ods	2023-02-02 10:12:53	jkermakov@toloka.ai
push2az	2023-02-06 01:05:43	lkozhinov@toloka.ai
raw	2023-02-02 10:12:46	jkermakov@toloka.ai
rep	2023-02-16 17:22:32	Odbdb0bb-38bd-48a7-bd67-ea30c6c74c4c
svc	2023-02-01 18:21:15	lkozhinov@toloka.ai

# Development enviroment

Объект формируется по пути `users.<user_name>.<domain_name>_<layer_name>_<model_name>`

The screenshot displays the Data Explorer interface for a user named 'jkermakov'. The breadcrumb path is 'Catalogs > users > users.jkermakov'. The left sidebar shows a tree view with 'users' expanded to 'jkermakov', listing various tables. The main panel shows a table list with columns: Name, Created at, Owner, and Popularity. A blue box highlights the breadcrumb path and the table list.

Name	Created at	Owner	Popularity <span>New</span>
cluster_dbus	2023-07-18 14:32:09	jkermakov@toloka.ai	----
svc_data_platform_cdm_fct_dbt_run_command	2023-07-13 11:59:50	jkermakov@toloka.ai	----
svc_data_platform_cdm_fct_dbt_run_command_w_dbu	2023-07-13 12:05:57	jkermakov@toloka.ai	----
svc_data_platform_cdm_fct_dbx_billing	2023-07-13 09:58:50	jkermakov@toloka.ai	----
svc_data_platform_ods_dbt_invocation_log	2023-07-13 11:52:57	jkermakov@toloka.ai	----
svc_data_platform_ods_dbt_run_log	2023-07-13 09:57:56	jkermakov@toloka.ai	----
svc_data_platform_ods_dbt_test_act	2023-07-13 09:57:58	jkermakov@toloka.ai	----
svc_data_platform_ods_dbx_billing_log	2023-07-31 13:02:06	jkermakov@toloka.ai	
svc_data_platform_raw_azure_azure_billing	2023-07-19 11:05:24	jkermakov@toloka.ai	----
svc_data_platform_raw_databricks_billing_logs	2023-07-31 12:53:09	jkermakov@toloka.ai	
svc_data_platform_rep_dbus_spending	2023-07-13 09:42:34	jkermakov@toloka.ai	----
svc_platform_backend_raw_eligible_pool_event_parsed	2023-08-16 16:46:43	jkermakov@toloka.ai	
svc_toloka_platform_raw_eligible_pool_event_errors	2023-07-24 18:09:55	jkermakov@toloka.ai	----
svc_toloka_platform_raw_eligible_pool_event_flat	2023-07-24 18:23:16	jkermakov@toloka.ai	----
svc_toloka_platform_raw_eligible_pool_event_parsed	2023-07-24 17:50:52	jkermakov@toloka.ai	----
test_udf	2023-07-21 15:42:12	jkermakov@toloka.ai	----
test_udf_ass	2023-07-23 16:57:38	jkermakov@toloka.ai	----
test_udf_ass_p	2023-07-23 17:28:51	jkermakov@toloka.ai	----

# dbt custom names

В dbt есть возможность переопределить макросы, отвечающие за имена

- › generate\_schema\_name
- › generate\_database\_name
- › generate\_alias\_name

Простая логика:

- › Если target==prod, то генерируем обычное имя объекта
- › Если target==dev, то генерируем имя в user-область

get\_custom\_alias.sql

```
{% macro generate_alias_name(custom_alias_name=none, node=none) -%}

    {%- if custom_alias_name -%}
        {{ custom_alias_name | trim }}

    {%- elif node.version -%}
        {{ return(node.name ~ "_v" ~ (node.version | replace(".", "_"))) }}

    {%- else -%}
        {{ node.name }}

    {%- endif -%}

{%- endmacro %}
```



# Плюсы\минусы решения

## Сильные стороны

- › Удобно и понятно пользователям
- › Возможность протестировать модель на prod-мощностях
- › Не нужно генерировать отдельные тестовые срезы

## Слабые стороны

- › Сложно тестировать изменения в макросах

**Вопрос: как запускать модели по расписанию?**

# Три кита и четыре столпа дата платформы

Что?



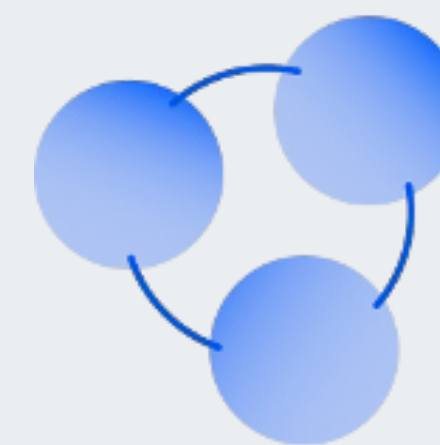
Архитектура  
=  
Data Lakehouse

Почему?



Процессы  
=  
Data mesh

Как?



Технологии  
=  
Modern Data Stack



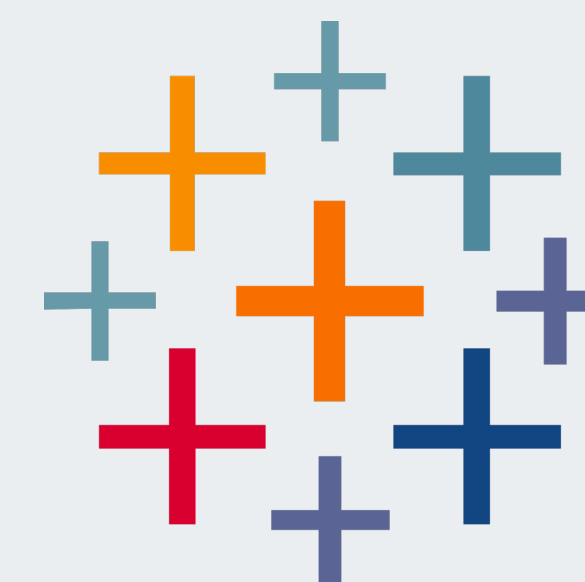
databricks



dbt



airflow



tableau

# airflow+dbt

**Основная идея:** у нас уже есть граф в dbt, на базе него можно создать граф в airflow

1. Собираем dbt

2. Получаем все зависимости в manifest.json

3. На базе зависимостей строим Dag

> dbt build



```
{
  "metadata": {
    "dbt_schema_version": "https://schemas.getdbt.com/dbt/manifest/v9.json",
    "dbt_version": "1.5.0",
    "generated_at": "2023-08-26T11:52:21.345068Z",
    "invocation_id": "1a0f379e-9731-4ede-b3de-e68c6869a9f0",
    "env": {},
    "project_id": "45e55f00b076b627b037b215b98e9f99",
    "user_id": null,
    "send_anonymous_usage_stats": true,
    "adapter_type": "databricks"
  },
  "nodes": {
```



```
dwh_etl > dags > system > dbt_dag.py > ...
You, 3 months ago | 3 authors (igsaf and others)
1 import os
2
3 from tlk_dbt_utils.dags import compile_dbt_dags
4 from tlk_dbt_utils.common.variables import DbtBuildVariables
5
6 # LABELS: dag, airflow
7 dags = compile_dbt_dags(
8     manifest_path=os.path.join(
9         DbtBuildVariables.load().project_dir,
10        'target/manifest.json',
11    ),
12    etl_service_name='dwh_etl',
13 )
14
15 for name, dag in dags.items():
16     globals()[name] = dag
17
```

# airflow+dbt

Основная идея: у нас уже есть граф в dbt, на базе него можно создать граф в airflow

The screenshot shows the Airflow web interface for a DAG named 'dbt\_advanced\_dag'. The interface includes a top navigation bar with 'Airflow', 'DAGs', 'Security', 'Browse', 'Admin', 'Docs', and 'Astronomer'. The DAG title is 'DAG: dbt\_advanced\_dag' with a subtitle 'A dbt wrapper for airflow'. The status is 'None' and the schedule is '1 day, 0:00:00'. Below the title are navigation tabs: 'Tree View', 'Graph View' (selected), 'Task Duration', 'Task Tries', 'Landing Times', 'Gantt', 'Details', and 'Code'. A filter bar shows a date '2020-12-22T21:02:14Z', 'Runs' count of '25', and a 'Run' button. A legend below the DAG shows various task statuses: 'queued', 'running', 'success', 'failed', 'up\_for\_retry', 'up\_for\_reschedule', 'upstream\_failed', 'skipped', 'scheduled', and 'no\_status'. The DAG graph itself starts with a 'start' node, followed by 'singer\_tap\_extract' and 'singer\_target\_load'. From 'singer\_target\_load', the flow branches into several 'model' tasks: 'model.jaffle\_shop.stg\_orders', 'model.jaffle\_shop.stg\_payments', 'model.jaffle\_shop.customer\_orders', 'model.jaffle\_shop.stg\_customers', 'model.jaffle\_shop.customer\_payments', 'model.jaffle\_shop.order\_payments', and 'model.jaffle\_shop.hidden\_model'. These 'model' tasks then lead to corresponding 'test' tasks: 'test.jaffle\_shop.customer\_orders', 'test.jaffle\_shop.dim\_customers', 'test.jaffle\_shop.stg\_customers', 'test.jaffle\_shop.customer\_payments', 'test.jaffle\_shop.fct\_orders', 'test.jaffle\_shop.order\_payments', 'test.jaffle\_shop.hidden\_test', 'test.jaffle\_shop.stg\_orders', and 'test.jaffle\_shop.stg\_payments'. All 'test' tasks converge into an 'end' node.

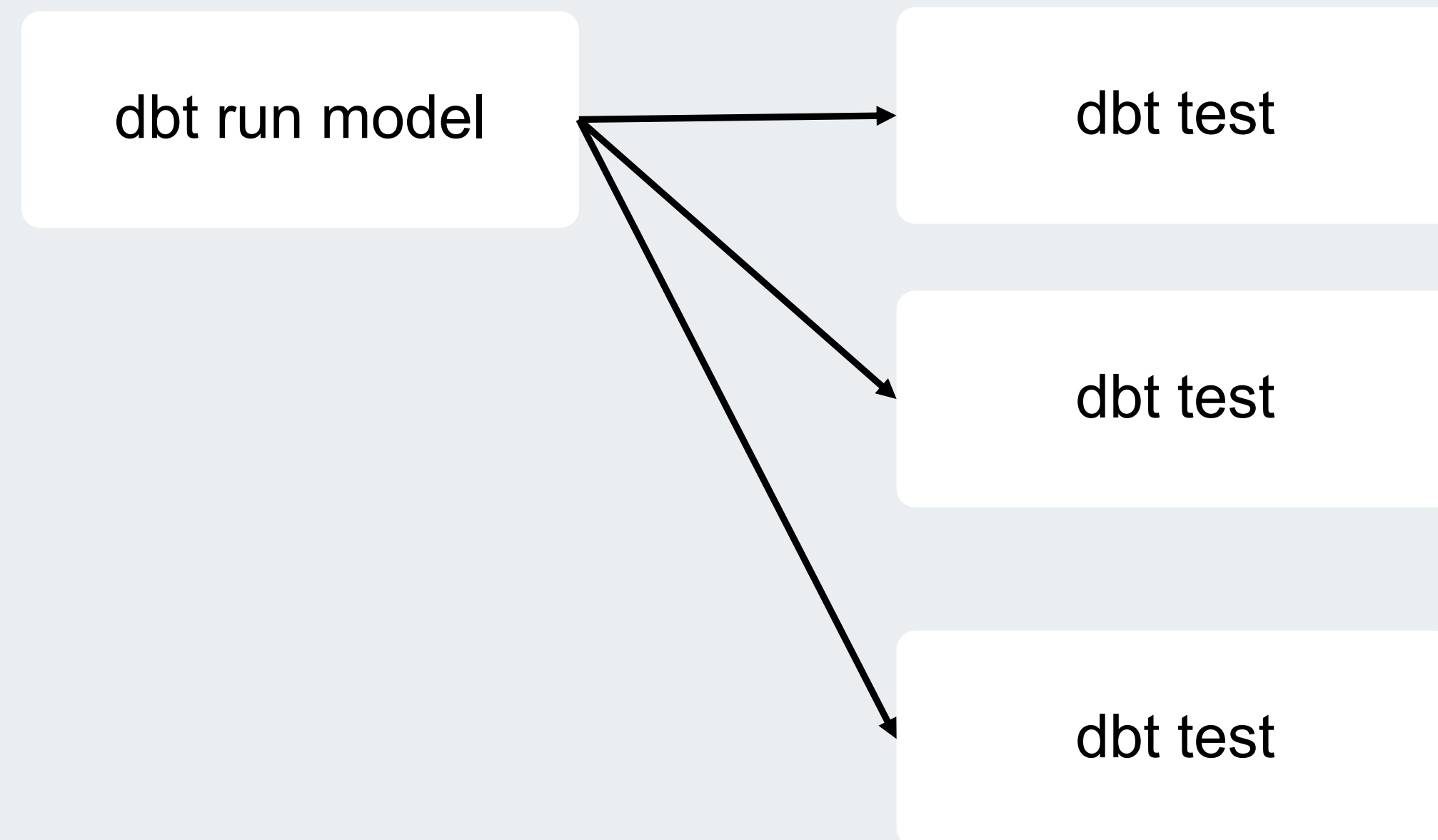
## model task

- › Основной оператор
- › Отвечает за построение модели

```
dbt run model
```

# model task

- › Основной оператор
- › Отвечает за построение модели
- › Тестирование модели
- › В зависимости от тега стартуют
  - › в конце taskgroup
  - › в конце dag
  - › раз в неделю

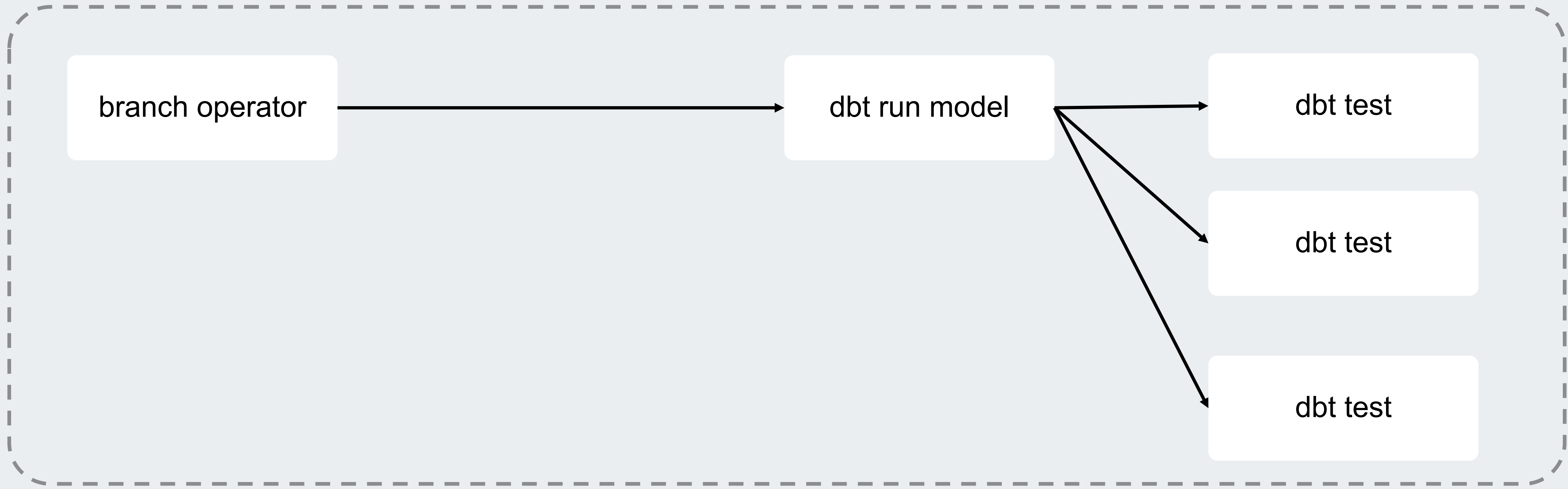


# model task

- › Проверяет, включена ли модель

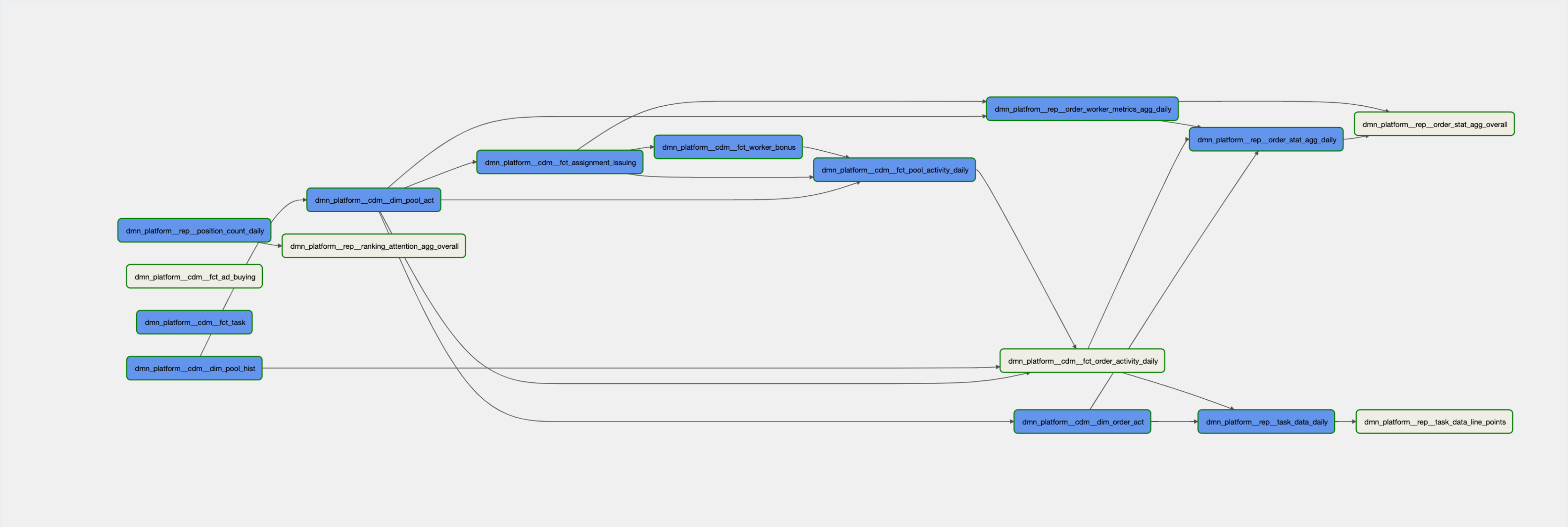
- › Основной оператор
- › Отвечает за построение модели

- › Тестирование модели
- › В зависимости от тега стартуют
  - › в конце taskgroup
  - › в конце dag
  - › раз в неделю





# Airflow Dag



# Плюсы\минусы решения

## Сильные стороны

- › Единый и понятный интерфейс для запуска задач - airflow
- › Возможность перезапускать модели и делать backfill
- › Интеграция со всеми остальными инструментами

## Слабые стороны

- › Дополнительная самописная интеграция
- › Запуск через BashOperator

**Вопрос: как соединить с обычными dags ?**

# Airflow dataset

С версии airflow 2.4 доступен Data-aware scheduling

```
from airflow.datasets import Dataset

with DAG(...):
    MyOperator(
        # this task updates example.csv
        outlets=[Dataset("s3://dataset-bucket/example.csv")],
        ...,
    )

with DAG(
    # this DAG should be run when example.csv is updated (by dag1)
    schedule=[Dataset("s3://dataset-bucket/example.csv")],
    ...,
):
    ...
```

# Airflow dataset

- › Зарегистрировали все dbt модели как dataset в airflow
- › Любой даг можно поставить в зависимость от dbt модели

The image shows a screenshot of the Airflow Datasets interface on the left and a DAG diagram on the right.

**Datasets Interface:**

Filter datasets with updates in the past: All Time 30 days 7 days 24 hours 1 hour

Search by URI...

URI	LAST UPDATE
svc_apps.ods.app_project Total Updates: 3325	2023-09-01, 08:11:40 UTC
svc_toloka_backend.ods.requester_profile_act Total Updates: 5109	2023-09-01, 08:11:37 UTC
svc_apps.ods.infra_project Total Updates: 3256	2023-09-01, 08:11:31 UTC
svc_toloker_backend.ods.fingerprint_log Total Updates: 225	2023-09-01, 08:11:28 UTC
svc_toloker_backend.ods.worker_profile_skill_act Total Updates: 225	2023-09-01, 08:11:25 UTC
svc_toloka_backend.raw.assignment_workflow_parsed Total Updates: 5525	2023-09-01, 08:11:22 UTC
svc_requester_backend.ods.project_act Total Updates: 225	2023-09-01, 08:11:12 UTC
svc_uhrs.ods.uhrs_worker_stat_5m_snp Total Updates: 2844	2023-09-01, 08:11:12 UTC
svc_toloka_backend.ods.worker_pool_selection_log Total Updates: 4708	2023-09-01, 08:11:11 UTC
svc_toloker_backend.ods.withdrawal_account_log Total Updates: 225	2023-09-01, 08:11:03 UTC
svc_requester_backend.ods.project_log Total Updates: 225	2023-09-01, 08:11:00 UTC
svc_toloka_backend.ods.captcha_log Total Updates: 4851	2023-09-01, 08:10:47 UTC
svc_toloka_backend.raw.worker_status_event Total Updates: 5046	2023-09-01, 08:10:42 UTC
svc_toloka_backend.raw.worker_profile_event Total Updates: 5271	2023-09-01, 08:10:37 UTC
svc_requester_backend.raw.pool_event_flatten Total Updates: 225	2023-09-01, 08:10:26 UTC

**DAG Diagram:**

The DAG diagram shows the following tasks and dependencies:

- svc\_web\_analytics\_\_backfill and svc\_web\_analytics\_\_daily depend on svc\_web\_analytics.ods.session.
- svc\_finance\_\_backfill and svc\_finance\_\_daily depend on svc\_finance.raw.bud23\_geo\_parsed and svc\_finance.ods.bud23\_geo\_parsed.
- svc\_toloker\_backend\_\_daily and svc\_toloker\_backend\_\_backfill depend on a large number of datasets (represented by many vertical lines).

# Плюсы\минусы решения

## Сильные стороны

- › Простая возможность интегрировать dbt и custom dags
- › Нативно для airflow

## Слабые стороны

- › Сложный дебаг в случае ошибок

**Вопрос: как уменьшить сложность графа?**

## Упрощение dag

- › Доменная организация – каждая модель принадлежит домену
- › Модели могут считаться с разной частотой

**Dag = Domains X shedule**



# Упрощение dag

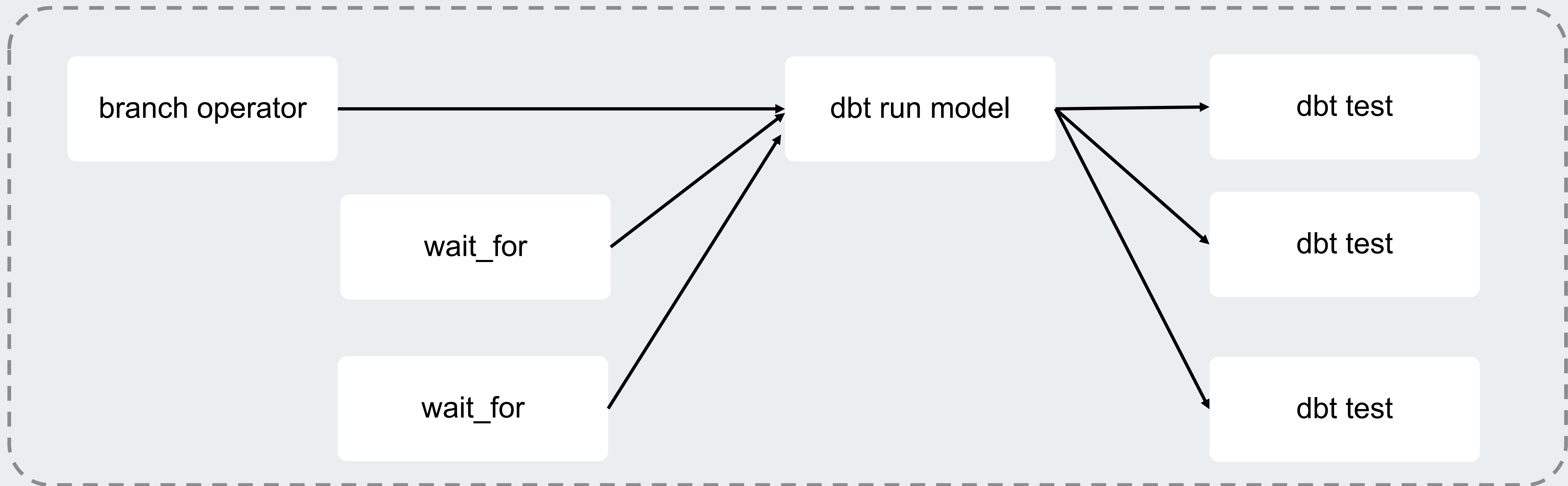
- › Доменная организация – каждая модель принадлежит домену
- › Модели могут считаться с разной частотой

Dag = Domains X shedule

<input checked="" type="checkbox"/>	<b>svc_requester_backend__daily</b> dbt frontier svc_requester_backend	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 186 <input type="checkbox"/> <input checked="" type="checkbox"/> 57	@daily <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_requester_backend__hourly</b> dbt frontier svc_requester_backend	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 5838 <input checked="" type="checkbox"/> 1 <input type="checkbox"/>	@hourly <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_toloka_backend__daily</b> dbt frontier svc_toloka_backend	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 152 <input type="checkbox"/> <input checked="" type="checkbox"/> 33	@daily <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_toloka_backend__hourly</b> dbt frontier svc_toloka_backend	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 3116 <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 1947	@hourly <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_toloka_frontend__hourly</b> dbt frontier svc_toloka_frontend	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 2396 <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2247	@hourly <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_toloker_backend__daily</b> dbt frontier svc_toloker_backend	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 193 <input type="checkbox"/> <input checked="" type="checkbox"/> 50	@daily <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_toloker_backend__hourly</b> dbt frontier svc_toloker_backend	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 5616 <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 222	@hourly <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_uhrs__daily</b> dbt frontier svc_uhrs	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 223 <input checked="" type="checkbox"/> 2 <input checked="" type="checkbox"/> 21	@daily <input type="checkbox"/>
<input checked="" type="checkbox"/>	<b>svc_uhrs__hourly</b> dbt frontier svc_uhrs	airflow	<input type="checkbox"/> <input checked="" type="checkbox"/> 2730 <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 365	@hourly <input type="checkbox"/>

## model task

- › Проверяет, включена ли модель
- › external\_task
- › Ожидает построения связанной модели
  - › из другого домена
  - › с другой частотой
- › Основной оператор
- › Отвечает за построение модели
- › Тестирование модели
  - › В зависимости от тега стартуют
    - › в конце taskgroup
    - › в конце dag
    - › раз в неделю



# model task

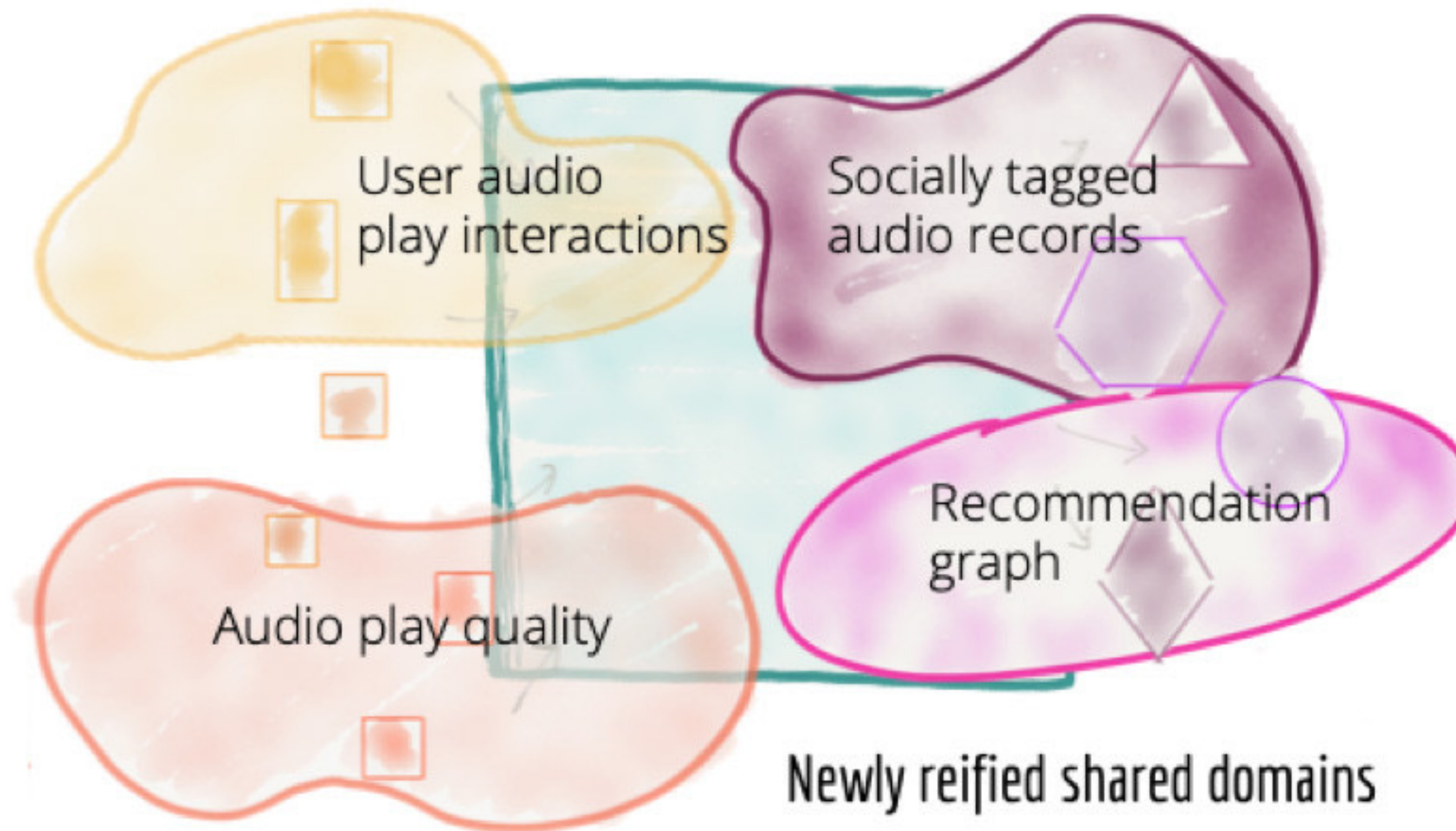
- › Проверяет, включена ли модель
- › external\_task
- › Ожидает построения связанной модели
  - › из другого домена
  - › с другой частотой
- › Основной оператор
- › Отвечает за построение модели
- › Тестирование модели
  - › В зависимости от тега стартуют
    - › в конце taskgroup
    - › в конце dag
    - › раз в неделю



# Data Mesh domain types

Domains aligned with the source

Domains aligned with the consumption



# Classic service (source) domain

Airflow DAGs Datasets Security Browse Admin Docs Toloka 06:59 UTC

DAG: **svc\_platform\_backend\_hourly** HERE COULD BE YOUR ADVERTISEMENT success Schedule: @hourly Next Run: 2023-09-01, 06:00:00

Grid Graph Calendar Task Duration Task Tries Landing Times Gantt Details Code Audit Log

2023-09-01T05:00:01Z Runs 25 Run scheduled\_\_2023-09-01T05:00:00+00:00 Layout Left > Right Update Find Task...

BranchPythonOperator DbtRun DbtTest EmptyOperator deferred failed queued removed restarting running scheduled shutdown skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

Auto-refresh

```
graph LR; subgraph Path1; T1_1[svc_platform_backend_raw_ban_event_parsed] --> T1_2[svc_platform_backend_raw_ban_event_flatten]; T1_2 --> T1_3[svc_platform_backend_ods_ban_log]; end; subgraph Path2; T2_1[svc_platform_backend_raw_eligible_pool_event_parsed] --> T2_2[svc_platform_backend_raw_eligible_pool_event_flatten]; T2_2 --> T2_3[svc_platform_backend_ods_eligible_pool_log]; end; subgraph Path3; T3_1[svc_platform_backend_raw_worker_bonus_event_parsed] --> T3_2[svc_platform_backend_raw_worker_bonus_event_flatten]; T3_2 --> T3_3[svc_platform_backend_ods_worker_bonus_log]; T3_3 --> T3_4[svc_platform_backend_ods_worker_bonus_act]; end; subgraph Path4; T4_1[svc_platform_backend_raw_worker_pool_selection_event_parsed] --> T4_2[svc_platform_backend_raw_worker_pool_selection_event_flatten]; T4_2 --> T4_3[svc_platform_backend_ods_worker_pool_selection_log]; end; subgraph Path5; T5_1[svc_platform_backend_raw_worker_skill_event_parsed] --> T5_2[svc_platform_backend_raw_worker_skill_event_flatten]; T5_2 --> T5_3[svc_platform_backend_ods_worker_skill_log]; end;
```

# Classic analytics (consumption) domain

**DAG: dmn\_requesters\_\_daily** HERE COULD BE YOUR ADVERTISEMENT success Schedule: @daily Next Run: 2023-09-01, 00:00:00

Grid **Graph** Calendar Task Duration Task Tries Landing Times Gantt Details <> Code Audit Log

2023-08-31T00:00:01Z Runs 25 Run scheduled\_2023-08-31T00:00:00+00:00 Layout Left > Right Update Find Task...

DbtExternalSensor DbtRun DbtTest EmptyOperator deferred failed queued removed restarting running scheduled shutdown skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

Auto-refresh

```
graph LR; A[dmn_requesters_cdm_dim_requester_act] --> B[dmn_requesters_cdm_dim_project_hist]; A --> C[dmn_requesters_cdm_fct_project_activity]; A --> D[dmn_requesters_rep_fct_requester_balance_daily]; A --> E[dmn_requesters_rep_fct_requester_topup_daily]; A --> F[dmn_requesters_rep_requester_activity_agg_daily]; A --> G[dmn_requesters_rep_app_project_stat_agg_daily]; A --> H[dmn_requesters_rep_platform_and_uhrs_project_stat_agg_daily]; B --> I[dmn_requesters_cdm_dim_project_act]; C --> J[dmn_requesters_cdm_dim_project_metrics_act]; I --> K[dmn_requesters_rep_requester_main_event_agg_act]; J --> K; D --> K; E --> K; F --> K; G --> L[dmn_requesters_rep_project_stat_agg_daily]; H --> L; L --> K;
```

# Плюсы\минусы решения

## Сильные стороны

- › У каждой доменной команды свой набор графов и контроль за ними
- › Ответственность за dag у каждой отдельной доменной команды
- › Легко определять зависимости

## Слабые стороны

- › Крайне сложно проводить backfill в случае ошибки
- › Запуск dag по времени может приводить к излишним ожиданиям

## 4. Выводы

# Что у нас получилось?



# Dbt at Toloka Data Platform

## Ingestion

Event Streaming



Connectors / EL



## Data Governance

Data Quality



## Transformation

Streaming Processing



Batch Processing



## Storage

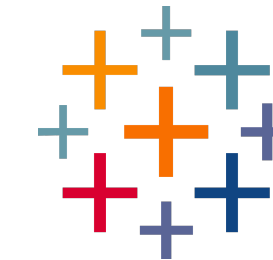


## Consumption

Quering



Business Intelligence



Analytical tools



## Orchestration



Data Catalog \ Data Observability



Monitoring



# Три кита и четыре столпа дата платформы

Что?



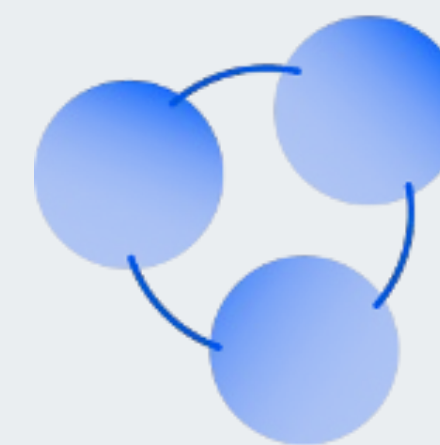
Архитектура  
=  
Data Lakehouse

Почему?



Процессы  
=  
Data mesh

Как?



Технологии  
=  
Modern Data Stack



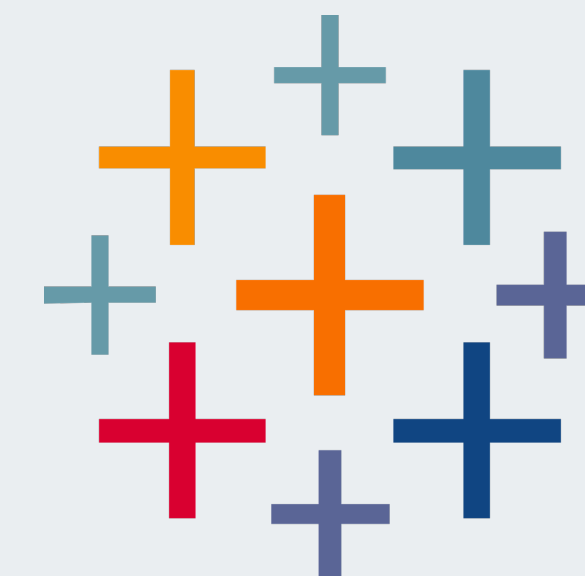
databricks



dbt



airflow



tableau

# Итого

## Сильные стороны

- › dbt – простой инструмент, скрывающий сложности работы с данными
- › dbt+airflow позволяет использовать преимущества обоих продуктов
- › dbt позволяет реализовать data mesh через ввод доменов и analytics engineers
- › Возможность удобной работы как разработчику, так и аналитику

## Слабые стороны

- › Слабости монорепы
- › Дополнительная самописная библиотека для интеграции
- › Сложность во взаимодействии между доменами в случае ошибки

# Thank you!



[jkermakov@toloka.ai](mailto:jkermakov@toloka.ai)



[www.toloka.ai](http://www.toloka.ai)

[iJKos.com](http://iJKos.com)