



# Build reliable data pipelines using Modern Data Stack in the cloud



**Michał Soszko**

Data Analyst / Analytics Engineer  
*GetInData*



**Andrzej Swatowski**

Data Engineer  
*GetInData*



# Readiness check

## Checklist:

- Chrome or Microsoft Edge
- Invitation to join GitLab project
- Please send your google email address to:  
**andrzej.swatowski@getindata.com**
- Visit:  
**<https://gitlab.com/bdtw-mdp-workshop>**

# Today's agenda

## Session 1 - Introduction to Modern Data Stack

- What is Modern Data Stack? Intro
- Key components of MDS
- Core concepts of dbt
- Hands-on exercises

## Session 2 - Simple end-to-end data pipeline

- Transforming data using SQL with dbt
- Data catalog (data search, discovery, profiling)
- Hands-on exercises

## Session 3 - data pipeline scheduling, deployment, advanced dbt concepts

- Apache Airflow as a workflow scheduler
- Advanced concepts of dbt
- Connecting to BI Tool: Looker Studio
- Hands-on exercises

## Today's schedule

- |                 |              |
|-----------------|--------------|
| → 09:00 - 10:30 | - Session 1  |
| → 10:45 - 12:00 | - Session 2a |
| → 13:00 - 14:15 | - Session 2b |
| → 14:30 - 16:00 | - Session 3  |

Lunch Break: 12:00 - 13:00

# Key takeaways

After this workshop you should...

- Know what **Modern Data Stack** is about
- Have a **hands-on** experience with **transforming** data in **dbt**
- Learn how to **orchestrate** your **data pipelines** using best **engineering practices** without IT dependencies
- Know how to use **modern data catalogs** to **search** and **explore** the data
- Be able to prevent **data quality issues** by **data testing & monitoring**

## Session 1 - Introduction to Modern Data Stack

- ➔ What is Modern Data Stack? Intro
- ➔ Key components of MDS
- ➔ Core concepts of dbt
- ➔ Hands-on exercises: creating your own workspace

# What is modern data stack?

Saves time, effort, and money

Business (not IT) focused operating model

Airbyte, dbt, Snowflake, Looker, etc.

Cloud based, flexible & scalable, plug-and-play, no long term commitments

Shift from once-off analytics to operational BI and AI

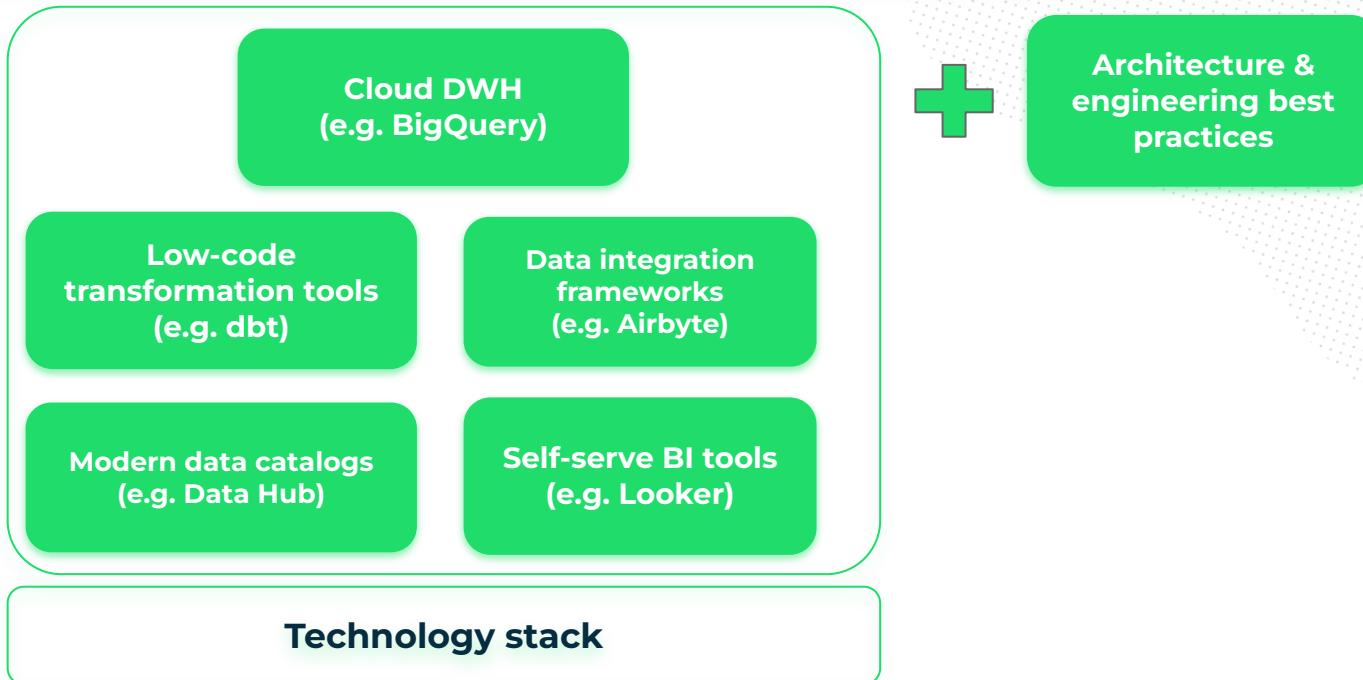
Data governance as a first-class citizen

***Modern Data Stack (MDS) is a set of tools hosted in the cloud that enables an organization for highly efficient data integration (...)***

***MDS creates clean, trustworthy, and always available data that can empower business users to make self-service discoveries, enabling a truly data-driven culture.***

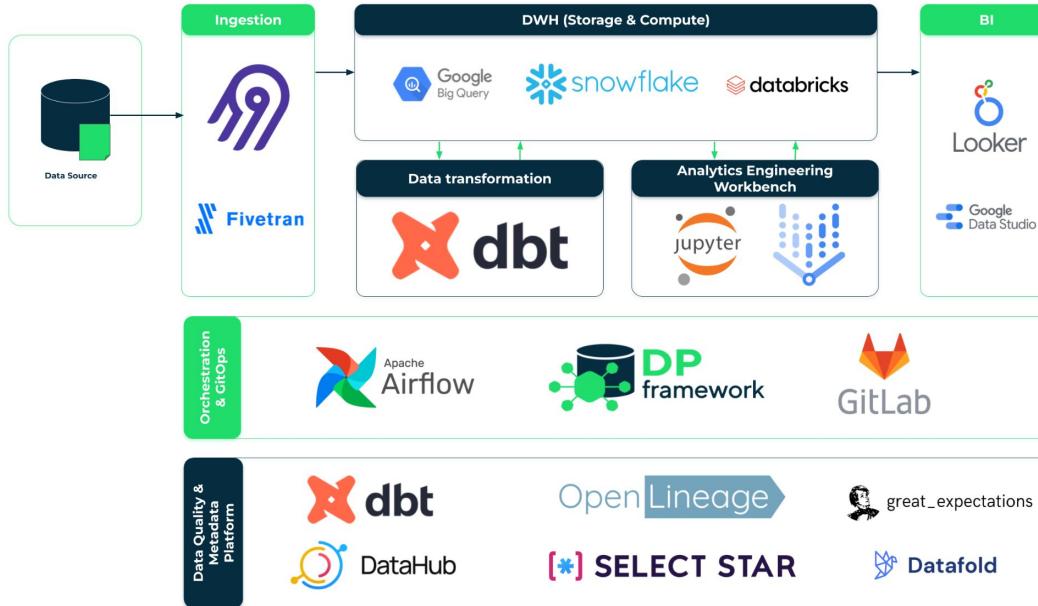
*Source: neptune.ai*

# What is modern data stack?



**The Platform**

# GetInData Modern Data Platform - our framework (GCP example)

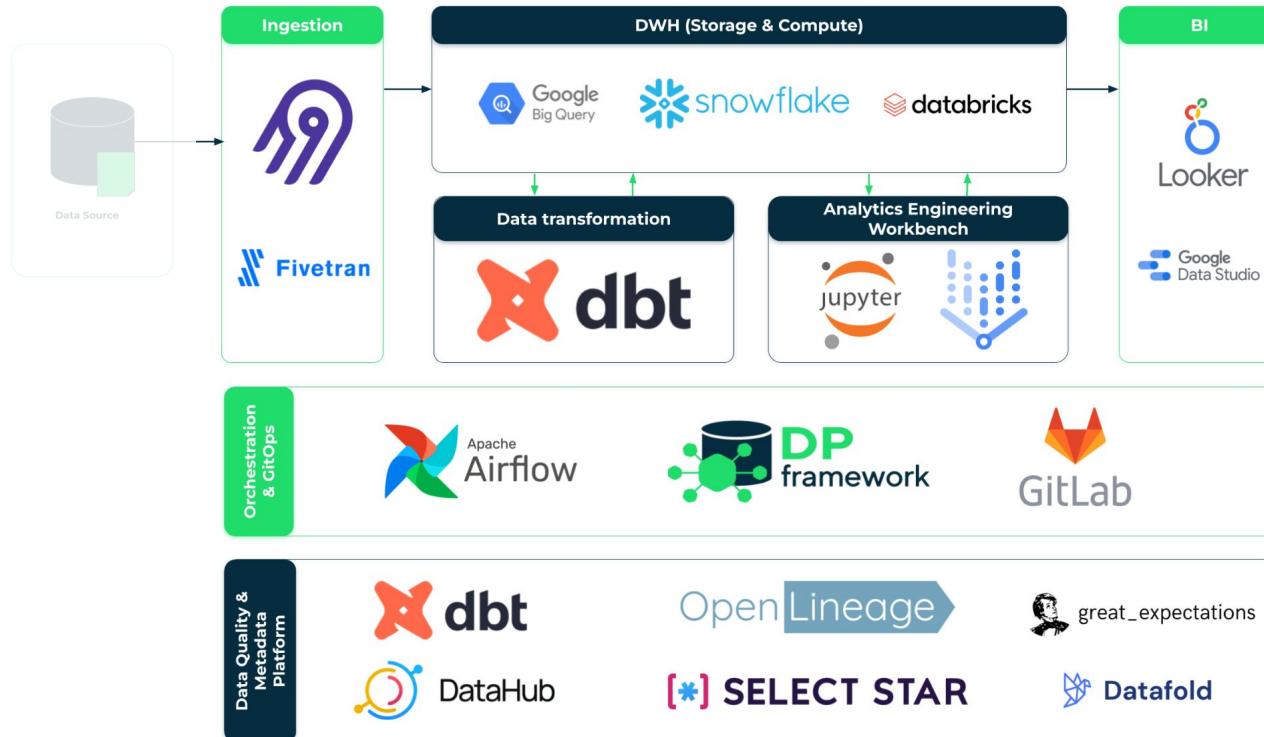


## Solution optimized for:

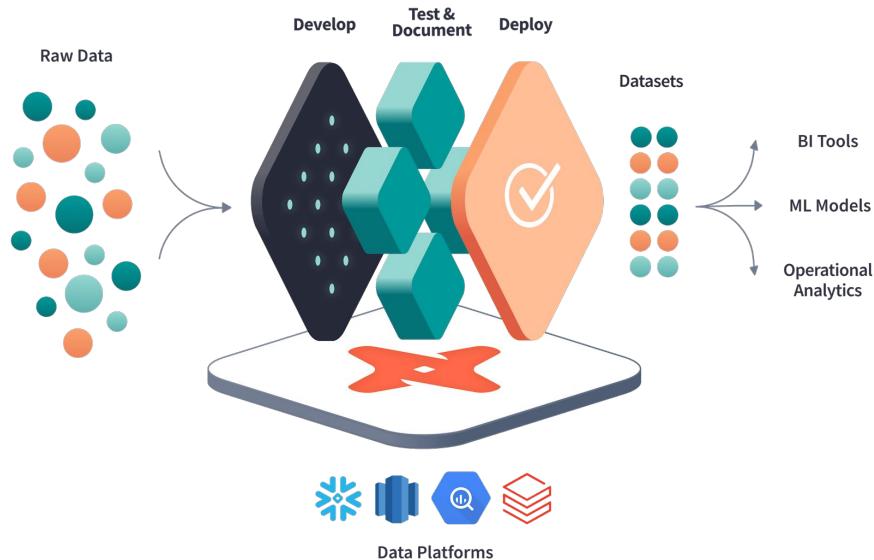
- any cloud
- flexibility & extensibility
- self-service
- data mesh readiness
- automation
- observability & discovery



# GetInData Modern Data Platform - our framework (GCP example)



# Core concepts of dbt

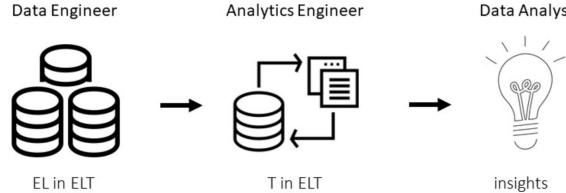


*"dbt™ is a transformation workflow that lets teams quickly and collaboratively deploy analytics code following software engineering best practices like modularity, portability, CI/CD, and documentation. Now anyone who knows SQL can build production-grade data pipelines."*

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

# Core concepts of dbt

## What problem does dbt solve?



- Provides software engineering-grade way of writing data analysis code
  - **Compiler & runner**
  - Automated testing
  - documentation
  - project structuring
  - resolving dependencies
  - environments configuration



Python  
(recently)

# Core concepts of dbt



SQL

```
select *\nfrom table_A\nwhere column_B > 0
```

jinja

```
{{ function( 'var1', 'var2', .... ) }}\n{{ config ('....') }}
```

CLI

```
C:\project\dbt run – select my_model.sql\nC:\project\dbt compile
```

# Core concepts of dbt



## dbt model

Select statement

SQL file



YML file

metaconfig

```
✖ stg_coupon.sql M ✎  
models > staging > dunnhumby > ✖ stg_coupon.sql  
1 {{ config(  
2   enabled=true  
3 )}}  
4  
5 with source as (  
6   select * from {{ source('dunnhumby_source') }},  
7   renamed as [  
8     select  
9       coupon_upc      as coupon_id,  
10      campaign        as campaign_id  
11    ]  
12  )
```

```
! stg_coupon.yml M ✎  
models > staging > dunnhumby > ! stg_coupon.yml  
1 version: 2  
2  
3 models:  
4   - name: stg_coupon  
5     description: '{{ doc("stg_coupon") }}'  
6     columns:  
7       - name: coupon_id  
8         description: '{{ doc("coupon") }}'  
9         tests:  
10           - not_null
```

### DEMOCRACY

- > analyses
- > config
- > dag
- > dbt\_packages
- > docs
- > logs
- > macros
- > models
- > data\_mart
- > intermediate
- > source
- > staging
- > BQML
- > dunnhumby
- ✖ stg\_coupon.sql
- ! stg\_coupon.yml
- .gitkeep
- > seeds
- > snapshots

# Core concepts of dbt

Create models, calculate dependencies

```
{{ source( "schema", "table" ) }}
```

**table\_A.sql:**

```
select *  
from {{ source( "my_schema", "raw_table" ) }}
```

**raw\_table**

```
{{ ref( "dbt_model" ) }}
```

**table\_B.sql:**

```
select *  
from {{ ref( "table_A" ) }}
```

# Core concepts of dbt

Create models, calculate dependencies

```
{% source( "schema", "table" ) %}  
compiled
```

```
{% ref( "dbt_model" ) %}  
compiled
```

```
create or replace table_A as (  
  
select *  
from "my_schema.raw_table"  
)
```

```
create or replace table table_B as (  
  
select *  
from "my_schema.table_A"  
)
```

raw\_table

table\_A

table\_B

Lineage

# Core concepts of dbt

Connection to DWH, different environments

~/.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
```

Project name → jaffle\_shop

Environment → dev

Config (dev, local, prod...) → outputs

DWH type → dev

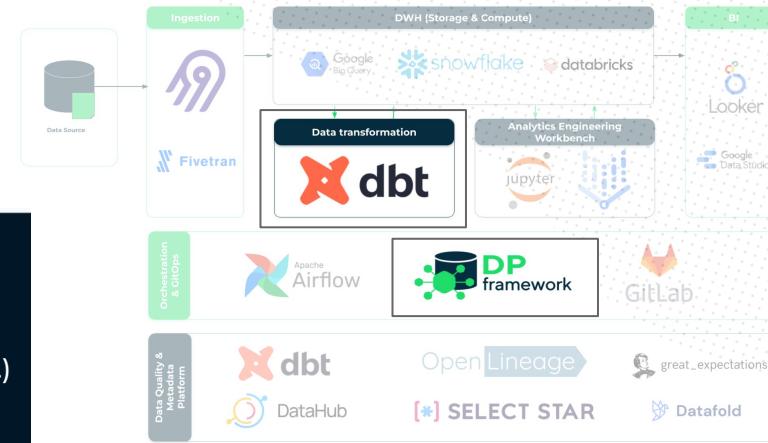
Address → host

Credentials → user

DB name → dbname

Default working schema → schema

Number of threads → threads



In Modern Data Platform:

- dbt instance is installed and configured
- config is stored as a part of DP framework
- DWH access is granted through user's notebook
- However! User should be aware what database name or GCP project-id his dbt project will use

# Core concepts of dbt

## Hands-on:

- Create your workspace
  - DP-customized VertexAI notebook
  - Gitlab project
- Create and initialize dbt data transformation project
- Inspect data in Bigquery

... coffee break

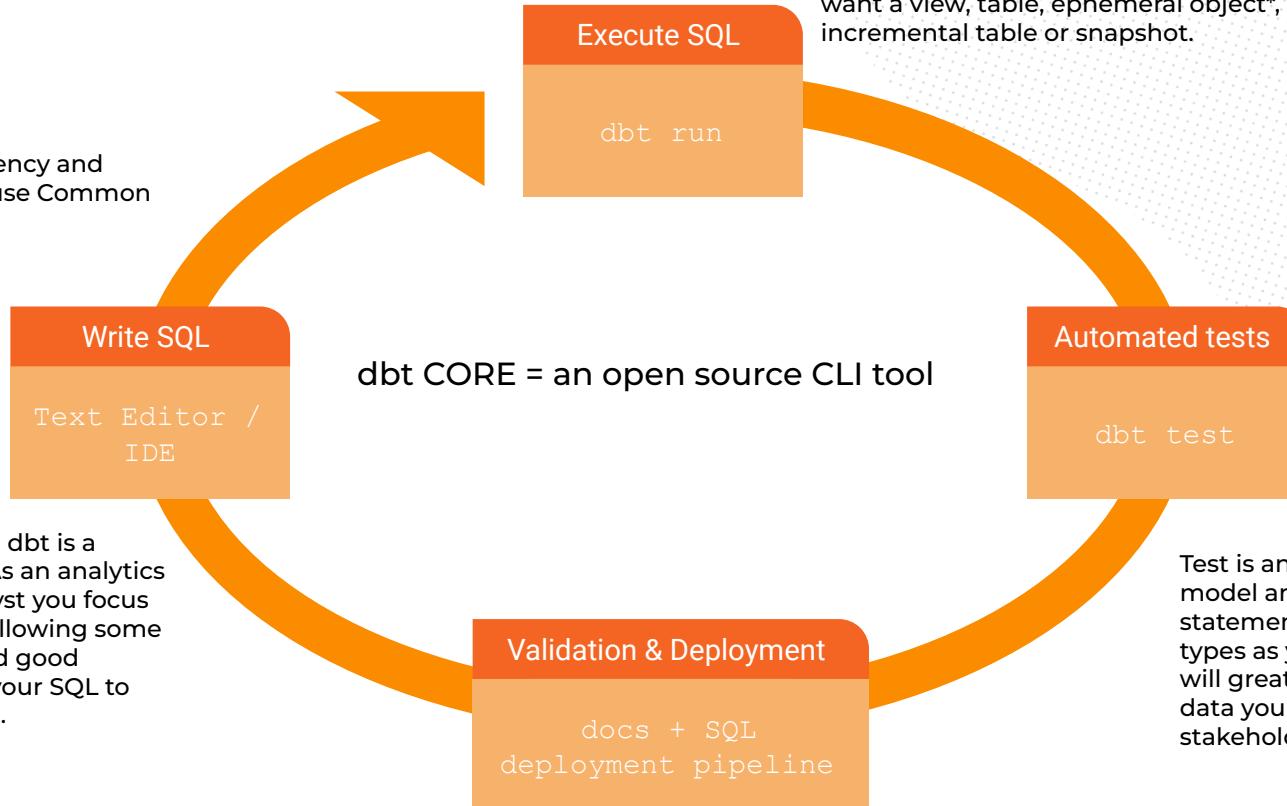
## Session 2 - Simple end-to-end data pipeline

- Transforming data using SQL with dbt
- Data catalog (data search, discovery, profiling)
- Hands-on exercises

# Transforming data with dbt



For better transparency and future debugging use Common Table Expressions



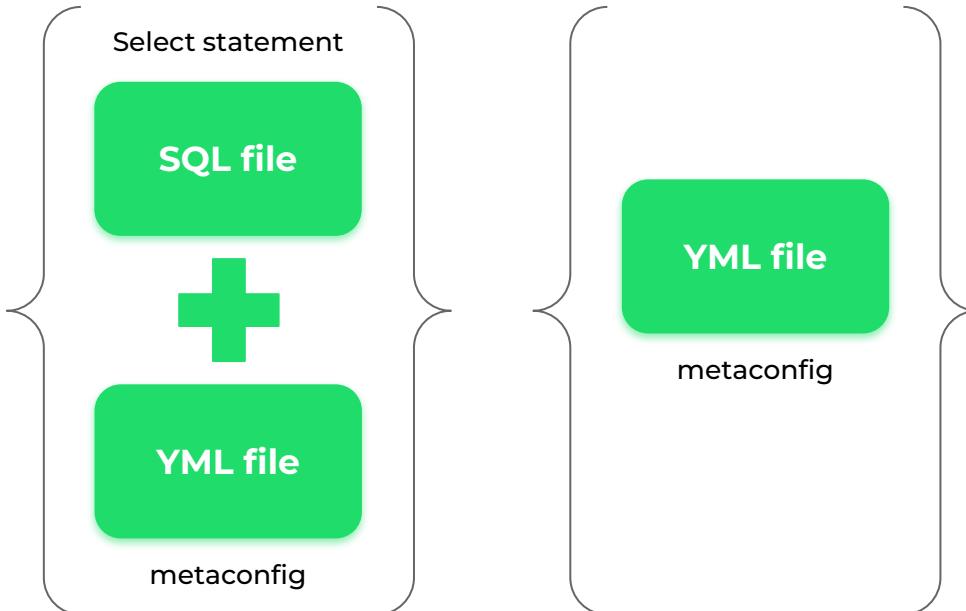
# Transforming data with dbt

## Sources: define your input data

### dbt model

vs

### dbt source



- In dbt `sources` refer to tables that **already exist** in data warehouse
- If data in the source table changes - it is not a result of a dbt transformation
- `dbt sources` are defined in `yaml` file / files alone
- Sources are referenced by using the `source` function
- You can put `tests` and `documentation` on `dbt sources`

# Transforming data with dbt

## Sources: define your input data

EXPLORER      ...      ! source\_order\_items.yml X

JUPYTER

- ✓ jupyter
- ✓ michal-soszko-data-transformations-p...
- > analyses
- > build
- > config
- > dag
- > dbt\_packages
- > docs
- > logs
- > macros
- ✓ models
- ❖ .gitignore
- ! source\_order\_items.yml
- > seeds
- > snapshots
- > targets
- > tests

Michal-Soszko-Data-Transformations-Project > models > ! source\_order\_items.yml

```

1 version: 2
2
3 sources:
4 - name: raw_data
5   tables:
6     - name: order_items
    
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS 1    QUERY RESULTS    JUPYTER    bash - michal-soszko-data-transformations-project

(base) root@c1379ad31d5d:/home/jupyter/michal-soszko-data-transformations-project# dbt compile

09:46:19 Running with dbt=1.0.8

09:46:20 [WARNING]: Configuration paths exist in your dbt\_project.yml file which do not apply to any resources.

There are 1 unused configuration paths:

- models

09:46:20 Found 0 models, 0 tests, 0 snapshots, 0 analyses, 584 macros, 0 operations, 0 seed files, 1 source, 0 exposures, 0 metrics

09:46:20

09:46:20 [WARNING]: Nothing to do. Try checking your model configs and model specification args

09:46:20 Done.

(base) root@c1379ad31d5d:/home/jupyter/michal-soszko-data-transformations-project#

▼ bdtw-mdp-workshop

► External connections

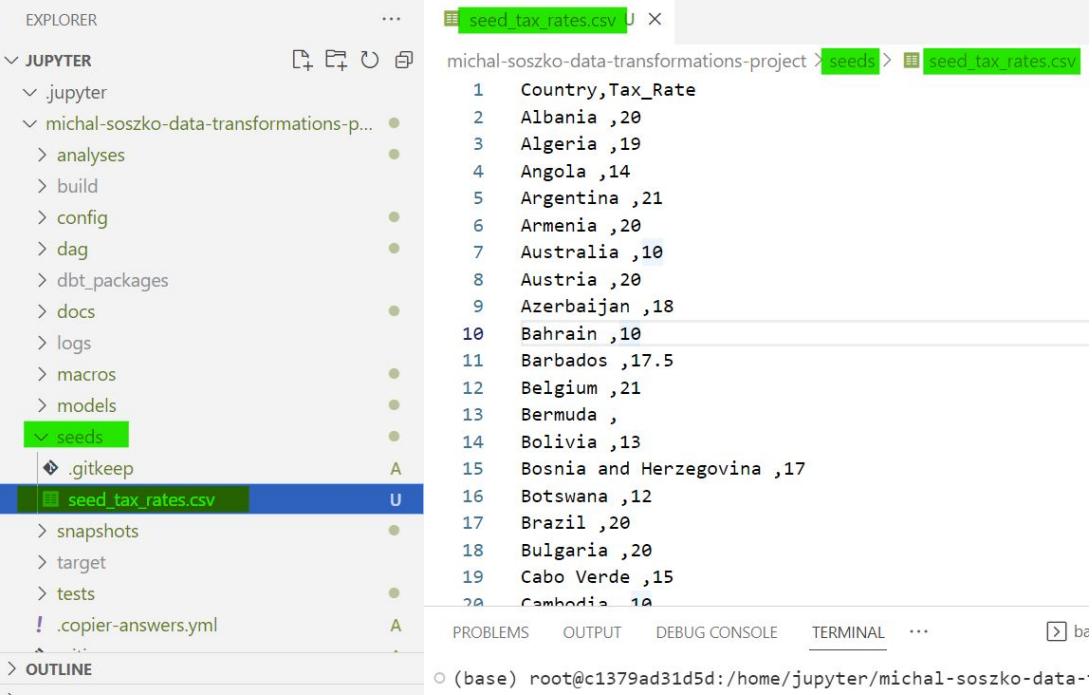
■ raw\_data

■ distribution\_centers



# Transforming data with dbt

## Seeds: add custom CSV data to your project



The screenshot shows a Jupyter Notebook interface with the following structure:

- EXPLORER**: A sidebar showing the project structure:
  - JUPYTER
  - jupyter
  - michal-soszko-data-transformations-p...
    - > analyses
    - > build
    - > config
    - > dag
    - > dbt\_packages
    - > docs
    - > logs
    - > macros
    - > models
    - > seeds (highlighted)
    - .gitkeep
  - > snapshots
  - > target
  - > tests
  - ! .copier-answers.yml
- CELLS**: A list of cells:
  - seed\_tax\_rates.csv
- TERMINAL**: A terminal window showing the command: `(base) root@c1379ad31d5d:/home/jupyter/michal-soszko-data-`

- In dbt, **seeds** are small CSV files you can add to your project
- Put your CSV file into **seeds** folder
- Run command **dbt seed** to ingest the file as a table into the DWH
- You can add metaconfig yaml file to your [seed](#)
- Seeds are referenced by using the **ref** function

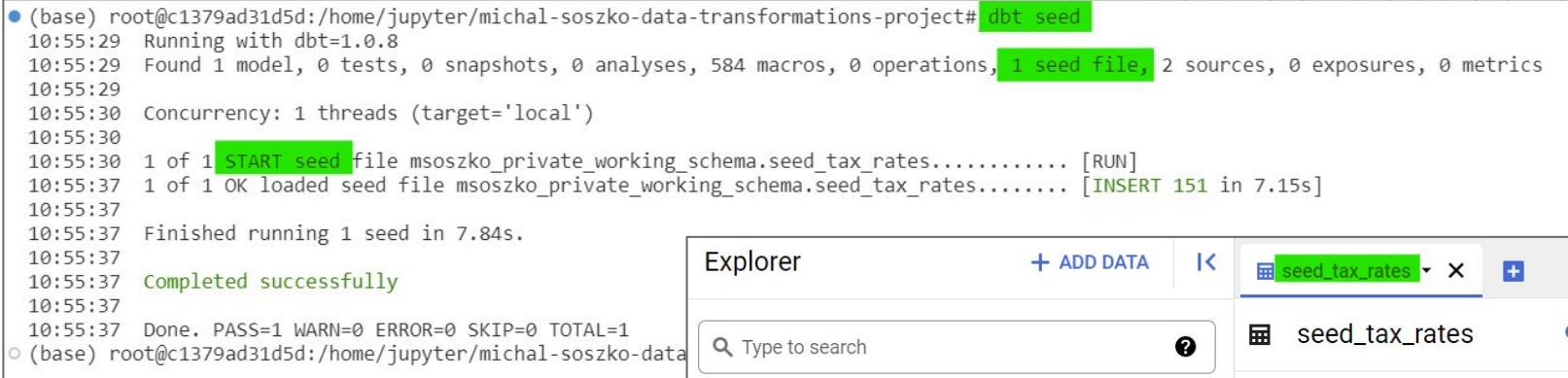
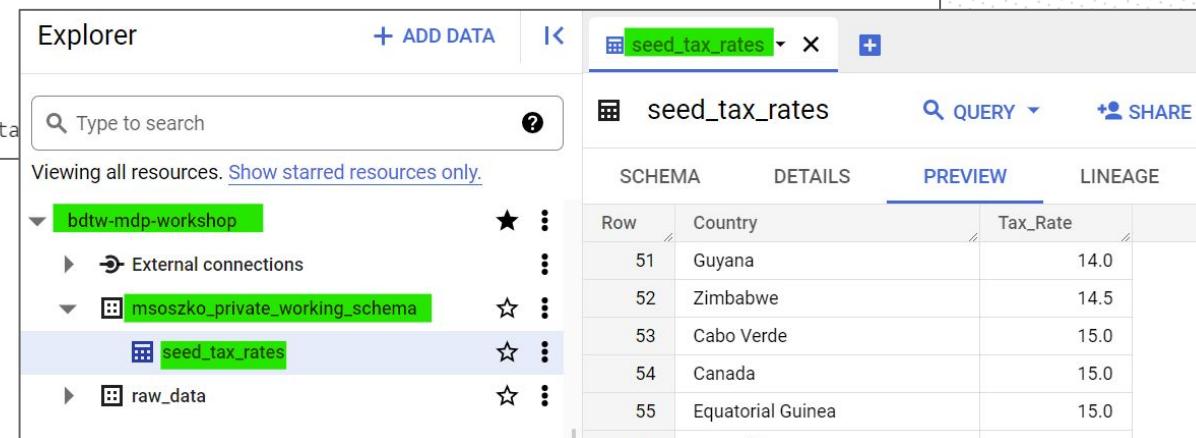
# Transforming data with dbt

## Seeds: add custom CSV data to your project

```
● (base) root@c1379ad31d5d:/home/jupyter/michal-soszko-data-transformations-project# dbt seed
10:55:29  Running with dbt=1.0.8
10:55:29  Found 1 model, 0 tests, 0 snapshots, 0 analyses, 584 macros, 0 operations, 1 seed file, 2 sources, 0 exposures, 0 metrics
10:55:29
10:55:30  Concurrency: 1 threads (target='local')
10:55:30
10:55:30  1 of 1 START seed file msoszko_private_working_schema.seed_tax_rates..... [RUN]
10:55:37  1 of 1 OK loaded seed file msoszko_private_working_schema.seed_tax_rates..... [INSERT 151 in 7.15s]
10:55:37
10:55:37  Finished running 1 seed in 7.84s.
10:55:37
10:55:37  Completed successfully
10:55:37
10:55:37  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
○ (base) root@c1379ad31d5d:/home/jupyter/michal-soszko-data
```

To run a seed type:

```
dbt seed
dbt seed --select seed_file
```

The screenshot shows the dbt Explorer interface. On the left, the 'Explorer' sidebar lists a project structure with a 'bdtw-mdp-workshop' node expanded, showing 'External connections' and a 'msoszko\_private\_working\_schema' node. The 'msoszko\_private\_working\_schema' node has a 'seed\_tax\_rates' seed file selected, indicated by a blue highlight. On the right, the main area displays the 'seed\_tax\_rates' seed file in a table format. The table has columns for 'Row', 'Country', and 'Tax\_Rate'. The data rows are: Row 51 (Guyana, 14.0), Row 52 (Zimbabwe, 14.5), Row 53 (Cabo Verde, 15.0), Row 54 (Canada, 15.0), and Row 55 (Equatorial Guinea, 15.0). The table is currently in 'PREVIEW' mode.

Row	Country	Tax_Rate
51	Guyana	14.0
52	Zimbabwe	14.5
53	Cabo Verde	15.0
54	Canada	15.0
55	Equatorial Guinea	15.0

# Transforming data with dbt

## Models: foundation of a dbt data pipeline

EXPLORER

JUPYTER

- > jupyter
- > michal-soszko-data-transformations-project...
- > analyses
- > build
- > config
- > dag
- > dbt\_packages
- > docs
- > logs
- > macros
- > models**
- .gitkeep**
- model\_order\_items\_with\_country.sql**
- ! model\_order\_items\_with\_country.yml
- ! source\_order\_items.yml
- ! source\_users.yml
- > seeds

...

**model\_order\_items\_with\_country.sql** U X

michal-soszko-data-transformations-project > models > **model\_order\_items\_with\_country.sql** U X

```

1  with order_items as (
2    |   select * from {{ source('raw_data', 'order_items') }}
3  ),
4    users as (
5      |   select * from {{ source('raw_data', 'users') }}
6    )
7
8    select
9      oi.id          as order_item_id,
10     oi.order_id    as order_id,
11     oi.user_id     as user_id,
12     oi.product_id  as product_id,
13     oi.status       as order_status,
14     oi.sale_price   as order_item_sale_price,
15     u.country      as user_country
16   from
17     order_items as oi
18   left join
19     users as u on oi.user_id = u.id

```

- Model is a select statement stored in a SQL file.
- Model metaconfig **yml** file is **optional**
- Executing a model with the **dbt run** command will create a **table / view** in the DWH
- Models are referenced within dbt using the **ref** function
- There are different materialization modes to choose from:
  - **table**
  - **view**
  - **incremental table**
  - **ephemeral**

# Transforming data with dbt

## Models: foundation of a dbt data pipeline

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   1   QUERY RESULTS   JUPYTER   bash - michal-soszko-data-transformations-project

```
● (base) root@c1379ad31d5d:/home/jupyter/michal-soszko-data-transformations-project# dbt run
10:48:15 Running with dbt=1.0.8
10:48:16 Found 1 model, 0 tests, 0 snapshots, 0 analyses, 584 macros, 0 operations, 1 seed file, 2 sources, 0 exposures, 0 metrics
10:48:16
10:48:17 Concurrency: 1 threads (target='local')
10:48:17
10:48:17 1 of 1 START view model msoszko_private_working_schema.model_order_items_with_country
10:48:18 1 of 1 OK created view model msoszko_private_working_schema.model_order_items_with_country
10:48:18
10:48:18 Finished running 1 view model in 2.35s.
10:48:18
10:48:18 Completed successfully
10:48:18
10:48:18 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
● (base) root@c1379ad31d5d:/home/jupyter/michal-soszko-data-transformations-project#
```

**Explorer**

+ ADD DATA                     +

Type to search   ?

Viewing all resources. [Show starred resources only.](#)

- bdtw-mdp-workshop
  - External connections
- msoszko\_private\_working\_schema
  - model\_order\_items\_with\_country**   ★   
  - seed\_tax\_rates
- raw\_data
  - distribution\_centers
  - events
  - inventory\_items
  - order\_items
  - orders

**model\_order\_items\_with\_country**      

QUERY

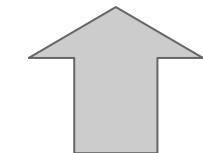
SCHEMA	DETAILS	LINEAGE
Filter Enter property name or value		
<input type="checkbox"/> Field name	Type	Mode
<input type="checkbox"/> <u>order_item_id</u>	INTEGER	NULLABLE
<input type="checkbox"/> <u>order_id</u>	INTEGER	NULLABLE
<input type="checkbox"/> <u>user_id</u>	INTEGER	NULLABLE
<input type="checkbox"/> <u>product_id</u>	INTEGER	NULLABLE
<input type="checkbox"/> <u>order_status</u>	STRING	NULLABLE
<input type="checkbox"/> <u>order_item_sale_price</u>	FLOAT	NULLABLE
<input type="checkbox"/> <u>user_country</u>	STRING	NULLABLE

To run a model type:

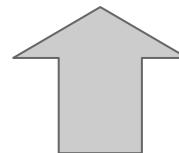
```
dbt run
dbt run --select your_model
dbt run --select +your_model+
```

# Transforming data with dbt

## Models: foundation of a dbt data pipeline



dbt sources



dbt model

```
{{ source ('schema', 'table') }}
```

Explorer + ADD IK

Type to search ?

Viewing workspace resources.  
SHOW STARRED ONLY

- bdtw-mdp-workshop
  - External connections
  - msoszko\_private\_working\_schema
    - model\_order\_items\_with\_country
    - seed\_tax\_rates
- raw\_data
  - distribution\_centers
  - events
  - inventory\_items
  - order\_items
  - orders
  - products
  - users

# Transforming data with dbt

## Models: foundation of a dbt data pipeline

EXPLORER

- JUPYTER
  - jupyter
  - michal-soszko-data-transformations-project
    - analyses
    - build
    - config
    - dag
    - dbt\_packages
    - docs
    - logs
    - macros
    - models
      - .gitkeep
      - ~~model\_order\_items\_with\_country.sql~~
      - ~~model\_order\_items\_with\_country.yml~~
      - ~~model\_order\_items\_with\_tax.sql~~ U
      - ~~source\_order\_items.yml~~
      - ~~source\_users.yml~~
    - seeds
      - .gitkeep
      - seed\_tax\_rates.csv
    - snapshots
    - target
    - tests
    - .copier-answers.yml
    - .gitignore

model\_order\_items\_with\_tax.sql

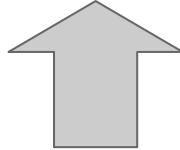
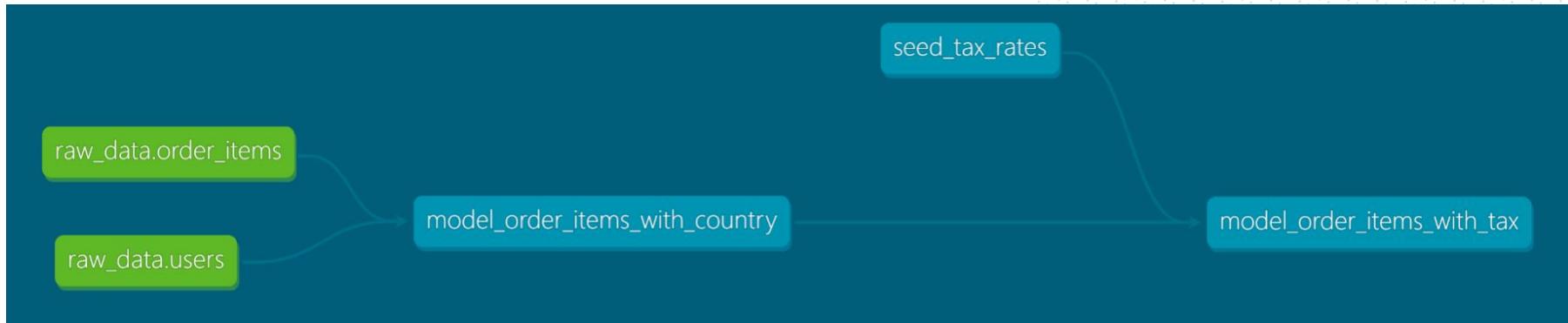
```

1 {{ config(
2   persist_docs={"relation": true, "columns": true}
3 ) }}
4
5
6 with _order_items_with_country as (
7   select * from {{ ref('model_order_items_with_country') }}
8 ),
9 tax_rates as (
10  select * from {{ ref('seed_tax_rates') }}
11 )
12
13 select
14   order_item_id,
15   order_id,
16   user_id,
17   product_id,
18   order_status,
19   order_item_sale_price,
20   user_country,
21   tr.Tax_Rate      as tax_rate,
22
23   round(order_item_sale_price * (tr.Tax_Rate / 100), 2)    as order_item_sale_VAT
24
25 from
26   _order_items_with_country as oi
27 left join
28   tax_rates as tr on oi.user_country = trim(tr.Country)

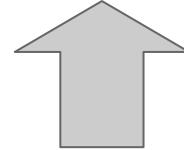
```

# Transforming data with dbt

## Models: foundation of a dbt data pipeline

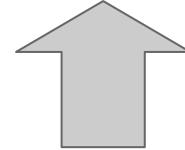


dbt sources



dbt model  
referencing sources

```
 {{ source( 'schema', 'table' ) }}
```



dbt models  
referencing other dbt entities

```
 {{ ref( 'model_name' ) }}  
 {{ ref( 'seed_name' ) }}
```

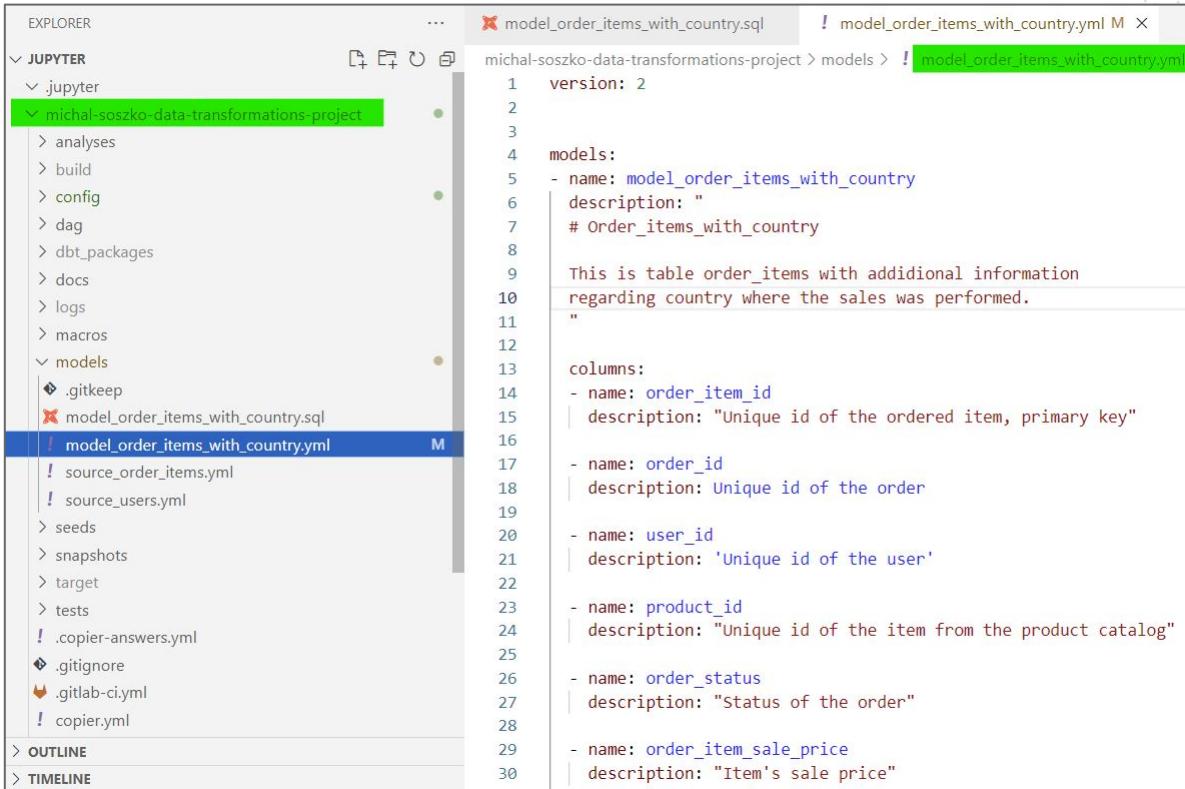
# Coffee Break...!

...and hands-on exercises:

Open `Session_2a.md` and create your local data pipeline

# Transforming data with dbt

## Models: metaconfig YAML files



```

version: 2

models:
  - name: model_order_items_with_country
    description: "
      # Order_items_with_country

      This is table order_items with additional information
      regarding country where the sales was performed.
    "
    columns:
      - name: order_item_id
        description: "Unique id of the ordered item, primary key"
      - name: order_id
        description: Unique id of the order
      - name: user_id
        description: 'Unique id of the user'
      - name: product_id
        description: "Unique id of the item from the product catalog"
      - name: order_status
        description: "Status of the order"
      - name: order_item_sale_price
        description: "Item's sale price"

```

- Metaconfig **YAML** file stores descriptive information about the model.
- The **YAML** file is optional
- Allows documenting all **dbt** entities (models, seeds & sources) at their source
- Allows adding **tests** to all **dbt** entities
- **Whitespace sensitive!**
- You can store all model, source and seed metadata in one big yaml file (however, we recommend splitting them to keep the highest granularity - ie. 1 SQL file per 1 YAML)

# Transforming data with dbt

## Data quality in dbt: tests

EXPLORER

JUPYTER

- > docs
- > logs
- > macros
- > models
- > seeds
- > snapshots
- > target
- tests**
- ◆ .gitkeep
- ☒ assertion\_there\_is\_at\_most\_10\_items\_in\_order.sql A
- ! .copier-answers.yml

assertion\_there\_is\_at\_most\_10\_items\_in\_order.sql 4 ×

michal-soszko-data-transformations-project > tests > assertion\_there\_is\_at\_most\_10\_items\_in\_order.sql 4 ×

```

1   with tested_order_items as (
2     select * from {{ ref('model_order_items_with_tax') }}
3   ),
4   precalc as (
5     select
6       *,
7       count(*) over (partition by order_id) as cnt_windowed
8     from tested_order_items
9   )
10  select
11    *
12  from precalc
13  where cnt_windowed >= 10
14

```

To run tests type:

dbt test

dbt test --select model

dbt test --select assertion

- In dbt test is an assertion about your model
- it is a SELECT statement
- Test **fails** whenever at least one record is returned
- Test **passes** whenever no data is returned
- There are two types of tests: **generic** and **singular**

# Transforming data with dbt

## Data quality in dbt: tests

### Success

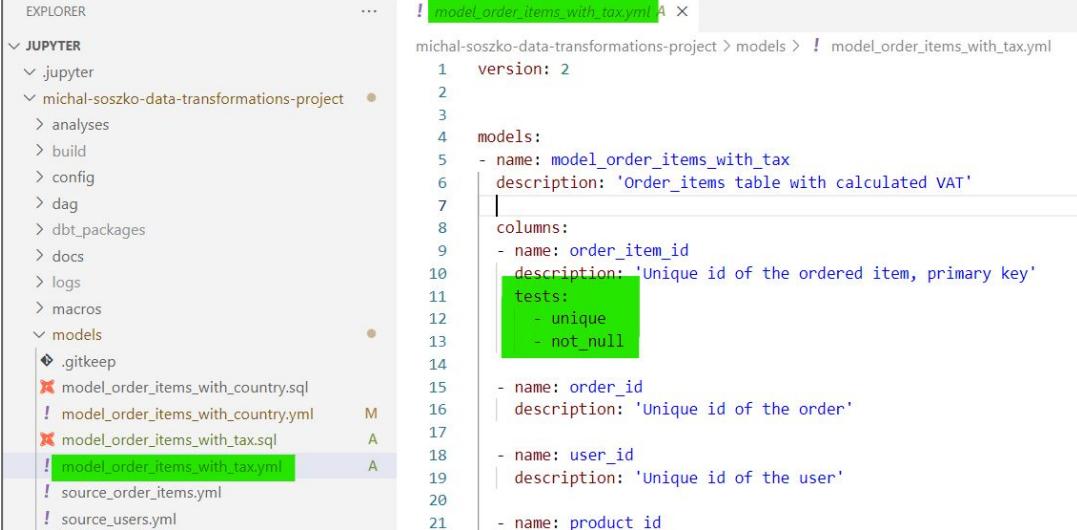
```
● (base) root@0037bbec830b:/home/jupyter/michal-soszko-data-transformations-project# dbt test --select assertion_there_is_at_most_10_items_in_order
20:45:08  Running with dbt=1.0.8
20:45:08  Found 2 models, 5 tests, 0 snapshots, 0 analyses, 584 macros, 0 operations, 1 seed file, 2 sources, 0 exposures, 0 metrics
20:45:08
20:45:09  Concurrency: 1 threads (target='local')
20:45:09
20:45:09  1 of 1 START test assertion_there_is_at_most_10_items_in_order..... [RUN]
20:45:10  1 of 1 PASS assertion_there_is_at_most_10_items_in_order..... [PASS in 1.49s]
20:45:10
20:45:10  Finished running 1 test in 1.89s.
20:45:10
20:45:10  Completed successfully
20:45:10
20:45:10  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
○ (base) root@0037bbec830b:/home/jupyter/michal-soszko-data-transformations-project# █
```

### Failure

```
✖ (base) root@0037bbec830b:/home/jupyter/michal-soszko-data-transformations-project# dbt test --select assertion_there_is_at_most_10_items_in_order
20:47:36  Running with dbt=1.0.8
20:47:36  Found 2 models, 5 tests, 0 snapshots, 0 analyses, 584 macros, 0 operations, 1 seed file, 2 sources, 0 exposures, 0 metrics
20:47:36
20:47:37  Concurrency: 1 threads (target='local')
20:47:37
20:47:37  1 of 1 START test assertion_there_is_at_most_10_items_in_order..... [RUN]
20:47:38  1 of 1 FAIL 182316 assertion_there_is_at_most_10_items_in_order..... [FAIL 182316 in 1.68s]
20:47:38
20:47:38  Finished running 1 test in 2.01s.
20:47:38
20:47:38  Completed with 1 error and 0 warnings:
20:47:38
20:47:38  Failure in test assertion_there_is_at_most_10_items_in_order (tests/assertion_there_is_at_most_10_items_in_order.sql)
20:47:38    Got 182316 results, configured to fail if != 0
20:47:38
20:47:38    compiled SQL at target/compiled/michal_soszko_data_transformations_project/tests/assertion_there_is_at_most_10_items_in_order.sql
20:47:38
20:47:38  Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
○ (base) root@0037bbec830b:/home/jupyter/michal-soszko-data-transformations-project# █
```

# Transforming data with dbt

## Data quality in dbt: tests



```

EXPLORER
...
JUPYTER
  ✓ jupyter
  michal-soszko-data-transformations-project
    > analyses
    > build
    > config
    > dag
    > dbt_packages
    > docs
    > logs
    > macros
    ✓ models
      .gitkeep
      ✘ model_order_items_with_country.sql
      ! model_order_items_with_country.yml M
      ✘ model_order_items_with_tax.sql
      ! model_order_items_with_tax.yml A
      ! source_order_items.yml
      ! source_users.yml

! model_order_items_with_tax.yml A X
michal-soszko-data-transformations-project > models > ! model_order_items_with_tax.yml
1   version: 2
2
3
4   models:
5     - name: model_order_items_with_tax
6       description: 'Order_items table with calculated VAT'
7
8   columns:
9     - name: order_item_id
10       description: 'Unique id of the ordered item, primary key'
11       tests:
12         - unique
13         - not_null
14
15     - name: order_id
16       description: 'Unique id of the order'
17
18     - name: user_id
19       description: 'Unique id of the user'
20
21     - name: product_id

```

- Generic tests can be applied to any column
  - There are 4 main generic tests:
    - **not\_null**
    - **unique**
    - **accepted\_values**
    - **relationships**
  - Users can define their own generic test as jinja macros
  - dbt community has served hundreds of custom generic tests; you can find them in many public packages:
    - **dbt-labs / dbt\_utils**
    - **calogica / dbt\_expectations**
    - and many more:
- <https://hub.getdbt.com/>

# Transforming data with dbt

## Data quality in dbt: tests

EXPLORER

JUPYTER

michal-soszko-data-transformations-project

- > analyses
- > build
- > config
- > dag
- > dbt\_packages
- > docs
- > logs
- macro
- generic\_tests
  - is\_positive\_value.sql
  - .gitkeep
- generate\_schema\_name.sql
- models
- seeds

! model\_order\_items\_with\_tax.yml M is\_positive\_value.sql X

```

1  {% test is_positive_value(model, column_name) %}
2
3  with validation as (
4    select
5      {{ column_name }} as tested_value
6      from {{ model }}
7  ),
8  validation_errors as (
9    select
10       tested_value
11      from validation
12      where tested_value < 0
13 )

```

description: 'Unique id of the item from the product catalog'

name: order\_status  
description: 'Status of the order'

name: order\_item\_sale\_price  
description: "Item's sale price"

tests:
 - is\_positive\_value

name: user\_country  
description: 'Country where the sales was performed'

name: tax\_rate  
description: 'Tax rate for a given country'

- Generic tests can be applied to any column
- There are 4 main generic tests:
  - not\_null
  - unique
  - accepted\_values
  - relationships
- Users can define their own generic test as jinja macros

hundreds of  
n find them in

tions

on

# Transforming data with dbt

## Documentation and graphical lineage

**dbt**

Overview

Project Database raw\_data

**model\_order\_items\_with\_country**

SCHEMA DETAILS LINEAGE

**View info**

View ID	bdtw-mdp-workshop.msoszko_private_working_schema.model_order_items_with_country
Created	Mar 20, 2023, 11:09:02PM UTC+1
Last modified	Mar 20, 2023, 11:09:03PM UTC+1
View expiration	NEVER
Use Legacy SQL	false
Description	# Order_Items_with_Country This is table order_items with additional information regarding country where the sales was performed.

Labels

Filter Enter property name or value

Field name	Type	Mode	Collation	Default Value	Policy Tags	Description
<input type="checkbox"/> <a href="#">order_item_id</a>	INTEGER	NULLABLE				Unique id of the ordered item, primary key
<input type="checkbox"/> <a href="#">order_id</a>	INTEGER	NULLABLE				Unique id of the order
<input type="checkbox"/> <a href="#">user_id</a>	INTEGER	NULLABLE				Unique id of the user
<input type="checkbox"/> <a href="#">product_id</a>	INTEGER	NULLABLE				Unique id of the item from the product catalog
<input type="checkbox"/> <a href="#">order_status</a>	STRING	NULLABLE				Status of the order
<input type="checkbox"/> <a href="#">order_item_sale_price</a>	FLOAT	NULLABLE				Item's sale price
<input type="checkbox"/> <a href="#">user_country</a>	STRING	NULLABLE				Country where the sales was performed.

Depends On

```
models:
- name: model_order_items_with_country
  description: "# Order_Items_with_Country"
  This is table order_items with additional information regarding country where the sales was performed.
  "
  columns:
    - name: order_item_id
      description: "Unique id of the ordered item, primary key"
    - name: order_id
      description: Unique id of the order
    - name: user_id
      description: 'Unique id of the user'
    - name: product_id
      description: "Unique id of the item from the product catalog"
    - name: order_status
      description: "Status of the order"
    - name: order_item_sale_price
      description: "Item's sale price"
```

# Transforming data with dbt

## Documentation and graphical lineage



- In regular **dbt-core** instance accessing the local documentation requires two commands:
  - dbt docs generate**
  - dbt docs serve**

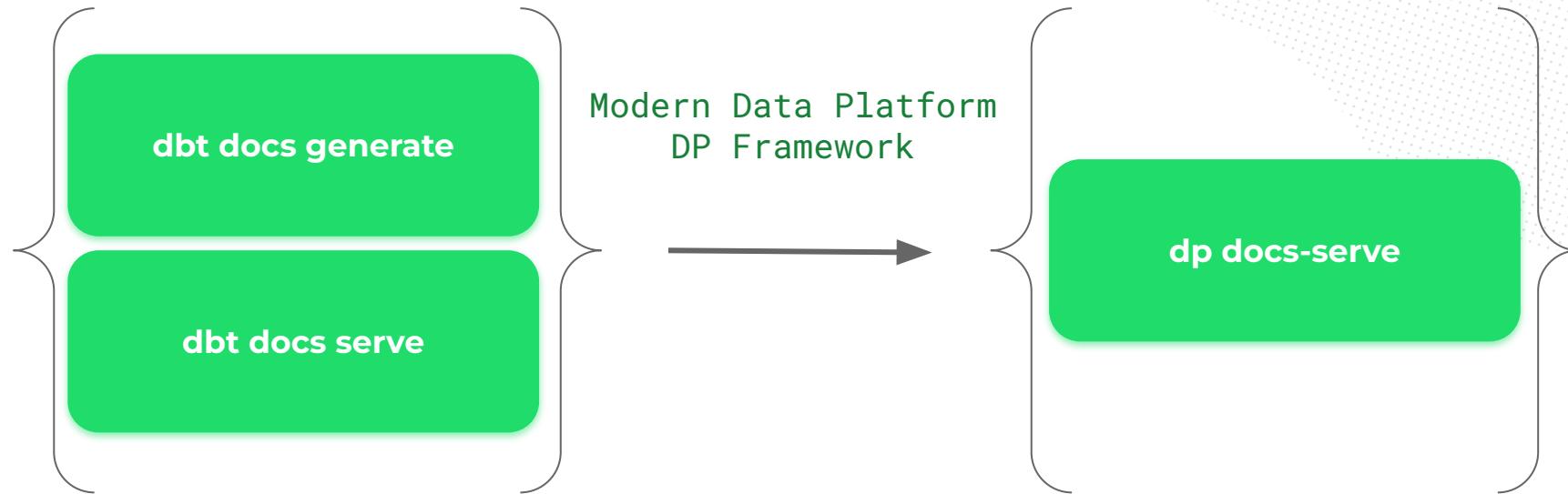
- dbt** comes with its own documentation generator
- Rendered documentation can be viewed via web-browser
- Model** and project documentation (table / column level descriptions, project description etc.) is stored as a part of the dbt entity metaconfig YAML file.
- In MDP, you can access your local documentation using the **dp docs-serve** command

```

BI is disabled
dbt docs serve --port 9328
21:34:01  Running with dbt=1.0.8
21:34:01  Serving docs at 0.0.0.0:9328
21:34:01  To access from your browser, navigate to:  http://localhost:9328
21:34:01
21:34:01
21:34:01  Press Ctrl+C to exit.
127.0.0.1 - - [20/Mar/2023 21:34:05] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Mar/2023 21:34:05] "GET /manifest.json?cb=1679348044835 HTTP/1.1" 200 -
127.0.0.1 - - [20/Mar/2023 21:34:05] "GET /catalog.json?cb=1679348044835 HTTP/1.1" 200 -
  
```

# Transforming data with dbt

## Documentation and graphical lineage



# Coffee Break...!

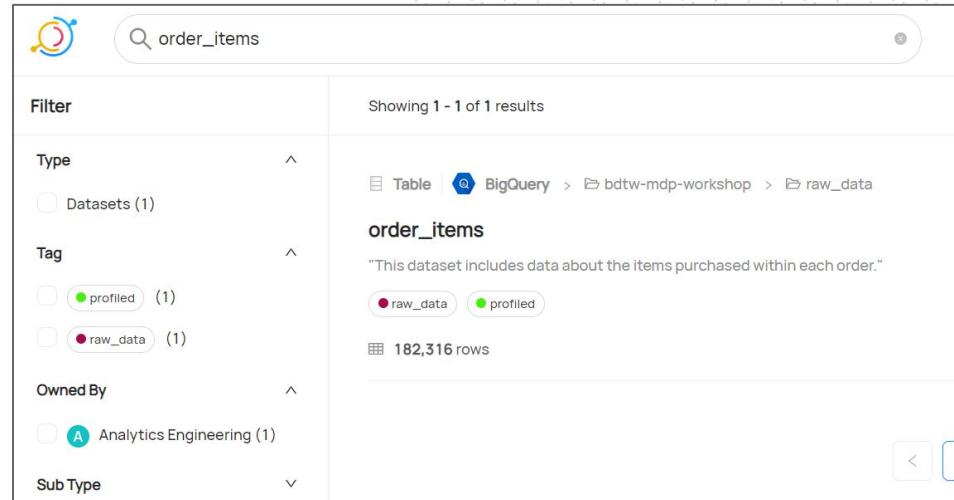
...and hands-on exercises:

Open `Session_2b.md` and ensure high quality of your models by adding documentation and tests to your local pipeline!

# Data Catalog

## Single source of knowledge about your data

- Not everyone in the organization have / wants an access to the data warehouse
- Data catalog: an external tool storing information related to data.
- In MDP data catalog it is synchronized with the code repository - it is always up to date with recent changes in tables.
- Data catalogs usually have rich metadata grouping and categorization functionalities, as well as search capabilities, which makes it the first place where the user will find both technical and business information about the data.

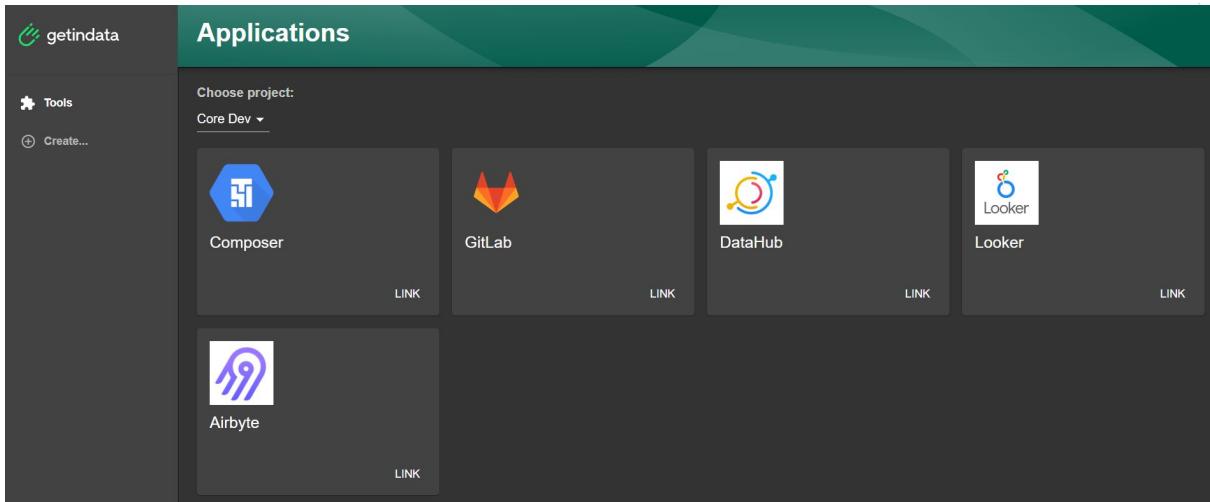


The screenshot shows a data catalog interface with the following details:

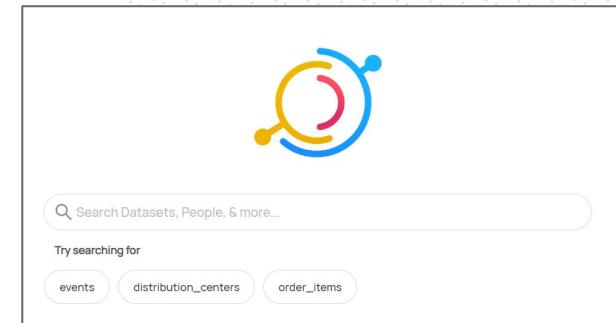
- Search Bar:** Displays "order\_items".
- Filter Section:**
  - Type:** Datasets (1)
  - Tag:** profiled (1), raw\_data (1)
  - Owned By:** Analytics Engineering (1)
  - Sub Type:**
- Dataset Details:**
  - Table:** BigQuery > bdwtw-mdp-workshop > raw\_data
  - Name:** order\_items
  - Description:** "This dataset includes data about the items purchased within each order."
  - Status:** raw\_data, profiled
  - Rows:** 182,316 rows

# Data Catalog

## Single source of knowledge about your data



The screenshot shows the 'Applications' section of the getindata Data Catalog. On the left, there's a sidebar with the getindata logo, 'Tools' (with a gear icon), and a 'Create...' button. The main area has a dark background with a teal header bar containing the title 'Applications'. Below it, a sub-header says 'Choose project:' followed by a dropdown menu set to 'Core Dev'. There are five application cards: 'Composer' (blue hexagon icon), 'GitLab' (red fox icon), 'DataHub' (yellow and blue circular icon), 'Looker' (white square with colorful dots), and 'Airbyte' (purple stylized icon). Each card has a 'LINK' button below it.



The screenshot shows the DataHub search interface. It features a large, stylized magnifying glass icon composed of yellow and blue arcs. Below it is a search bar with the placeholder text 'Search Datasets, People, & more...'. Underneath the search bar, there's a section titled 'Try searching for' with three buttons: 'events', 'distribution\_centers', and 'order\_items'.

## Datahub: demo

# Data Catalog

## Single source of knowledge about your data

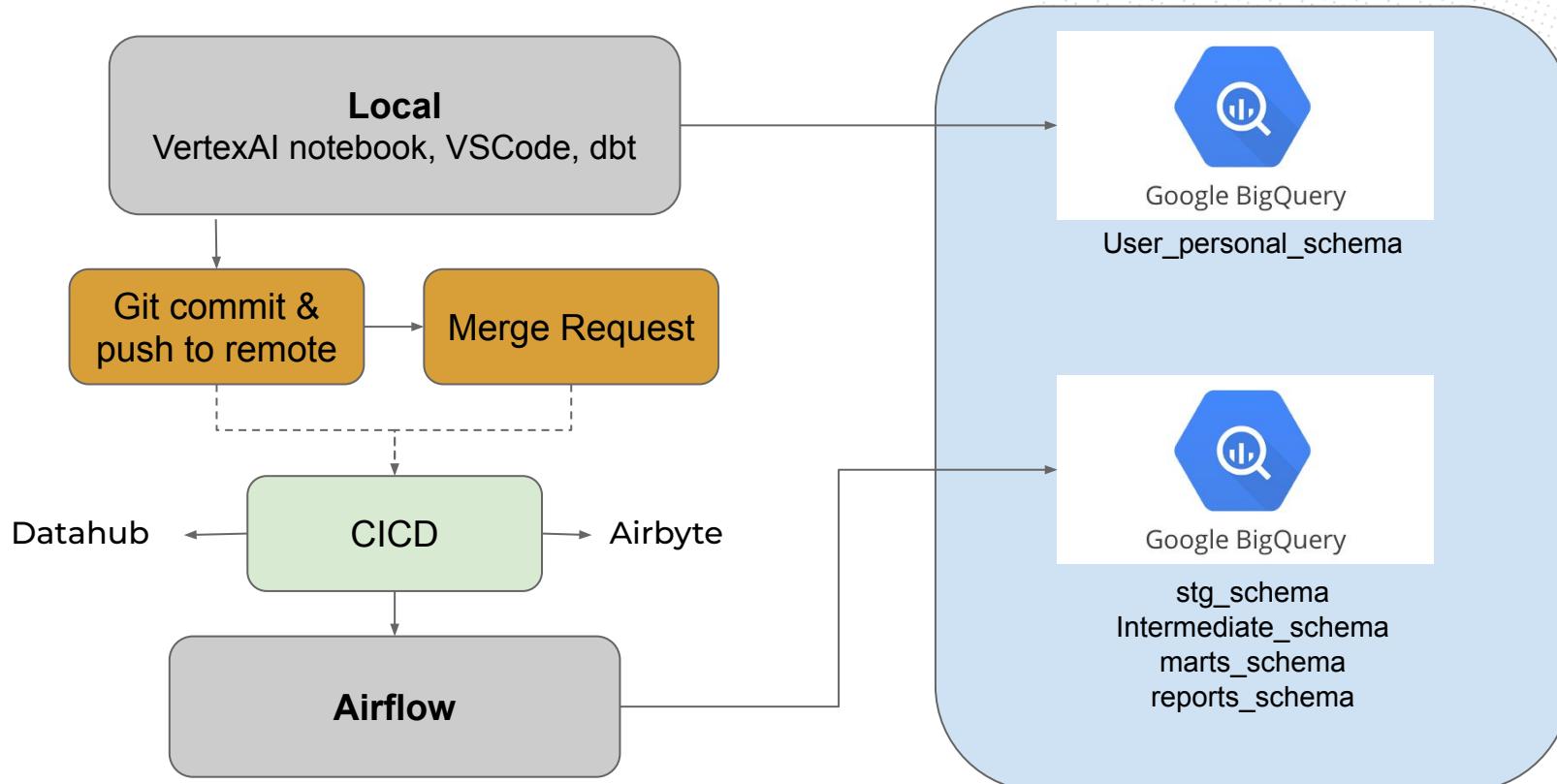
Datahub access:

User name & Password: -> check on Slack

## **Session 3 - Scheduling, deployment, advanced dbt concepts**

- Productionalization of dbt models
- Advanced concepts of dbt
- Connecting to BI Tool: Looker Studio
- Hands-on exercises

# Transferring from local model development to DEV / PROD



# dbt compile & the manifest file

- Whenever one runs `dbt run` or `dbt test`, `dbt compile` runs first.
- `dbt compile` parses SQL and YAML files, *renders* Jinja templates in those, and generates `manifest.json` file.
- `dbt` uses `manifest.json` file to decide in what order it should run the models, which one should be materialized as tables and which as views, etc.

```
{  
  "metadata": {  
    "dbt_schema_version": "https://schemas.getdbt.com/dbt/manifest/v5.json",  
    "dbt_version": "1.4.0",  
    "generated_at": "2023-03-20T08:51:59.212464Z",  
    "invocation_id": "9ae5f35d-d9a0-4f71-9136-0bfc52128bea",  
    "env": {},  
    "project_id": "cf6ef8b0d94b1214ee1c83dc66bb4eb5",  
    "user_id": "19c06c51-fdaf-4be6-95fa-6d3724afa750",  
    "send_anonymous_usage_stats": true,  
    "adapter_type": "bigquery"  
  },  
  "nodes": {  
    "model.example_schema.users_basic_info": { ... },  
    "test.example_schema.unique_users_basic_info_id.a92193f514": { ... },  
    "test.example_schema.not_null_users_basic_info_id.76846ee519": { ... }  
  },  
  "sources": {  
    "source.example_schema.raw_ecommerce_eu.users": { ... }  
  },  
  "macros": { ... },  
  "docs": { ... },  
  "exposures": {},  
  "metrics": {},  
  "selectors": {},  
  "disabled": {},  
  "parent_map": { ... },  
  "child_map": { ... }  
}
```

# dbt compile & the manifest file manifest.json's node example

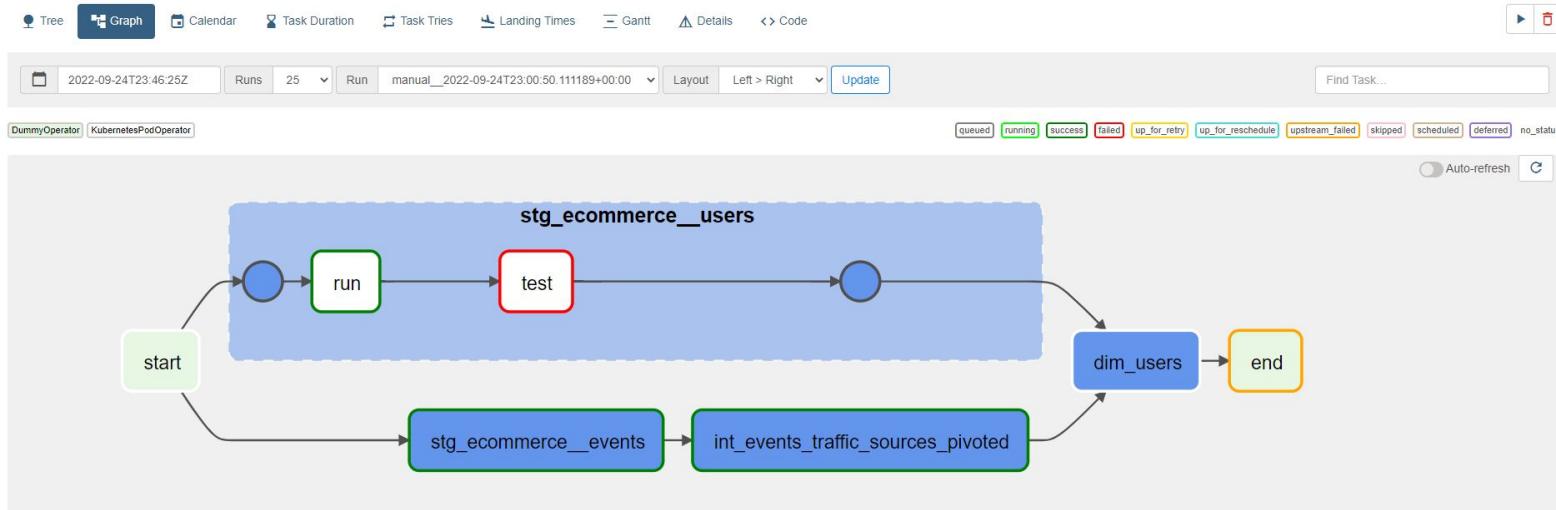
```
"model.example_schema.users_basic_info": {  
    "raw_sql": "...",  
    "compiled": true,  
    "resource_type": "model",  
    "depends_on": {  
        "macros": [  
            "macro.dbt.run_hooks",  
            "macro.dbt.statement",  
            "macro.dbt.persist_docs"  
        ],  
        "nodes": [  
            "source.example_schema.raw_ecommerce_eu.users"  
        ]  
    },  
    "config": {  
        ...  
    },  
    "materialized": "table"  
},  
  
    "database": "bdtw-mdp-workshop",  
    "schema": "example_schema",  
    "name": "users_basic_info",  
    "sources": [  
        [  
            "raw_ecommerce_eu",  
            "users"  
        ]  
    ],  
    "description": "This is a table with basic information about app users: first name, last name & email",  
    "columns": { ... },  
    "compiled_sql": "...",  
    "relation_name":  
        "`bdtw-mdp-workshop`.`example_schema`.`users_basic_info`"  
    }  
}
```

# Directed Acyclic Graph

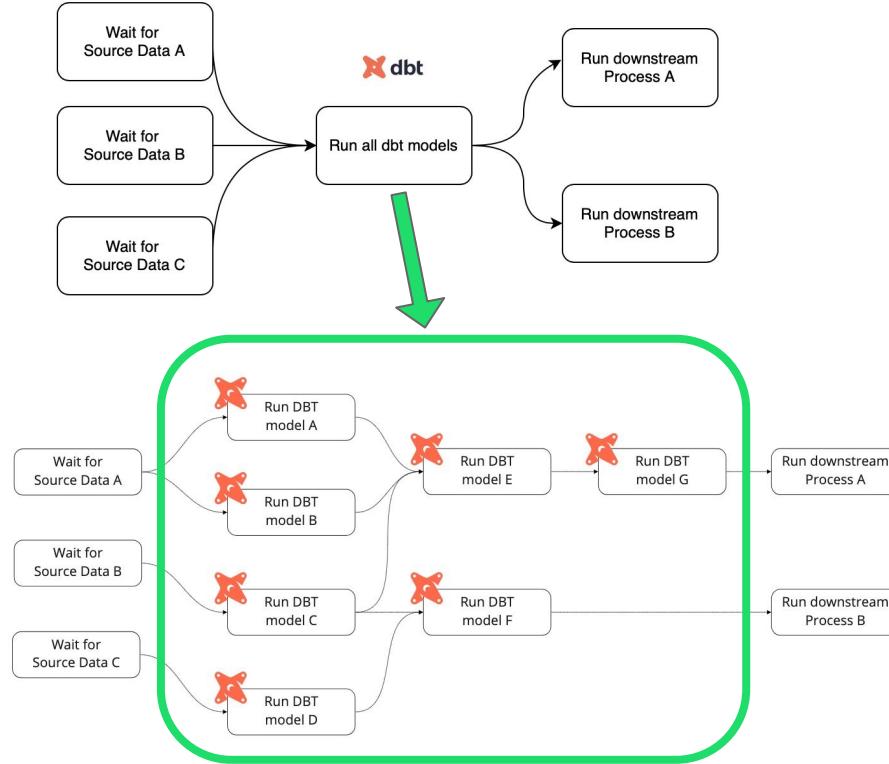
## aka show me the order of operations



# Apache Airflow as a workflow scheduler

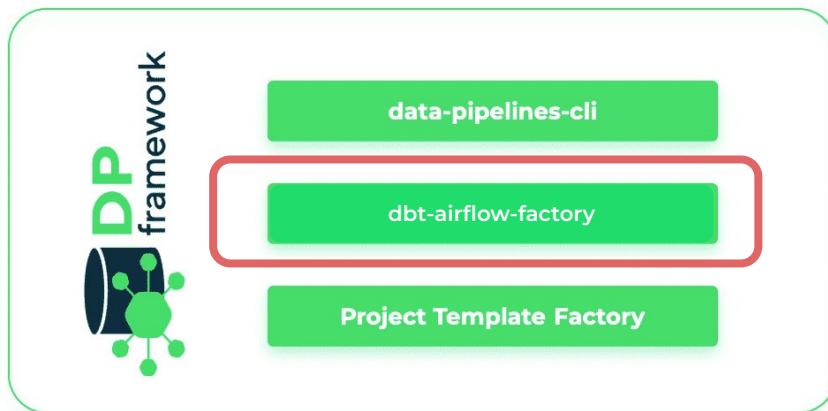


# Apache Airflow as a workflow scheduler



- dbt and Airflow integrations are very common
- run jobs on a cron schedule
- view logs
- configure error notifications ( Slack )
- you can orchestrate your dbt jobs to wait for data sources

# Apache Airflow as a workflow scheduler



- dbt-airflow-factory is a library for parsing DBT manifest files and building Airflow DAG
- The library is expected to be used inside an Airflow environment with a Kubernetes image referencing dbt
- Although the tools was created by GetInData and are used in their project it is open-sourced and everyone is welcome to use and contribute to make it better and even more useful

# Apache Airflow as a workflow scheduler

```

from datetime import datetime, timedelta
import json
from typing import cast, Any, Dict

from airflow import DAG
from airflow.operators.bash import BashOperator

dag = DAG(
    dag_id="dbt_dag",
    start_date=datetime(2023, 3, 28),
    description="An Airflow DAG to invoke simple dbt commands",
    schedule=timedelta(days=1),
)

local_filepath = "/usr/local/airflow/dags/dbt/target/manifest.json"
with open(local_filepath) as f:
    data = cast(Dict[str, Any], json.load(f))

dbt_tasks = {}
for node_id in data["nodes"].keys():
    if node_id.split(".")[-1] != "model":
        continue

    dbt_tasks[node_id] = BashOperator(
        task_id=node_id,
        bash_command=f"dbt run --select {node_id.split('.')[-1]}",
        dag=dag
    )
    dbt_tasks[f"{node_id}_test"] = BashOperator(
        task_id=node_id,
        bash_command=f"dbt test --select {node_id.split('.')[-1]} --indirect-selection=cautious",
        dag=dag
    )

    dbt_tasks[node_id] >> dbt_tasks[f"{node_id}_test"]

for node_id, node in data["nodes"].items():
    if node_id.split(".")[-1] != "model":
        continue

    for upstream_node in node["depends_on"]["nodes"]:
        upstream_node_type = upstream_node.split(".")[-1]
        if upstream_node_type == "model":
            dbt_tasks[upstream_node] >> dbt_tasks[node_id]

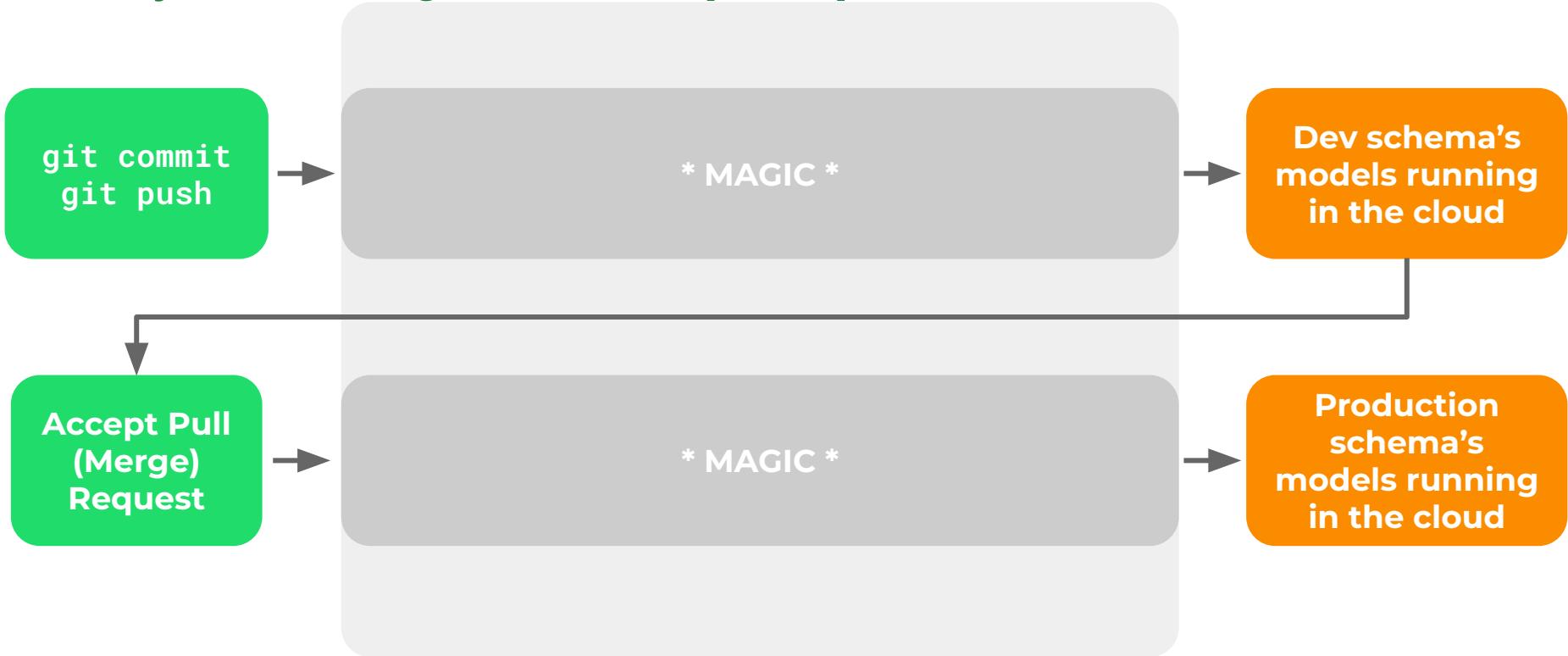
```

```

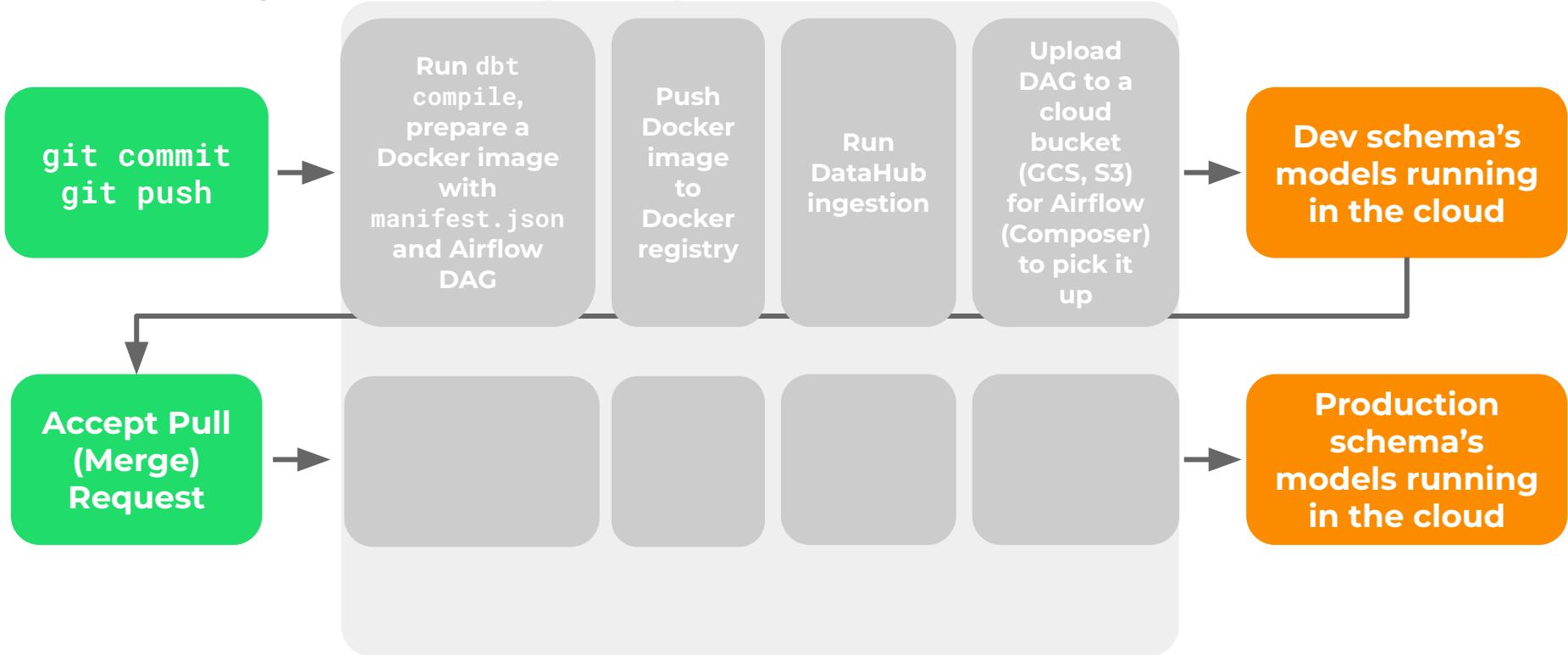
1 from os import path
2 from airflow.models import Variable
3 from dbt_airflow_factory.airflow_dag_factory import AirflowDagFactory
4
5
6 dag = AirflowDagFactory(path.dirname(path.abspath(__file__)), Variable.get("env")).create()

```

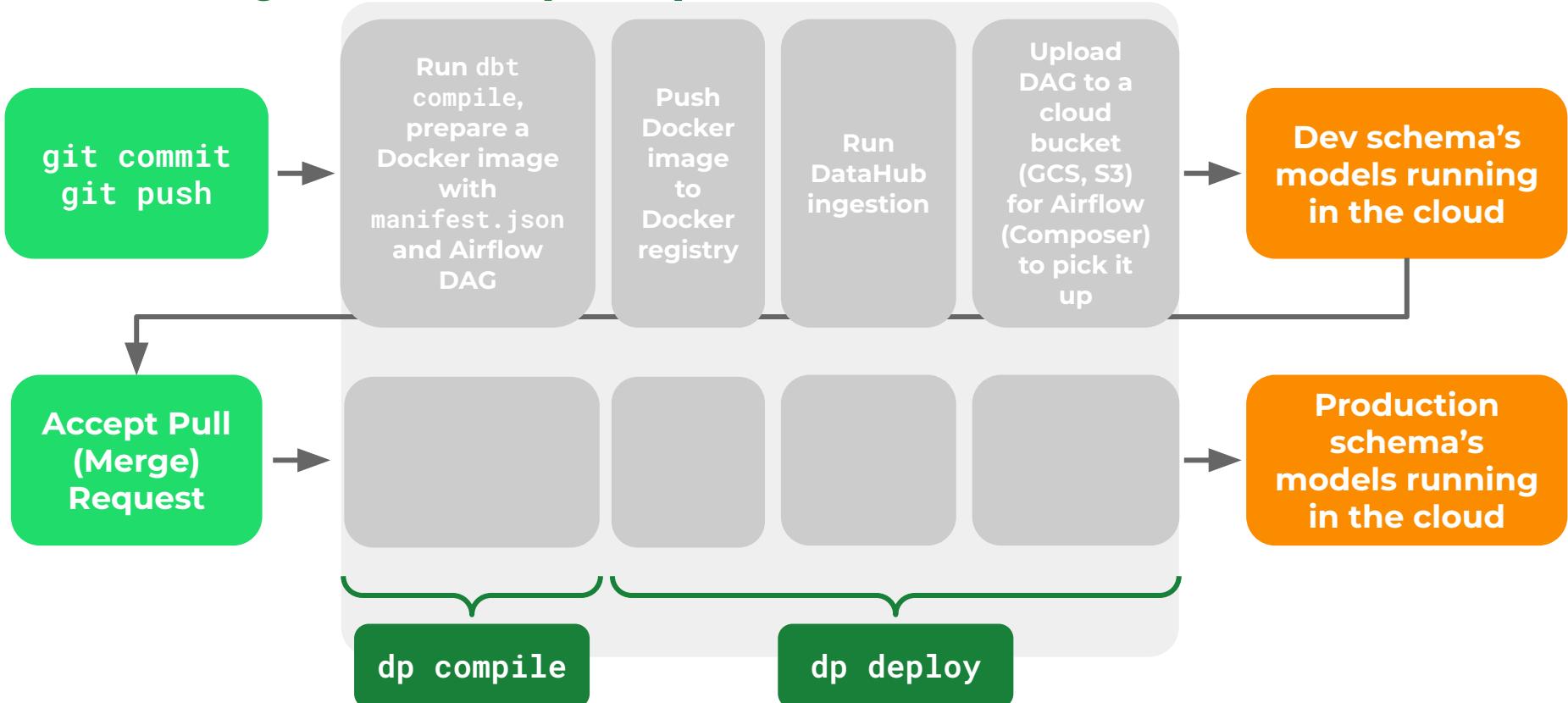
## Analytics Engineer's perspective



# Data Engineer's perspective

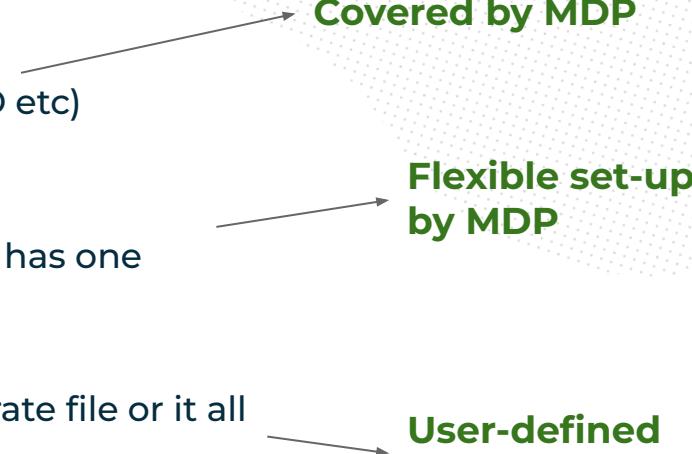


# Data Engineer's perspective



# Advanced concepts of dbt

## Core dbt configuration

- **profiles.yml**
    - configuring your environments (local, DEV, PROD etc)
  - **dbt\_project.yml**
    - Specify operations/metaconfig. Every dbt project has one
  - **specific dbt object .yml files**
    - Each dbt object can have its config done in separate file or it all can be grouped in schema.yml files in respective directories
- 
- Covered by MDP**
- Flexible set-up by MDP**
- User-defined objects**

# Advanced concepts of dbt

## Model materialization

- **Table**
  - Model deployed as table in DWH, recalculating on every dbt run
- **View**
  - Model deployed as view in DWH, recreating on every dbt run
- **Ephemeral**
  - Model deployed is NOT deployed in DWH but it's transferred to downstream SELECT statements in form of CTE
- **Incremental**
  - Model deployed as table, appending **new records** if detected per every dbt run
- **Snapshot\*\***
  - deployed as table, appending new records when there has been a **change** in existing data, executed in separate run

# Advanced concepts of dbt

## Structuring the project

### Raw

- dbt sources
- **no** transformations allowed
- tests allowed
- dbt does not materialize sources
- sources  $=/$  models

### Snapshots

- SCD-2
- can serve as a pre-staging layer
- JOINs - **strongly not** recommended
- aggregations - **strongly not** recommended

### Staging

- “Engineering” layer
- sublayers: data sources
- base and staging models
- Initial cleaning & transformations
- applying column naming conventions
- concatenating
- splitting
- filtering
- applying simple formulas
- formatting
- JOINs- **not** recommended
- aggregations - **strongly not** recommended
- tests recommended
- docs recommended

### Intermediate

- “Transition” layer
- sublayers: business teams
- advanced transformations
- pivoting
- aggregations
- JOINs
- applying advanced business logic
- advanced formulas and functions
- tests recommended
- docs recommended

### Data Mart

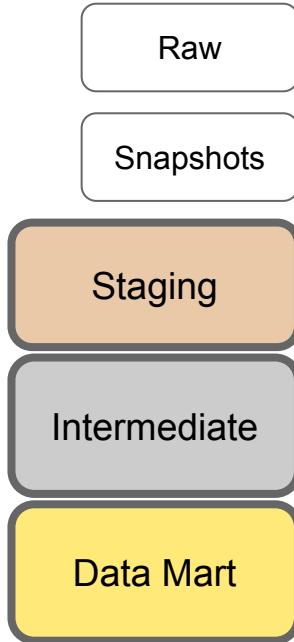
- “Business” layer
- sublayers: business teams
- final JOINs
- final aggregations
- final filtering
- quality checked (tests are obligatory)
- documentation is obligatory (detailed description for columns, tables, tests, exposures etc)

# Advanced concepts of dbt

## Structuring the project

```

models
└── 00_raw_data
    └── ! source_order_items.yml
    └── ! source_users.yml
└── 01_staging
    └── ✘ stg_order_items.sql
    └── ✘ stg_tax_rates.sql
    └── ✘ stg_users.sql
└── 02_intermediate
    └── ✘ int_order_items_with_country.sql
└── 03_mart
    └── ✘ dm_order_items.sql
    └── ! dm_order_items.yml
  
```



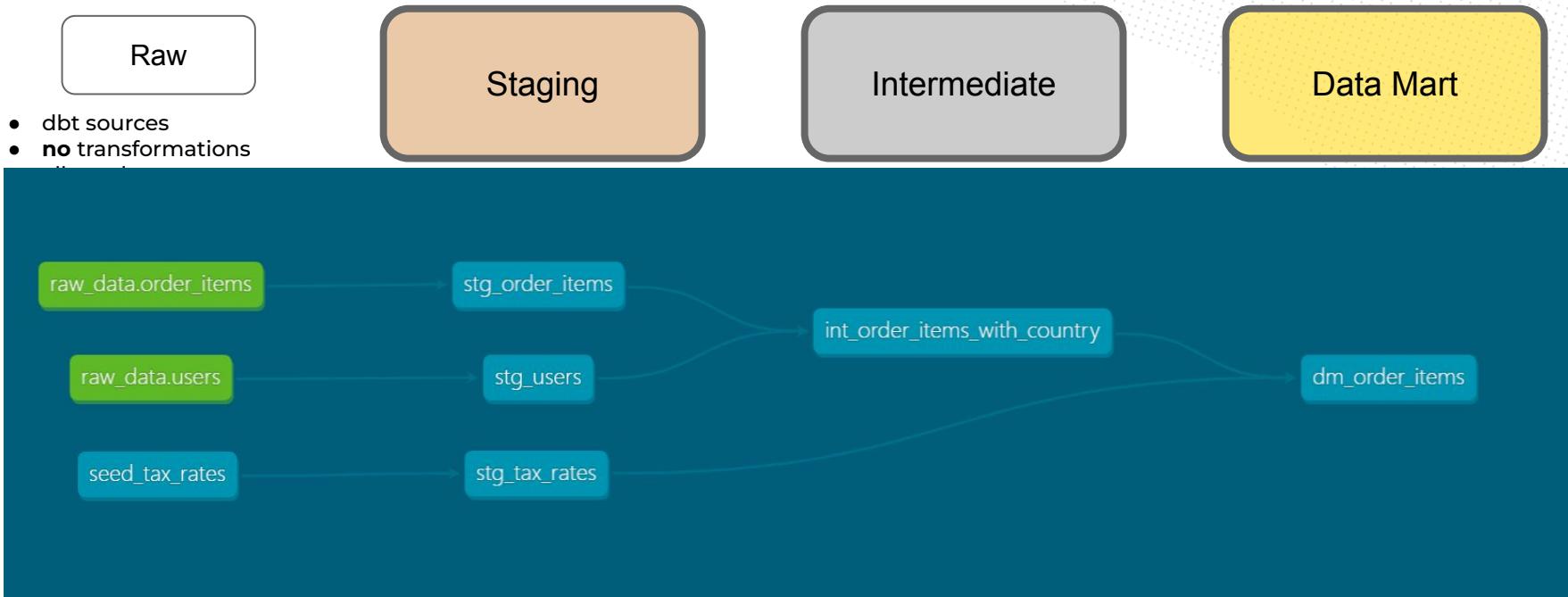
dbt\_project.yml

```

1 # Name your project! Project names should contain only lowercase characters
2 # and underscores. A good package name should reflect your organization's
3 # name or the intended use of these models
4 name: 'michal_soszko_data_transformations_project'
5 version: '1.0.0'
6 config-version: 2
7
8 # This setting configures which "profile" dbt uses for this project.
9 profile: 'bigquery'
10
11 # These configurations specify where dbt should look for different types of files.
12 # The 'model-paths' config, for example, states that models in this project can be
13 # found in the "models/" directory. You probably won't need to change these!
14 model-paths: [ "models" ]
15 docs-paths: [ "docs" ]
16 analysis-paths: [ "analyses" ]
17 test-paths: [ "tests" ]
18 seed-paths: [ "seeds" ]
19 macro-paths: [ "macros" ]
20 snapshot-paths: [ "snapshots" ]
21
22 target-path: "target" # directory which will store compiled SQL files
23 clean-targets: # directories to be removed by 'dbt clean'
24   - "target"
25   - "dbt_packages"
26
27 models:
28   michal_soszko_data_transformations_project:
29     staging:
30       +materialized: view
31       +schema: michal_soszko_staging
32     intermediate:
33       +materialized: ephemeral
34       +schema: michal_soszko_intermediate
35     mart:
36       +materialized: table
37       +schema: michal_soszko_mart
  
```

# Advanced concepts of dbt

## Structuring the project



# Advanced concepts of dbt

## Naming convention

Raw

source\_<dataset>\_\_ <table>  
src\_<dataset>\_\_<table>  
src\_<table>

Snapshots

snapshot\_<dataset>\_\_<table>  
snapshot\_<table>

Staging

staging\_<dataset>\_\_ <table>  
stg\_<dataset>\_\_<table>  
stg\_<table>  
base\_<dataset>\_\_ <table>  
base\_<table>

Intermediate

int\_<descriptive\_name>  
example:  
int\_active\_users\_with\_mapped\_payments\_agg\_daily

Data Mart

dm\_<name>  
mart\_<name>  
dm\_<business\_unit>\_\_<name>  
mart\_<business\_unit>\_\_<name>  
<name>  
<business\_unit>\_<name>  
dm\_<short\_descriptive\_name>  
mart\_<short\_descriptive\_name>  
examples:  
dm\_users, mart\_active\_subscriptions\_aggd,  
mart\_marketing\_campaigns

## DEMO:

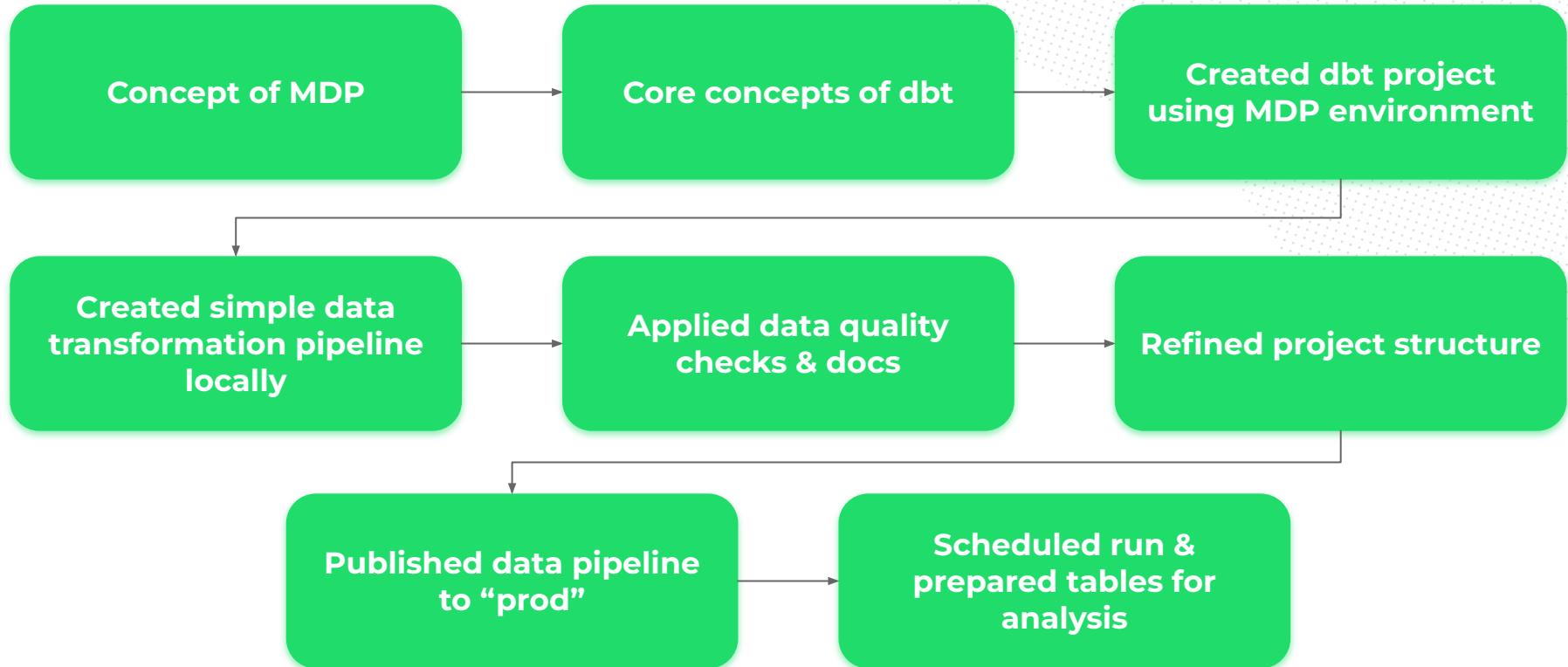
- Simple dbt pipeline scheduled with Airflow
- Connection to BI Tools



# hands-on exercises:

Open `Session_3.md` and start reforging your dbt pipeline into well structured project and then use Gitlab to publish your work on prod.

# Let's summarize



## Data-Driven Fast-Track



TECH NEWS

### Data-driven fast-track: 3 steps to make your company more data-driven



Piotr Menclewicz | 13 October 2022 | 11 min read



## Modern Data Platform for Volt.io

SUCCESS STORIES

### How we built a Modern Data Platform in 4 months for Volt.io, a FinTech scale-up



Paweł Kociński



Rafał Zalewski

| 4 October 2022 |

19 min read

## Thank you!

**Contact us!**

<https://getindata.com/estimate>

**Our blog**

<https://getindata.com/blog>

**Our podcast**

[Radio DaTa](#)

