



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

Session 2 - Simple end-to-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

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

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?

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

Why we talk about modern data stack now?



Challenges



Needs



Enablers

Data platform challenges

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**)

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)

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)

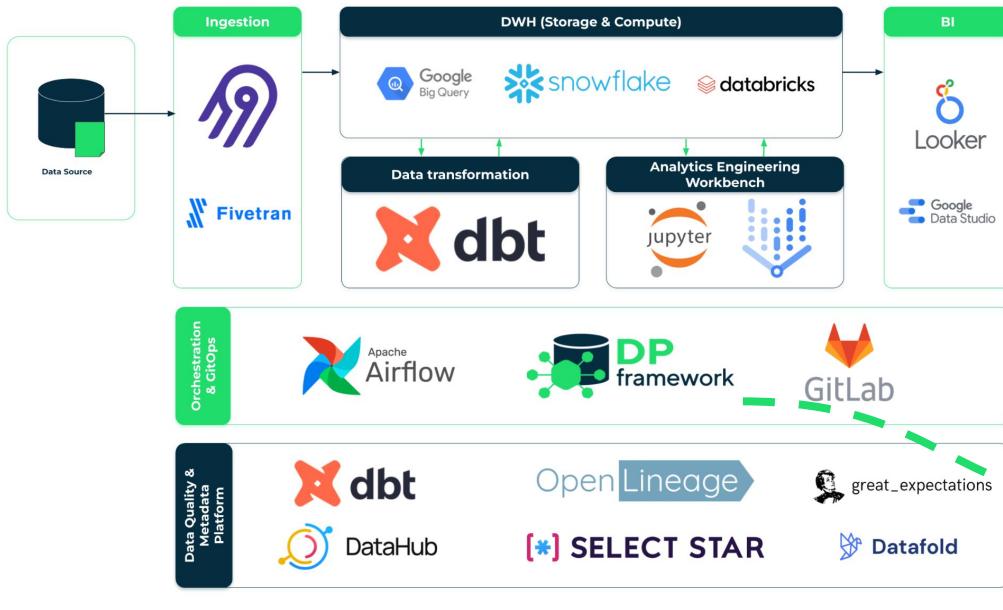
The technology is already here!



Architecture &
engineering best
practices



GetInData Modern Data Platform - our framework (GCP example)



Solution optimized for:

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

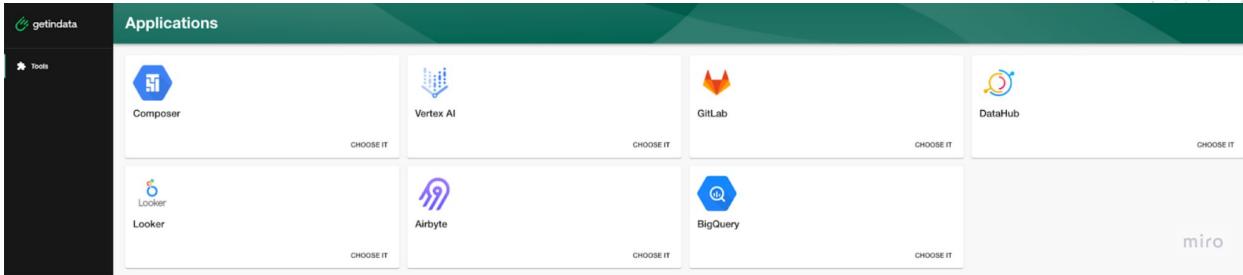


`data-pipelines-cli`

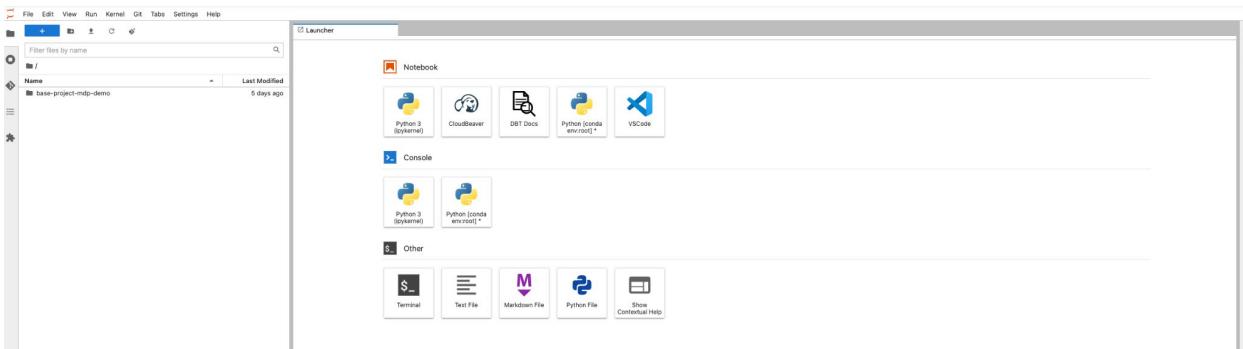
`data-airflow-factory`

`Project Template Factory`

MDP - user interface

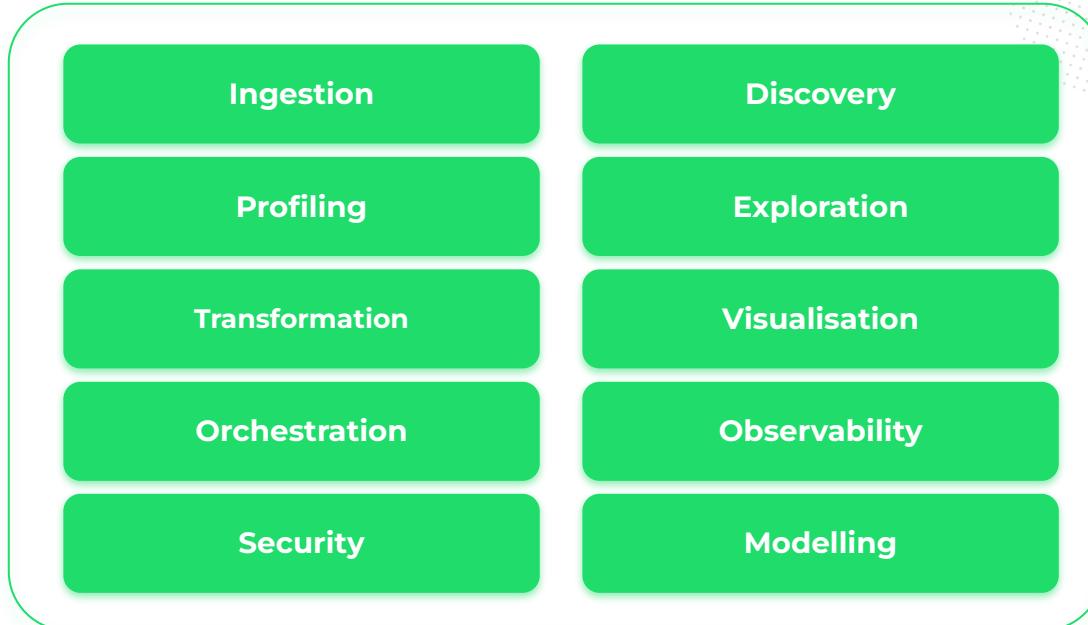


Access all your apps
via **GID Portal**

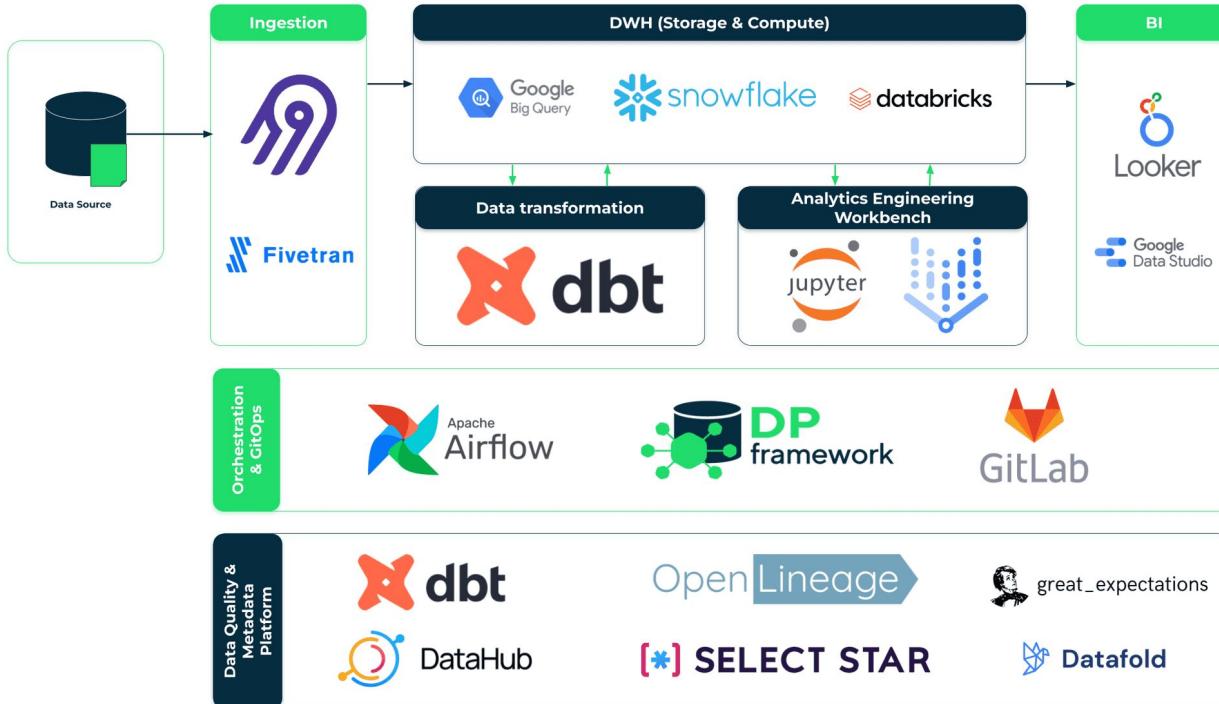


All you need for data
processing is in
Analytics Workbench

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



Key components of MDS

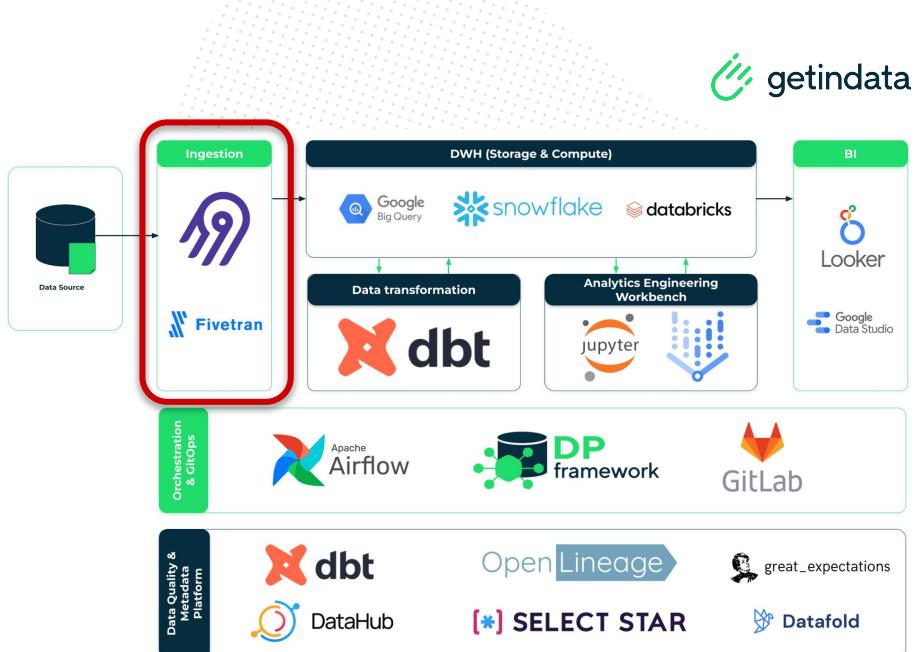


Key components of MDS



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!**



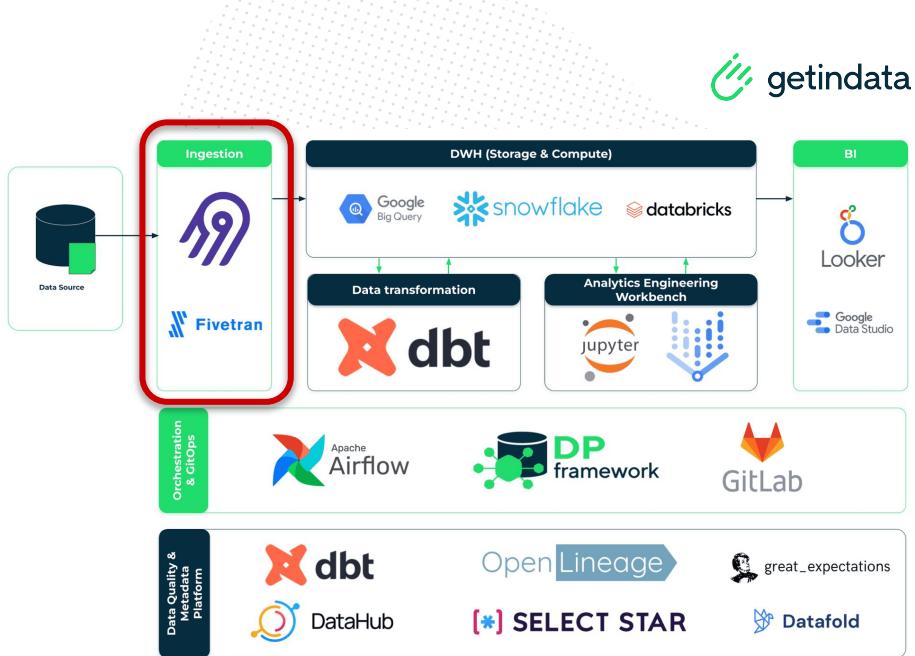
Modern data integration tools are already here!

Key components of MDS



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**

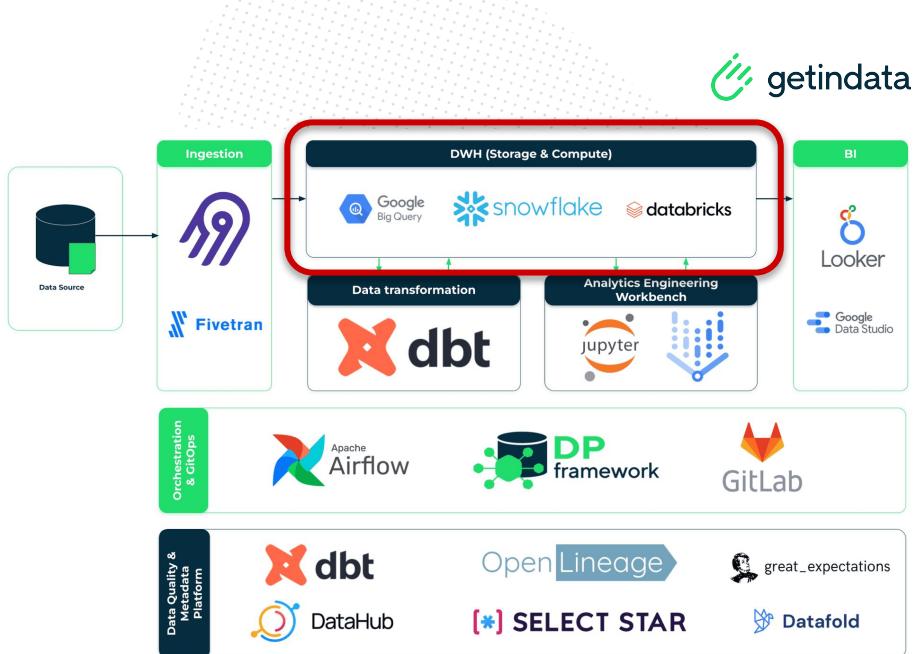


Key components of MDS



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

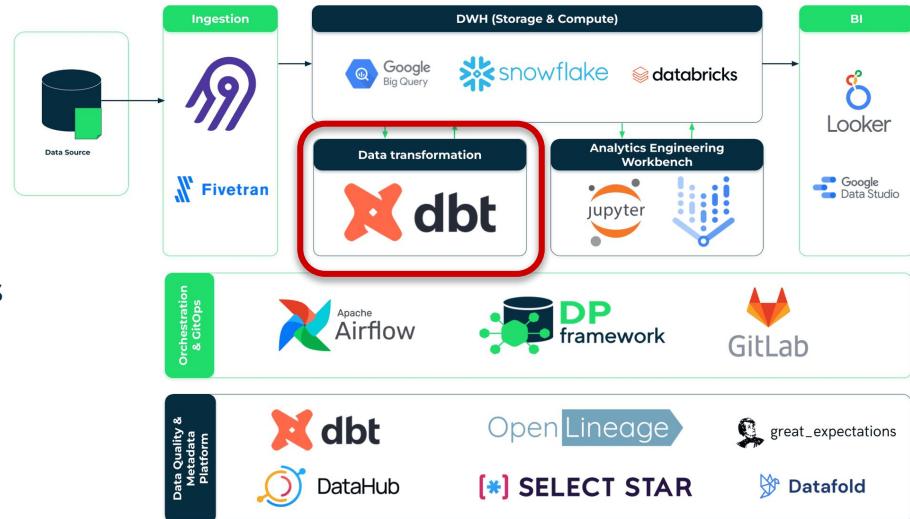


Key components of MDS



Data Transformation

- **dbt (Data-Building-Tool, 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**)

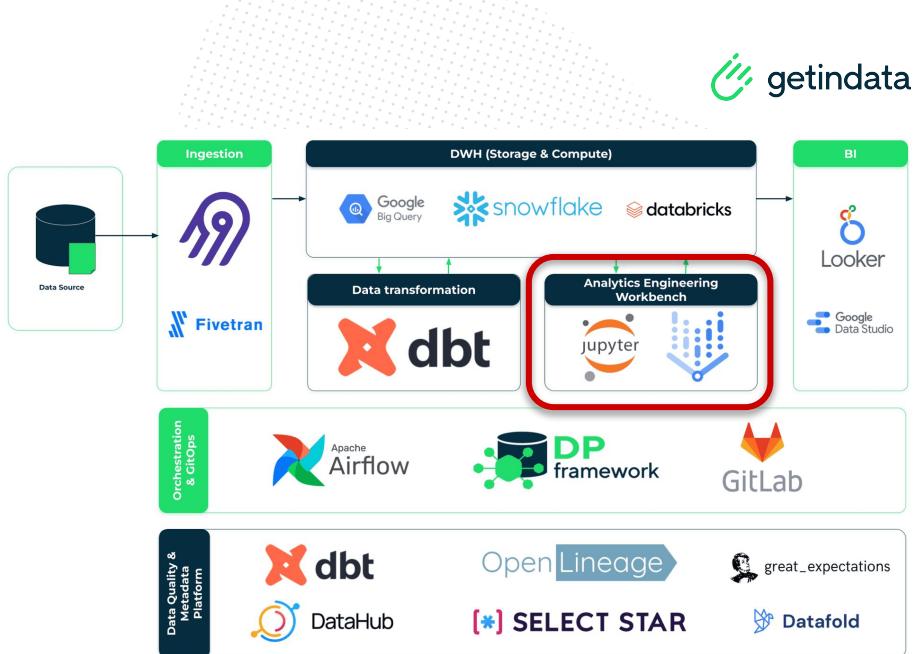


Key components of MDS



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)

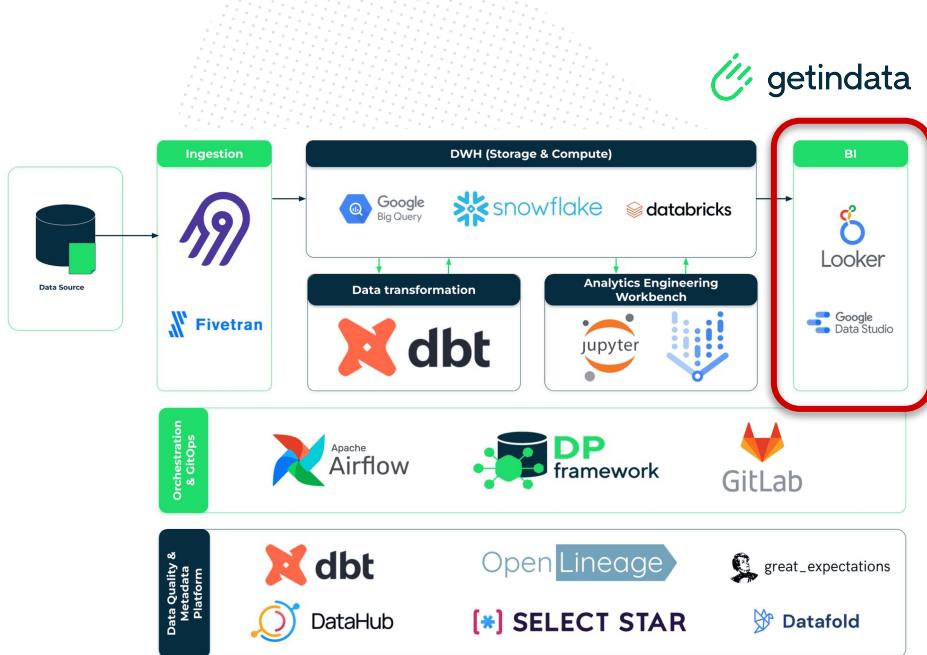


Key components of MDS



Business Intelligence

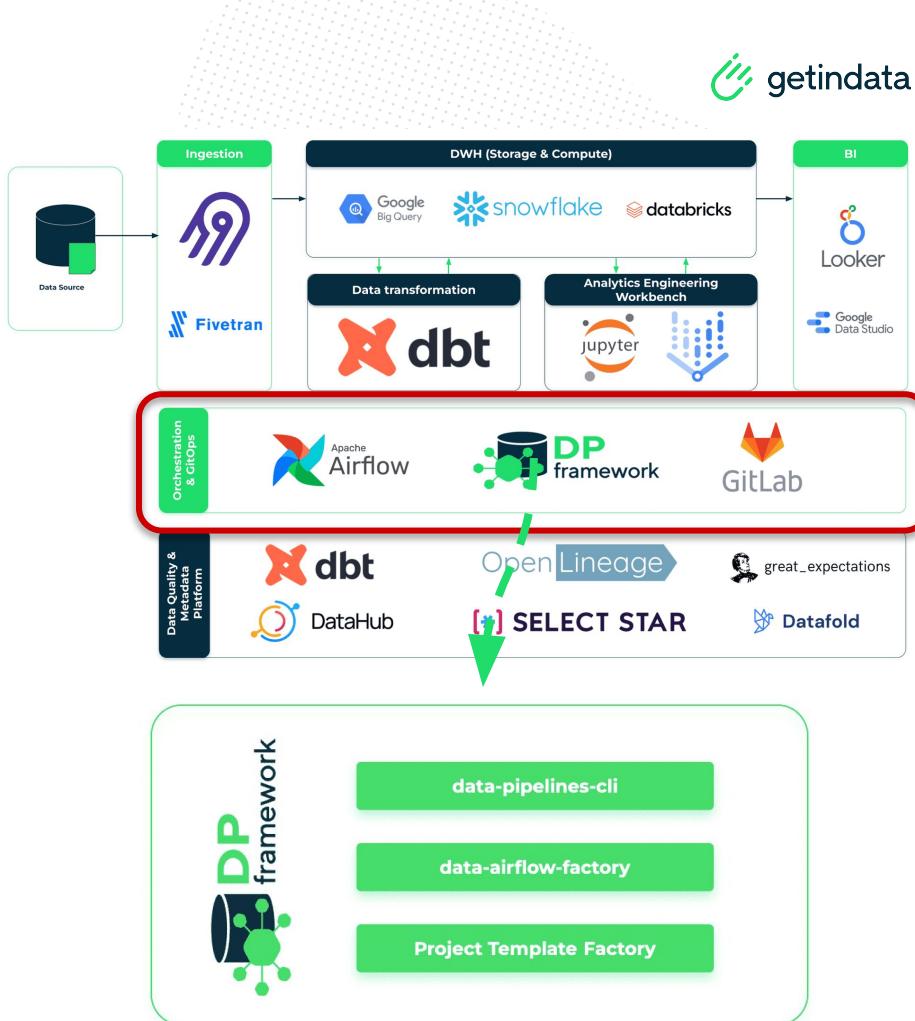
- **Self-service data consumption tool**
- **Integration with data transformation layer**, deployment automation
- **Version control** of artifacts (LookML files)
- Integrated data access control
- Data **visualization & interactive reporting**



Key components of MDS

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

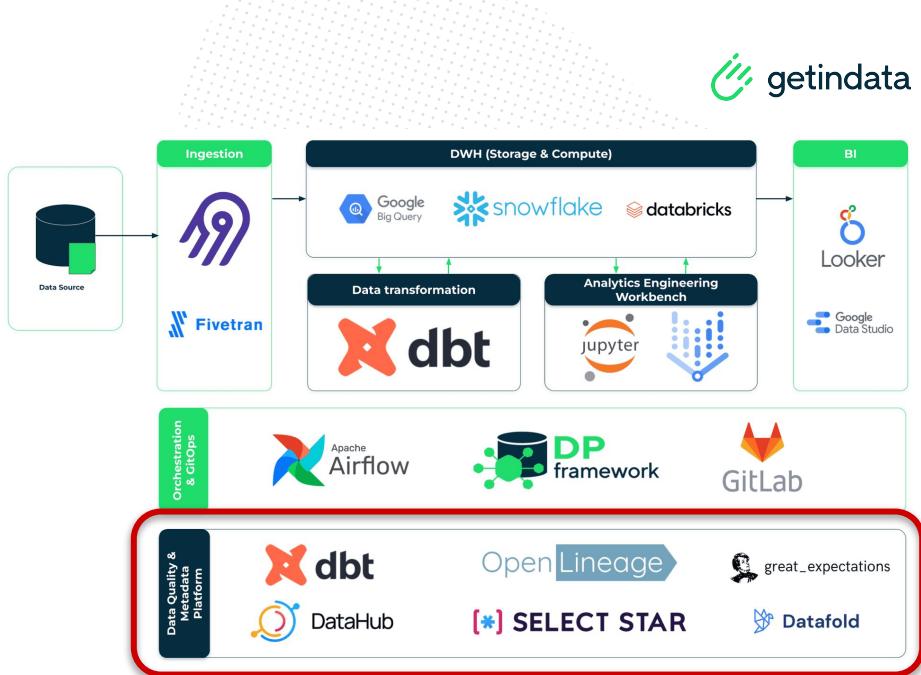


Key components of MDS

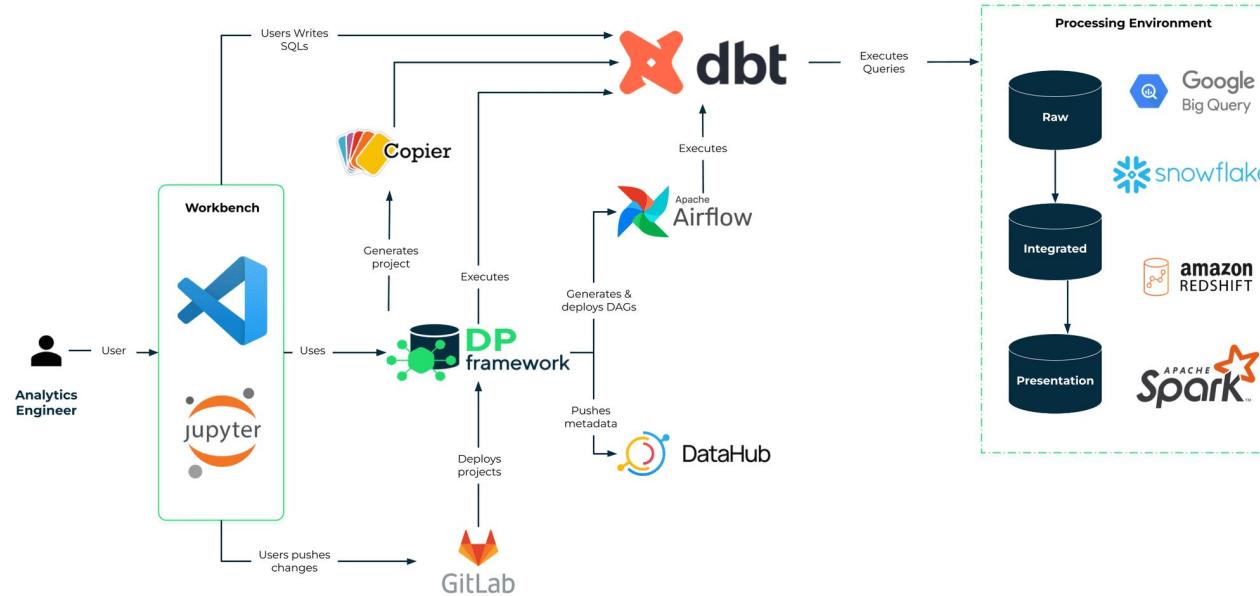


Data Discovery & Observability

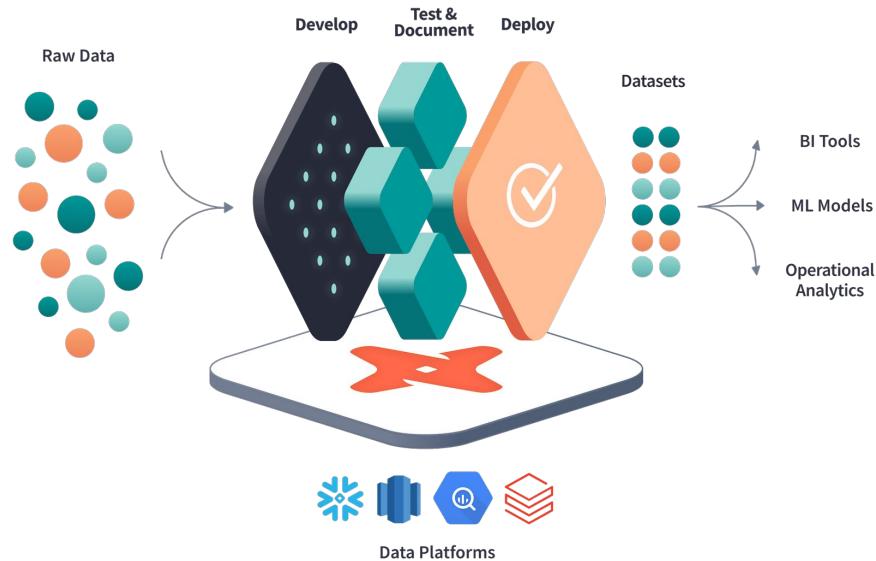
- Automation of data **quality tests**
- Data **monitoring & alerting**
- Data **search & lineage**
- Data **profiling**
- Business **glossaries, data tagging**
- Data **documentation**
- Support for best-in-class **open-source or proprietary** solutions



GetInData Modern Data Platform - WoW



Core concepts of dbt



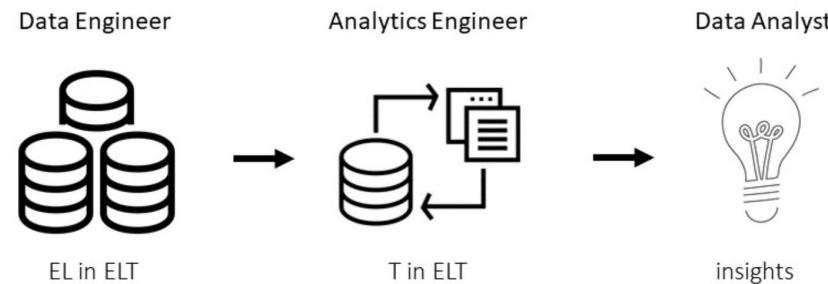
"dbt™ is a transformation workflow that lets teams quickly and collaboratively deploy analytics code following software engineering best practices like modularity, portability, CI/CD, and documentation. Now anyone who knows SQL can build production-grade data pipelines."

getdbt.com

Core concepts of dbt

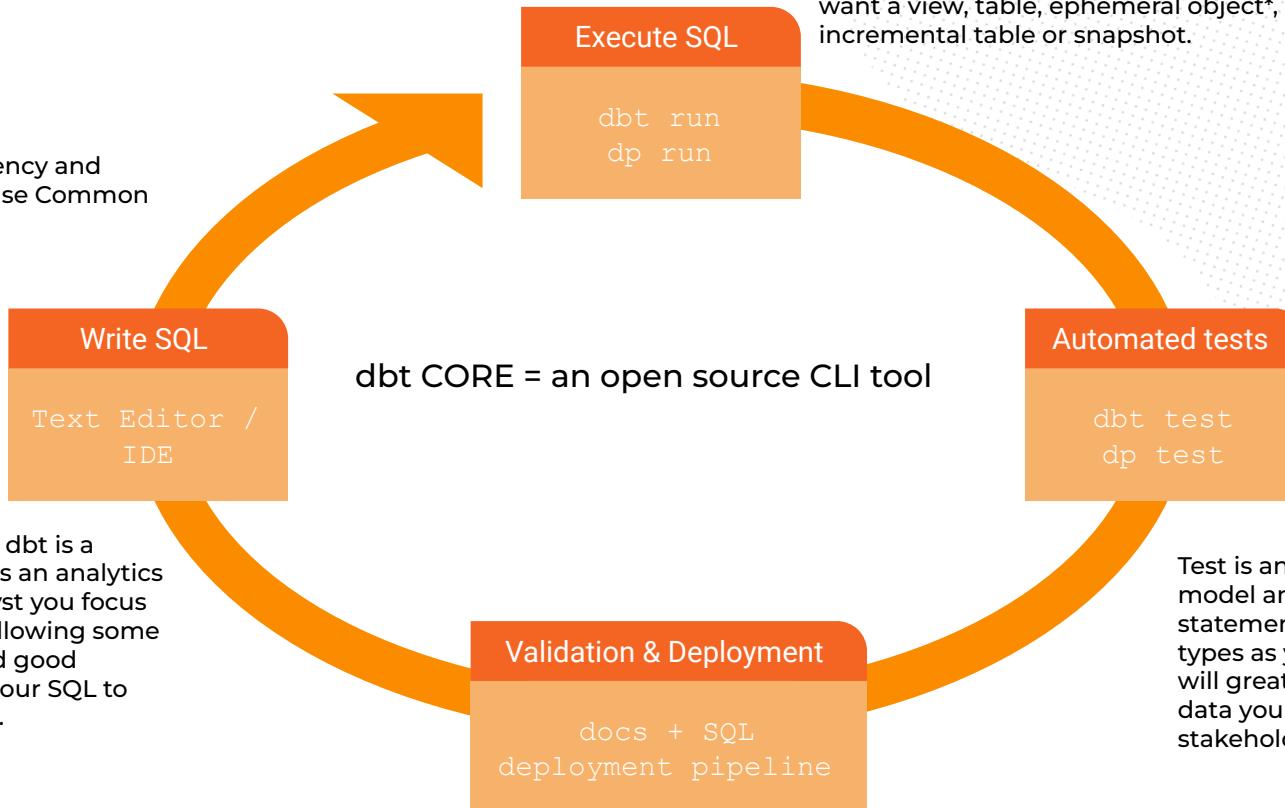
What problem does dbt solve?

- 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

For better transparency and future debugging use Common Table Expressions



Core concepts of dbt

DEMO:

- Create the workspace
- First steps into dbt



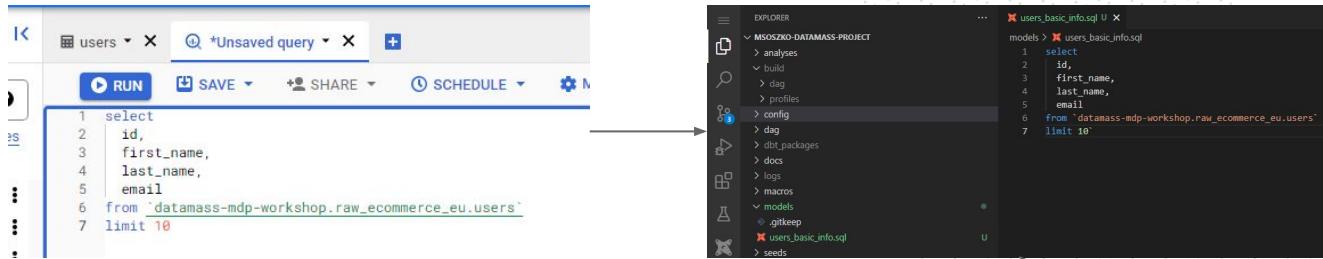
Core concepts of dbt



Hands-on exercises:

<https://github.com/getindata/gid-mdp-workshop>

DEMO



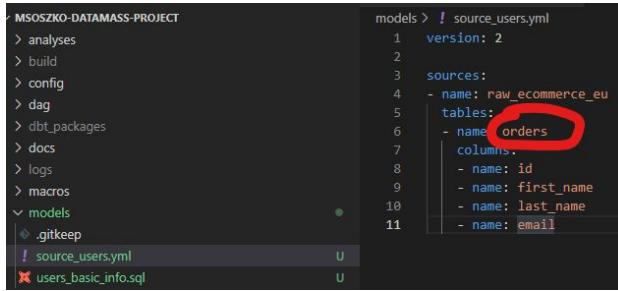
In vertex AI workbench we use **dp** and **dbt** commands. That is because **dp** works on top of **dbt** and integrates it with other components. It is recommended that at least the last run you perform is done with DP.

```
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#
```

```
models > users_basic_info.sql
  1 {{ config(materialized='table')
  2   }
  3   }
  4
  5 select
  6   id,
  7   first_name,
  8   last_name,
  9   email
 10  from `datamass-mdp-workshop.raw_ecommerce_eu.users`
```

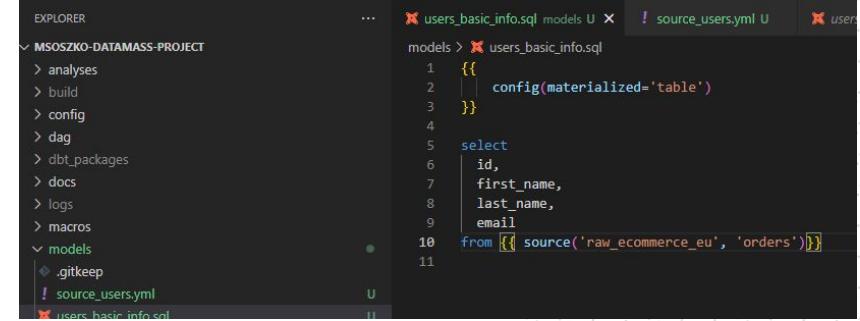
Option: inspect and compare compiled SQL in target / run



```

MSOSZKO-DATAMASS-PROJECT
  > analyses
  > build
  > config
  > dag
  > dbt_packages
  > docs
  > logs
  > macros
  > models
    > .gitkeep
    ! source_users.yml U
  ✘ users_basic_info.sql U
models > ! source_users.yml
  1 version: 2
  2
  3 sources:
  4   - name: raw_ecommerce_eu
  5     tables:
  6       - name: orders
  7         columns:
  8           - name: id
  9             - name: first_name
  10            - name: last_name
  11            - name: email

```



```

EXPLORER
MSOSZKO-DATAMASS-PROJECT
  > analyses
  > build
  > config
  > dag
  > dbt_packages
  > docs
  > logs
  > macros
  > models
    > .gitkeep
    ! source_users.yml U
  ✘ users_basic_info.sql U
models > ✘ users_basic_info.sql
  1 {{
  2   | config(materialized='table')
  3 }
  4
  5 select
  6   | id,
  7     first_name,
  8     last_name,
  9     email
 10    from {{ source('raw_ecommerce_eu', 'orders') }}
 11

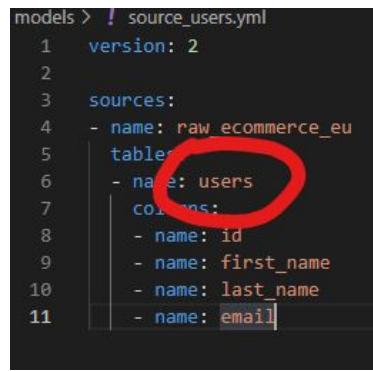
```

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.

```

22:33:58 Found 1 model, 0 tests, 0
22:33:58
22:34:00 Concurrency: 1 threads (target)
22:34:00
22:34:00 1 of 1 START table model msoszko_private_working_schema.users_basic_info
22:34:03 1 of 1 OK created table msoszko_private_working_schema.users_basic_info
22:34:03
22:34:03 Finished running 1 table msoszko_private_working_schema.users_basic_info
22:34:03
22:34:03 Completed successfully
22:34:03
22:34:03 Done. PASS=1 WARN=0 ERROR=0
(base) root@7e7478c36fa8:/home/jupyter/msoszko-datamass-project#

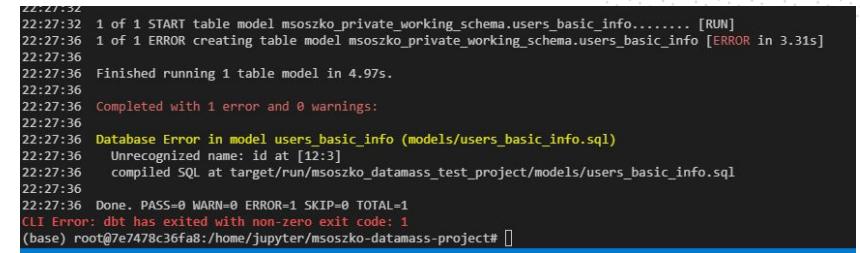
```



```

models > ! source_users.yml
  1 version: 2
  2
  3 sources:
  4   - name: raw_ecommerce_eu
  5     tables:
  6       - name: users
  7         columns:
  8           - name: id
  9             - name: first_name
 10               - name: last_name
 11               - name: email

```



```

22:27:32
22:27:32 1 of 1 START table model msoszko_private_working_schema.users_basic_info..... [RUN]
22:27:36 1 of 1 ERROR creating table model msoszko_private_working_schema.users_basic_info [ERROR in 3.31s]
22:27:36
22:27:36 Finished running 1 table model in 4.97s.
22:27:36
22:27:36 Completed with 1 error and 0 warnings:
22:27:36
22:27:36 Database Error in model users_basic_info (models/users_basic_info.sql)
22:27:36 Unrecognized name: id at [12:3]
22:27:36 compiled SQL at target/run/msoszko_datamass_test_project/models/users_basic_info.sql
22:27:36
22:27:36 Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
CLI Error: dbt has exited with non-zero exit code: 1
(base) root@7e7478c36fa8:/home/jupyter/msoszko-datamass-project# 

```

DEMO



```
! users_basic_info.sql models U ! users_basic_info.yml U ! source_users.yml U ✘ users_basic_info.sql .../msoszko_datamass_test_project  
models > ! users.basic_info.yml  
1 version: 2  
2  
3 models:  
4   - name: users_basic_info  
5     description: "This is a table with basic information about app users: first name, last name & email"  
6     columns:  
7       - name: id  
8         description: "This is users unique identification code"  
9       - name: first_name  
10      description: "This is users first name"  
11       - name: last_name  
12      description: "This is users last name"  
13       - name: email  
14      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.

```
RUN SAVE SHARE SCHEDULE MORE  
1 insert into `datamass-mdp-workshop.msoszko_private_working_schema.users_basic_info`  
2 values (99999999, "george", "duplicated", "george_duplicated@test_fail.com"),  
3 (99999999, "george", "duplicated", "george_duplicated@test_fail.com");  
  
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
```

```
! users_basic_info.sql U ! users.basic_info.yml U ! source_users.yml U ✘ users_basic_info.sql .../msoszko_datamass_test_project  
models > ! users.basic_info.yml  
1 version: 2  
2  
3 models:  
4   - name: users_basic_info  
5     description: "This is a table with basic information"  
6     columns:  
7       - name: id  
8         description: "This is users unique identification code"  
9         tests:  
10        - unique  
11        - not_null  
12        - max_length  
13        - min_length  
14        - max  
15        - min  
16       - name: first_name  
17      description: "This is users first name"  
18       - name: last_name  
19      description: "This is users last name"  
20       - name: email  
21      description: "This is users email address"
```

Field name	Type	Mode	Collation	Default Value	Policy Tags	Description
<input type="checkbox"/> id	INTEGER	NULLABLE				This is users unique identification code
<input type="checkbox"/> first_name	STRING	NULLABLE				This is users first name
<input type="checkbox"/> last_name	STRING	NULLABLE				This is users last name
<input type="checkbox"/> email	STRING	NULLABLE				This is users email address

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

1 {{ config(
2   materialized='table',
3   persist_docs={"relation":true, "columns": true}
4 )
5 }
6 }

7

8
9 with source as (
10   select * from {{ source('raw_ecommerce_eu', 'users') }}
11 ),
12 users as (
13
14   select
15     id          as user_id,
16     first_name  as user_first_name,
17     last_name   as user_last_name,
18     email       as user_email,
19     age         as user_age,
20     gender      as user_gender,
21     state       as user_address_state,
22     street_address as user_street_address,
23     postal_code  as user_address_postal_code,
24     city         as user_address_city,
25     country      as user_address_country,
26     latitude     as user_address_latitude,
27     longitude    as user_address_longitude,
28     traffic_source as user_traffic_source,
29     created_at   as user_account_created_at
30
31   from source
32
33
34   select * from users
35 }
```

```
user_basic_info.sql U ✘ ! user_basic_info.yml U
msoszko-datamass-project > models > user_basic_info.yml

1 version: 2
2
3 models:
4   - name: user_basic_info
5     description: "This is a table with basic information about app users: first name, last name & email"
6     columns:
7       - name: user_id
8         description: "This is users unique identification code"
9         tests:
10           - unique
11           - not_null
12       - name: user_first_name
13         description: "This is users first name"
14       - name: user_last_name
15         description: "This is users last name"
16       - name: user_email
17         description: "This is users email address"
```

Coffee break!

 getindata

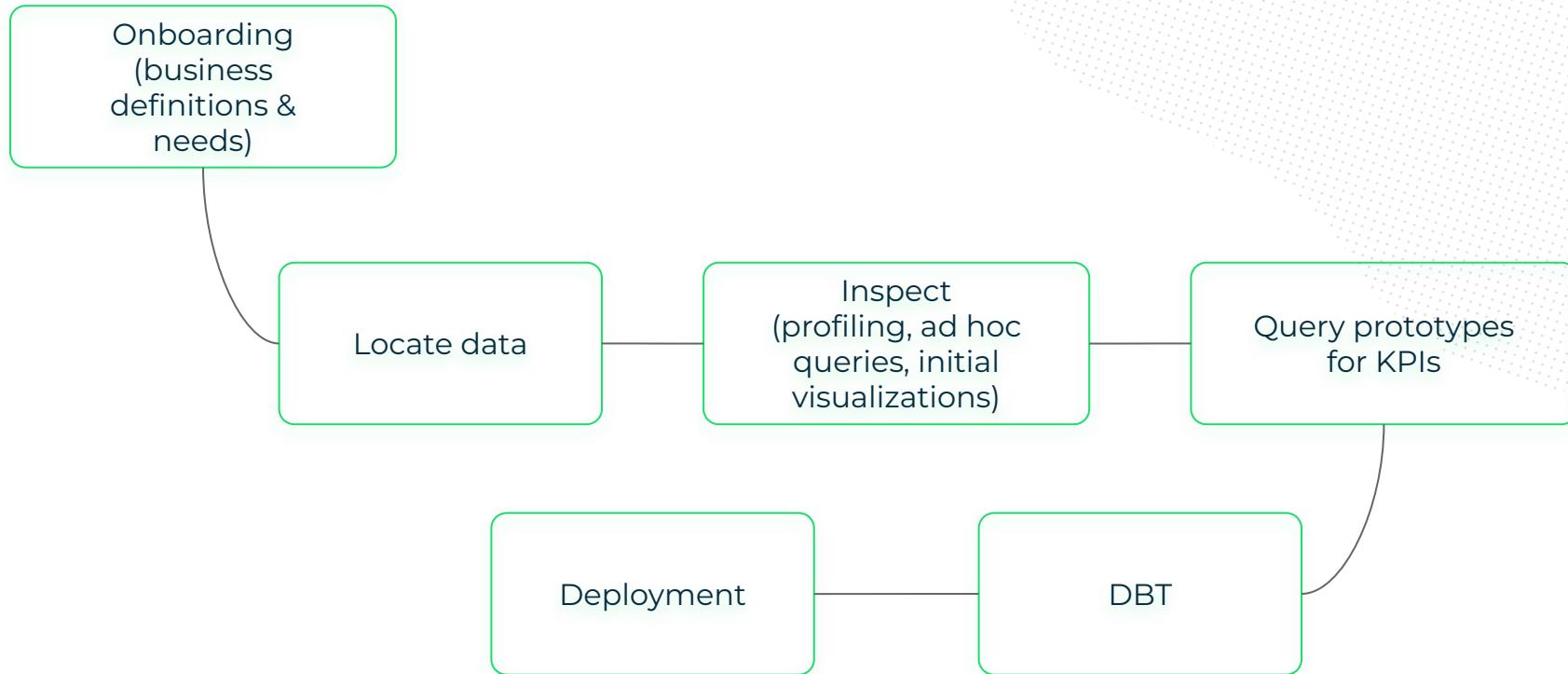


Session 2 - Simple end-to-end data pipeline

- 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



Discovery, exploration, profiling

- 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

products: cost

Products total count

29,120

Min. Cost

0.01

Max. Cost

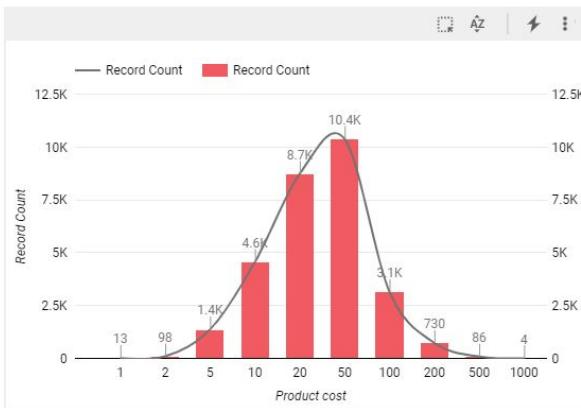
557.15

Avg. Cost

28.48

Std.Dev Cost

30.62

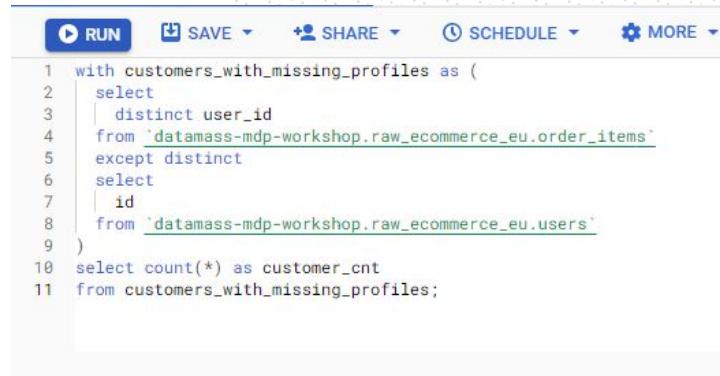


Field name	Type	Mode	Collation	Default Value	Policy Tags
<u>id</u>	INTEGER	NULLABLE			
<u>cost</u>	FLOAT	NULLABLE			
<u>category</u>	STRING	NULLABLE			
<u>name</u>	STRING	NULLABLE			
<u>brand</u>	STRING	NULLABLE			
<u>retail_price</u>	FLOAT	NULLABLE			
<u>department</u>	STRING	NULLABLE			
<u>sku</u>	STRING	NULLABLE			
<u>distribution_center_id</u>	INTEGER	NULLABLE			

- 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

- 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



```

1 with customers_with_missing_profiles as (
2   select
3     distinct user_id
4   from `datamass-mdp-workshop.raw_ecommerce_eu.order_items`
5   except distinct
6   select
7     id
8   from `datamass-mdp-workshop.raw_ecommerce_eu.users`
9 )
10 select count(*) as customer_cnt
11 from customers_with_missing_profiles;

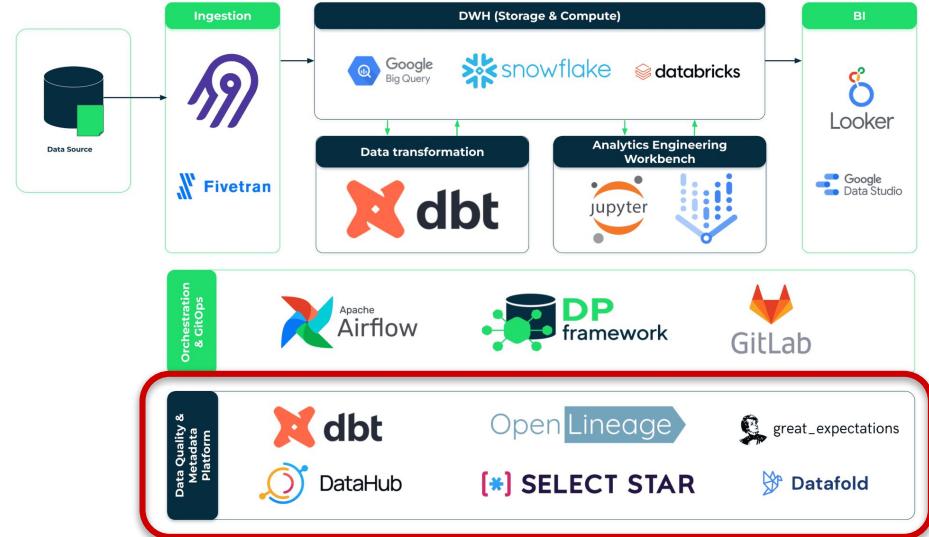
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	customer_cnt			
1	0			

Discovery, exploration, profiling

- 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

Dataset | BigQuery

raw_product_type

Source: product_type • It consists of product type number, product type and product name.

(dbt-product_type) (pipeline-example) (customer_retention_rate-project)

Table | BigQuery

raw_transactions

Source: fact transactions • Fact table. It contains information about transactions. • The primary key is t

(dbt) (pipeline-example) (customer_retention_rate-project)

raw_transactions

29 752 863 rows

Schema Documentation Properties Lineage Queries Stats Validation

Field	Description



Datasets

Datasets

dataops-demo-prod

Search Datasets, People, & more...

Try searching for

base_fact_transactions raw_customers raw_articles

Explore your Metadata

13

Platforms

dbt 109

BigQuery 15



Discovery, exploration, profiling

raw_transactions

29 752 863 rows

Schema	Documentation	Properties	Lineage	Queries	Stats	Validation
Edit + Add Link						

Source: fact transactions

Fact table. It contains information about transactions. The primary key is transaction_id consisting of distinguishing orders is order_id concatenating t_dat and customer_id, sales_channel_id.

raw_transactions

29 752 863 rows

Schema	Documentation	Properties	Lineage	Queries	Stats	Validation
Downstream Upstream						
Filters						

Filter

Degree of Dependencies

<input checked="" type="checkbox"/> 1	Dataset  BigQuery
<input type="checkbox"/> 2	base_fact_transactions
<input type="checkbox"/> 3+	 customer_retention_rate_project

raw_transactions

29 752 863 rows

Schema	Documentation	Properties	Lineage	Queries	Stats	Validation
--------	---------------	------------	---------	---------	-------	------------

Name	Value
------	-------

catalog_schema <https://schemas.getdbt.com/dbt/catalog/v1.json>

catalog_type table

raw_transactions

29 752 863 rows

Schema	Documentation	Properties	Lineage	Queries	Stats	Validation
--------	---------------	------------	---------	---------	-------	------------

Latest	Historical
------------------------	----------------------------

Table Stats

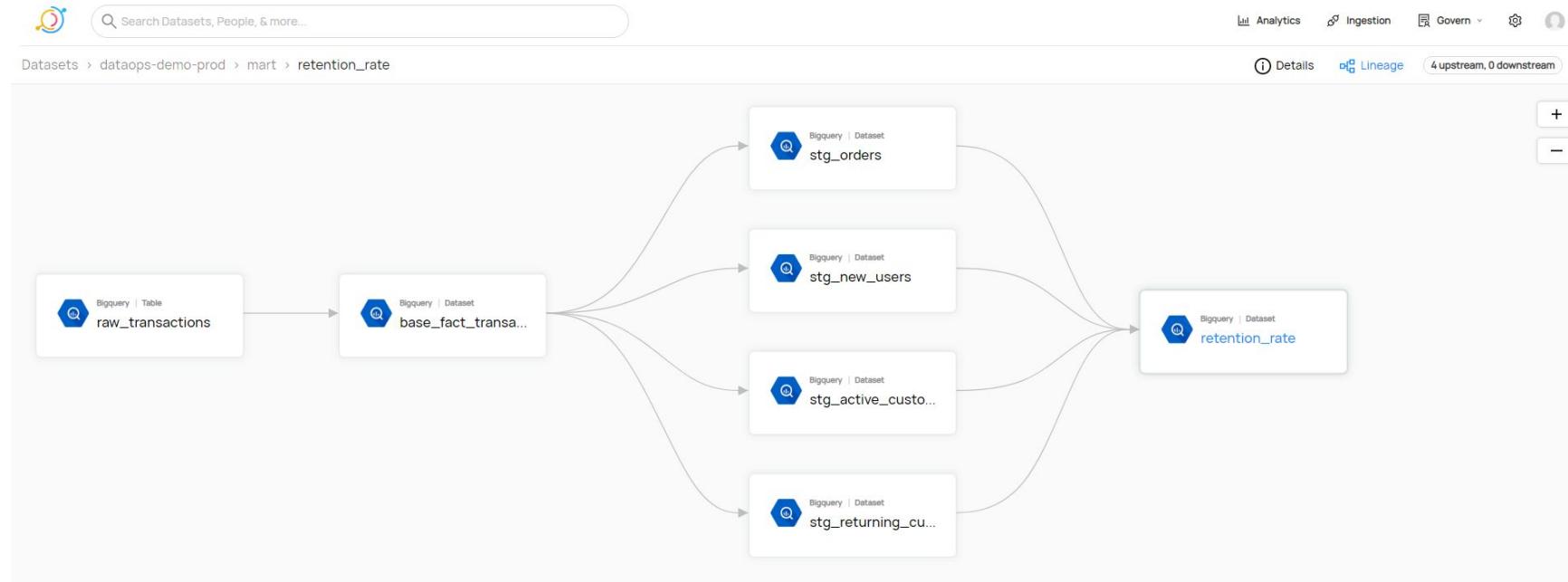
Rows

29,8M

Column Stats

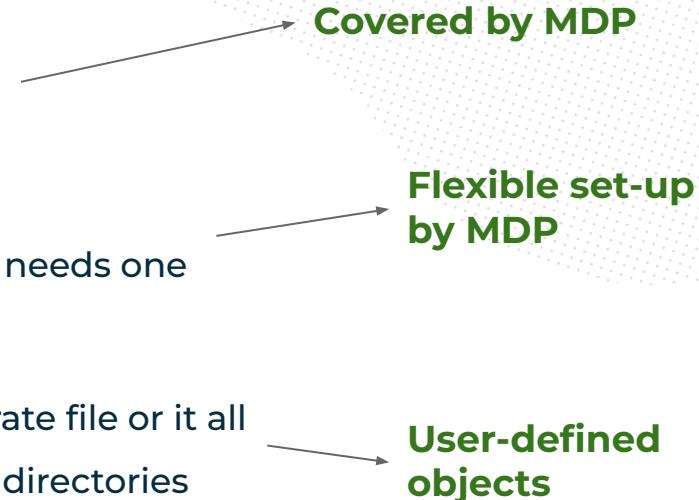
Name	Min	Max	Mean	Median	Null Count	Null %	Distinct Count	Distinct %
transaction_id	unknown	unknown	unknown	unknown	0	0.00%	29937627	100.00%

Discovery, exploration, profiling



Transforming data using SQL with dbt

Core dbt configuration

- Profiles.yml
 - configuring your environments (DEV, PROD etc)
 - Dbt_project.yml
 - Specify operations/metaconfig. Every dbt project needs 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**

Transforming data using SQL with 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

Transforming data using SQL with 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)

Transforming data using SQL with dbt

Naming convention

Raw

source_<dataset>_<table>
src_<dataset>_<table>
src_<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

Snapshots

snapshot_<dataset>_<table>
snapshot_<table>

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

DEMO:

- Simple dbt pipeline in dev

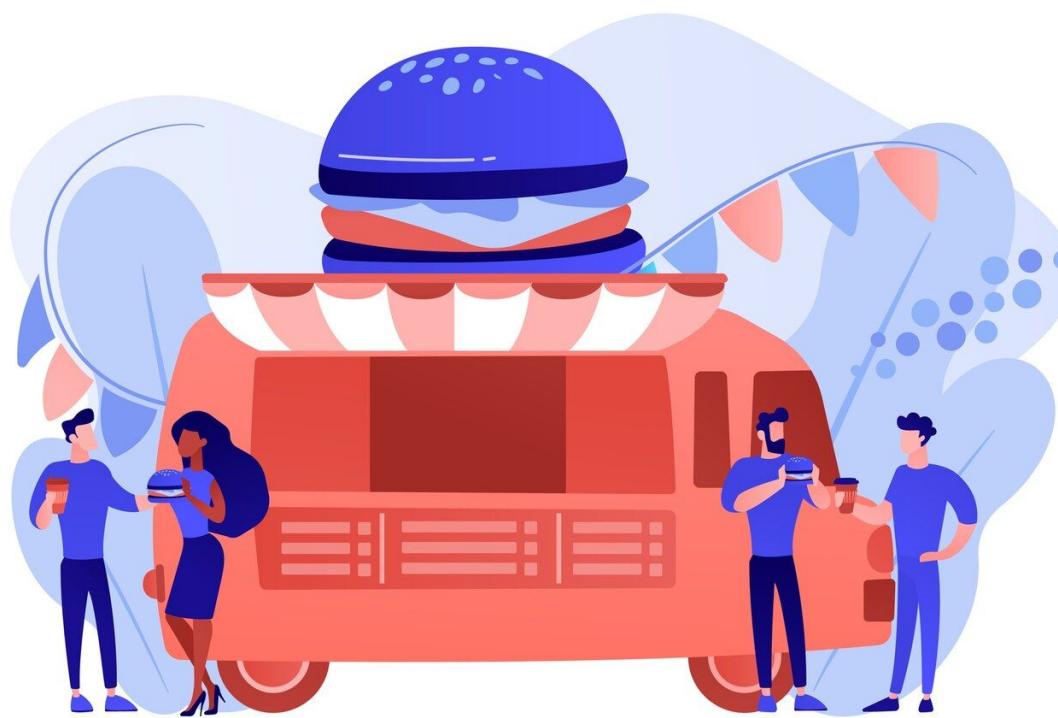


Hands-on exercises



Lunch!

 getindata



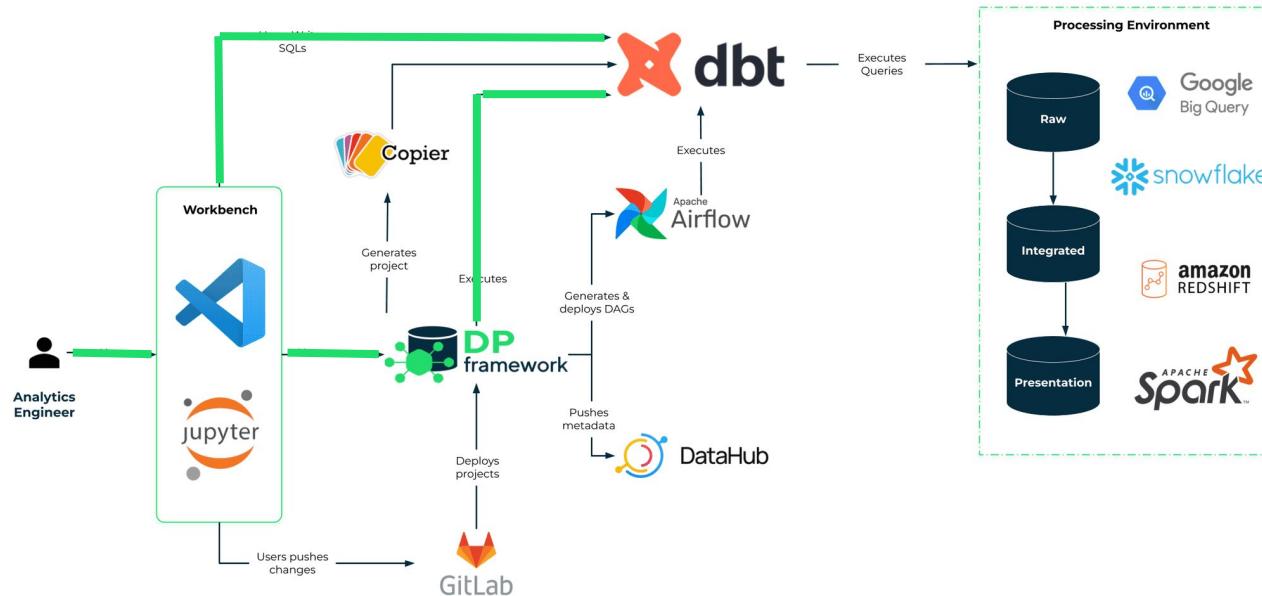
Session 3 - Data pipeline - scheduling, deployment & advanced features

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



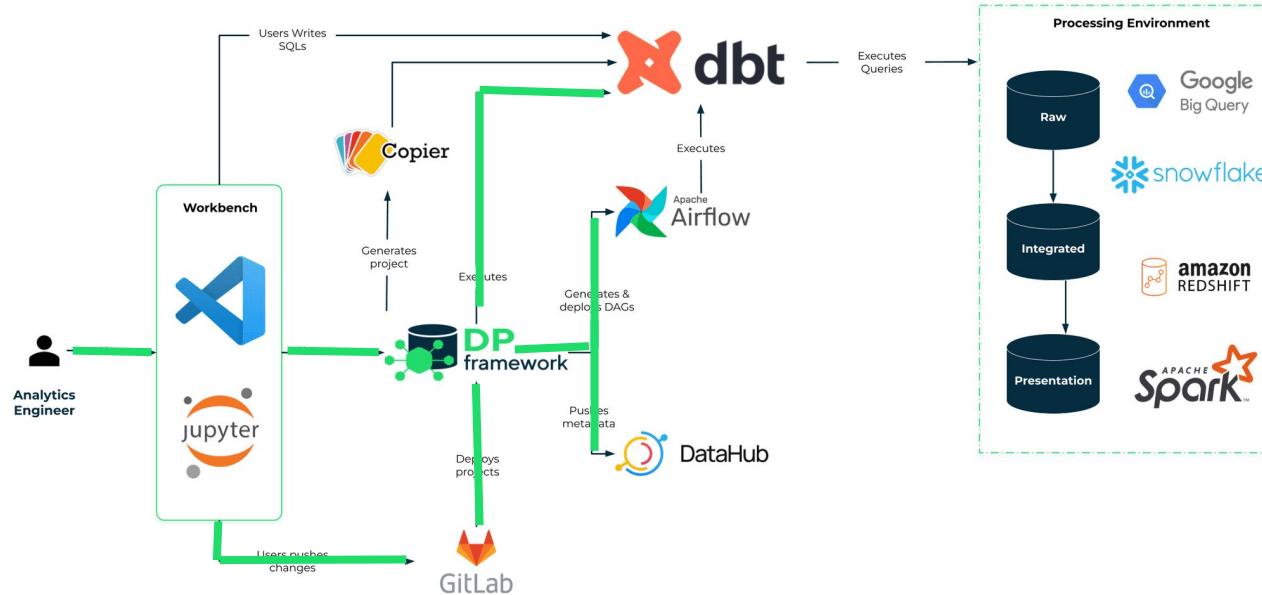
Session #3

Data pipeline - scheduling, deployment & advanced features



Session #3

Data pipeline - scheduling, deployment & advanced features



CICD: Gitlab phase

- **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

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

```
# 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

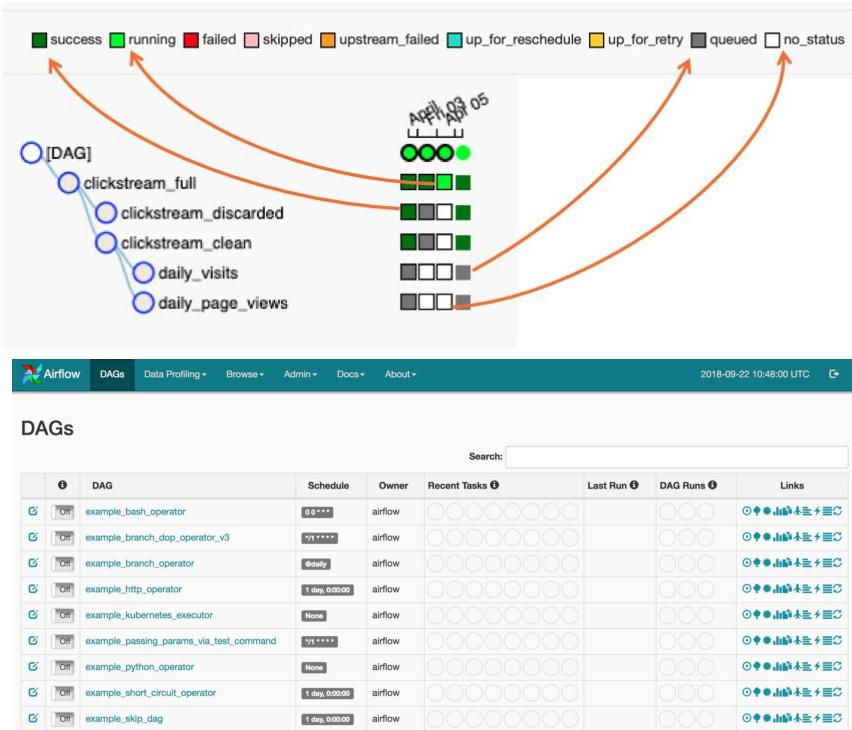
DEMO:

- Commit our work
- Watch CI-CD



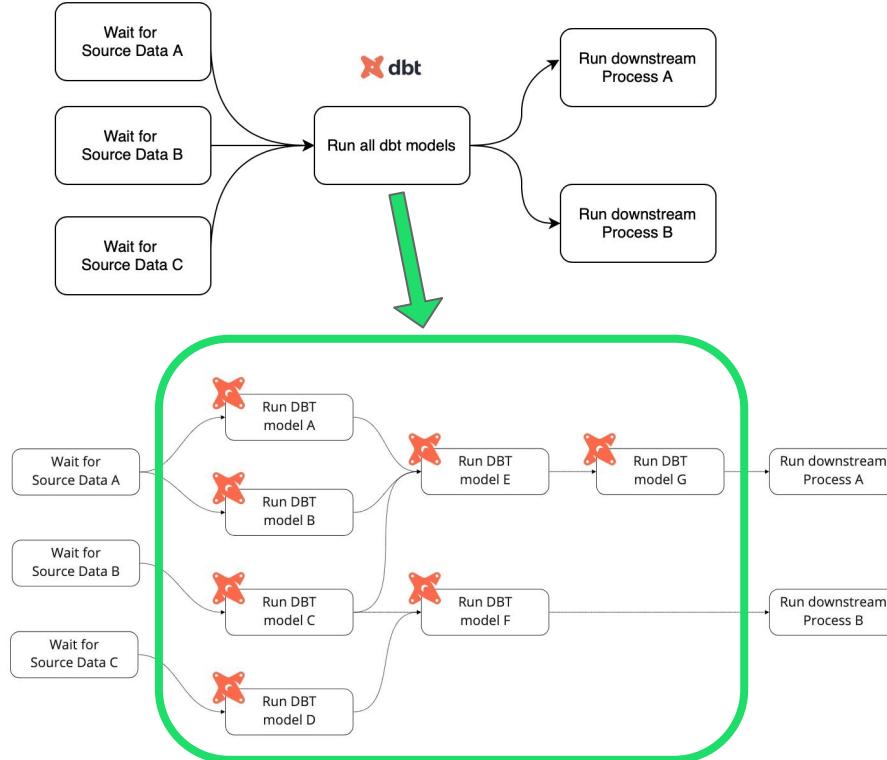
Apache Airflow as a workflow scheduler

- 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



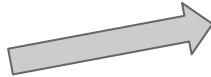
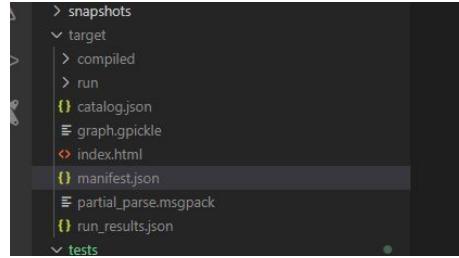
- dbt and Airflow integrations are very common
- run jobs on a cron schedule
- view logs
- configure error notifications (Slack)
- render your project's documentation
- you can orchestrate your dbt jobs to wait for data sources

Apache Airflow as a workflow scheduler



- 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



```

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()

```

```

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 * * *",
    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,
            )
        """
    )

```

```

manifest.json ×

target > manifest.json > {} nodes > {} model.datamass_project.int_events_traffic_sources_pivoted > raw_sql
1 {"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\Michał\U0142_Soszko\Projects\Workshops\msoszko-datamass-project", "path": "ISO_1
"seeds\ISO_like_Countries-Continents", "name": "ISO_like_Countries-Continents", "alias":
"c62d61682f55ec9b83356d00cf61bb2c24046d816c6c43fd17facae6861b61543", "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", persist_docs={"relation":true, "columns": true}\n    )\n    ('stg_ecommerce_events', })\n    ,\n    npivot_event_traffic_source_aggr_by_user as (\n        se
'Adwords' then 1 else 0 end) as adwords_traffic,\n        sum(case when event_traffic_sour
when event_traffic_source = 'Facebook' then 1 else 0 end) as facebook_traffic,\n        sum
organic_traffic,\n        sum(case when event_traffic_source = 'YouTube' then 1 else 0 end)
(event_traffic_source)\n        as total_traffic,\n        mi

```

Apache Airflow as a workflow scheduler

DEMO:

- Inspect DAG



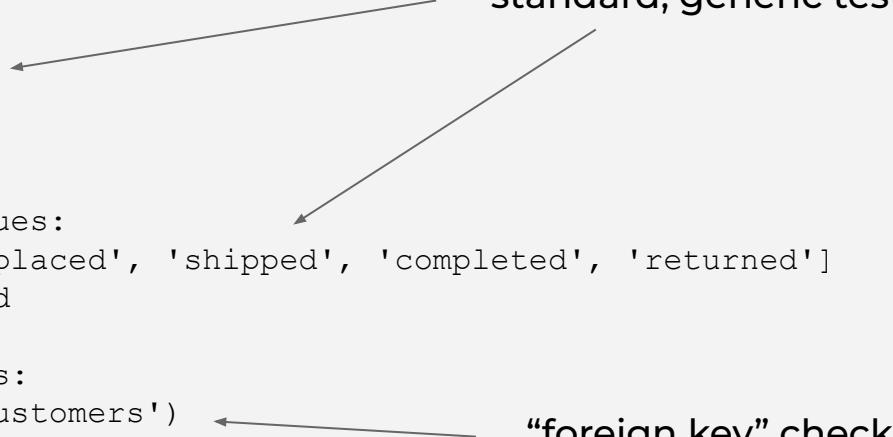
Data testing & data observability

- 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

```
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

dbt Packages storing dozen of generic tests and macros :

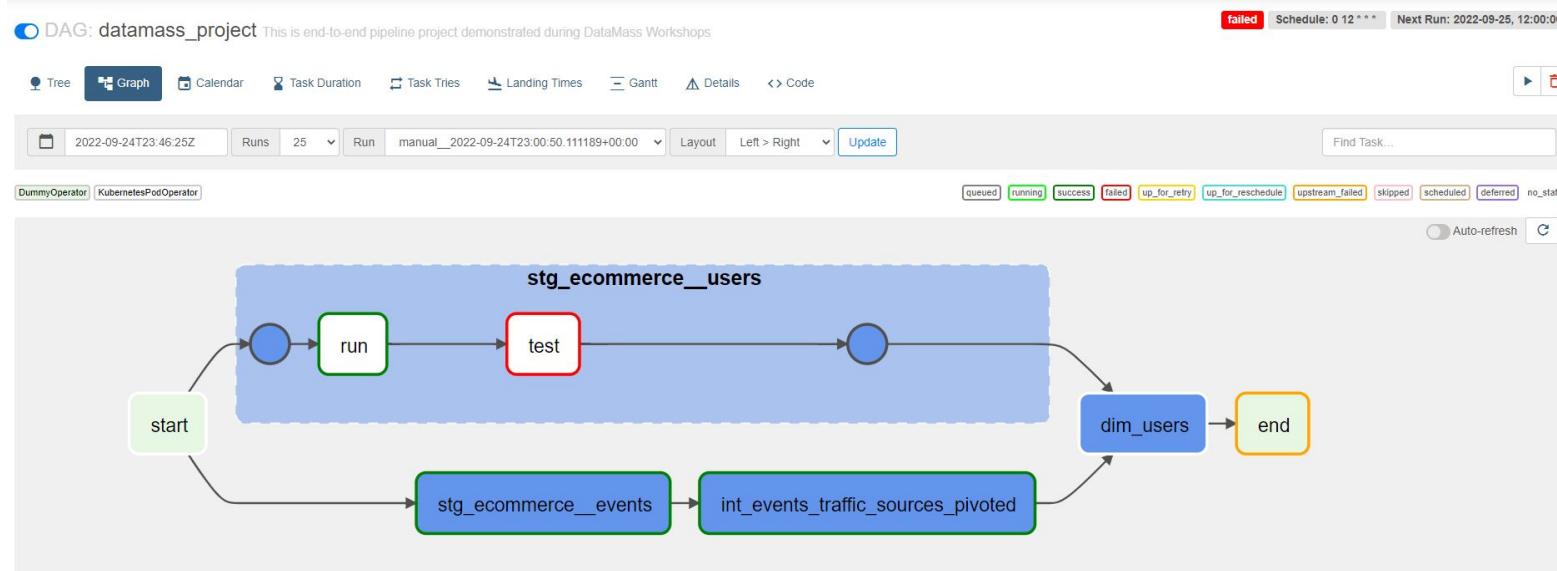
- dbt-labs/dbt_utils
- calogica/dbt_expectations
- If those do not provide tests you require, you can either write a macro / singular test of your own or... search for more packages -
<https://hub.getdbt.com/>

Data testing & data observability

DEMO:

- Custom tests in dbt

Data testing & data observability



Data testing & data observability

Search Datasets, People, & more...

Analytics Ingestion

Datasets > datamass-mdp-workshop > staging > stg_ecommerce__users

Table BigQuery > datamass-mdp-workshop > staging

stg_ecommerce__users X

100 000 rows

Schema Documentation Properties Lineage Queries Stats Validation

> Passed Column user_first_name values are not null X

> Passed Column user_last_name values are not null X

> Passed Column user_id values are not null X

> Passed Unique value proportion for column user_id is equal to 1 X

> Failed Column user_first_name values are not null X

> Failed Column user_last_name values are not null X

Data testing & data observability



The screenshot displays the DataMatters platform interface for a dataset named `stg_ecommerce__users`. The top navigation bar includes a search bar, Analytics, and Ingestion tabs. The main view shows the dataset's path: Datasets > datamass-mdp-workshop > staging > `stg_ecommerce__users`. Below this, there are tabs for Table, BigQuery, and the current dataset. The dataset details show 100,000 rows. A sidebar on the left provides navigation links for Slack Connect, Channels, and Apps, along with a Direct messages section and an Upgrade button. The central area displays a list of audit log entries for Data Pipelines Notifications, categorized by date: Sunday, September 18th, Monday, September 19th, and Wednesday, September 21st. Each entry details a failed task, its DAG, execution time, and log URL. The bottom of the screen shows a message from the `#dp-quality-alerting-dev` channel.

Exploring transformed data with Data Studio

Orders & Events Summary Report Prototype

This report contains basic summary for revenue, number of orders, customer lifetime value and web traffic. Generated traffic stands for total count of web events per user_id. You can click on country name to view details.

Status: In review

Country	Total Revenue	Average Revenue	Total orders
China	\$922,539.15	\$33.77	42,640
US	\$588,779.98	\$32.47	28,381
Brazil	\$393,923.75	\$34.15	18,153
Korea, South	\$146,658.56	\$34.11	6,755
France	\$137,993.87	\$36.93	5,884
United Kingdom	\$124,401.54	\$34.1	5,734
Germany	\$114,905.36	\$34.13	5,215
Spain	\$108,906.91	\$33.83	5,009
Japan	\$65,874.57	\$32.72	3,147
Australia	\$59,670.13	\$33.65	2,779

1 - 10 / 14 < >

Average Purchase Value
\$21.56

Average Purchase Frequency Rate
1.26

Customer Value
\$27.07

Average Customer Lifespan
0.22

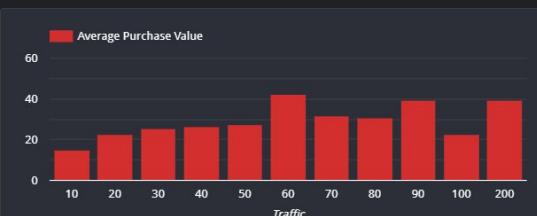
Customer Lifetime Value
\$5.98



Total number of orders

Count of users

Traffic



Average Purchase Value

Traffic

[Link](#)

Hands-on exercises



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



Thank you!

Contact us!

<https://getindata.com/estimate>

Our blog

<https://getindata.com/blog>

Our podcast

[Radio DaTa](#)

