getindata

Build reliable data pipelines using Modern Data Stack in the cloud

Michał Soszko
Data Analyst / Analytics Engineer
GetInData

Andrzej Swatowski
Data Engineer
GetInData

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

getindata

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-do-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

getindata

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**

Agenda

getindata

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?

getindata

Saves time, effort, and money

Shift from once-off analytics to operational BI and AI

Business (not IT) focused operating model

Airbyte, dbt, Snowflake, Looker, etc.

Data governance as a first-class citizen

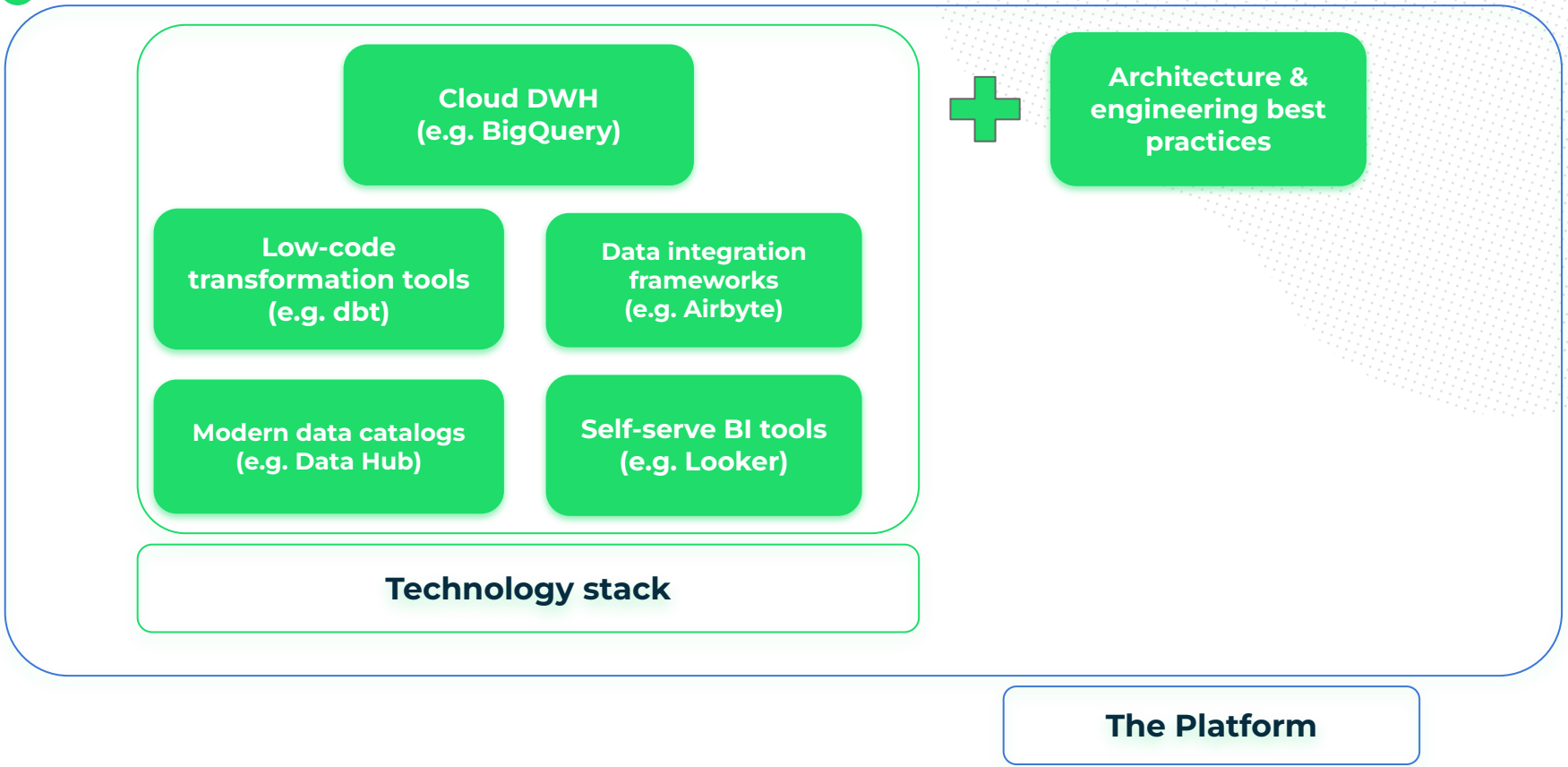
Cloud based, flexible & scalable, plug-and-play, no long term commitments

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

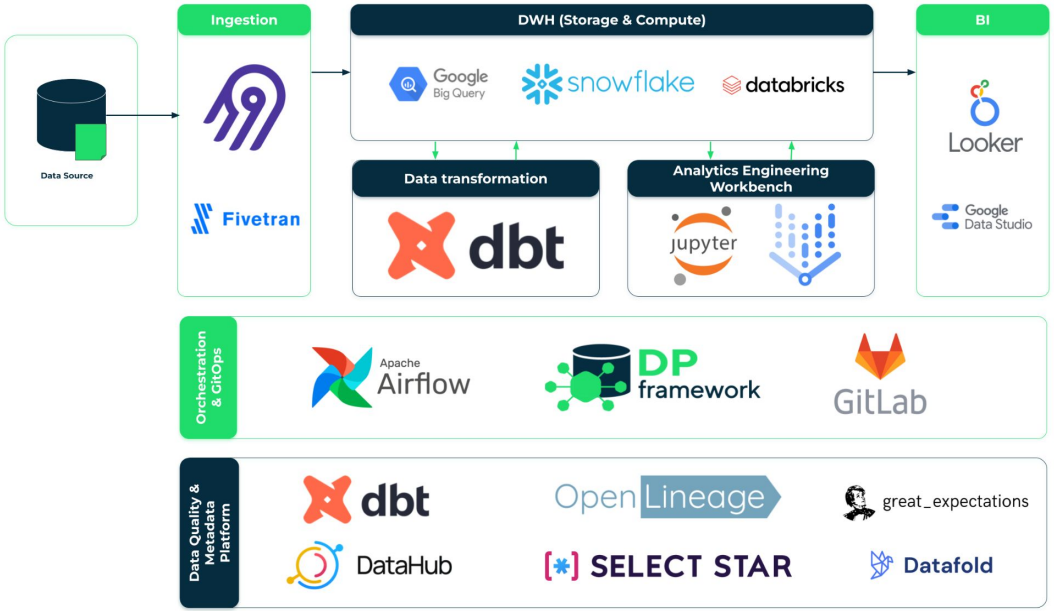
What is modern data stack?

getindata



GetInData Modern Data Platform - our framework (GCP example)

getindata



Solution optimized for:

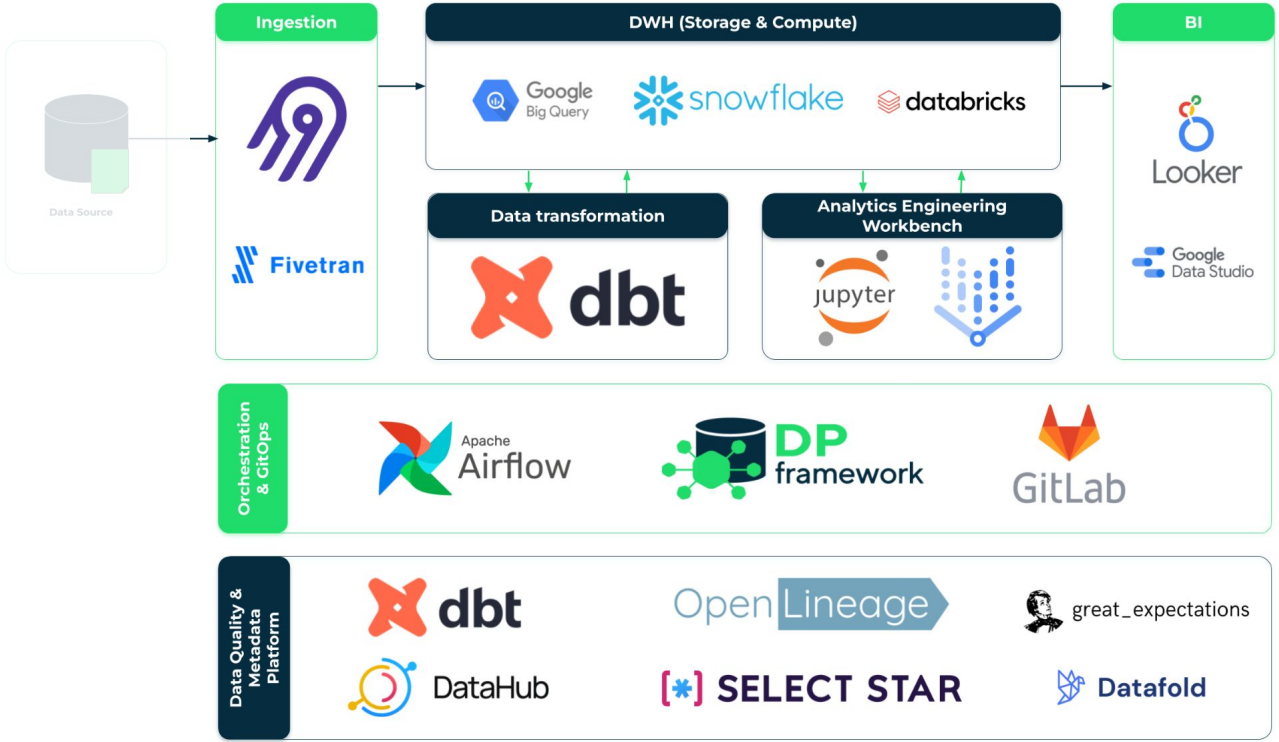
- any cloud
- flexibility & extensibility
- self-service
- data mesh readiness
- automation
- observability & discovery

DP framework

- data-pipelines-cli
- data-airflow-factory
- Project Template Factory

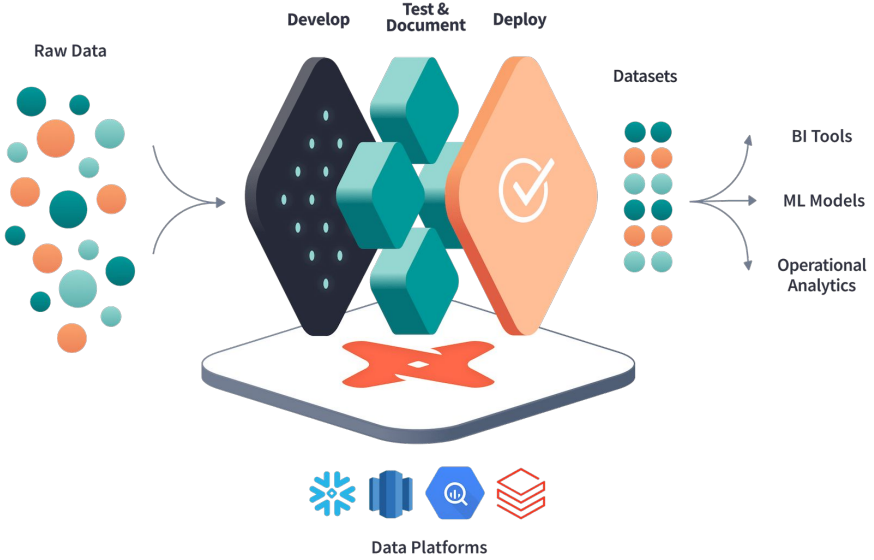
GetInData Modern Data Platform - our framework (GCP example)

getindata



Core concepts of dbt

getindata



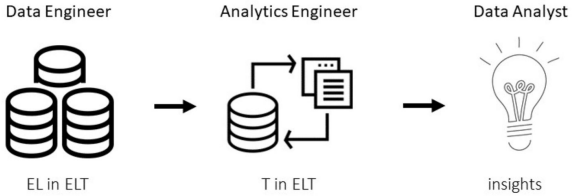
“dbt™ is a transformation workflow that lets teams quickly and collaboratively deploy analytics code following software engineering best practices like modularity, portability, CI/CD, and documentation. Now anyone who knows SQL can build production-grade data pipelines.”

getdbt.com

Source: getdbt.com

Core concepts of dbt

What problem does dbt solve?



getindata

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

SQL

Python
(recently)

Core concepts of dbt

getindata

SQL

```
select *
from table_A
where column_B > 0
```

jinja

```
{{ function( ‘var1’, ‘var2’, …. ) }}
{{ config (‘….’) }}
```

CLI

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

Core concepts of dbt

getindata

dbt model

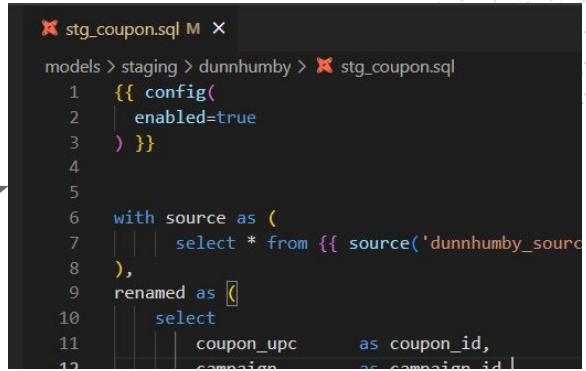
Select statement

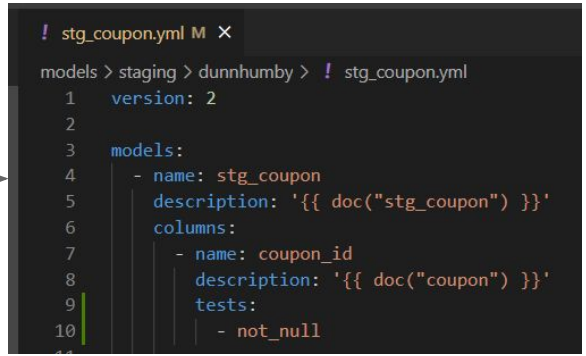
SQL file

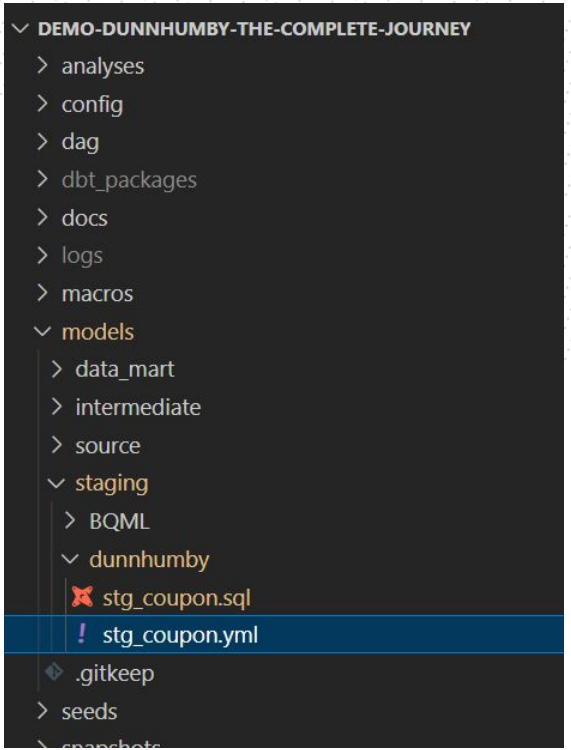
+

YML file

metaconfig







Core concepts of dbt

Create models, calculate dependencies

getindata

{{ source(“schema”, “table”) }}

{{ ref(“dbt_model”) }}

table_A.sql:

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

table_B.sql:

```
select *
from {{ ref( “table_A” ) }}
```

raw_table

Core concepts of dbt

Create models, calculate dependencies

getindata

{{ 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”
)
```

Lineage

Core concepts of dbt

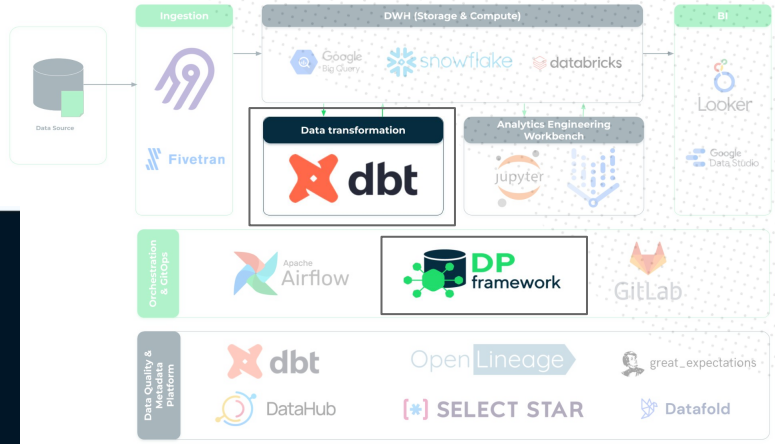
Connection to DWH, different environments

getindata

~/.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
```

- `jaffle_shop:` → Project name
- `target: dev` → Environment
- `outputs:` / `dev:` → Config (dev, local, prod...)
- `type: postgres` → DWH type
- `host: localhost` → Address
- `user: alice` / `password: <password>` → Credentials
- `port: 5432` / `dbname: jaffle_shop` → DB name
- `schema: dbt_alice` → Default working schema
- `threads: 4` → Number of 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

getindata

Hands-on:

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

... coffee break

Agenda

getindata

Session 2 - Simple end-to-end data pipeline

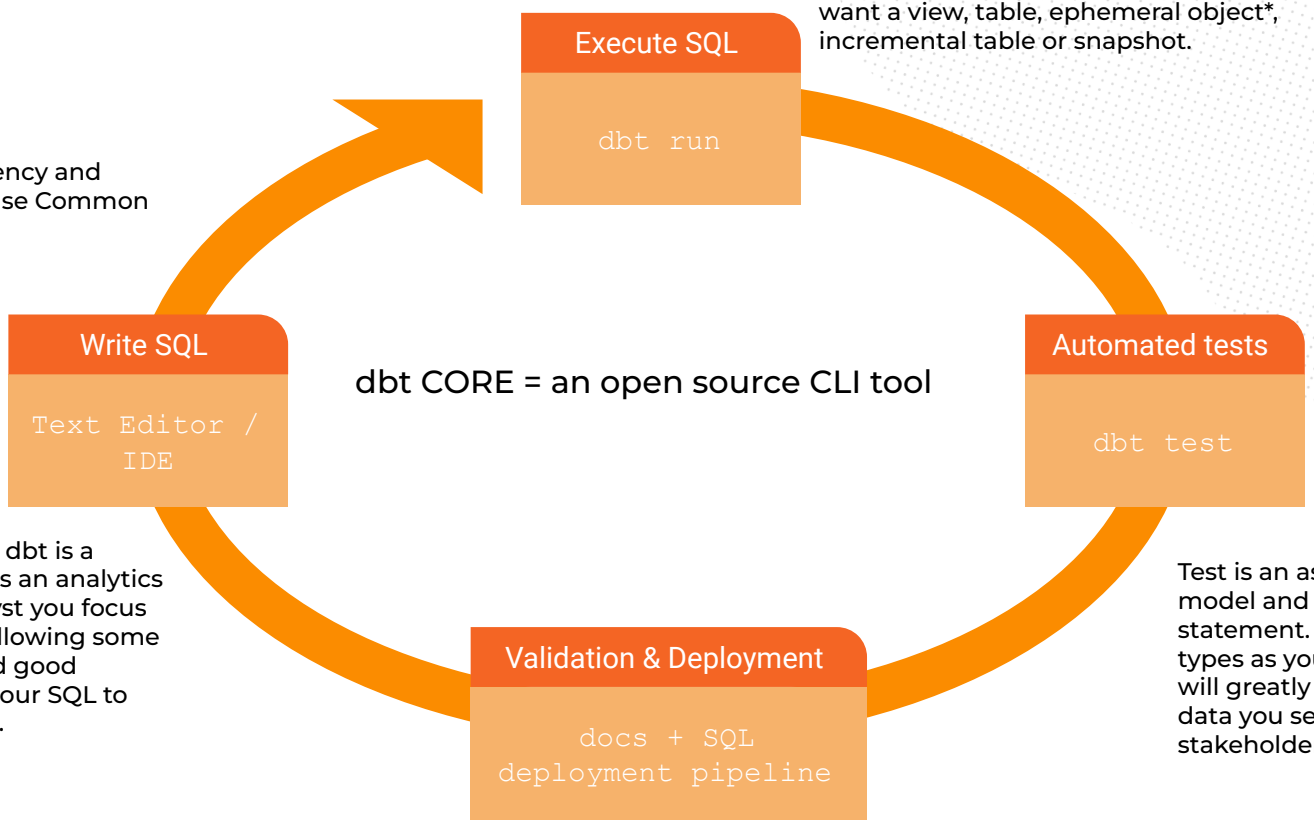
- ➔ Transforming data using SQL with dbt
- ➔ Data catalog (data search, discovery, profiling)
- ➔ Hands-on exercises

Transforming data with dbt

SQL = model is executed with **dbt run** command (CLI). While executing it, dbt will materialize an entity in your data warehouse. You can choose whether you want a view, table, ephemeral object*, incremental table or snapshot.

getindata

For better transparency and future debugging use Common Table Expressions



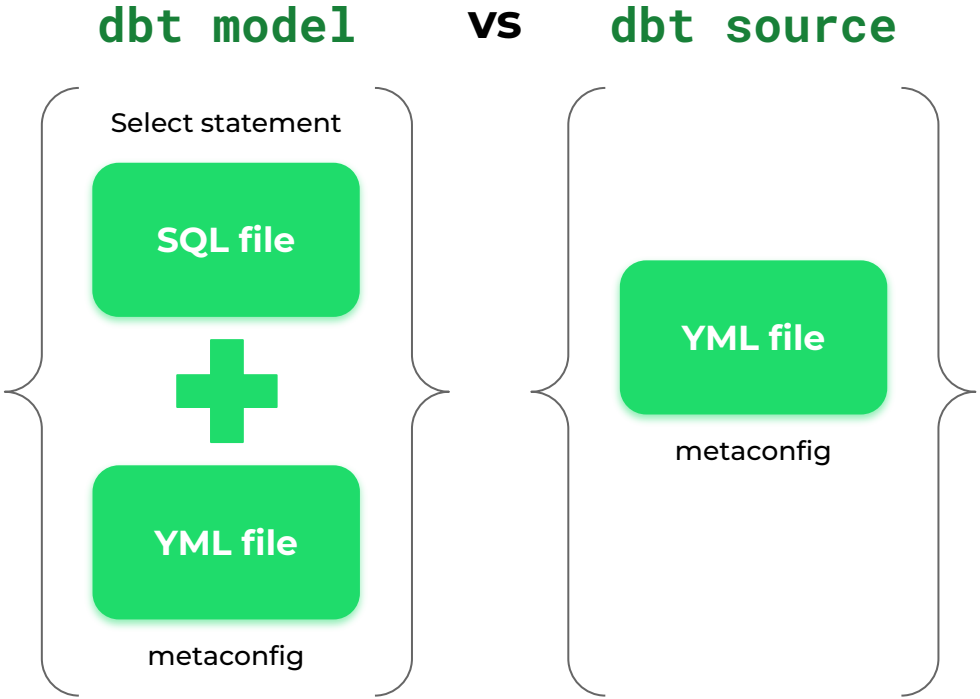
Almost everything in dbt is a SELECT statement. As an analytics engineer / data analyst you focus on business logic. Following some internal dbt rules and good practices will cause your SQL to become a dbt model.

Test is an assertion about your model and it is also a SELECT statement. Use as much test types as you can think of. This will greatly improve a trust in data you serve to your team & stakeholders.

Transforming data with dbt

Sources: define your input data

getindata

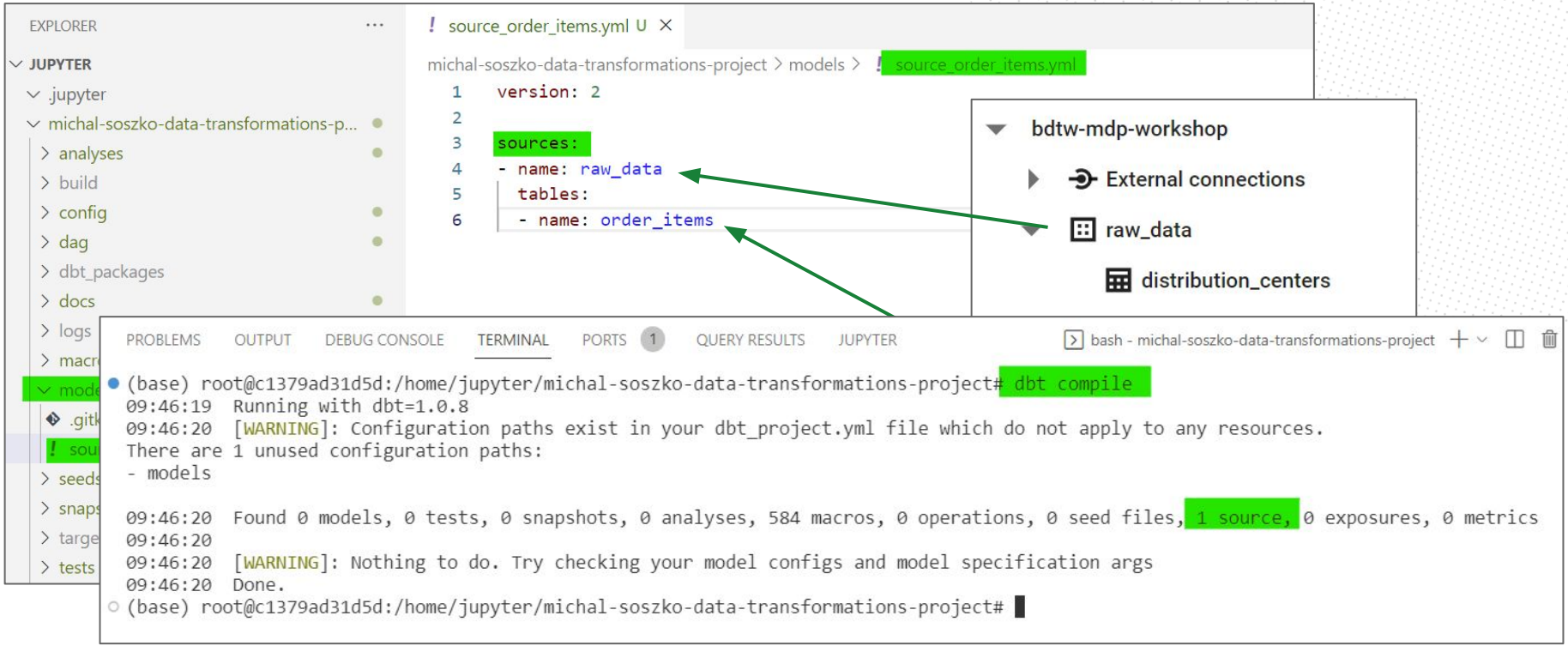


- In dbt `sources` refer to tables that **already exist** in data warehouse
- If data in the source table changes - it is not a result of a dbt transformation
- `dbt sources` are defined in `yaml` file / files alone
- Sources are referenced by using the `source` functon
- You can put `tests` and `documentation`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

Transforming data with dbt

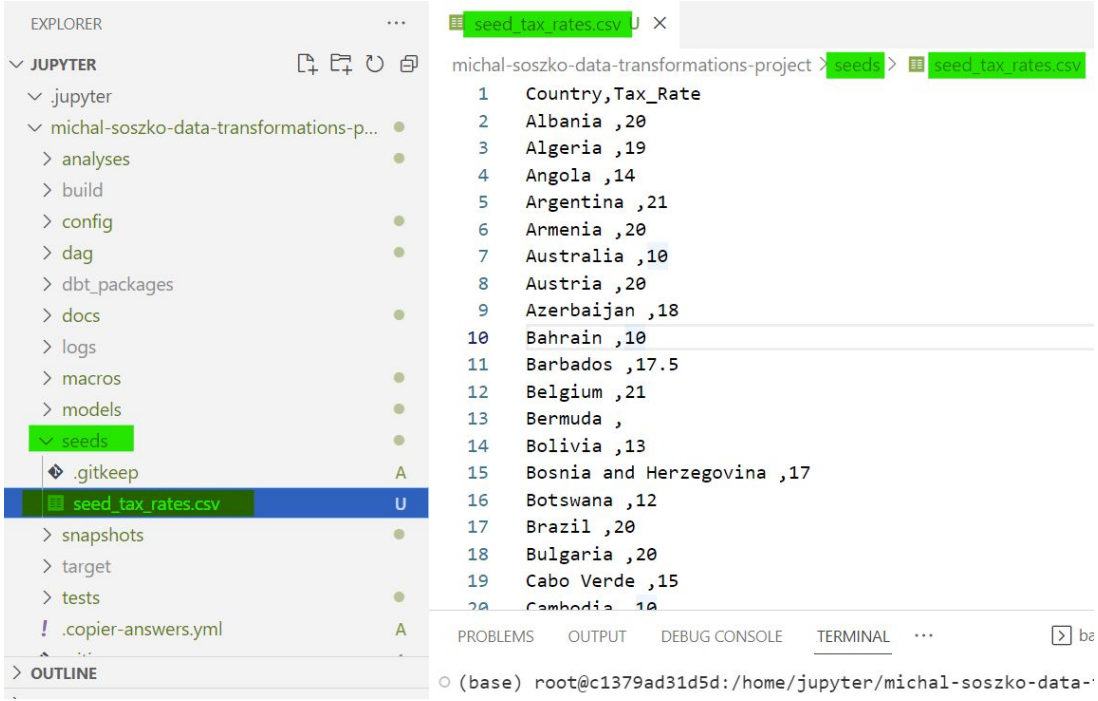
Sources: define your input data

getindata



Transforming data with dbt

Seeds: add custom CSV data to your project



- 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

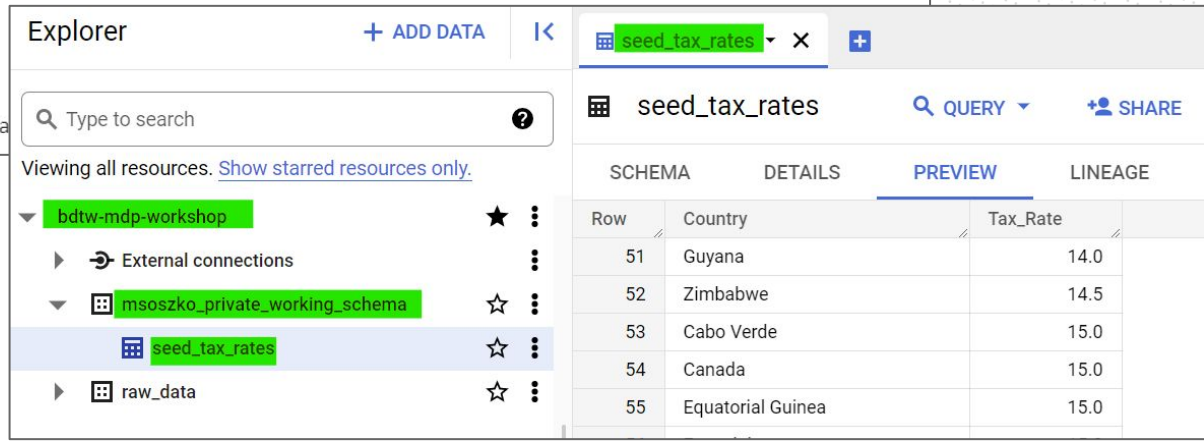
Transforming data with dbt

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
```



Transforming data with dbt

Models: foundation of a dbt data pipeline

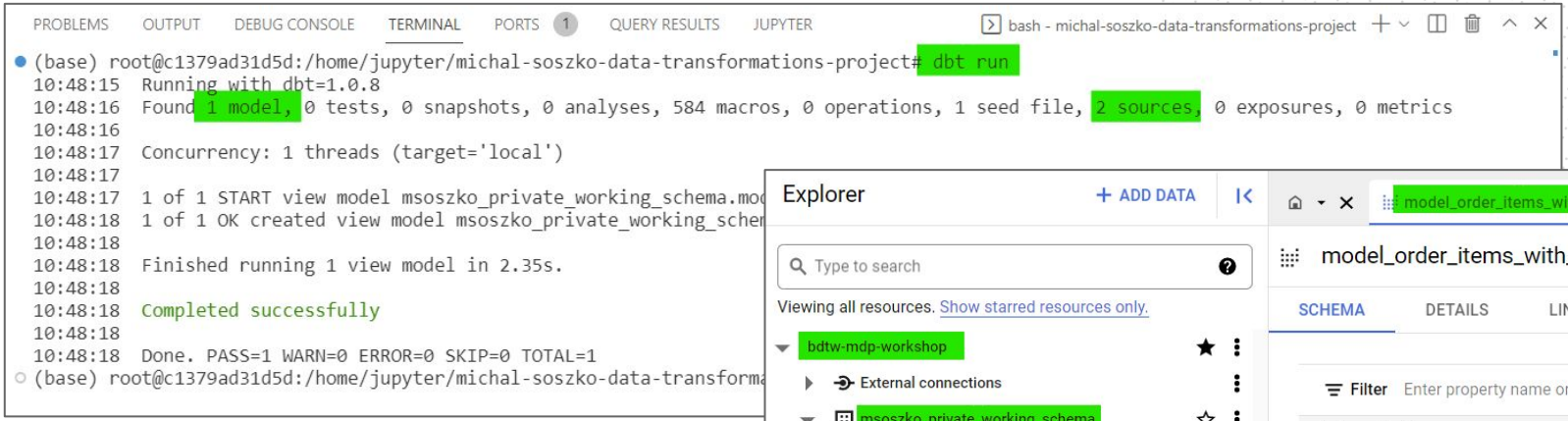
getindata

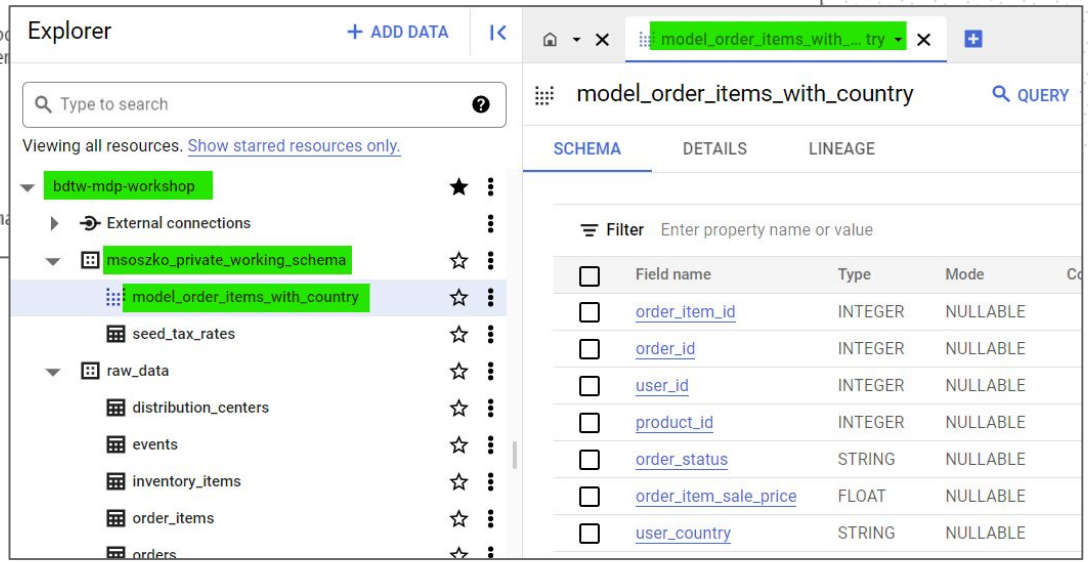
- `Model` is a select statement stored in a `SQL` file.
- `Model` metaconfig `yml` file is **optional**
- Executing a model with the `dbt run` command will create a `table` / `view` in the DWH
- Models are referenced within dbt using the `ref` function
- There are different materialization modes to choose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

Transforming data with dbt

Models: foundation of a dbt data pipeline

getindata





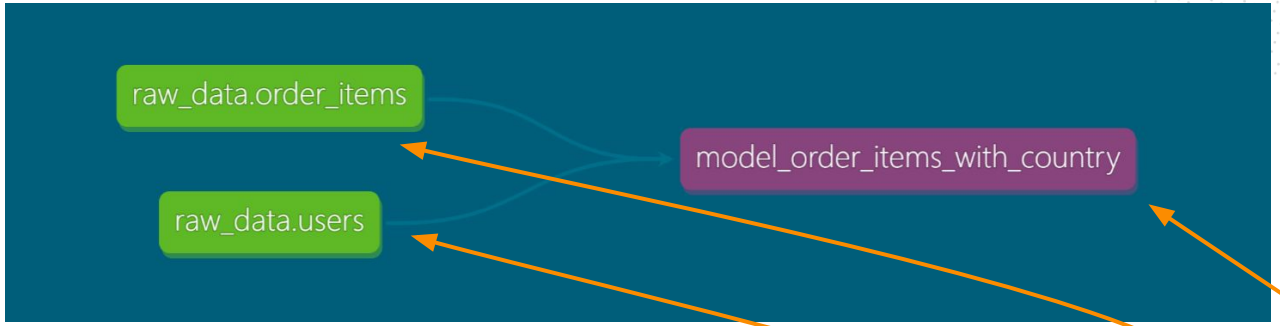
To run a model type:

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

Transforming data with dbt

Models: foundation of a dbt data pipeline

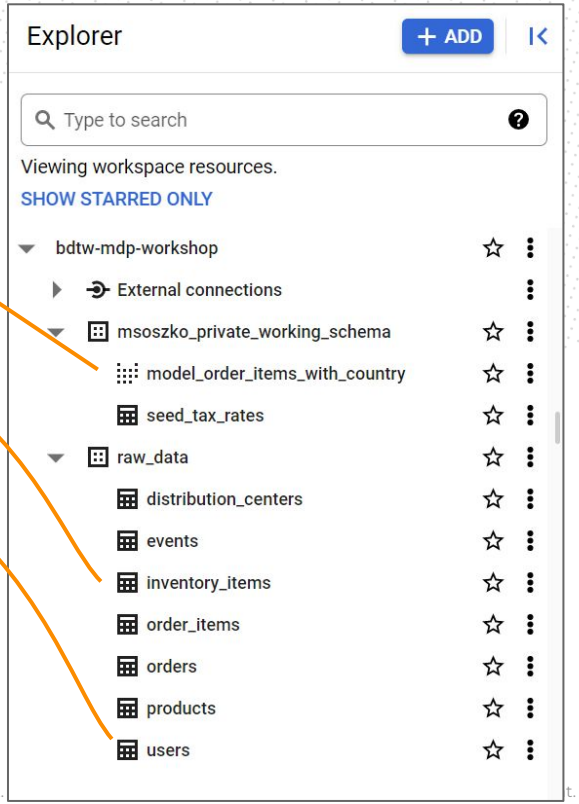
getindata



dbt sources

dbt model

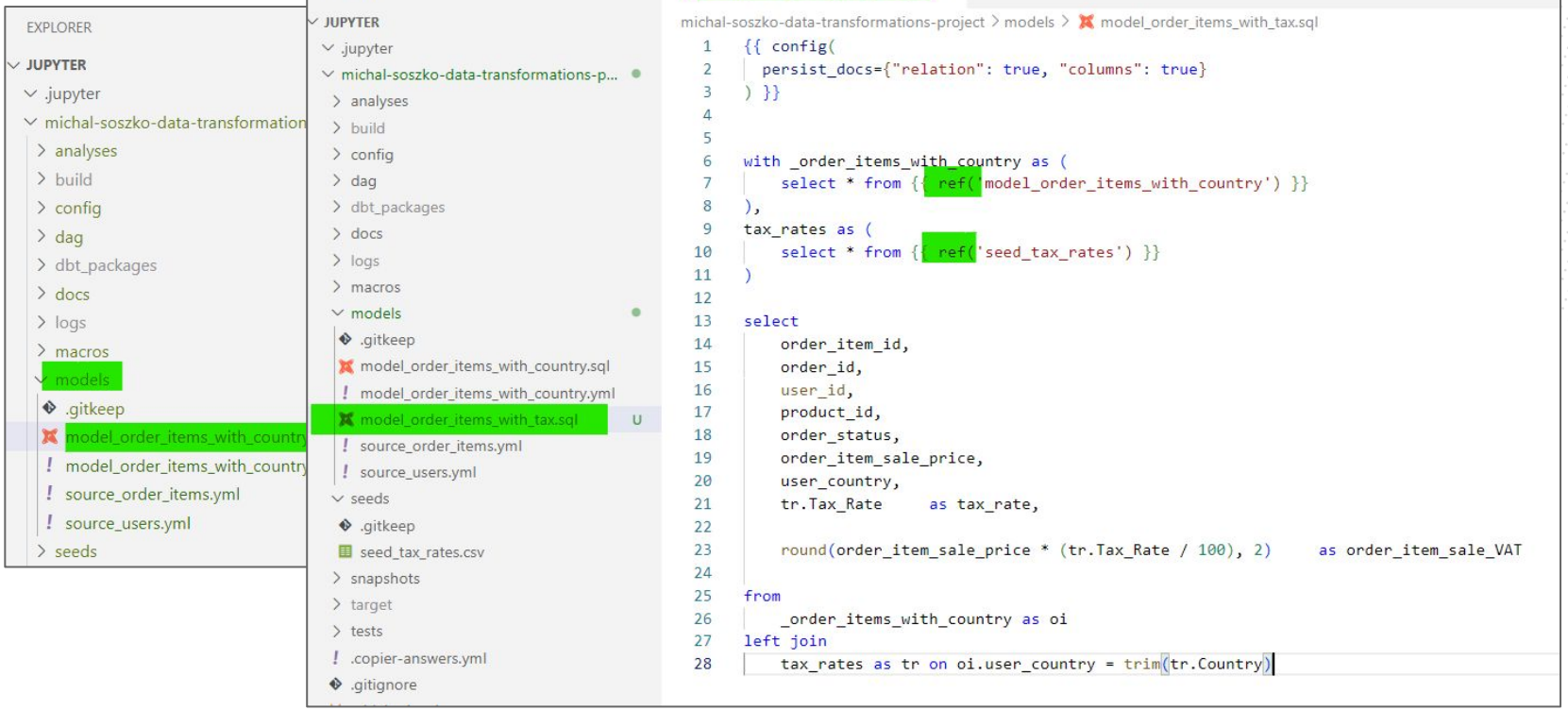
```
{{ source ( 'schema', 'table' ) }}
```



Transforming data with dbt

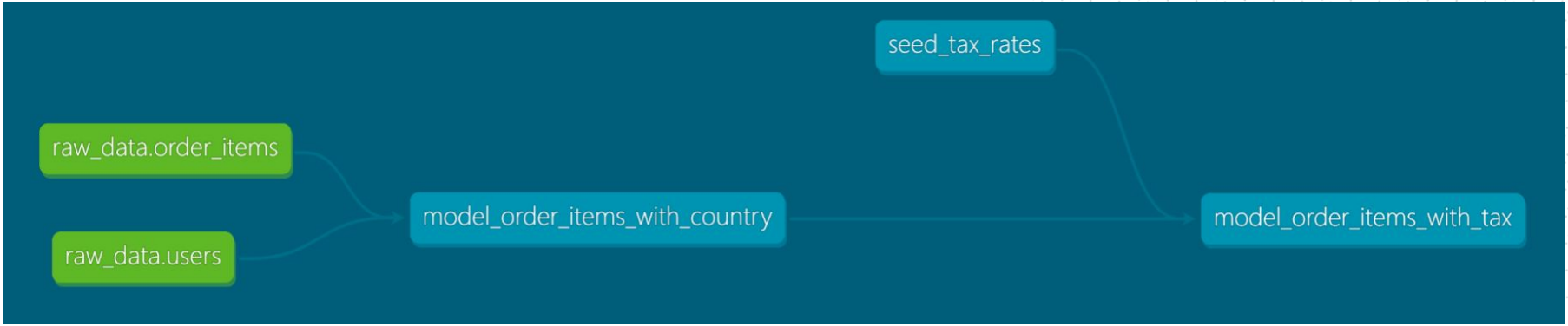
Models: foundation of a dbt data pipeline

getindata



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...!

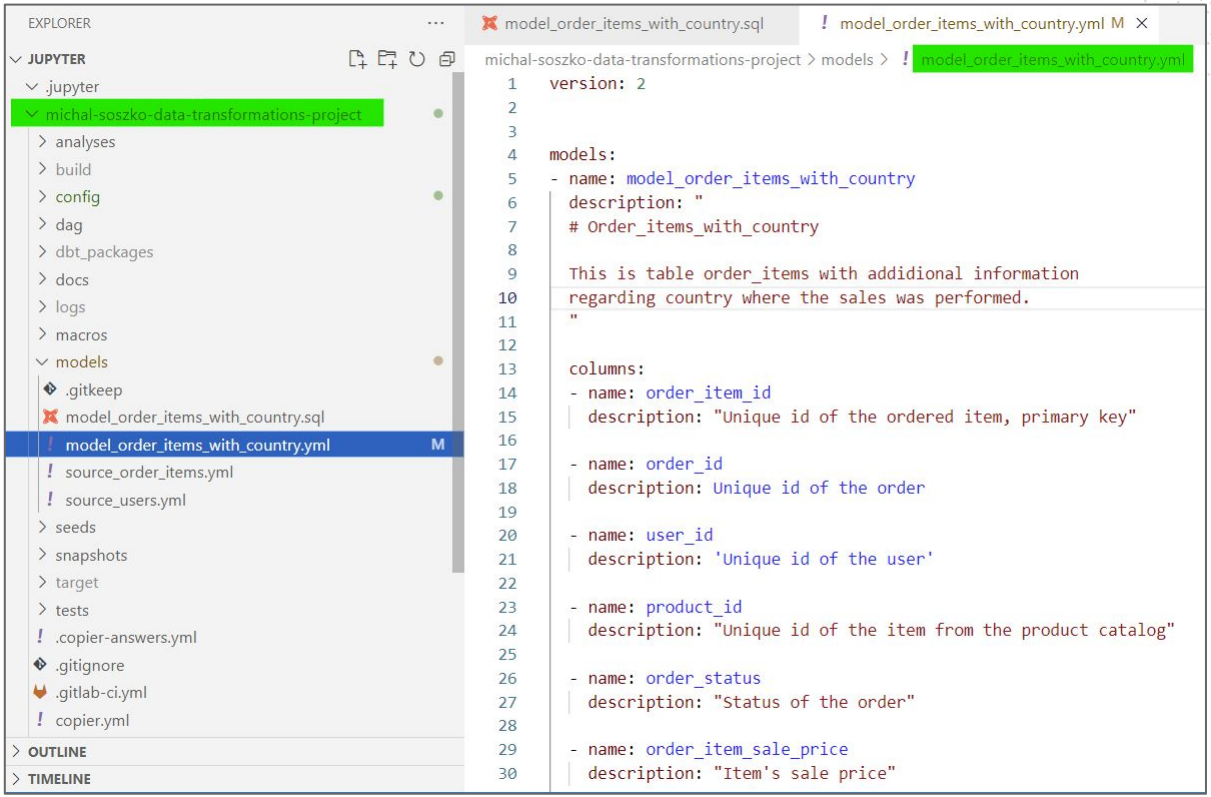
...and hands-on exercises:

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

Transforming data with dbt

Models: metaconfig YAML files

getindata

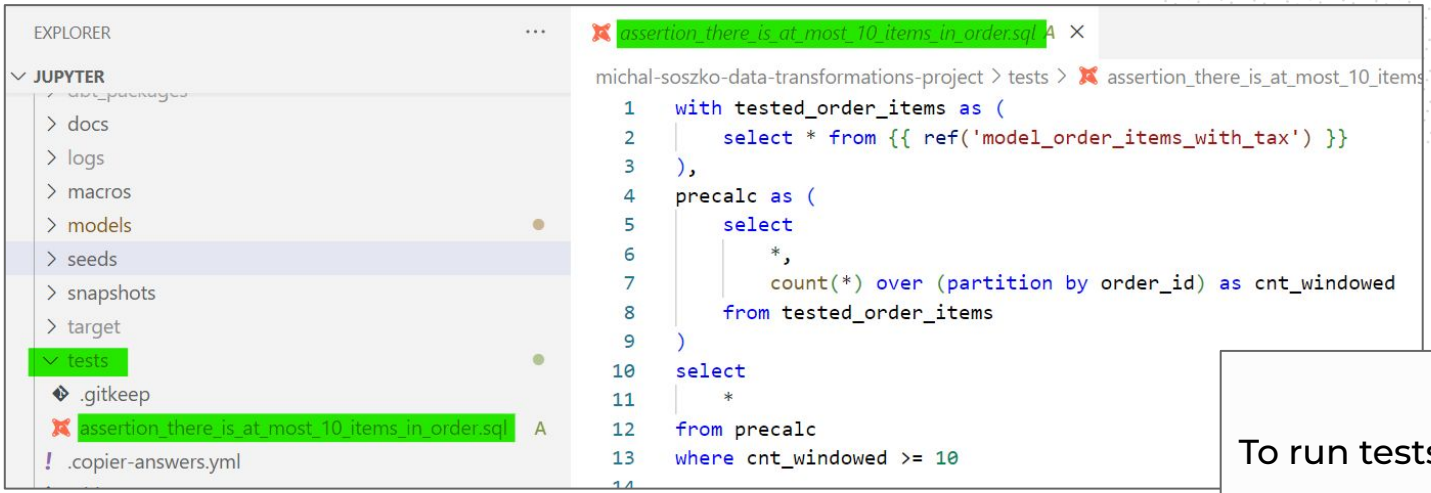


- Metaconfig `YAML` file stores descriptive information about the model.
- The `YAML` file is optional
- Allows documenting all `dbt` entities (models, seeds & sources) at their source
- Allows adding **tests** to all `dbt` entities
- **Whitespace sensitive!**
- You can store all model, source and seed 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

getindata



- 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**

To run tests type:

```
dbt test
dbt test --select model
dbt test --select assertion
```

Transforming data with dbt

Data quality in dbt: tests

getindata

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#
```

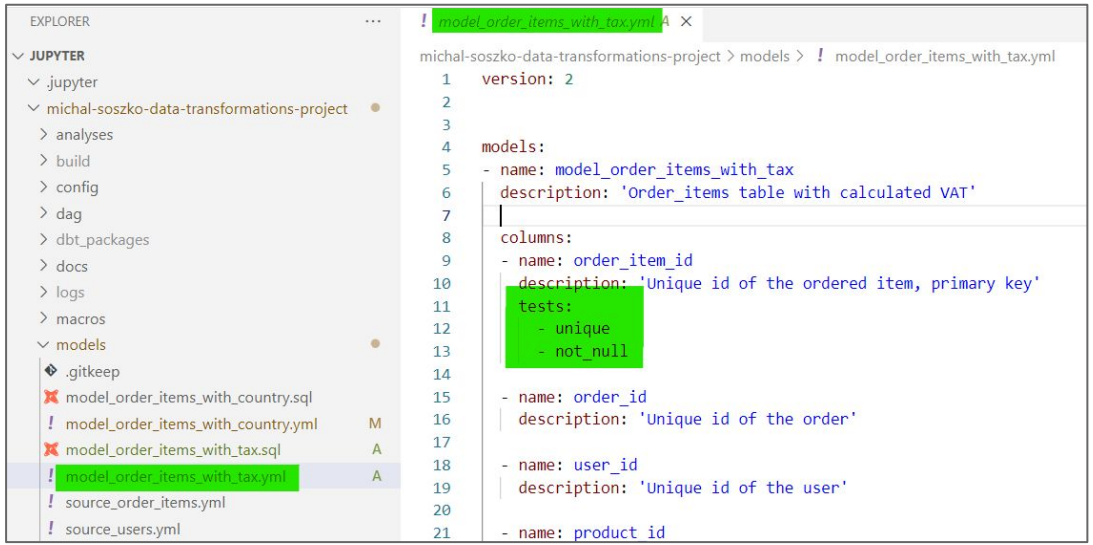
Failure

```
(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

getindata

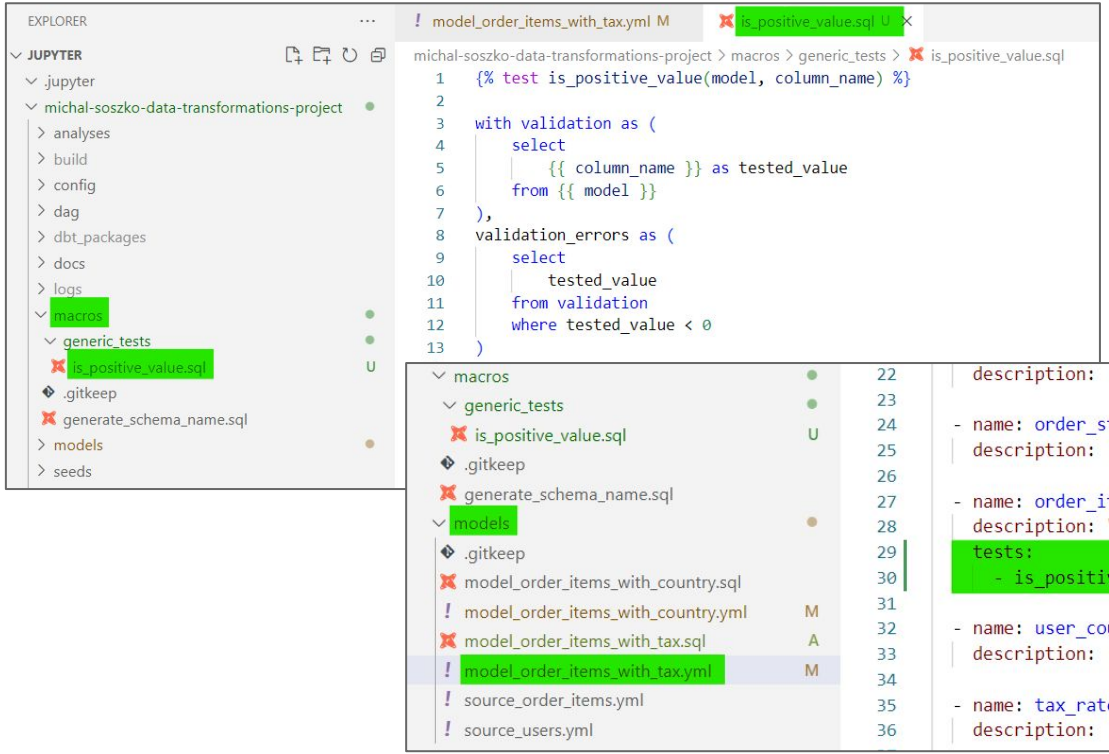


- Generic tests can be applied to any column
- There are 4 main generic tests:
  - not_null
  - unique
  - accepted_values
  - relationships
- Users can define their own generic test as jinja macros
- dbt community has served hundreds of custom generic tests; you can find them in many public packages:
  - dbt-labs / dbt_utils
  - calogica / dbt_expectations
  - and many more: https://hub.getdbt.com/

Transforming data with dbt

Data quality in dbt: tests

getindata



- 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**

undreds of
n find them in

tions

n/

Transforming data with dbt

Documentation and graphical lineage

getindata

Transforming data with dbt

Documentation and graphical lineage

getindata

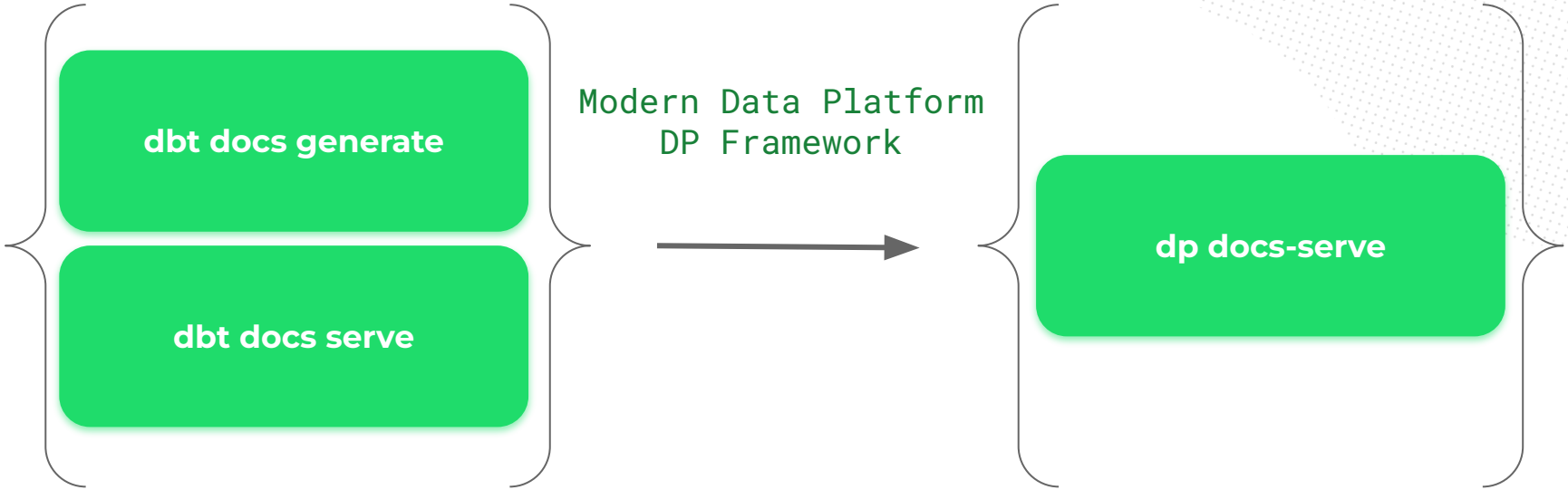
- `dbt` comes with its own documentation generator
- Rendered documentation can be viewed via web-browser
- `Model` and project documentation (table / column level descriptions, project description etc.) is stored as a part of the dbt entity metaconfig YAML file.
- In MDP, you can access your local documentation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

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

```
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



getindata

Coffee Break...!

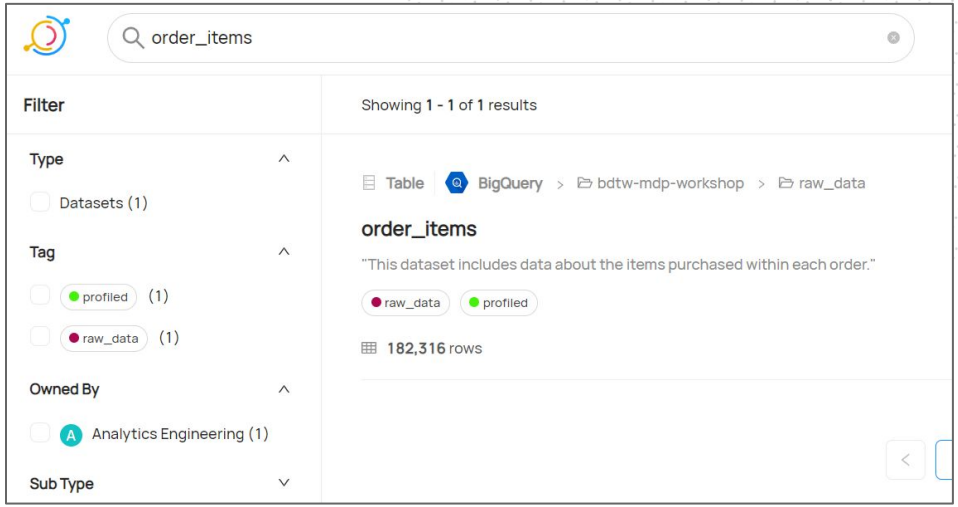
...and hands-on exercises:

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

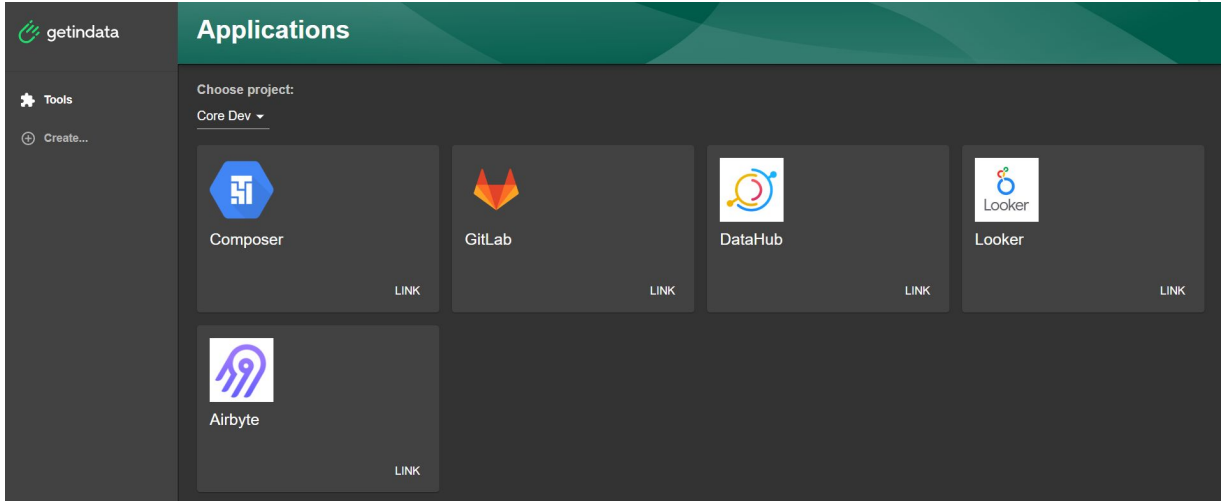
- Not everyone in the organization have / wants an access to the data warehouse
- Data catalog: an external tool storing information related to data.
- In MDP data catalog it is synchronized with the code repository - it is always up to date with recent changes in tables.
- Data catalogs usually have 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.

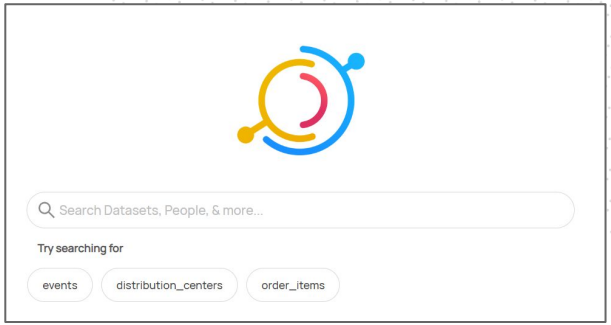


Data Catalog

Single source of knowledge about your data

getindata





Datahub: demo

Data Catalog

Single source of knowledge about your data

getindata

Datahub access:

User name & Password: -> check on Slack

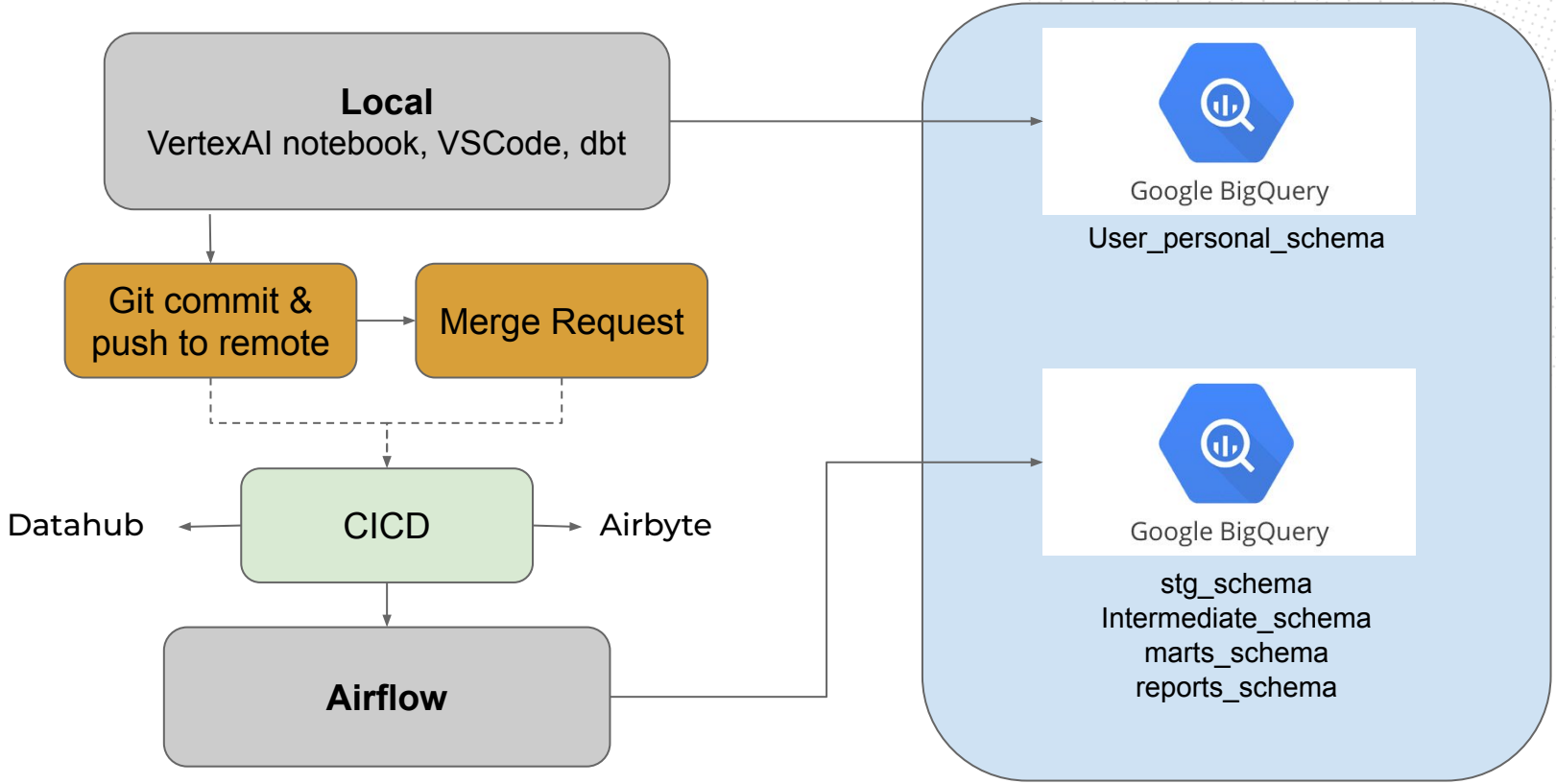
getindata

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

getindata



dbt compile & the manifest file

getindata

- 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

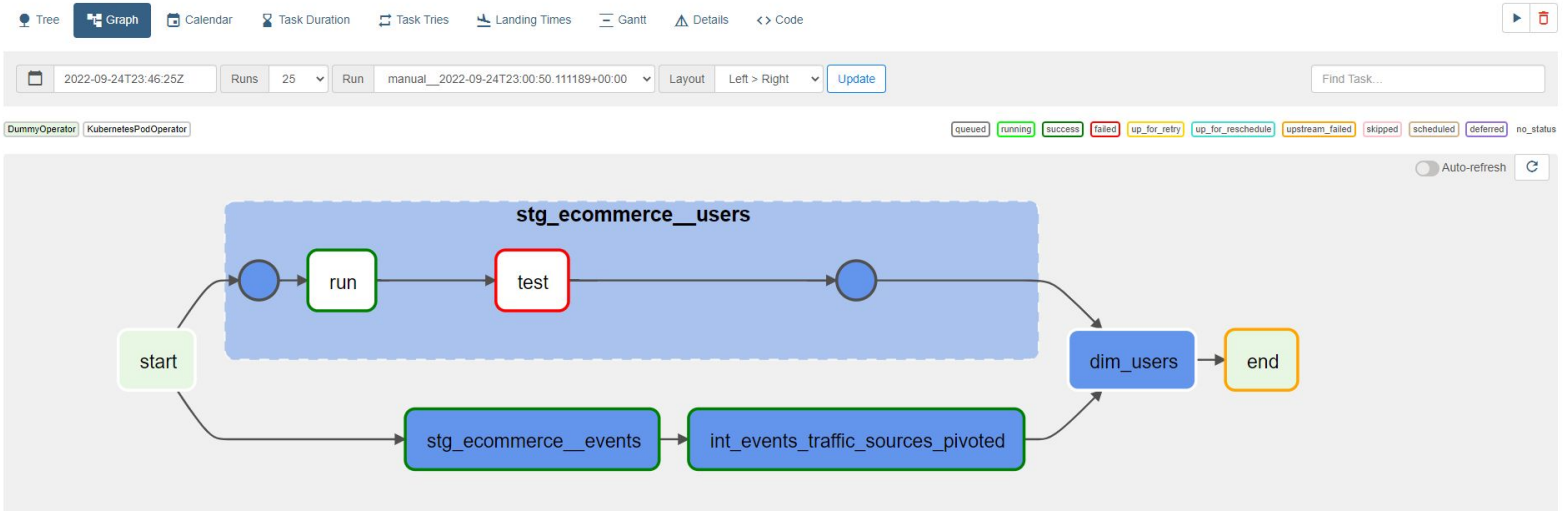
aka show me the order of operations

getindata

Source: getdbt.com

Apache Airflow as a workflow scheduler

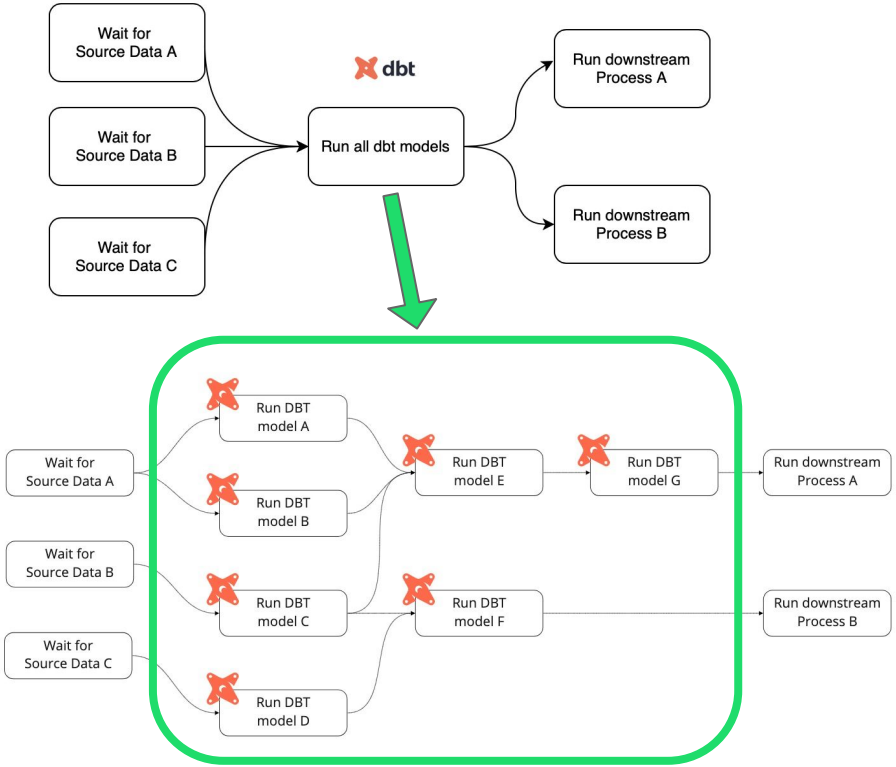
getindata



Source: getdbt.com

Apache Airflow as a workflow scheduler

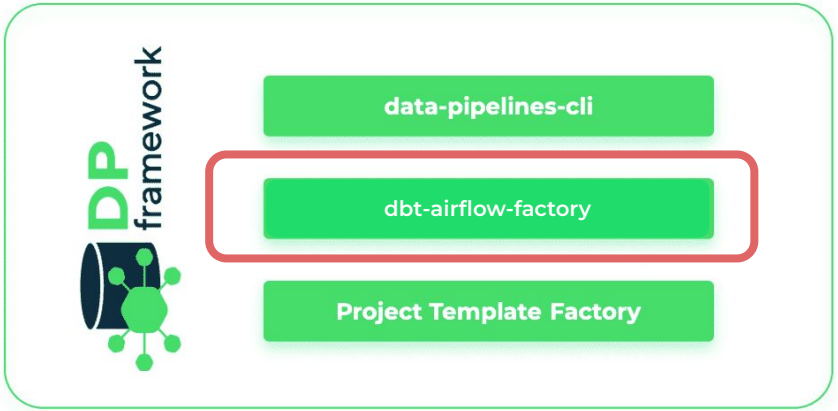
getindata



- 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

getindata



- `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

getindata

```python
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(".")[0] != "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(".")[0] != "model":
        continue

    for upstream_node in node["depends_on"]["nodes"]:
        upstream_node_type = upstream_node.split(".")[0]
        if upstream_node_type == "model":
            dbt_tasks[upstream_node] >> dbt_tasks[node_id]
```

```python
from os import path
from airflow.models import Variable
from dbt_airflow_factory.airflow_dag_factory import AirflowDagFactory


dag = AirflowDagFactory(path.dirname(path.abspath(__file__)), Variable.get("env")).create()
```

CI/CD

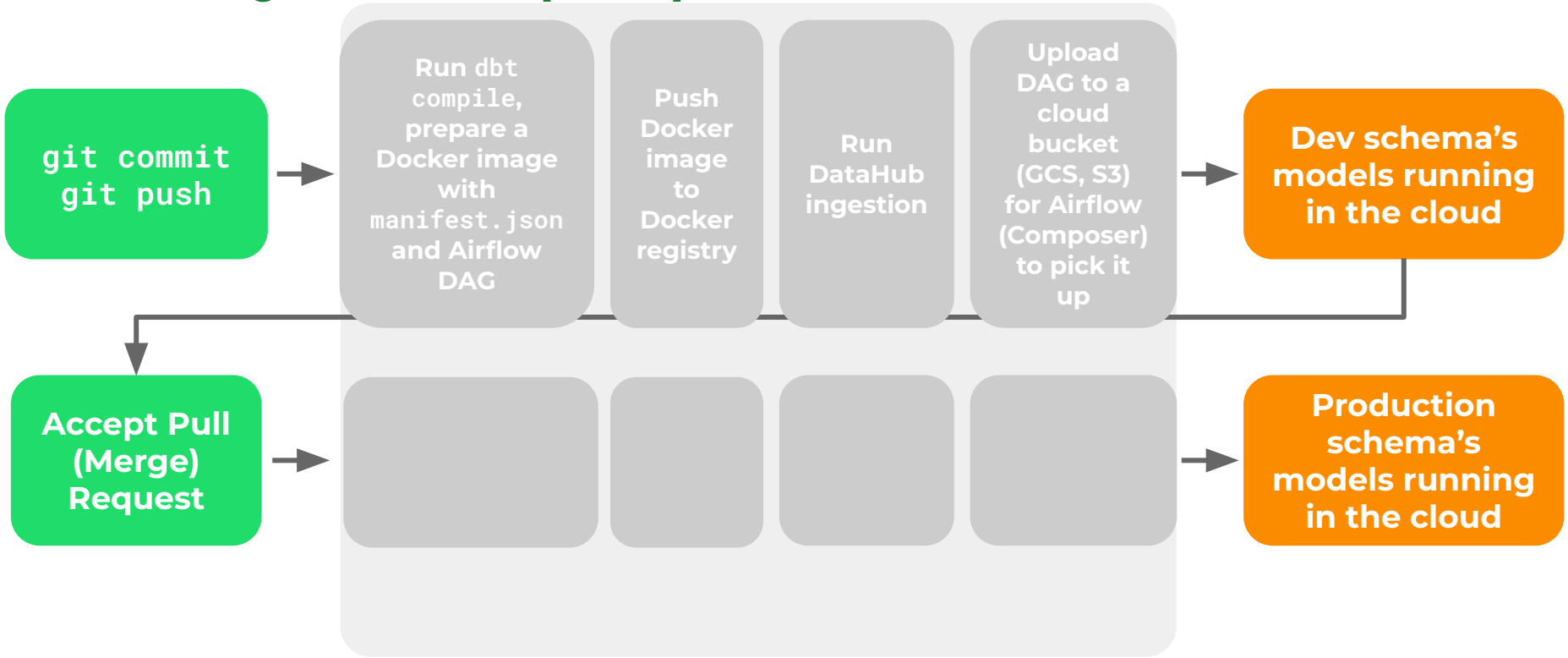
getindata

Analytics Engineer's perspective

CI/CD

getindata

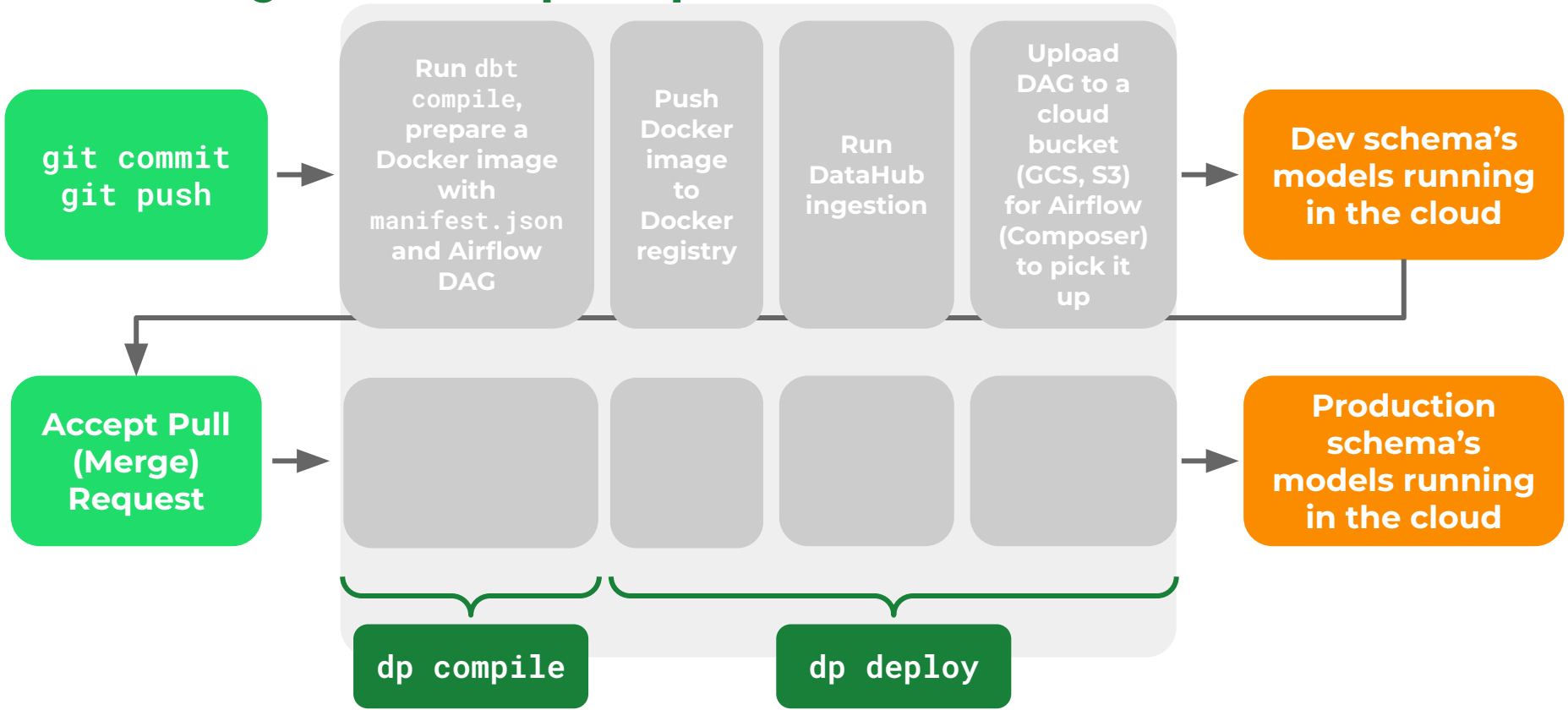
Data Engineer's perspective



CI/CD

getindata

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

getindata

- **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

getindata

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

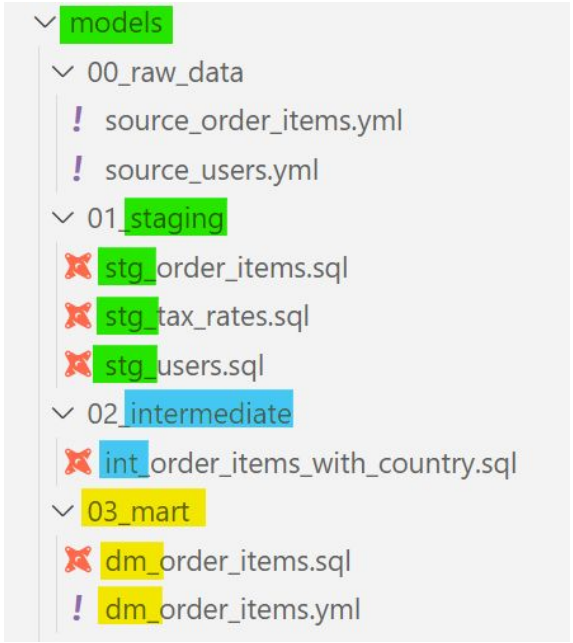
Data Mart

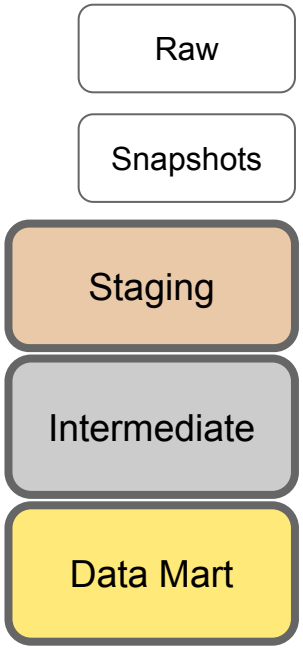
- “Business” layer
- sublayers: business teams
- final JOINs
- final aggregations
- final filtering
- quality checked (tests are obligatory)
- documentation is obligatory (detailed description for columns, tables, tests, exposures etc)

Advanced concepts of dbt

Structuring the project

getindata





dbt_project.yml 1.22 KiB

```yaml
# Name your project! Project names should contain only lowercase characters
# and underscores. A good package name should reflect your organization's
# name or the intended use of these models
name: 'michal_soszko_data_transformations_project'
version: '1.0.0'
config-version: 2

# This setting configures which "profile" dbt uses for this project.
profile: 'bigquery'

# These configurations specify where dbt should look for different types of files.
# The `model-paths` config, for example, states that models in this project can be
# found in the "models/" directory. You probably won't need to change these!
model-paths: [ "models" ]
docs-paths: ["docs"]
analysis-paths: [ "analyses" ]
test-paths: [ "tests" ]
seed-paths: [ "seeds" ]
macro-paths: [ "macros" ]
snapshot-paths: [ "snapshots" ]

target-path: "target"  # directory which will store compiled SQL files
clean-targets: # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"

models:
  michal_soszko_data_transformations_project:
    staging:
      +materialized: view
      +schema: michal_soszko_staging
    intermediate:
      +materialized: ephemeral
      +schema: michal_soszko_intermediate
    mart:
      +materialized: table
      +schema: michal_soszko_mart
```

Advanced concepts of dbt

Structuring the project

getindata

Raw

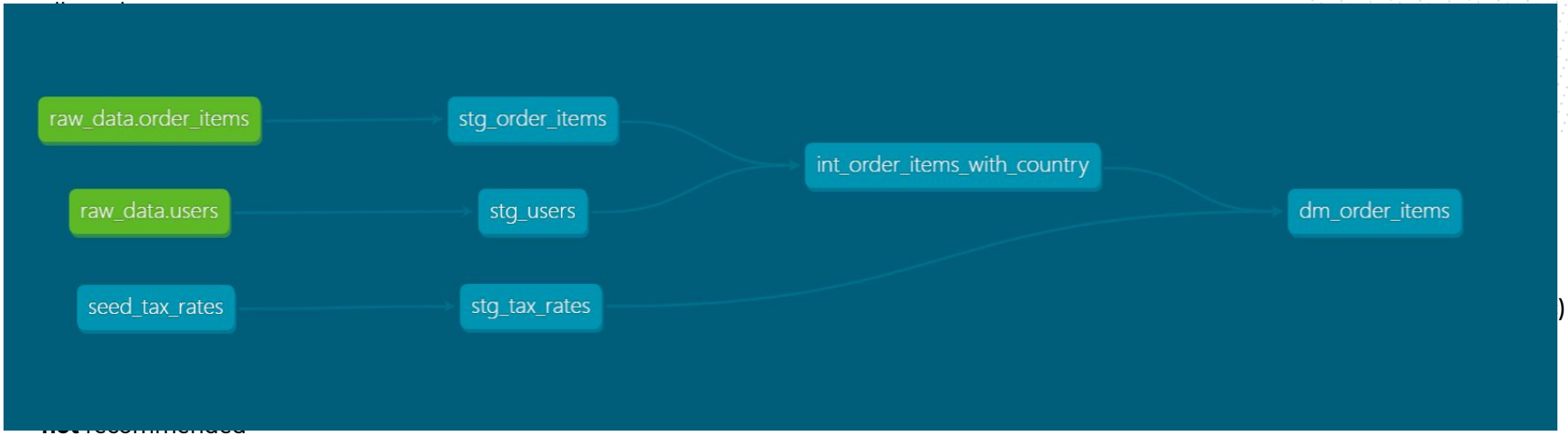
- dbt sources
- **no** transformations

Staging

- tests recommended
- docs recommended

Intermediate

Data Mart



Advanced concepts of dbt

Naming convention

getindata

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

getindata

DEMO:

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

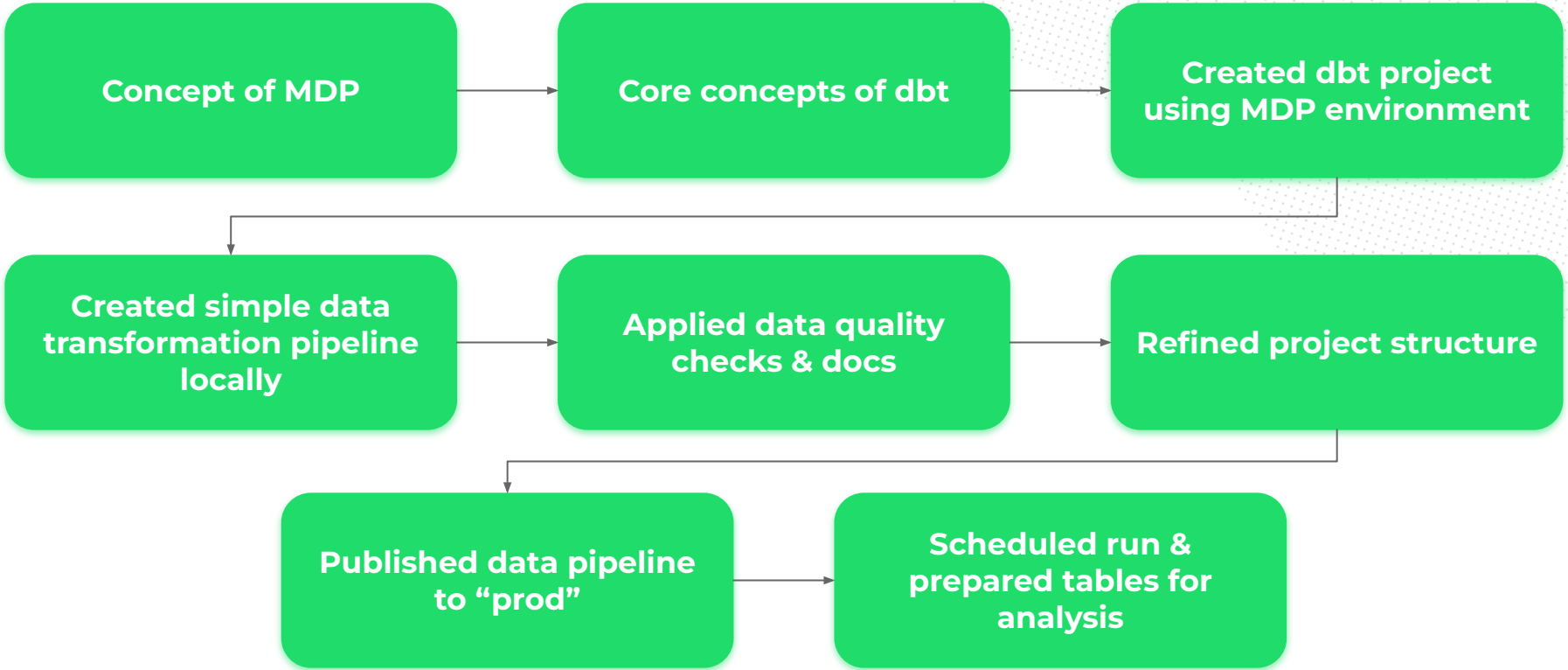
getindata

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

getindata



getindata
Part of Xebia

Solutions | Services | For Business | Company | Knowledge Base | Blog | Careers | FREE CONSULTATION

TECH NEWS

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

Piotr Menclewicz | 13 October 2022 | 11 min read

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

Wrap up

getindata

Thank you!

Contact us!
https://getindata.com/estimate

Our blog
https://getindata.com/blog

Our podcast
Radio DaTa