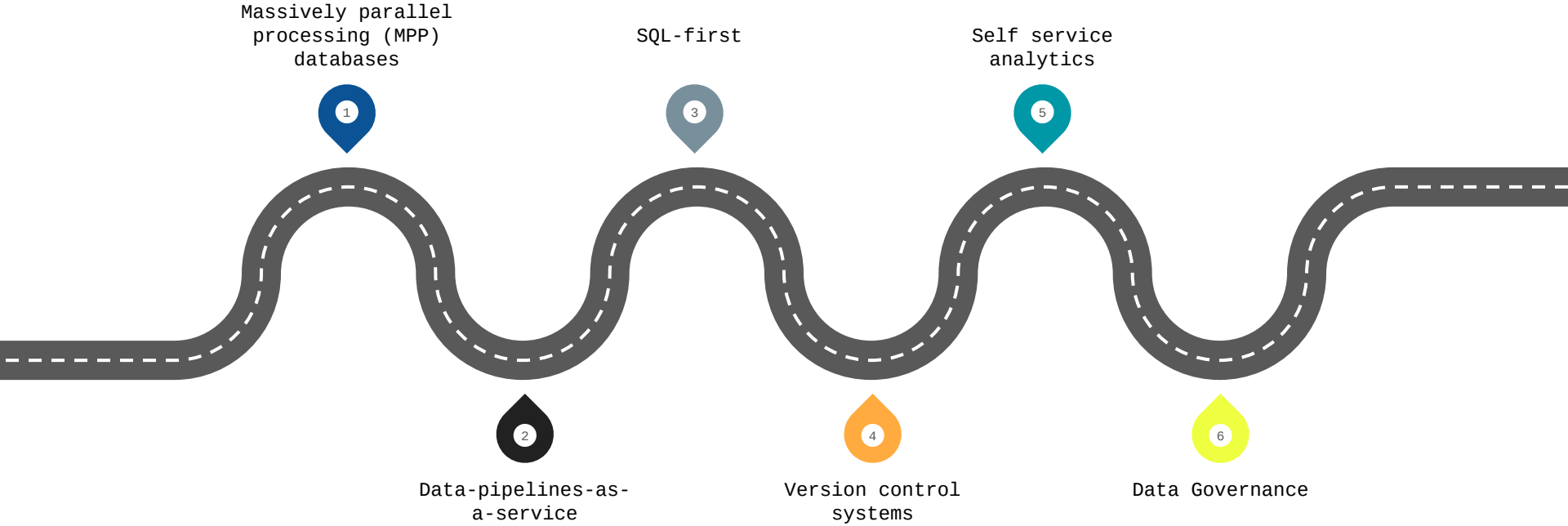
Data Engineering Zoomcamp

Analytics Engineering

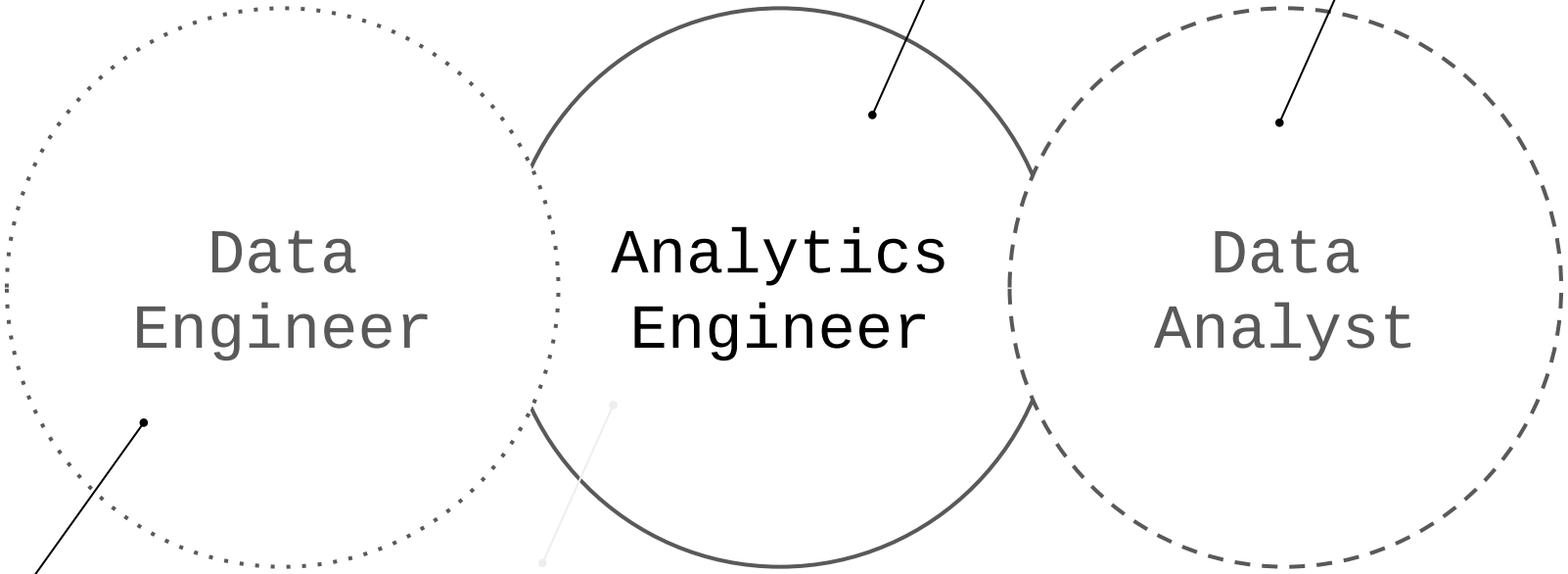
1

What is Analytics Engineering?

DATA DOMAIN DEVELOPMENTS



ROLES IN A DATA TEAM

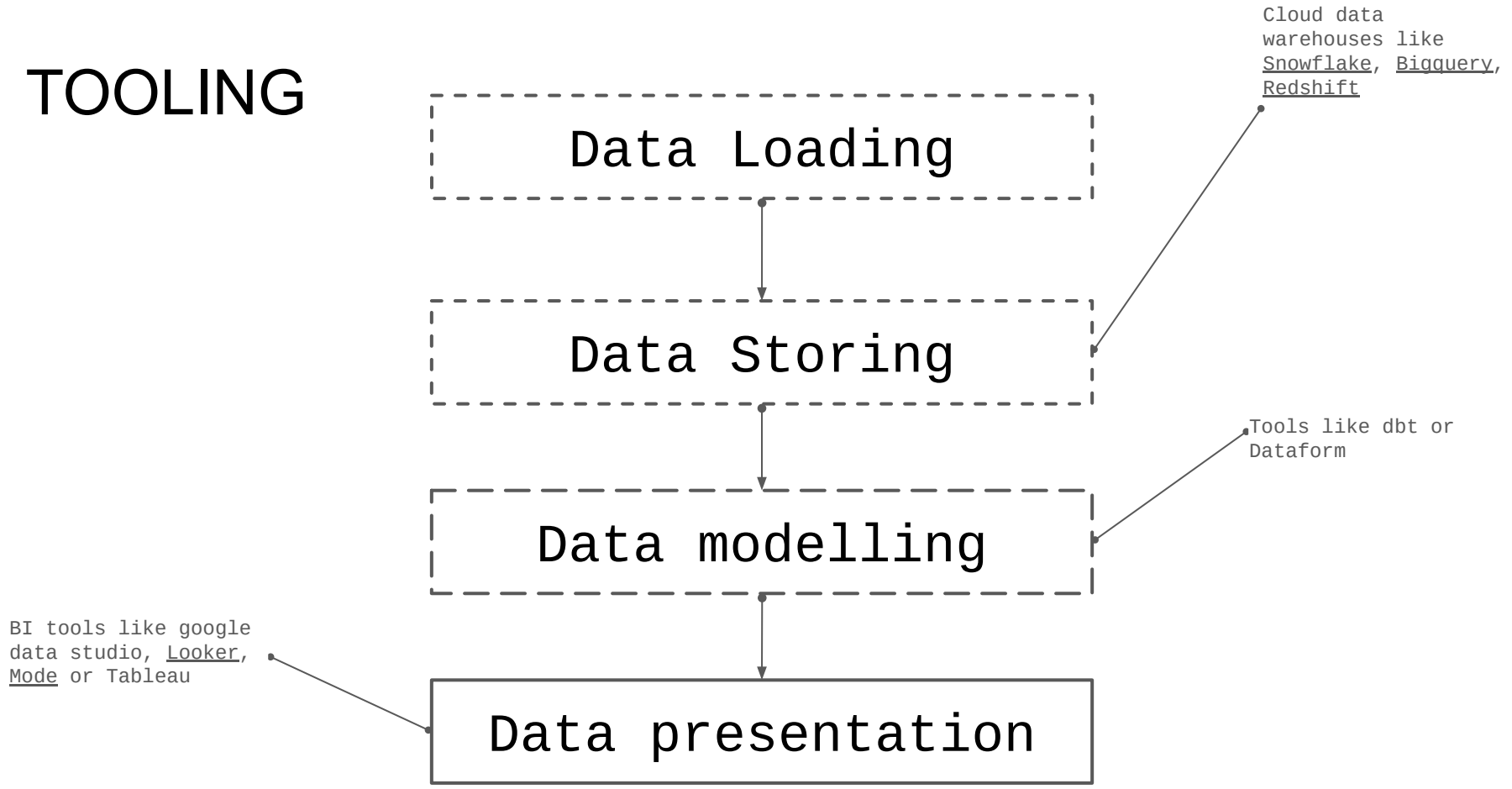


Introduces the good software engineering practices to the efforts of data analysts and data scientists

Uses data to answer questions and solve problems.

Prepares and maintain the infrastructure the data team needs.

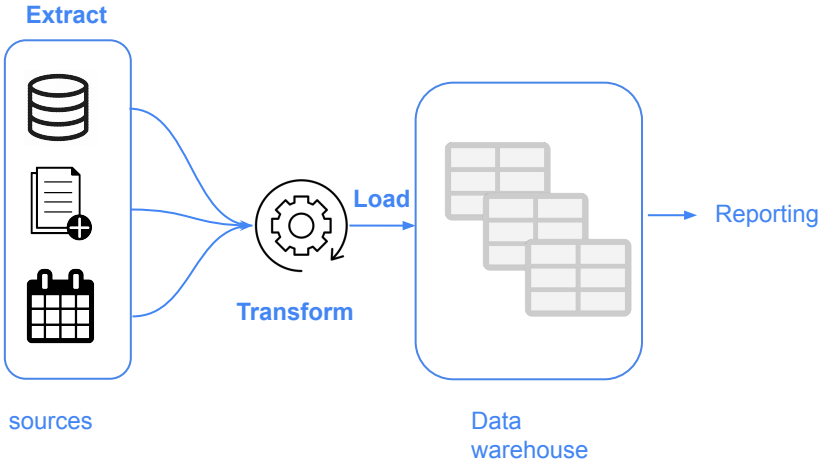
TOOLING

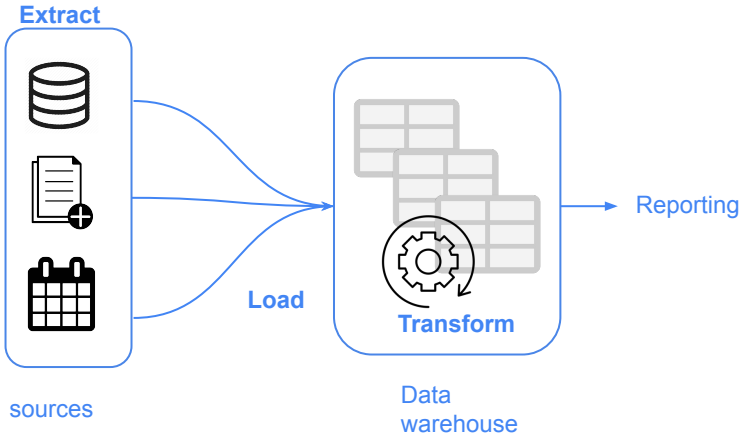

Data Loading
Data Storing
Data modelling
Data presentation
Cloud data warehouses like Snowflake, Bigquery, Redshift
Tools like dbt or Dataform
BI tools like google data studio, Looker, Mode or Tableau

2

Data Modelling concepts

ETL vs ELT





- Slightly more stable and compliant data analysis
- Higher storage and compute costs

- Faster and more flexible data analysis.
- Lower cost and lower maintenance

Kimball’s Dimensional Modeling

Objective

- Deliver data understandable to the business users
- Deliver fast query performance

Approach

Prioritise user understandability and query performance over non redundant data (3NF)

Other approaches

- Bill Inmon
- Data vault

Elements of Dimensional Modeling

Facts tables

- Measurements, metrics or facts
- Corresponds to a business *process*
- “verbs”

Dimensions tables

- Corresponds to a business *entity*
- Provides context to a business process
- “nouns”

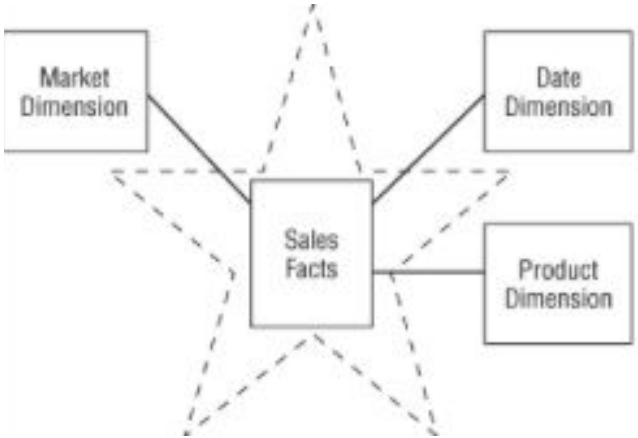


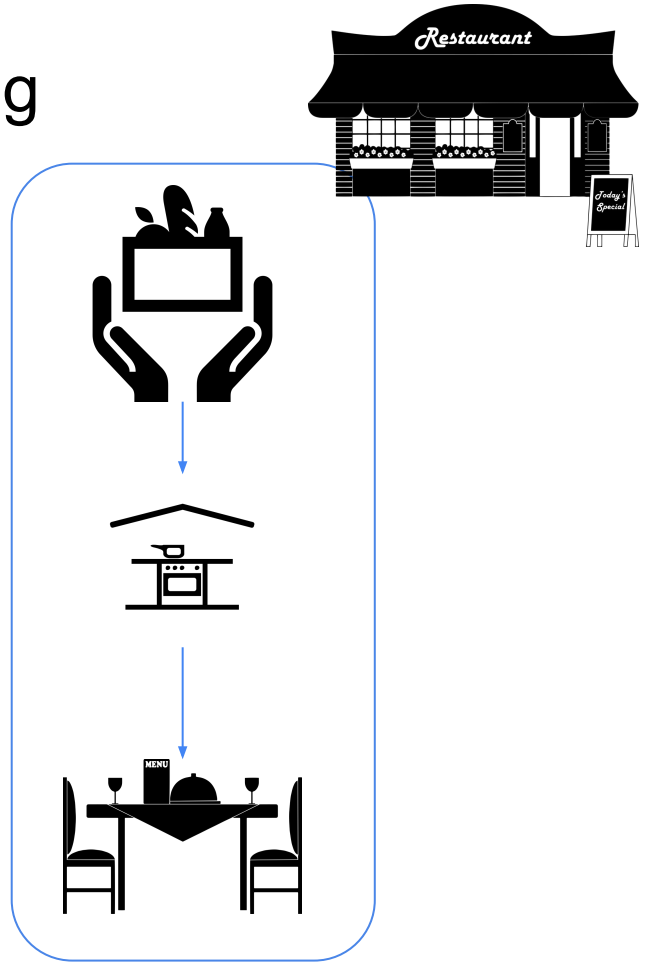
Image: The data warehouse toolkit

Architecture of Dimensional Modeling

Stage Area
- Contains the raw data
- Not meant to be exposed to everyone

Processing area
- From raw data to data models
- Focuses in efficiency
- Ensuring standards

Presentation area
- Final presentation of the data
- Exposure to business stakeholder

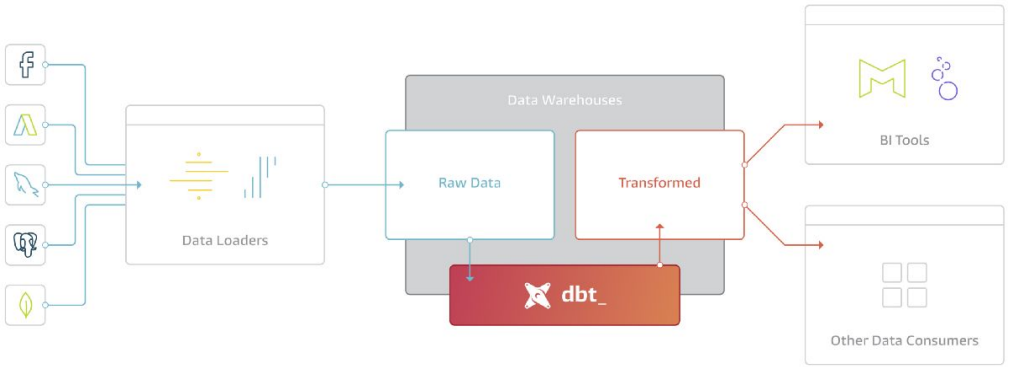


3

What is dbt?

What is dbt?

dbt is a transformation tool that allows anyone that knows SQL to deploy analytics code following software engineering best practices like modularity, portability, CI/CD, and documentation.



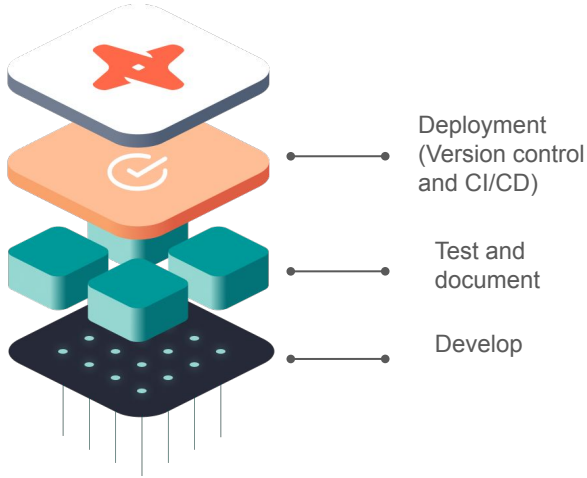
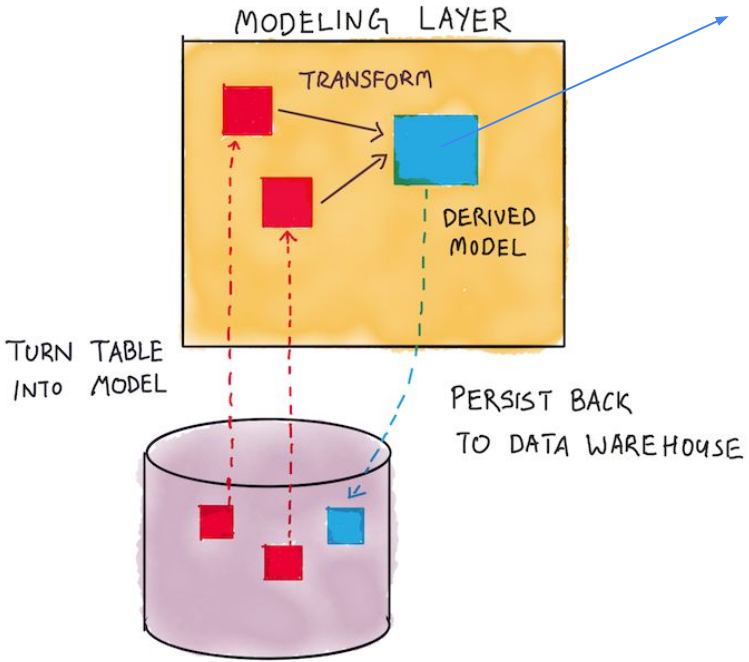


Image: getdbt.com

How does dbt work?



Each model is:

- A *.sql file
- Select statement, no DDL or DML
- A file that dbt will compile and run in our DWH

Image: holistics.io and getdbt.com

How to use dbt?

dbt Core
Open-source project that allows the data transformation
● Builds and runs a dbt project (.sql and .yml files)
● Includes SQL compilation logic, macros and database adapters
● Includes a CLI interface to run dbt commands locally
● Opens source and free to use

dbt Cloud
SaaS application to develop and manage dbt projects.
● Web-based IDE to develop, run and test a dbt project
● Jobs orchestration
● Logging and Alerting
● Integrated documentation
● Free for individuals (one developer seat)

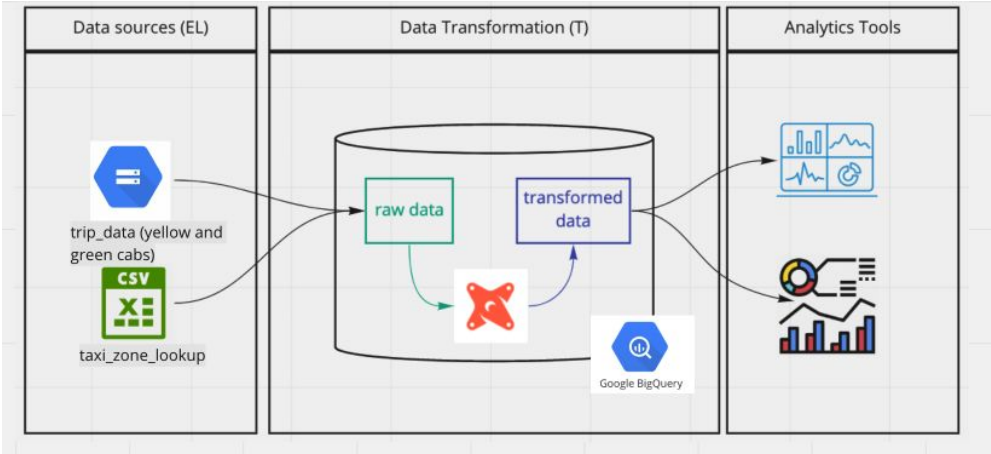
How are we going to use dbt?

BigQuery

- Development using cloud IDE
- No local installation of dbt core

Postgres

- Development using a local IDE of your choice.
- Local installation of dbt core connecting to the Postgres database
- Running dbt models through the CLI



4

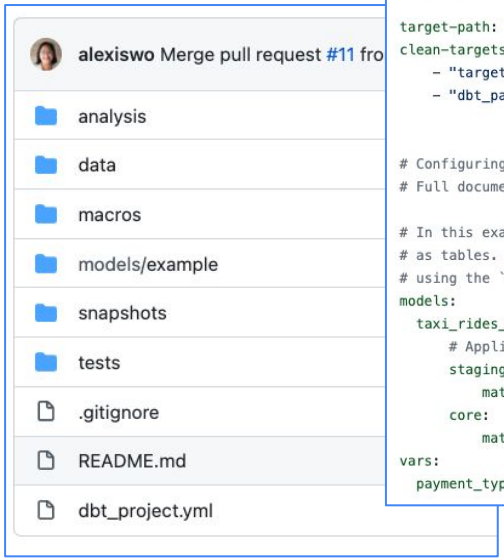
Starting a dbt project

Create a new dbt project

dbt provides an [starter project] with all the basic folders and files.
There are essentially two ways to use it:

With the CLI
After having installed dbt locally and setup the *profiles.yml*, run `dbt init` in the path we want to start the project to clone the starter project.

With dbt cloud
After having set up the dbt cloud credentials (repo and dwh) we can start the project from the web-based IDE



Starter project structure

```yaml
name: 'taxi_rides_ny'
version: '1.0.0'
config-version: 2

# This setting configures which "profile" dbt uses for this project.
profile: 'pg-dbt-workshop'

# These configurations specify where dbt should look for different types of files.
# The `source-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"]
analysis-paths: ["analysis"]
test-paths: ["tests"]
seed-paths: ["data"]
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"


# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring-models

# In this example config, we tell dbt to build all models in the example/ directory
# as tables. These settings can be overridden in the individual model files
# using the `{{ config(...) }}` macro.
models:
  taxi_rides_ny:
      # Applies to all files under models/.../
      staging:
          materialized: view
      core:
          materialized: table
vars:
  payment_type_values: [1, 2, 3, 4, 5, 6]
```

dbt_proyect.yml

Extra documentation link

5
Development of dbt models

Anatomy of a dbt model

dbt model

my_model.sql U

data-engineering-zoomcamp > week_5_analytics_enginee

```
1  {{
2      config(materialized='table' )
3  }}
4
5  Select *
6  from staging.source_table
7  where record_state = 'ACTIVE'
8
9
```

Compiled code

```
create table my_schema.my_model as (
    Select *
    from staging.source_table
    where record_state = 'ACTIVE'
)
```

Runs compiled code in the data warehouse

Several materialization strategies

- Table
- View
- Incremental
- Ephemeral

The FROM clause of a dbt model

taxi_zone_lookup.csv

data-engineering-zoomcamp > week_4_analytics_engineering > ta

You, a month ago | 1 author (You)

```
1  "locationid","borough","zone","service_zone"
```

```
select
    locationid,
    borough,
    zone,
    replace(service_zone,'Boro','Green') as service_zone
from {{ ref('taxi_zone_lookup') }}
```

Sources

- The data loaded to our dwh that we use as sources for our models
- Configuration defined in the yml files in the models folder
- Used with the source macro that will resolve the name to the right schema, plus build the dependencies automatically
- Source freshness can be defined and tested

```
sources:
    - name: staging
      database: production
      schema: trip_data_all

      loaded_at_field: record_loaded_at
      tables:
        - name: green_tripdata
        - name: yellow_tripdata
          freshness:
            error_after: {count: 6, period: hour}
```

```
    congestion_surcharge::double precision
from {{ source('staging','yellow_tripdata_2021_01') }}
where vendorid is not null
```

Seeds

- CSV files stored in our repository under the seed folder
- Benefits of version controlling
- Equivalent to a copy command
- Recommended for data that doesn't change frequently
- Runs with `dbt seed -s file_name`

The FROM clause of a dbt model

Ref

- Macro to reference the underlying tables and views that were building the data warehouse
- Run the same code in any environment, it will resolve the correct schema for you
- Dependencies are built automatically

dbt model

```
with green_data as (
    select *,
        'Green' as service_type
    from {{ ref('stg_green_tripdata') }}
),
```

```
with green_data as (
    select *,
        'Green' as service_type
    from "production"."dbt_victoria_mola"."stg_green_tripdata"
),
```

Compiled code

Macros

- Use control structures (e.g. if statements and for loops) in SQL
- Use environment variables in your dbt project for production deployments
- Operate on the results of one query to generate another query
- Abstract snippets of SQL into reusable macros — these are analogous to functions in most programming languages.

```
{#
    This macro returns the description of the payment_type
#}

{% macro get_payment_type_description(payment_type) -%}

    case {{ payment_type }}
        when 1 then 'Credit card'
        when 2 then 'Cash'
        when 3 then 'No charge'
        when 4 then 'Dispute'
        when 5 then 'Unknown'
        when 6 then 'Voided trip'
    end

{%- endmacro %}
```

Definition of the macro

```
select
    {{ get_payment_type_description('payment_type') }} as payment_type_description,
    congestion_surcharge::double precision
from {{ source('staging','green_tripdata_2021_01') }}
where vendorid is not null
```

Usage of the macro

```
create or alter view production.dbt_victoria_mola.stg_green_tripdata as
select
    case payment_type
        when 1 then 'Credit card'
        when 2 then 'Cash'
        when 3 then 'No charge'
        when 4 then 'Dispute'
        when 5 then 'Unknown'
        when 6 then 'Voided trip'
    end as payment_type_description,
    congestion_surcharge::double precision
from "production"."staging"."green_tripdata_2021_01"
where vendorid is not null
```

Compiled code of the macro

Packages

- Like libraries in other programming languages
- Standalone dbt projects, with models and macros that tackle a specific problem area.
- By adding a package to your project, the package's models and macros will become part of your own project.
- Imported in the **packages.yml** file and imported by running `dbt deps`
- A list of useful packages can be find in [dbt package hub]

schema.yml U | ! packages.yml ×

lata-engineering-zoomcamp-main > week_4_an

You, a month ago | 1 author (You)

```
1  packages:        You, 2 months ago
2    - package: dbt-labs/dbt_utils
3      version: 0.8.0
```

Specifications of the packages to import in the project

schema.yml U | stg_green_tripdata.sql ×

a-engineering-zoomcamp-main > week_4_analytics_engineering > taxi_rides_ny > models > staging >

You, 11 hours ago | 1 author (You)

```
{{ config(materialized='view') }}


select
    -- identifiers
6   {{ dbt_utils.surrogate_key(['vendorid', 'lpep_pickup_datetime']) }} as tripid,
7   cast(vendorid as integer) as vendorid,
8   cast(ratecodeid as integer) as ratecodeid,
```

Usage of a macro from a package

Variables

- Variables are useful for defining values that should be used across the project
- With a macro, dbt allows us to provide data to models for compilation
- To use a variable we use the `{{ var('...') }}` function
- Variables can defined in two ways:
  - In the dbt_project.yml file
  - On the command line

```
-- dbt build --m <model.sql> --var 'is_test_run: false'
{% if var('is_test_run', default=true) %}

  limit 100

{% endif %}
```

Variable whose value we can change via CLI

```
vars:
  payment_type_values: [1, 2, 3, 4, 5, 6]
```

Global variable we define under project.yml

6

Testing and documenting dbt models

Tests

- Assumptions that we make about our data
- Tests in dbt are essentially a `select` sql query
- These assumptions get compiled to sql that returns the amount of failing records
- Test are defined on a column in the .yml file
- dbt provides basic tests to check if the column values are:
  - Unique
  - Not null
  - Accepted values
  - A foreign key to another table
- You can create your custom tests as queries

```yaml
- name: payment_type_description
  description: Description of the payment_type code
  tests:
    - accepted_values:
        values: [1,2,3,4,5]
        severity: warn
- name: Pickup_locationid
  description: locationid where the meter was engaged.
  tests:
    - relationships:
        to: ref('taxi_zone_lookup')
        field: locationid
        severity: wa
columns:
    - name: tripid
      description: Primary key for
      tests:
          - unique:
              severity: warn
          - not_null:
              severity: warn
```

Definition of basic tests in the .yml files

```sql
select *
from "production"."dbt_victoria_mola"."stg_yellow_tripdata"
where tripid is null
```

Compiled code of the not_null test

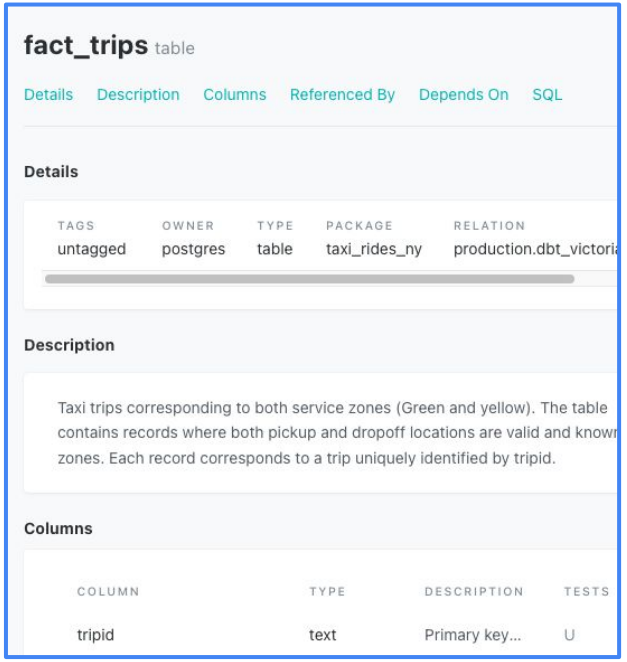
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

15:56:48
15:56:48  Finished running 2 view models, 1 seed, 12 tests, 3 table models in 2.53s.
15:56:48
15:56:48  Completed with 3 warnings:
15:56:48
15:56:48  Warning in test accepted_values_stg_green_tripdata_payment_type_description___var_paymen
15:56:48    Got 3 results, configured to warn if != 0
15:56:48
15:56:48    compiled SQL at target/compiled/taxi_rides_ny/models/staging/schema.yml/accepted_value
15:56:48
```

Warnings in the CLI from running dbt test

Documentation

- dbt provides a way to generate documentation for your dbt project and render it as a website.
- The documentation for your project includes:
  - **Information about your project**:
    - Model code (both from the .sql file and compiled)
    - Model dependencies
    - Sources
    - Auto generated DAG from the ref and source macros
    - Descriptions (from .yml file) and tests
  - **Information about your data warehouse (information_schema):**
    - Column names and data types
    - Table stats like size and rows
- dbt docs can also be hosted in dbt cloud




```
models:
  - name: dim_zones
    description: >
      List of unique zones idefied by locationid.
      Includes the service zone they correspond to (Green or yellow).
  - name: fact_trips
    description: >
      Taxi trips corresponding to both service zones (Green and yellow).
      The table contains records where both pickup and dropoff locations ar
      Each record corresponds to a trip uniquely identified by tripid.
```

7
Deployment of a dbt project

What is deployment?

- Process of running the models we created in our development environment in a production environment
- Development and later deployment allows us to continue building models and testing them without affecting our production environment
- A deployment environment will normally have a different schema in our data warehouse and ideally a different user
- A development - deployment workflow will be something like:
  - Develop in a user branch
  - Open a PR to merge into the main branch
  - Merge the branch to the main branch
  - Run the new models in the production environment using the main branch
  - Schedule the models

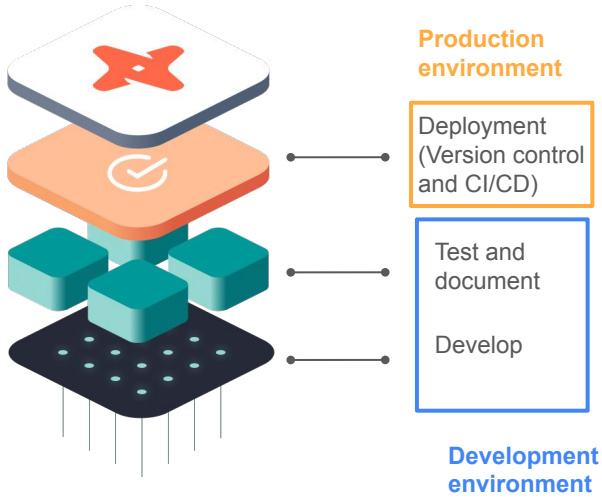
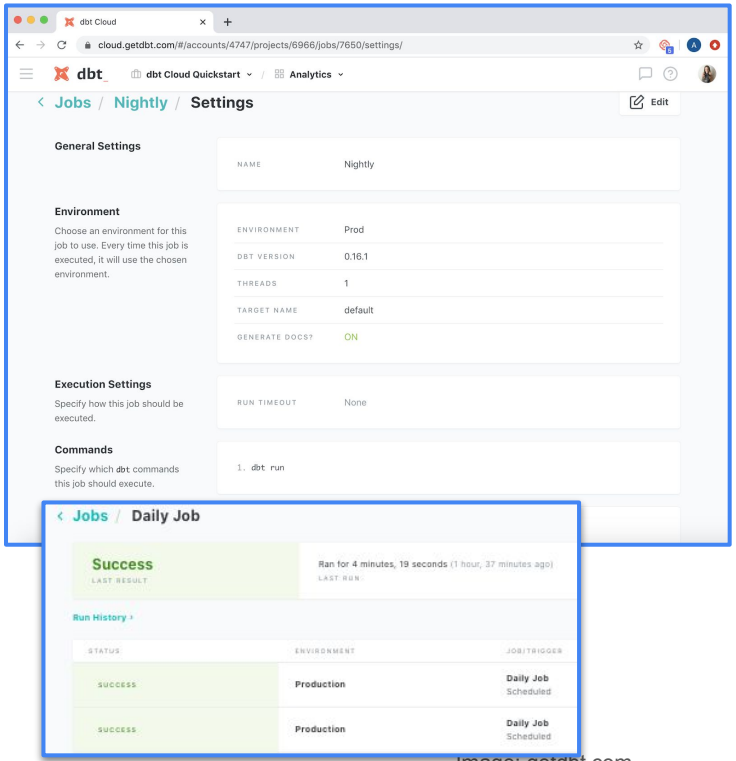


Image: getdbt.com

Running a dbt project in production

- dbt cloud includes a scheduler where to create jobs to run in production
- A single job can run multiple commands
- Jobs can be triggered manually or on schedule
- Each job will keep a log of the runs over time
- Each run will have the logs for each command
- A job could also generate documentation, that could be viewed under the run information
- If dbt source freshness was run, the results can also be viewed at the end of a job



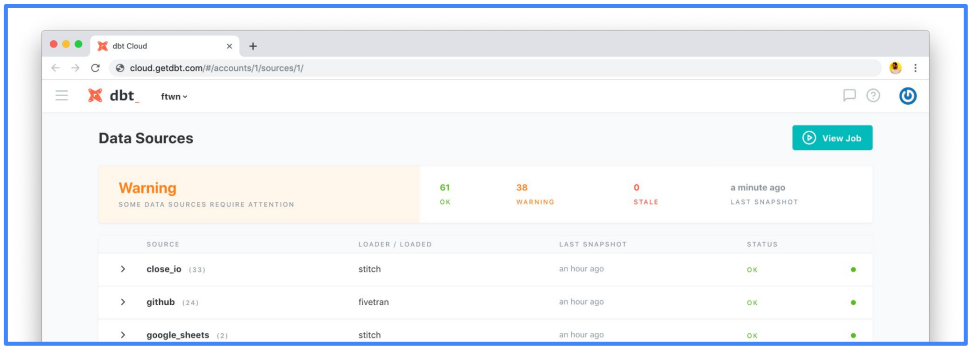


Image: getdbt.com

What is Continuous Integration (CI)?

- CI is the practice of regularly merge development branches into a central repository, after which automated builds and tests are run.
- The goal is to reduce adding bugs to the production code and maintain a more stable project.
- dbt allows us to enable CI on pull requests
- Enabled via webhooks from GitHub or GitLab
- When a PR is ready to be merged, a webhooks is received in dbt Cloud that will enqueue a new run of the specified job.
- The run of the CI job will be against a temporary schema
- No PR will be able to be merged unless the run has been completed successfully

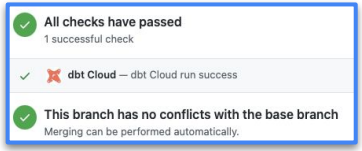


Image: getdbt.com

8

Visualising the transformed data