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**
- ➔ Vsit: **github.com/getindata/gid-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
  - Data models
  - Seeds, sources
  - Tests
  - Documentation
- Hands-on exercises

Session 2 - Simple end-do-end data pipeline

- Data discovery (data search, usage statistics, data lineage)
- Data profiling & exploration
- Transforming data using SQL with dbt
- Data consumption with BI tools
- Hands-on exercises

Session 3 - Data pipeline - scheduling, deployment & advanced features

- Apache Airflow as a workflow scheduler
- Data testing & data observability
- Exploring transformed data with Data Studio
- Hands-on exercises

Today's schedule

- **09:00 - 10:30 - Session 1**
- **10:30 - 10:45 - Coffee Break**
- **10:45 - 12:15 - Session 2**
- **12:15 - 13:15 - Lunch**
- **13:15 - 14:15 - Session 3**
- **14:15 - 14:30 - Coffee Break**
- **14:30 - 16:00 - Session 3 cont.**

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
  - Data models
  - Seeds, sources
  - Tests
  - Documentation
- Hands-on exercises

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

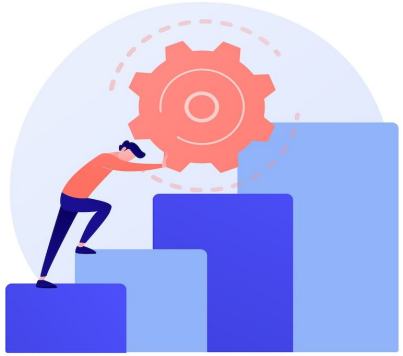
Cloud based, flexible & scalable, plug-and-play, no long term commitments

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

Why we talk about modern data stack now?

getindata



Challenges

Needs

Enablers

Data platform challenges

getindata

Sounds familiar?

Outdated data docs

Costly licensing

No flexibility

Too much engineering

Vendor lock-in

Chaotic project structures

Long time-to-value

Unreliable data

The Modern Data Platform manifest (needs)

getindata

- Self-service!
- Trusted & documented data!
- Project templates!
- Low-code transforms!
- All-in-one workbench!
- Adaptable!
- Data access security!
- Simple to set up & use!

We need to change the way we think about data platform...

Why is MDP possible? (enablers)

getindata

- Cloud DWH (e.g. BigQuery)
- Low-code transformation tools (e.g. dbt)
- Data integration frameworks (e.g. Airbyte)
- Modern data catalogs (e.g. Data Hub)
- Self-serve BI tools (e.g. Looker)

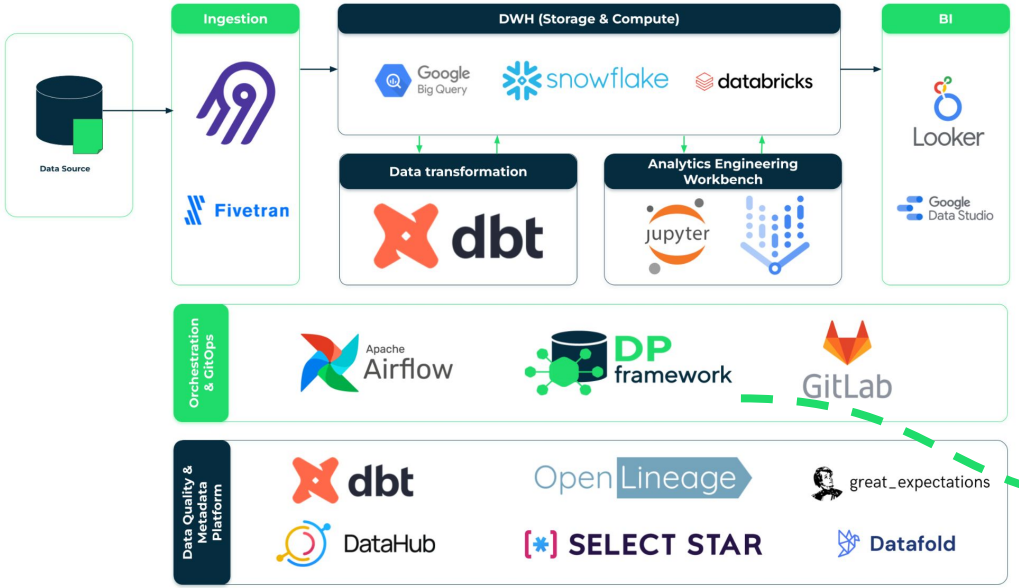
+

- Architecture & engineering best practices

The technology is already here!

GetInData Modern Data Platform - our framework (GCP example)

getindata



Solution optimized for:

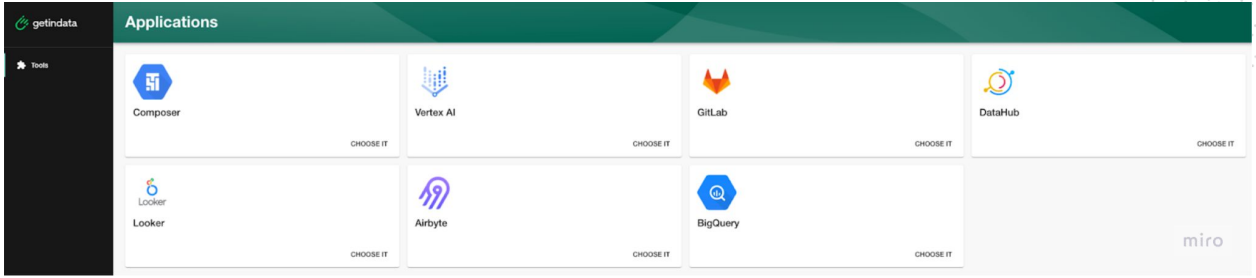
- any cloud
- flexibility & extensibility
- self-service
- data mesh readiness
- automation
- observability & discovery

DP framework

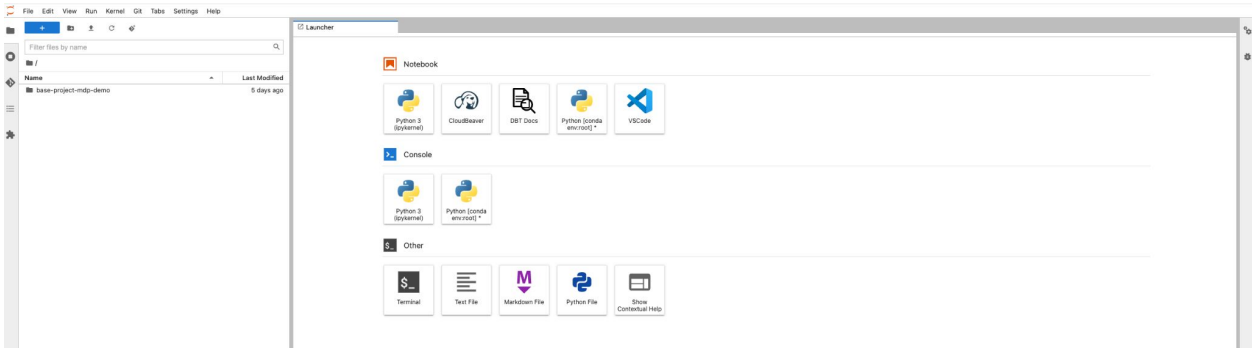
- data-pipelines-cli
- data-airflow-factory
- Project Template Factory

MDP - user interface

getindata



Access all your apps via **GID Portal**



All you need for data processing is in **Analytics Workbench**

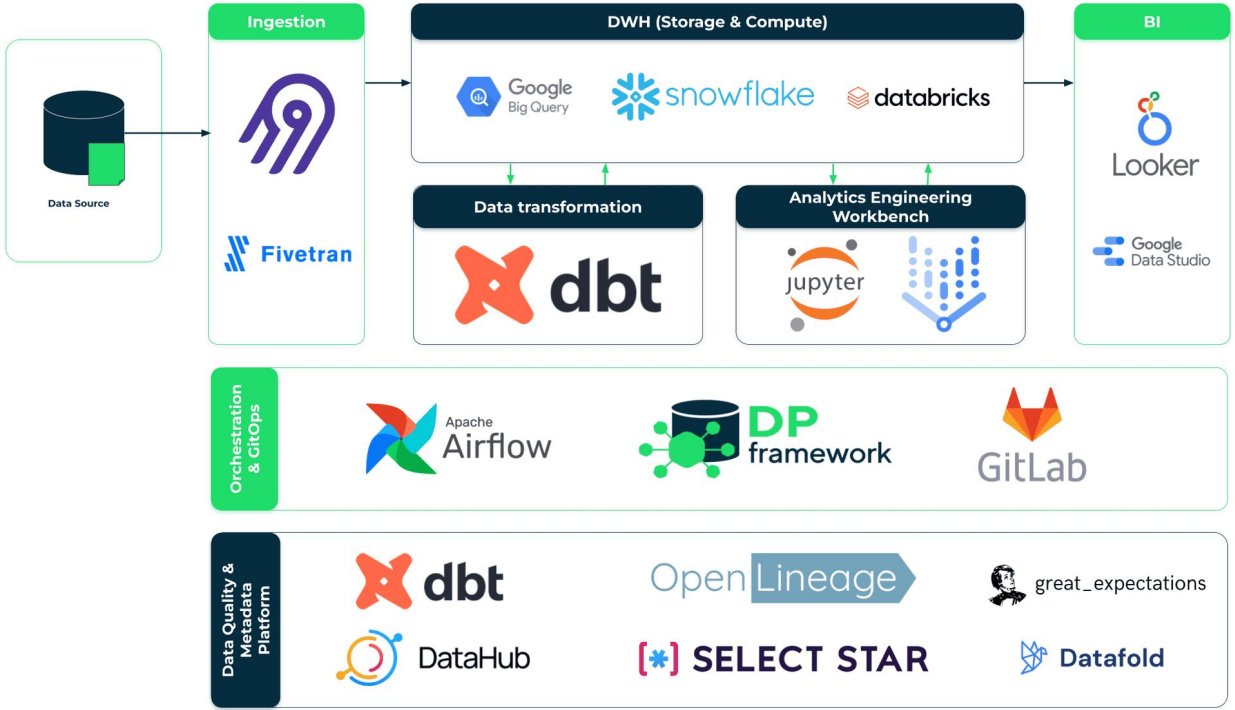
GetInData Modern Data Platform - self-service haven for analytics engineers

getindata

- Ingestion
- Discovery
- Profiling
- Exploration
- Transformation
- Visualisation
- Orchestration
- Observability
- Security
- Modelling

Key components of MDS

getindata

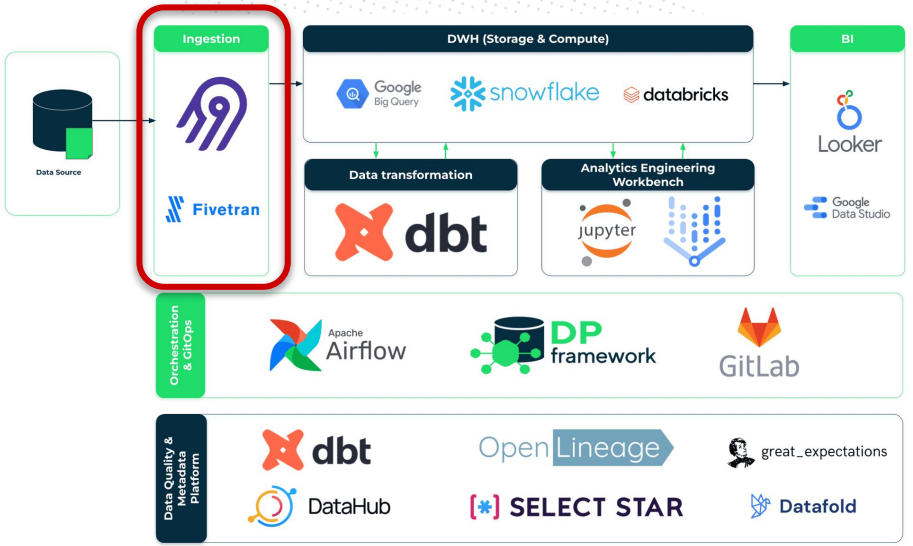


Key components of MDS

getindata

ETL vs. ELT

- **Storage** is now **cheap**
- Data **engineer's** time is **expensive**
- We need to be **flexible** & **scalable**
- Let's then **first load** the data, **then process** it - on **cloud DWH**!



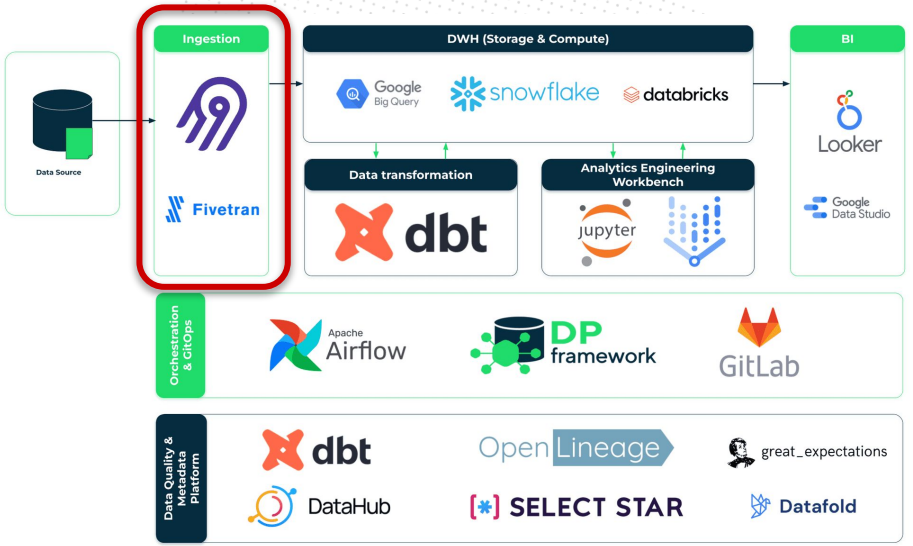
Modern data integration tools are already here!

Key components of MDS

getindata

Ingestion

- **Airbyte**: Open-source Extract & Load tool
- Promises to be the go-to OSS integration tool for **Modern Data Stack**
- Uses **dbt** on backend
- Hundreds of built-in connectors
- Lets you build own connectors with its SDK
- Can be orchestrated using Airflow but has own scheduler
- Wide adoption, large & dynamic community
- Main competitor: **Fivetran**

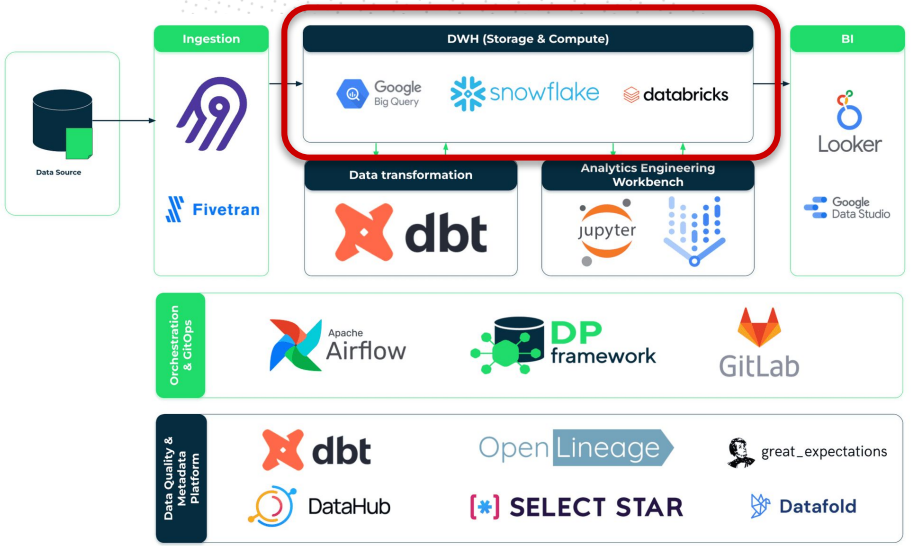


Key components of MDS

getindata

Storage & Compute

- Google **BigQuery** & **Google Cloud Storage** (other storage options also available)
- Mostly fully **managed**, near instant **scalability**, little-to-none infrastructure setup needed (**cost-effectiveness**)
- Many integrations with other tools **managed by cloud provider**
- **Security** and **governance** features built-in

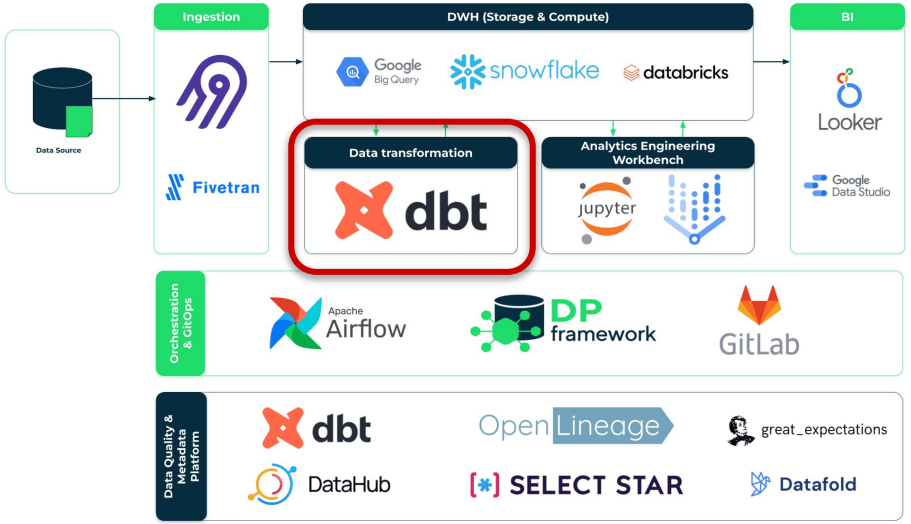


Key components of MDS

getindata

Data Transformation

- dbt (**D**ata-**B**uilding-**T**ool, 2016)
- **Low-code data transformation** tool (**SQL** plus simple yaml **configs**)...
- ...combined with **best engineering practices** of writing code
- **Integrations**: e.g. PostgreSQL, Redshift, Snowflake, Databricks, Spark, BigQuery
- Data **testing**, **documentation**, project **structuring**, **dependencies** management, environments configuration, and much more!
- Ideal tool for **analytics engineer**
- Also has a SaaS cloud version (**dbt cloud**)

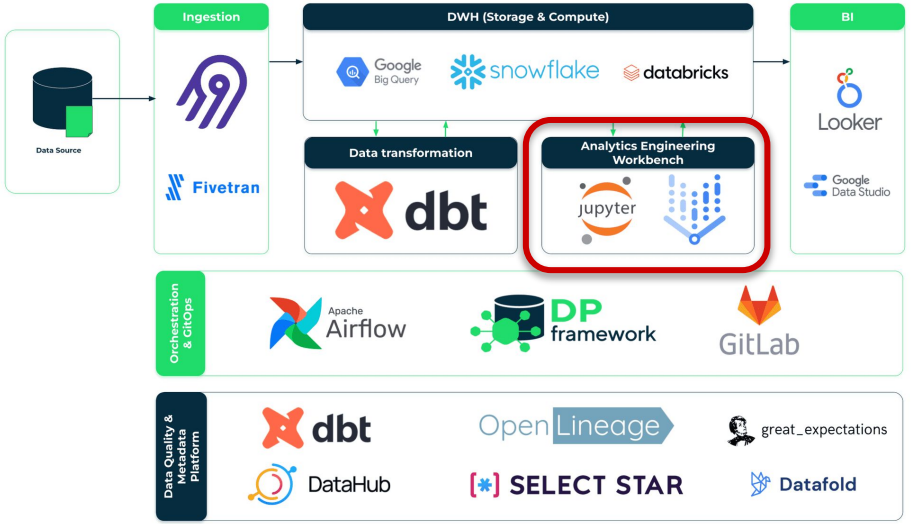


Key components of MDS

getindata

Analytics Engineering Workbench

- All the tools in **one place**!
- **JupyterLab** based images **deployed** on **Vertex AI Workbench**
- Data **transformation** (VS Code)
- Data **exploration** (Cloud Beaver, Jupyter notebook)
- Data pipelines **orchestration** (DP framework)
- Data **documentation** (dbt Docs)

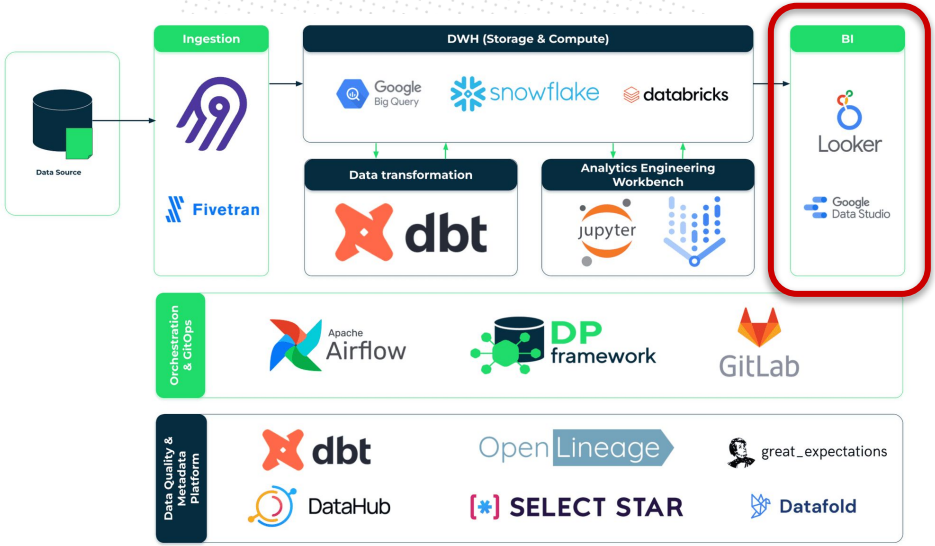


Key components of MDS

getindata

Business Intelligence

- **Self-service** data **consumption** tool
- **Integration** with **data transformation** layer, deployment automation
- **Version control** of artefacts (LookML files)
- Integrated data access control
- Data **visualization** & **interactive reporting**

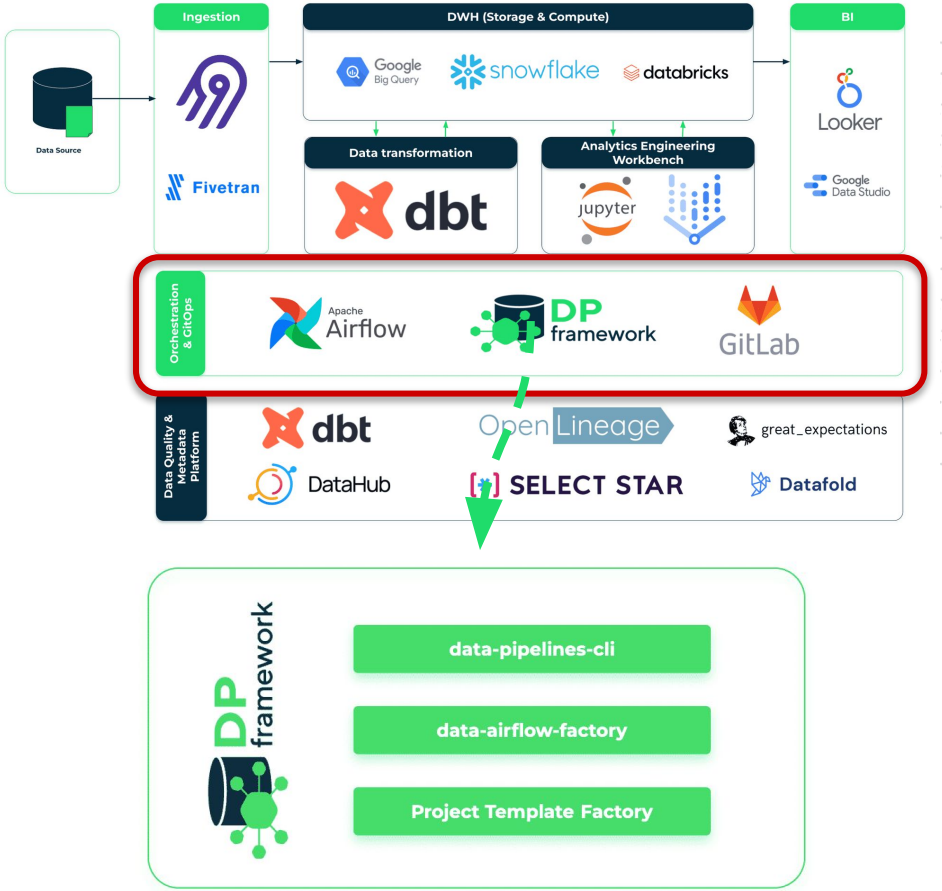


Key components of MDS

getindata

Orchestration & GitOps

- Data **pipelines orchestration** with **managed Apache Airflow (Cloud Composer)**
- **CI/CD & version control** with **GitLab**
- **DP Framework**
  - **data-pipelines-cli -** pipeline management in one place
  - **dbt-airflow-factory -** dbt model to DAG transformation
  - **Project Template Factory -** your standardized data projects library

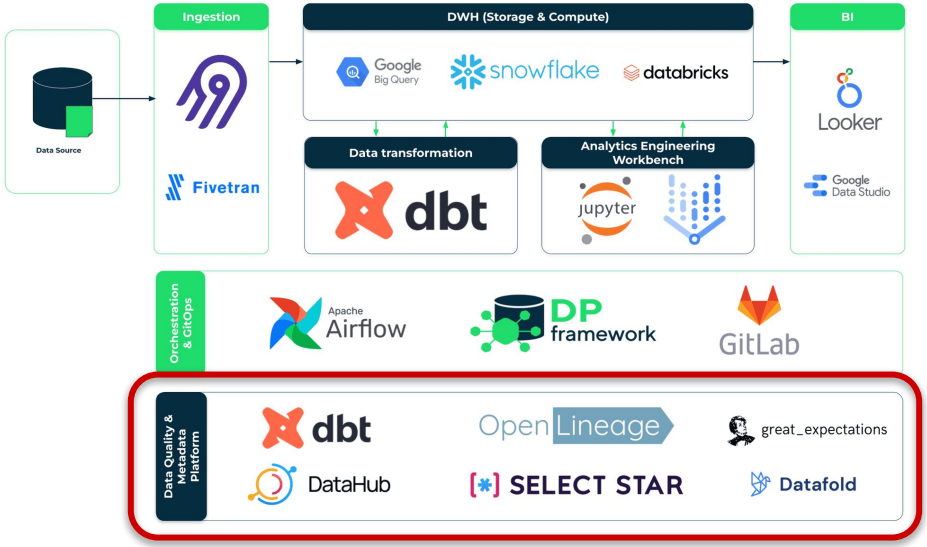


Key components of MDS

getindata

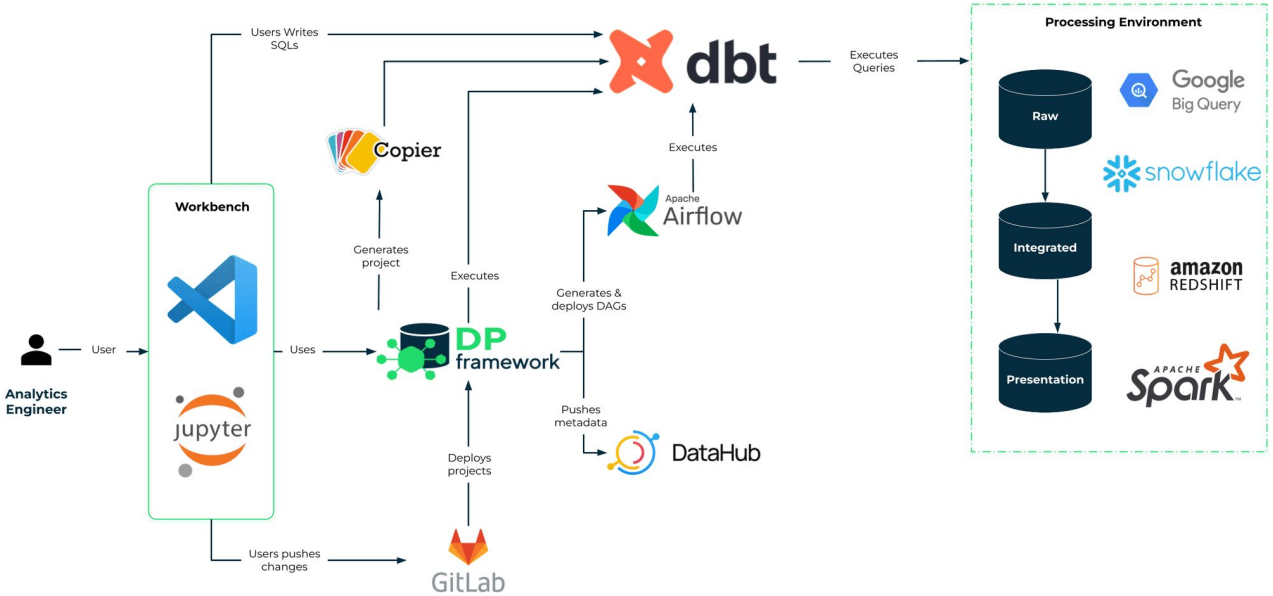
Data Discovery & Observability

- Automation of data **quality tests**
- Data **monitoring & alerting**
- Data **search** & **lineage**
- Data **profiling**
- Business **glossaries,** data **tagging**
- Data **documentation**
- Support for best-in-class **open-source** or **proprietary** solutions



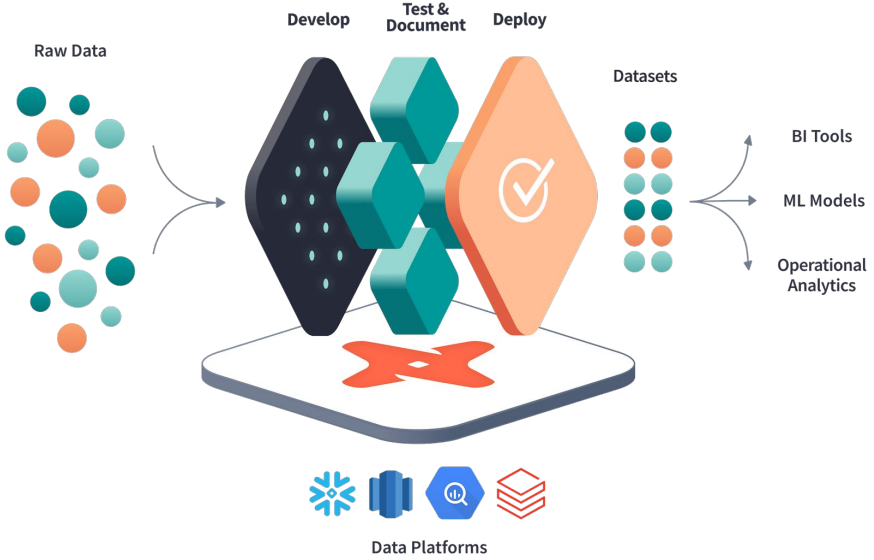
GetInData Modern Data Platform - WoW

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.”

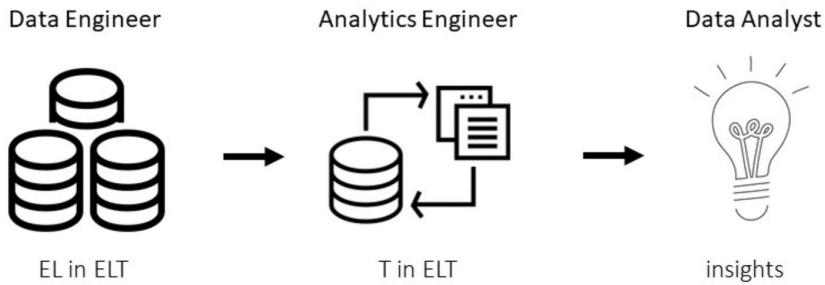
getdbt.com

Core concepts of dbt

What problem does dbt solve?

getindata

- Provides software engineering-grade way of writing data analysis code
  - Automated testing, documentation, project structuring, resolving dependencies, environments configuration
- It is designed for a particular role in the data pipeline process: analytics engineer

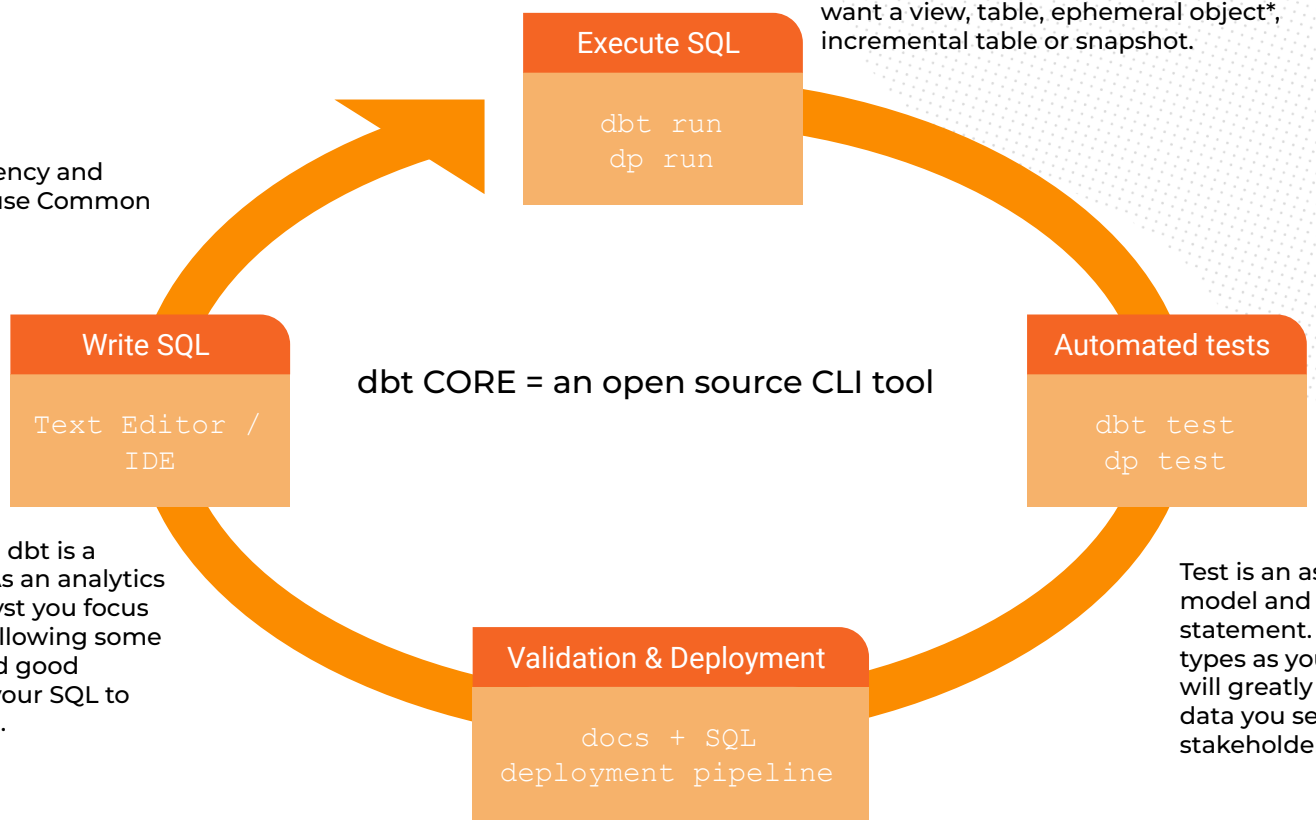


Core concepts of dbt

getindata

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.

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.

Core concepts of dbt

getindata

DEMO:

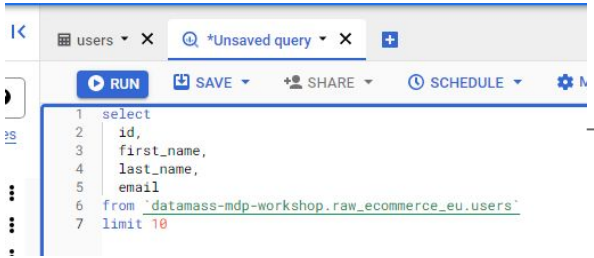
- Create the workspace
- First steps into dbt

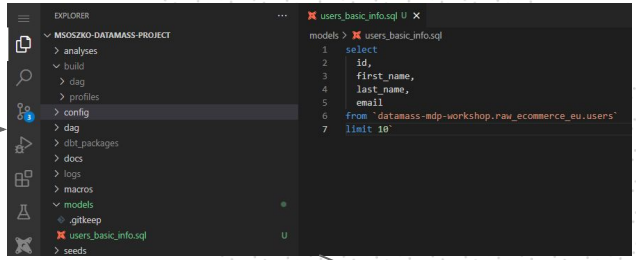
Core concepts of dbt

getindata

Hands-on exercises:
https://github.com/getindata/gid-mdp-workshop

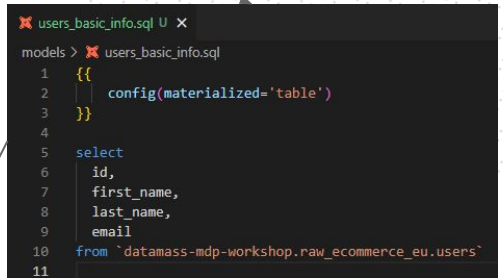
DEMO





getindata

In vertex AI workbench we use **dp** and **dbt** commands. That is because **dp** works on top of **dbt** and integrates it with other components. It is recommended that at least the last run you perform is done with DP.

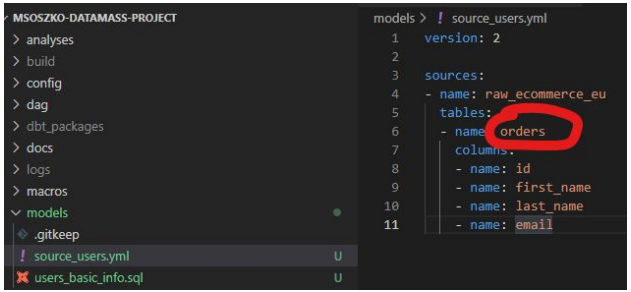


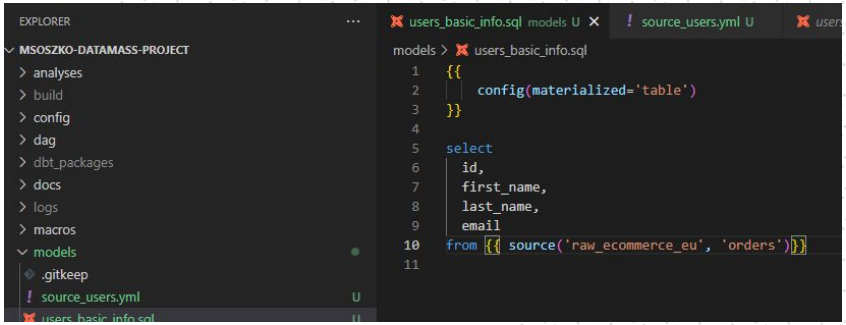

```
21:56:15  Running with dbt=1.0.8
21:56:16  Unable to do partial parsing because config vars, config profile, or config target have changed
21:56:16  Unable to do partial parsing because profile has changed
21:56:18  [WARNING]: Configuration paths exist in your dbt_project.yml file which do not apply to any resources.
There are 3 unused configuration paths:
- models.msoszko_datamass_test_project.base
- models.msoszko_datamass_test_project.staging
- models.msoszko_datamass_test_project.mart

21:56:18  Found 1 model, 0 tests, 0 snapshots, 0 analyses, 584 macros, 0 operations, 1 seed file, 0 sources, 0 exposures, 0 metrics
21:56:18
21:56:20  Concurrency: 1 threads (target='local')
21:56:20
21:56:20  1 of 1 START table model msoszko_private_working_schema.users_basic_info..... [RUN]
21:56:23  1 of 1 OK created table model msoszko_private_working_schema.users_basic_info... [CREATE TABLE (100.0k rows, 4.8 MB processed) in 2.93s]
21:56:23
21:56:23  Finished running 1 table model in 4.80s.
21:56:23
21:56:23  Completed successfully
21:56:23
21:56:23  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(base) root@7e7478c36fa8:/home/jupyter/msoszko-datamass-project#
```

Option: inspect and compare compiled SQL in target / run

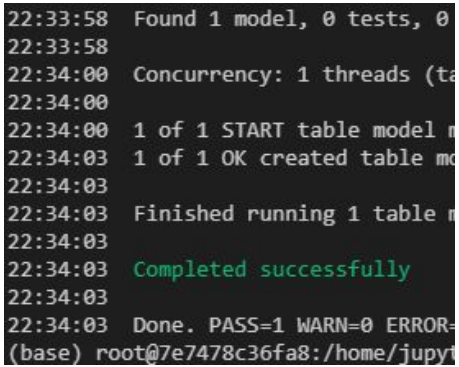
DEMO

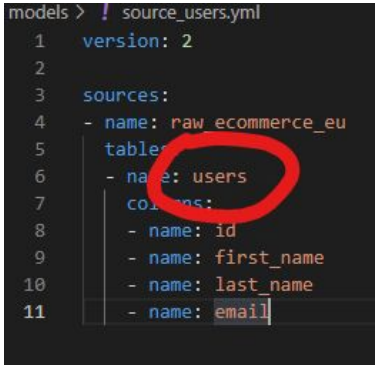
getindata

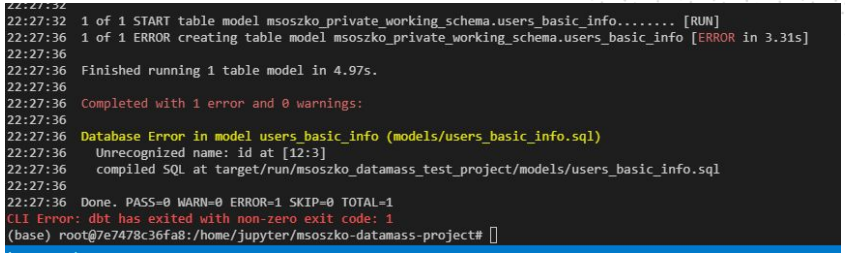




In vertex AI workbench we use **dp** and **dbt** commands. That is because **dp** works on top of **dbt** and integrates it with other components. It is recommended that at least the last run you perform is done with DP.







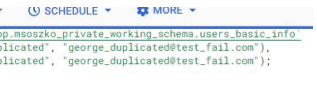
DEMO

getindata

```yaml
version: 2

models:
  - name: users_basic_info
    description: "This is a table with basic information about app users: first name, last name & email"
    columns:
      - name: id
        description: "This is users unique identification code"
      - name: first_name
        description: "This is users first name"
      - name: last_name
        description: "This is users last name"
      - name: email
        description: "This is users email address"
```

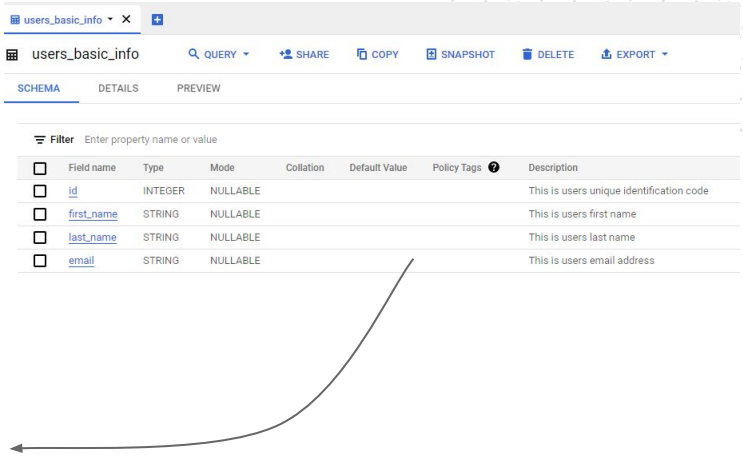
dp run

In vertex AI workbench we use **dp** and **dbt** commands. That is because **dp** works on top of **dbt** and integrates it with other components. It is recommended that at least the last run you perform is done with DP.




```
1 of 2 START test not_null_users_basic_info_id..... [RUN]
1 of 2 PASS not_null_users_basic_info_id..... [PASS in 1.30s]
2 of 2 START test unique_users_basic_info_id..... [RUN]
2 of 2 FAIL 1 unique_users_basic_info_id..... [FAIL 1 in 1.29s]

Finished running 2 tests in 3.81s.

Completed with 1 error and 0 warnings:

Failure in test unique_users_basic_info_id (models/users_basic_info.yml)
  Got 1 result, configured to fail if != 0

  compiled SQL at target/compiled/msoszko_datamass_test_project/models/users_basic_info.yml/unique_

Done. PASS=1 WARN=0 ERROR=1 SKIP=0 TOTAL=2
```

```yaml
version: 2

models:
  - name: users_basic_info
    description: "This is a table with basic information
    columns:
      - name: id
        description: "This is users unique identification
        tests:
          - unique
          - not_null
      - name: first_name
        description: "This is users first name"
      - name: last_name
        description: "This is users last name"
      - name: email
        description: "This is users email address"
```



DEMO

getindata

We have created a

- Configured
- Quality checked
- Documented
- Composed with respect to good practices...

... dbt model

user_basic_info.sql U × ! user_basic_info.yml U

msoszko-datamass-project > models > user_basic_info.sql

```sql
{{
    config(
      materialized='table',
      persist_docs={"relation":true, "columns": true}
    )
}}


with source as (
    select * from {{ source('raw_ecommerce_eu', 'users') }}
),
users as (

    select
        id                  as user_id,
        first_name          as user_first_name,
        last_name           as user_last_name,
        email               as user_email,
        age                 as user_age,
        gender              as user_gender,
        state               as user_address_state,
        street_address      as user_street_address,
        postal_code         as user_address_postal_code,
        city                as user_address_city,
        country             as user_address_country,
        latitude            as user_address_latitude,
        longitude           as user_address_longitude,
        traffic_source      as user_traffict_source,
        created_at          as user_account_created_at

    from source
)

select * from users

```

user_basic_info.sql U ! user_basic_info.yml U ×

msoszko-datamass-project > models > ! user_basic_info.yml

```yaml
version: 2

models:
  - name: user_basic_info
    description: "This is a table with basic information about app users: first name, last name & email"
    columns:
      - name: user_id
        description: "This is users unique identification code"
        tests:
          - unique
          - not_null
      - name: user_first_name
        description: "This is users first name"
      - name: user_last_name
        description: "This is users last name"
      - name: user_email
        description: "This is users email address"

```

Coffee break!

getindata

Agenda

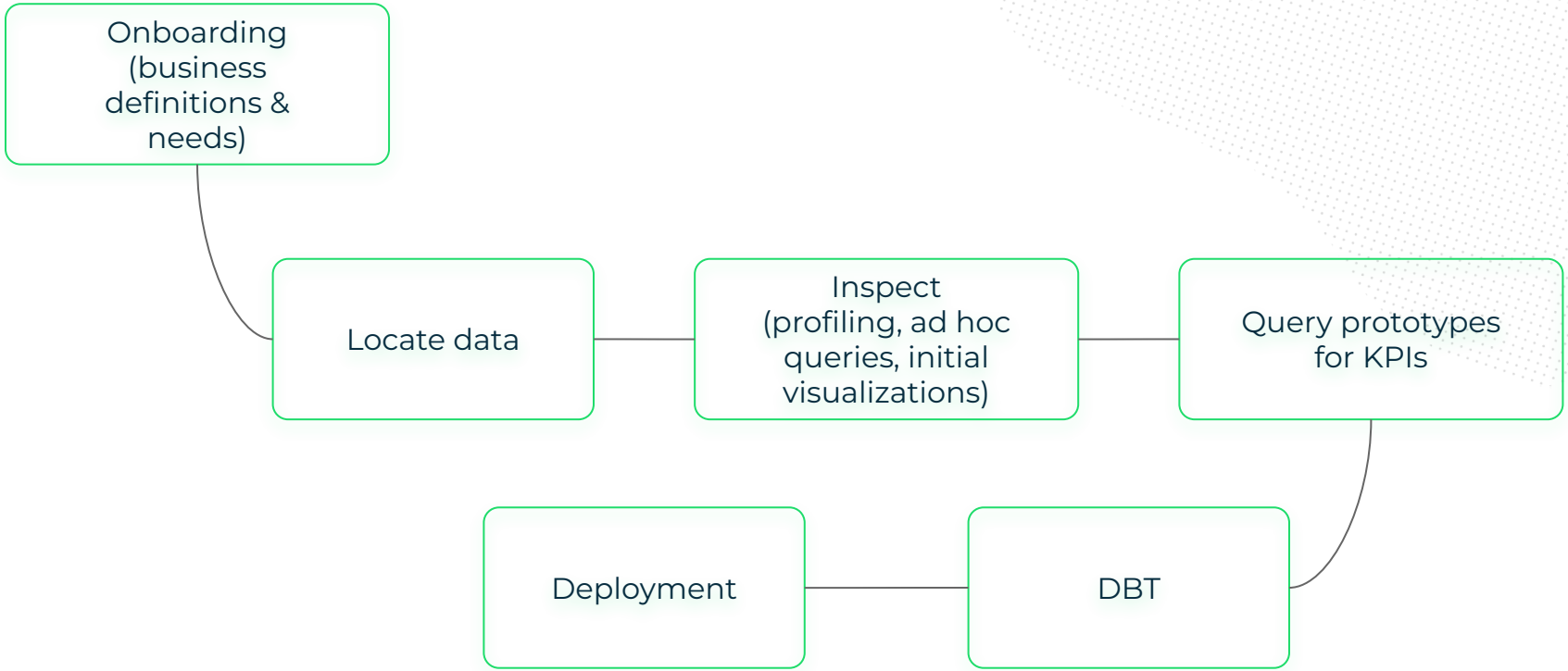
getindata

Session 2 - Simple end-do-end data pipeline

- ➔ Data discovery (data search, usage statistics, data lineage)
- ➔ Data profiling & exploration
- ➔ Transforming data using SQL with dbt
- ➔ Data consumption with BI tools
- ➔ Hands-on exercises

Typical workflow

getindata



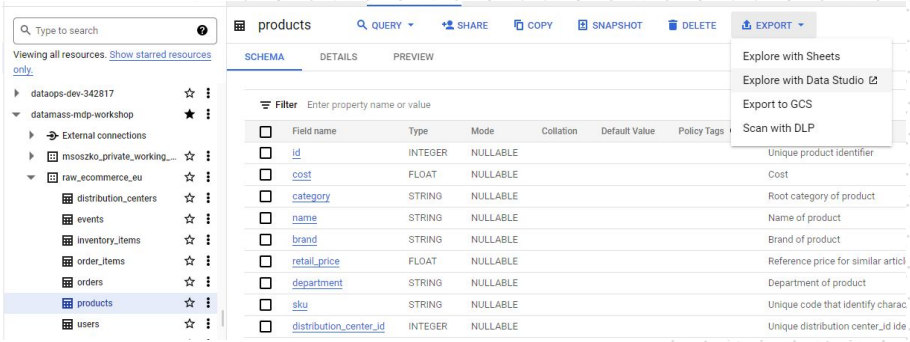
Discovery, exploration, profiling

getindata

- Data Discovery - making sure that all the data users can find and understand the data they need quickly and accurately
- Exploration - first / initial step in data analysis, semi-automated, visually summarised, often using ad hoc queries
- Profiling - a subset of the data exploration process for collecting statistics and informative summaries about data

Discovery, exploration, profiling

getindata

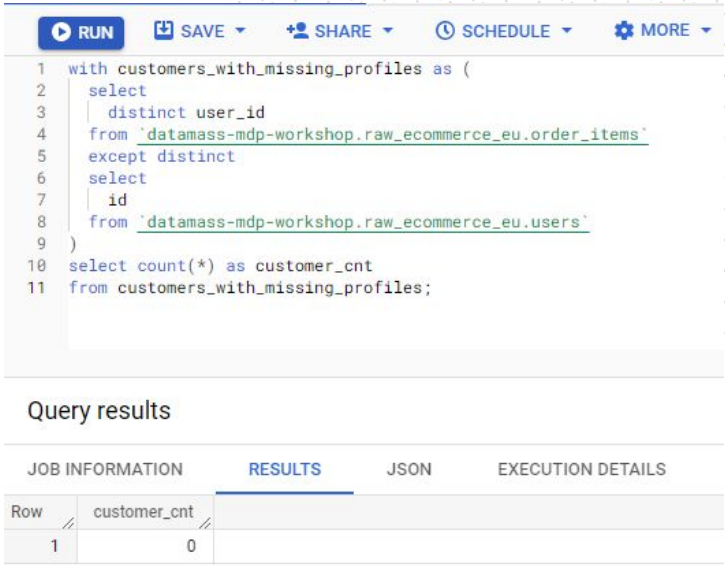


- Data Studio: basic BI tool hosted by GCP
- Allows quick look at data using simple graphs and tables (but they need to be created manually)
- Imperfect - ie. no histograms implementation (you need to define bins using case-when statements),
- Still helpful for some cases

Discovery, exploration, profiling

getindata

- Ad hoc exploration is performed using Bigquery console
- Raw data exploration results can be extracted to csv or visualized in DataStudio for quick preliminary reporting
- Final checks before “dbt” phase



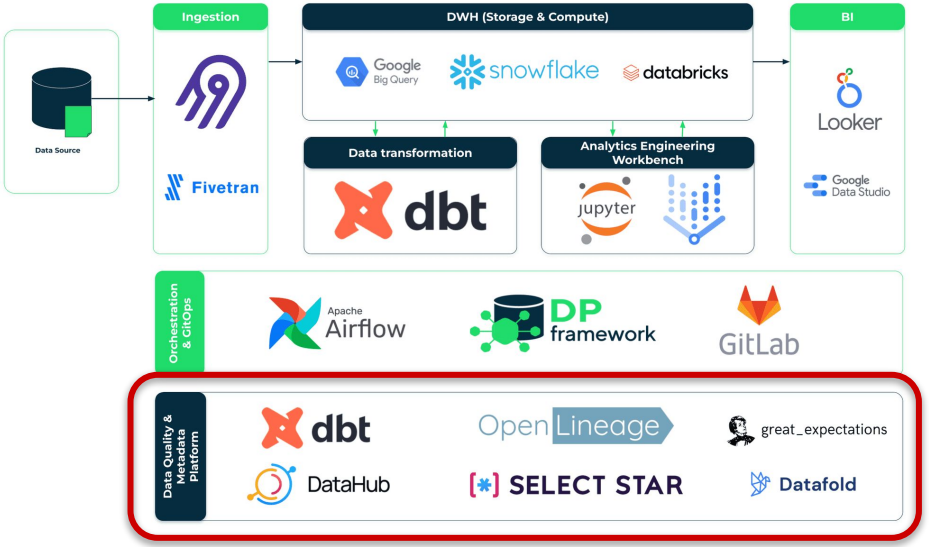
Discovery, exploration, profiling

getindata

- DataHub supports both push-based and pull-based metadata integration.
- Core concepts:
  - Sources (data systems where extracted metadata comes from)
  - Sinks (destination for metadata)
  - Recipes (configuration file, orchestrating ingestion with transformers etc.)

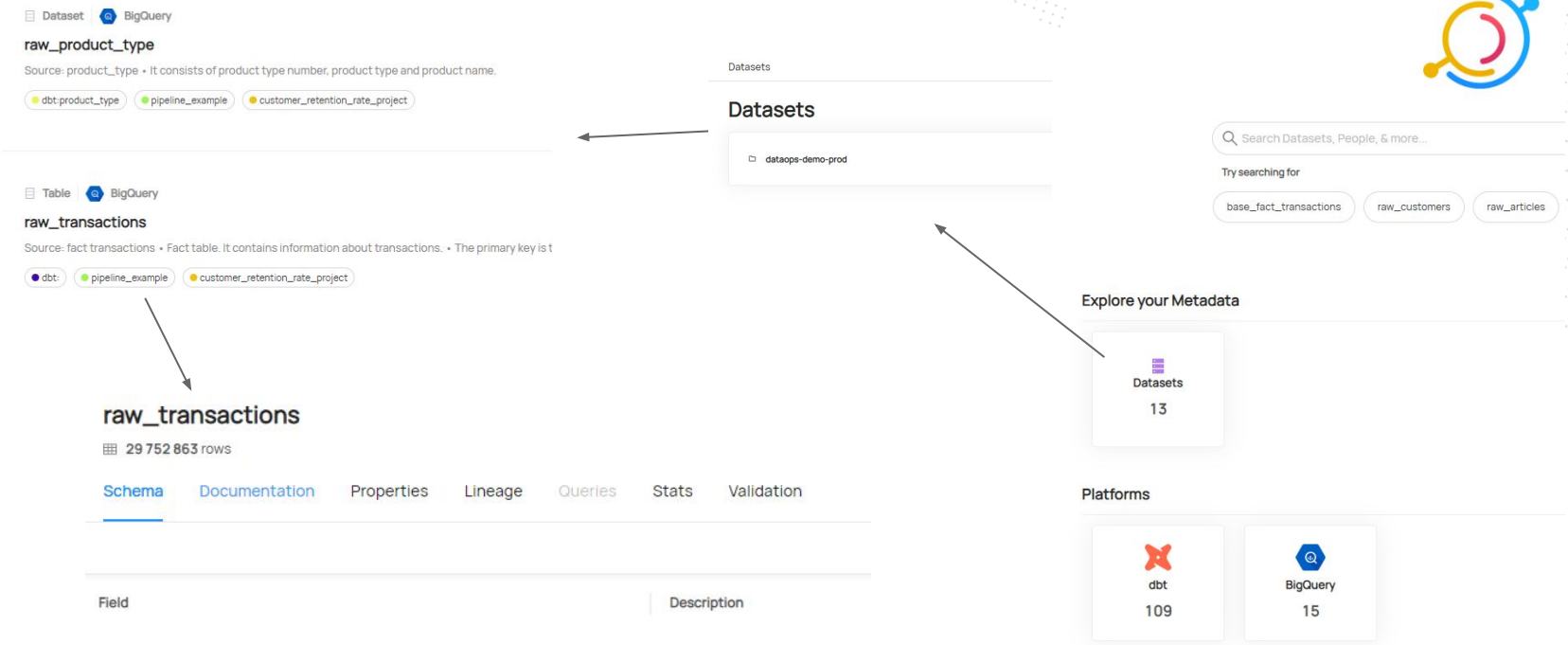
- As a data catalog - this could be the single source of truth for all data users

- Core features:
  - Data search
  - End-to-end lineage
  - Impact analysis
  - SQL Profiling



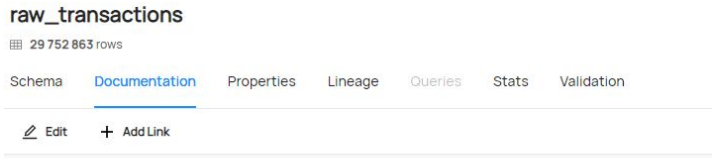
Discovery, exploration, profiling

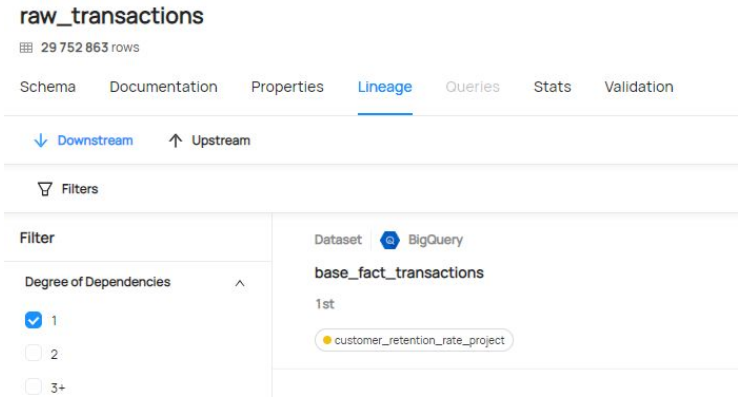
getindata

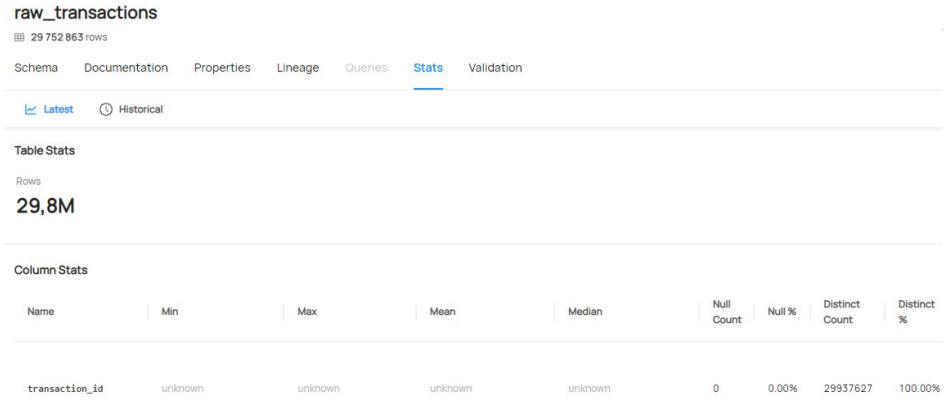


Discovery, exploration, profiling

getindata

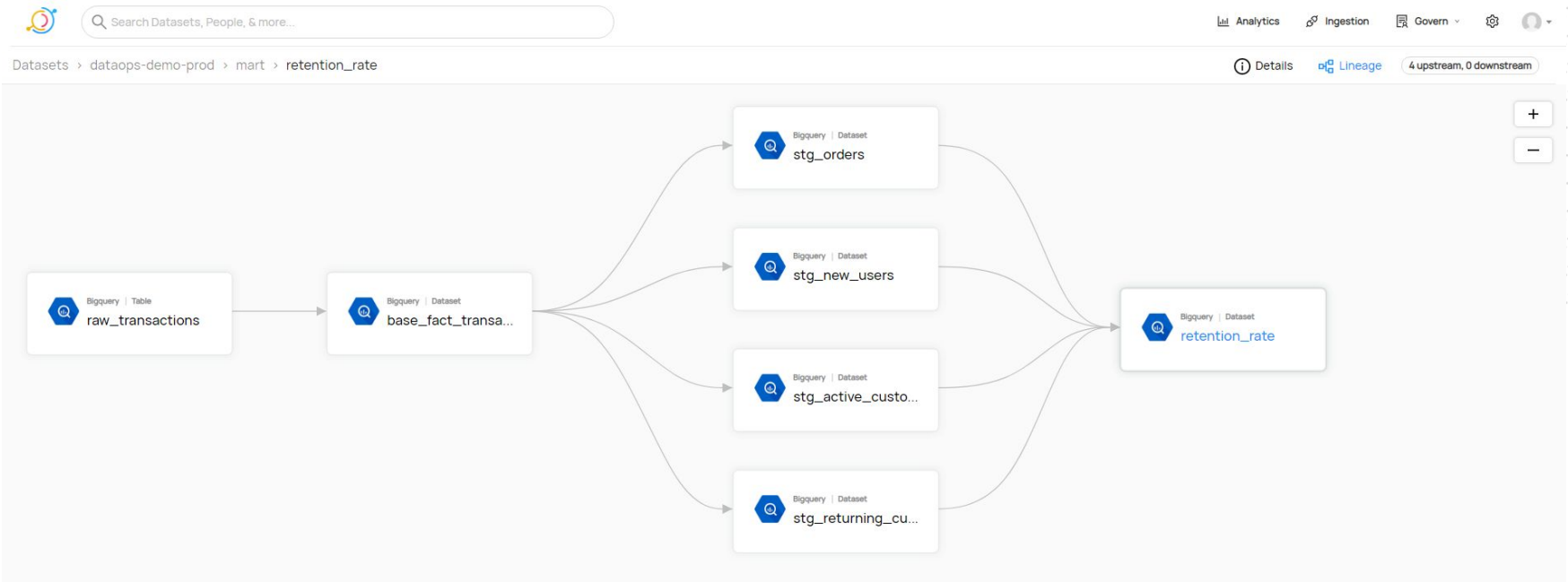






Discovery, exploration, profiling

getindata



Transforming data using SQL with dbt

Core dbt configuration

getindata

- Profiles.yml
  - configuring your environments (DEV, PROD etc)

Covered by MDP

- Dbt_project.yml
  - Specify operations/metaconfig. Every dbt project needs one

Flexible set-up by MDP

- 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

User-defined objects

Transforming data using SQL with 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

Transforming data using SQL with 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)

Transforming data using SQL with 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

Transforming data using SQL with dbt

getindata

DEMO:

- Simple dbt pipeline in dev

getindata

Hands-on exercises

Lunch!

getindata

Agenda

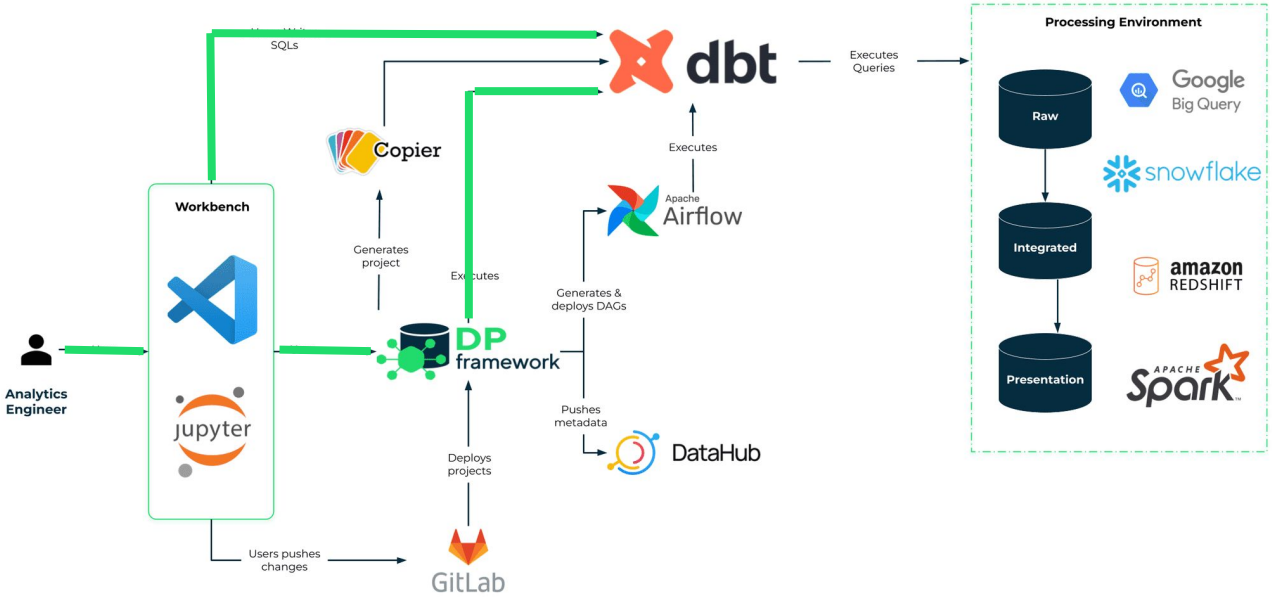
getindata

Session 3 - Data pipeline - scheduling, deployment & advanced features

- ➔ Apache Airflow as a workflow scheduler
- ➔ Data testing & data observability
- ➔ Exploring transformed data with Data Studio
- ➔ Hands-on exercises

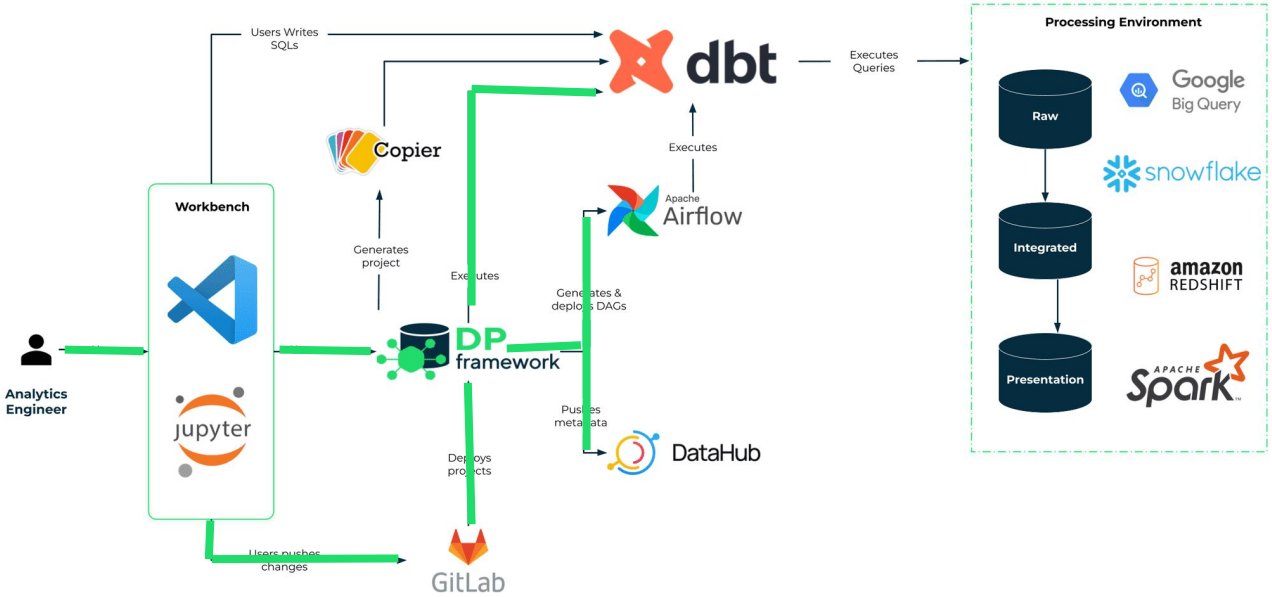
Session #3

Data pipeline - scheduling, deployment & advanced features



Session #3

Data pipeline - scheduling, deployment & advanced features



CICD: Gitlab phase

getindata

- **Git:** a version control system used to track changes in files
- **Gitlab:** a complete DevOps platform offering web-based Git repository
- Powerfull **CI/CD** pipelines
- **Continuous Integration**: managing changes in the code once it was published, while ensuring automated verification and review (in our case: dbt tests coverage, documentation, naming patterns check etc.)
- **Continuous Delivery**: apply and publish changes into repositories, automatic build creation and testing.
- With dbt on board we are able to work according to software engineering best practices - that includes CI/CD.
- In **Modern Data Platform** the **CI/CD** phase is preconfigured

CICD: Gitlab phase

getindata

dbt meta testing - a package to assert test and documentation coverage - especially useful for CI/CD

```yaml
# dbt_project.yml
...
models:
  project:
    +required_docs: true
    marts:
      +required_tests: {"unique.*|not_null": 1}
      model_2:
        +required_tests:
          "mocker.*|unique": 1
          "mock_schema_test": 1
          ".*data_test": 1
```

```
usr@home dbt-meta-testing $ dbt run-operation required_tests
Running with dbt=0.20.0
Encountered an error while running operation: Compilation Error in macro required_tests (macros/required_tests.sql)
  Insufficient test coverage from the 'required_tests' config on the following models:
  Model: 'model_1' Test: 'not_null' Got: 1 Expected: 2
  Model: 'model_1' Test: 'mock_schema_test' Got: 0 Expected: 1

  > in macro _evaluate_required_tests (macros/utils/required_tests/evaluate_required_tests.sql)
  > called by macro required_tests (macros/required_tests.sql)
  > called by macro required_tests (macros/required_tests.sql)
usr@home dbt-meta-testing $
```

```
usr@home dbt-meta-testing $ dbt run-operation required_docs
Running with dbt=0.20.0
Encountered an error while running operation: Compilation Error in macro required_docs (macros/required_docs.sql)
  The following models are missing descriptions:
   - model_2
  The following columns are missing from the model yml:
   - model_2.new
  The following columns are present in the model yml, but have blank descriptions:
   - model_1.id

  > in macro _evaluate_required_docs (macros/utils/required_docs/evaluate_required_docs.sql)
  > called by macro required_docs (macros/required_docs.sql)
  > called by macro required_docs (macros/required_docs.sql)
usr@home dbt-meta-testing $
```

CICD: Gitlab phase

getindata

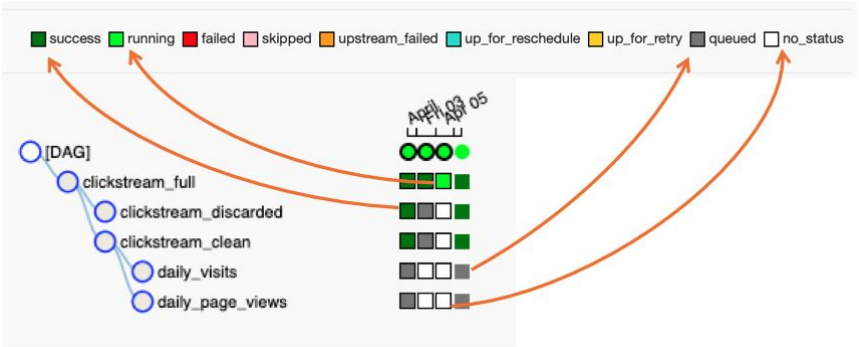
DEMO:

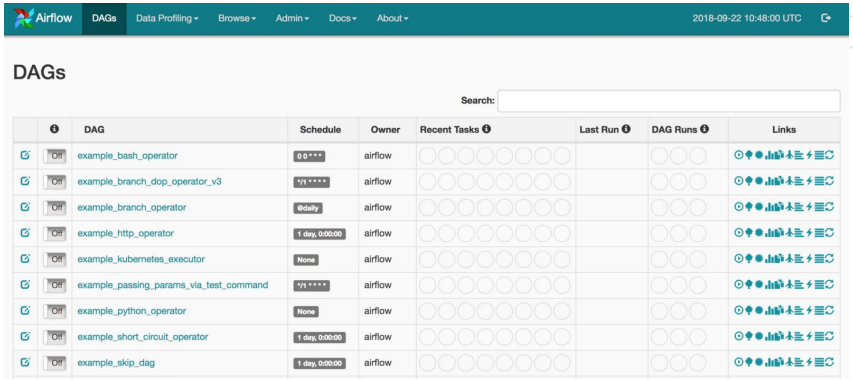
- Commit our work
- Watch CI-CD

Apache Airflow as a workflow scheduler

getindata

- an open-source workflow management platform for data engineering pipelines
- manage scheduling and running jobs and data pipelines
- Ensures jobs are ordered correctly based on dependencies
- Provides mechanisms for tracking the state of jobs and recovering from failure

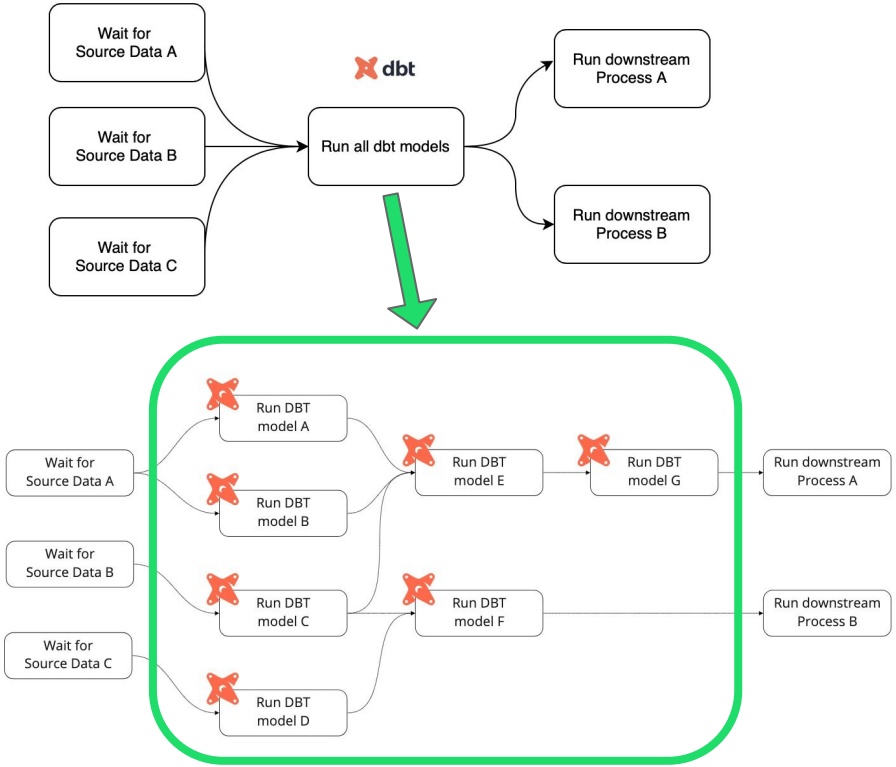




https://airflow-tutorial.readthedocs.io/

Apache Airflow as a workflow scheduler

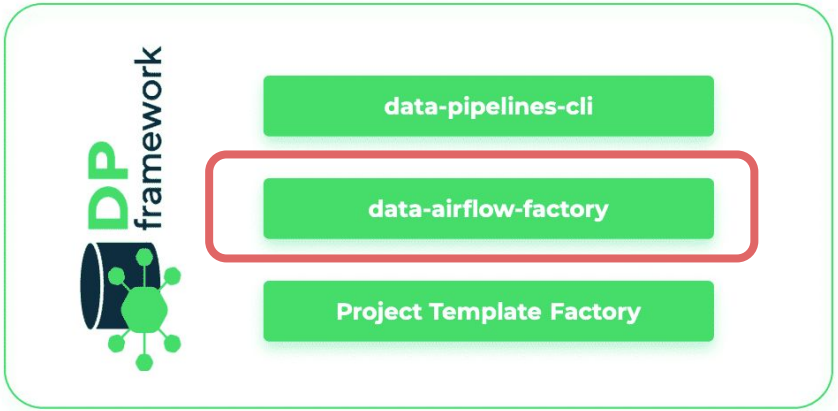
getindata



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

Apache Airflow as a workflow scheduler

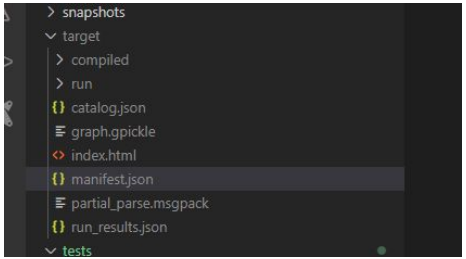
getindata



- The data-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 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 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()
```

manifest.json

target > {} manifest.json > {} nodes > {} model.datamass_project.int_events_traffic_sources_pivoted > raw_sql


```
{"metadata": {"dbt_schema_version": "https://schemas.getdbt.com/dbt/manifest/v6.json", "dbt_ve
"invocation_id": "b96b0f01-9d64-465b-a096-ec6172e74f0e", "env": {}, "project_id": "ad92464d329
"send_anonymous_usage_stats": true, "adapter_type": "bigquery"}, "nodes": {"seed.datamass_proj
"depends_on": {"macros": [], "nodes": []}, "config": {"enabled": true, "alias": null, "schema"
"seed", "persist_docs": {}, "quoting": {}, "column_types": {}, "full_refresh": null, "unique_k
null, "post-hook": [], "pre-hook": []}, "database": "datamass-mdp-workshop", "schema": "michal
"ISO_like_Countries-Continents"], "unique_id": "seed.datamass_project.ISO_like_Countries-Conti
"c:\\Users\\Micha\u0142 Soszko\\Projects\\Workshops\\msoszko-datamass-project", "path": "ISO_l
"seeds\\ISO_like_Countries-Continents.csv", "name": "ISO_like_Countries-Continents", "alias":
"c62d61682f55ec9b83356d00cf61bb2c24046d816c6c43fd17faca6861b61543"}, "tags": [], "refs": [], "
"docs": {"show": true}, "patch_path": "datamass_project://seeds\\ISO_like_Countries-continents.
"unrendered_config": {}, "created_at": 1664277093.1640687}, "model.datamass_project.int_events
materialized='table',\r\n      persist_docs={\"relation\":true, \"columns\": true}\r\n    )\r\
('stg_ecommerce__events') }}\r\n),\r\npivot_event_traffic_source_aggd_by_user as (\r\n    sele
'Adwords' then 1 else 0 end)    as adwords_traffic,\r\n        sum(case when event_traffic_sour
when event_traffic_source = 'Facebook' then 1 else 0 end)  as facebook_traffic,\r\n        sum
organic_traffic,\r\n        sum(case when event_traffic_source = 'YouTube' then 1 else 0 end)
(event_traffic_source)                                          as total_traffic,\r\n        mi
```


```python
import datetime
import pendulum
import os

import requests
from airflow.decorators import dag, task
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.providers.postgres.operators.postgres import PostgresOperator


@dag(
    dag_id="process-employees",
    schedule_interval="0 0 * * *",
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
)
def ProcessEmployees():
    create_employees_table = PostgresOperator(
        task_id="create_employees_table",
        postgres_conn_id="tutorial_pg_conn",
        sql="""
            CREATE TABLE IF NOT EXISTS employees (
                "Serial Number" NUMERIC PRIMARY KEY,
                "Company Name" TEXT,
                "Employee Markme" TEXT,
                "Description" TEXT,
```

Apache Airflow as a workflow scheduler

getindata

DEMO:

- Inspect DAG

Data testing & data observability

getindata

- Bespoke tests:
  - Write a SQL query that should return failing rows, put it under tests directory, execute with a single command
- Generic tests:
  - Define tests in special blocks within .yml files, which correspond to dbt objects such as models, columns, sources, snapshots or seeds.
  - There are a handful of built-ins and special plugins for various cases

```
unique

not_null

accepted_values

relationships
```

Data testing & data observability

getindata

```
version: 2

models:
  - name: orders
    columns:
      - name: order_id
        tests:
          - unique
          - not_null
      - name: status
        tests:
          - accepted_values:
              values: ['placed', 'shipped', 'completed', 'returned']
      - name: customer_id
        tests:
          - relationships:
              to: ref('customers')
              field: id
```

standard, generic tests

“foreign key” check

Data testing & data observability

getindata

dbt Packages storing dozen of generic tests and macros :

- dbt-labs/dbt_utils
- calogica/dbt_expectations

- If those do not provide tests you require, you can either write a macro / singular test of your own or... search for more packages - https://hub.getdbt.com/

Data testing & data observability

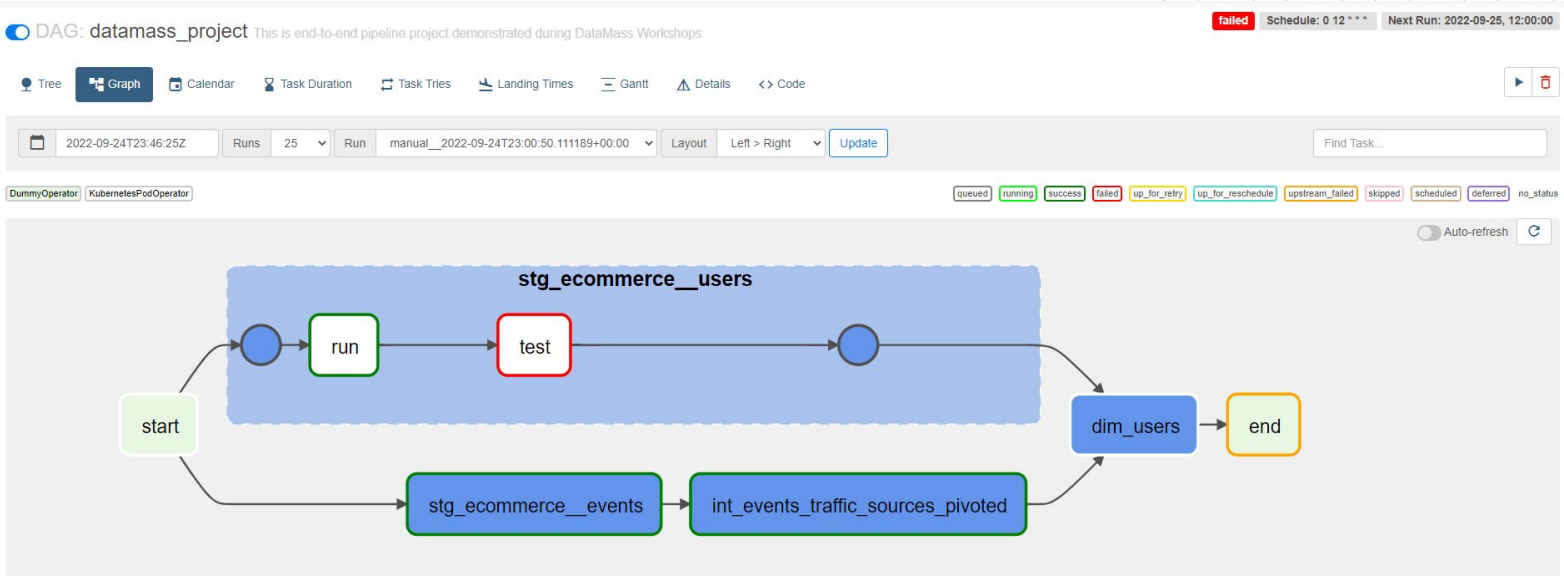
getindata

DEMO:

- Custom tests in dbt

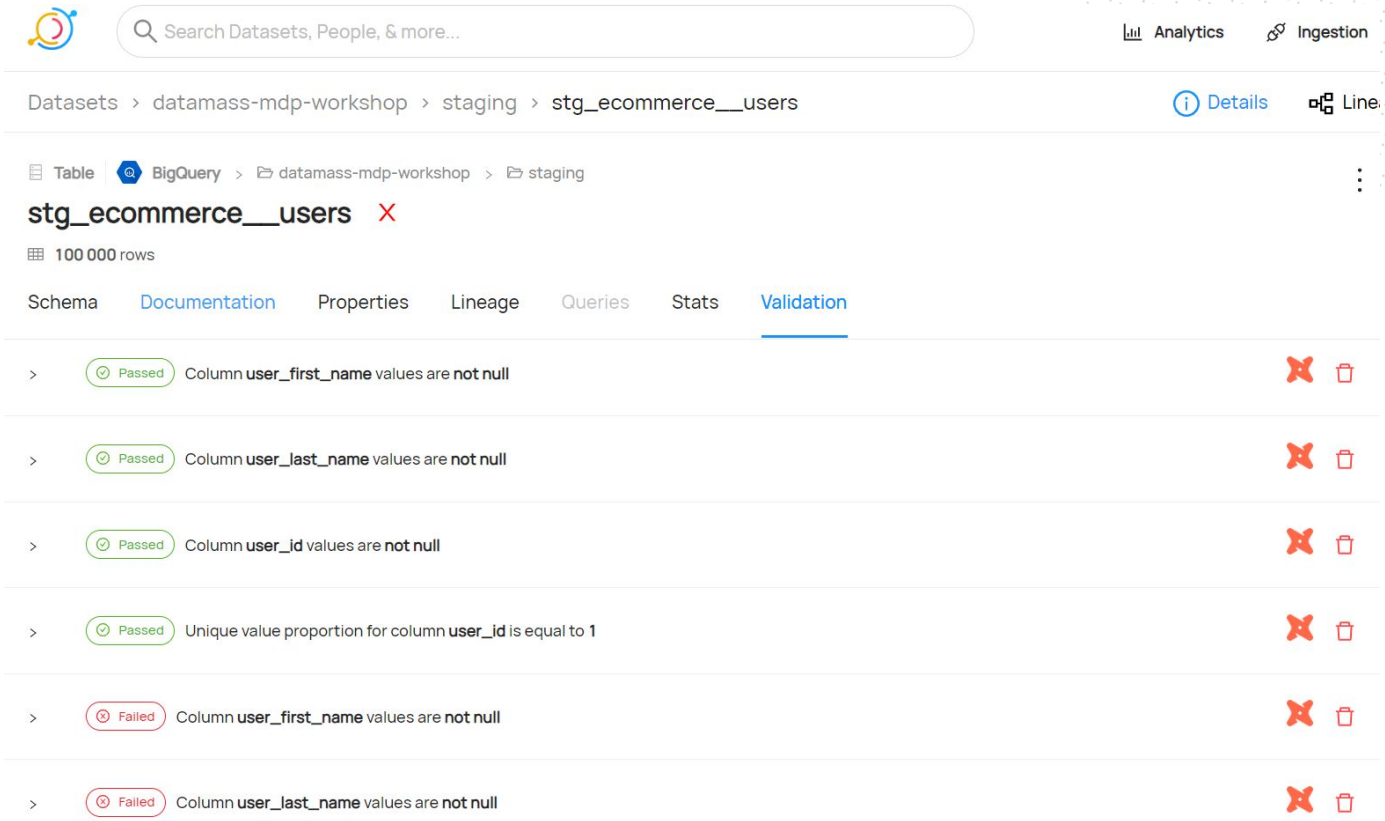
Data testing & data observability

getindata



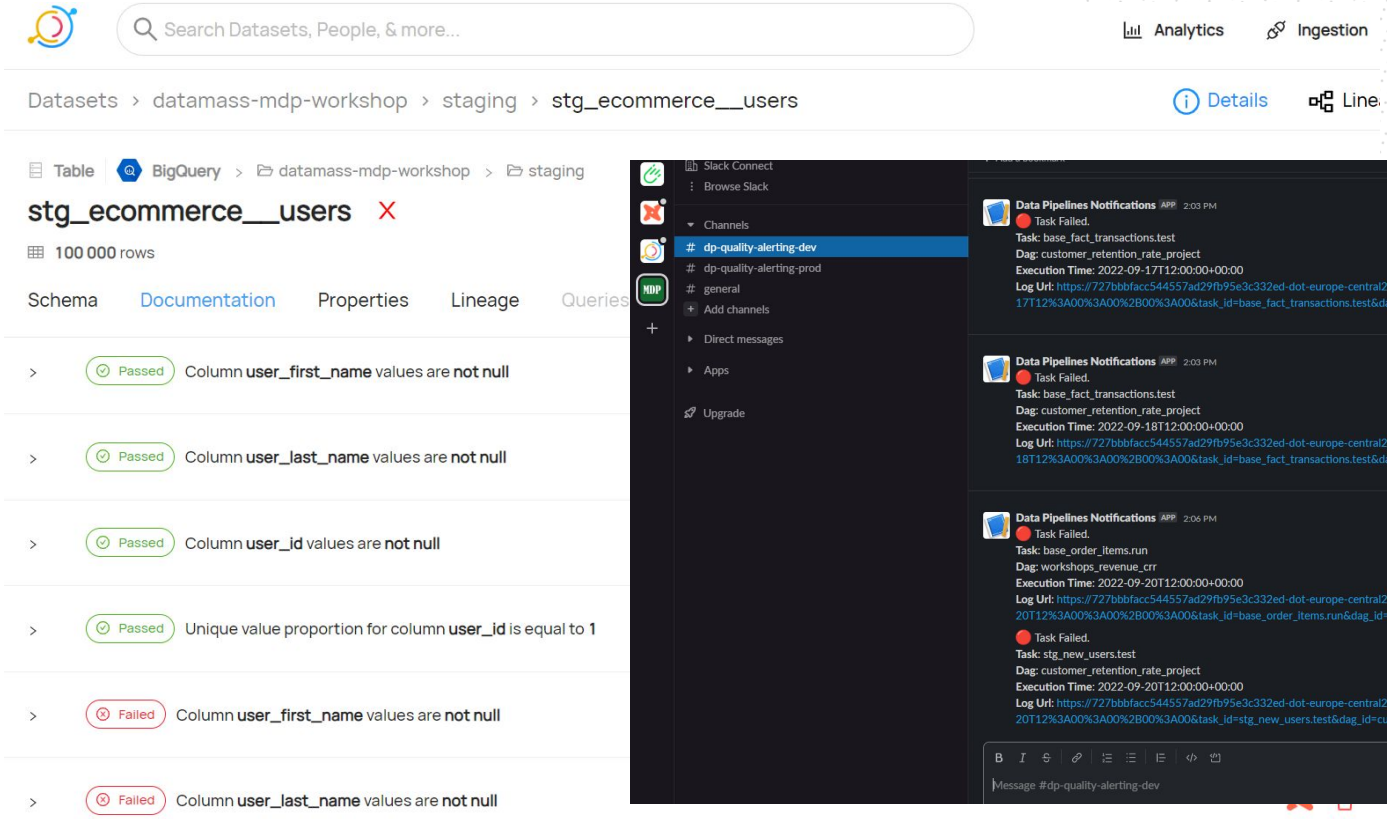
Data testing & data observability

getindata



Data testing & data observability

getindata



Exploring transformed data with Data Studio

getindata

[Link]

getindata

Hands-on exercises

Wrap up

getindata

In this workshop you've learnt:

- What are the key features and tools of MDP
- What is dbt and how to create a SQL pipeline
- How to deploy the dbt pipeline using MDP
- How to check and monitor the quality of your data with dbt and MDP

Wrap up

getindata

Thank you!

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

Our blog
https://getindata.com/blog

Our podcast
Radio DaTa