Introduction to DBT - What & Why?

بالعربي

dbt™

Key Learning Outcomes

1-Introduction to DBT

Understanding DBT's role in data transformation

2-Setting Up the Environment

Configuring DBT on platforms like BigQuery and Redshift

3-Writing Models with SQL

Building SQL models for data transformations

4-Data Testing

Adding tests to ensure data transformation correctness

5-Creating Data Documentation

Documenting models and creating data lineage

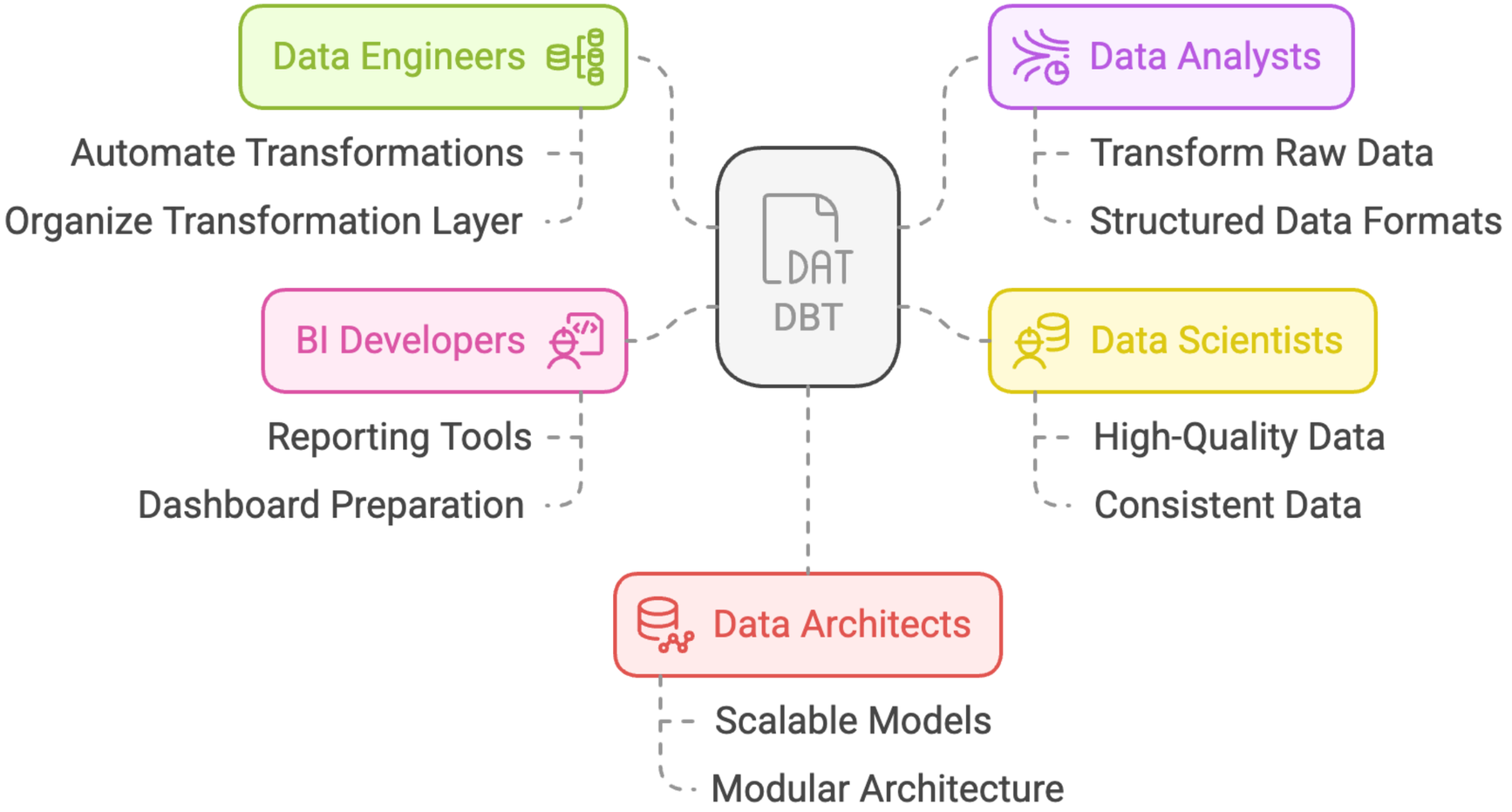
6-Version Control and Management

Integrating DBT with Git for version control

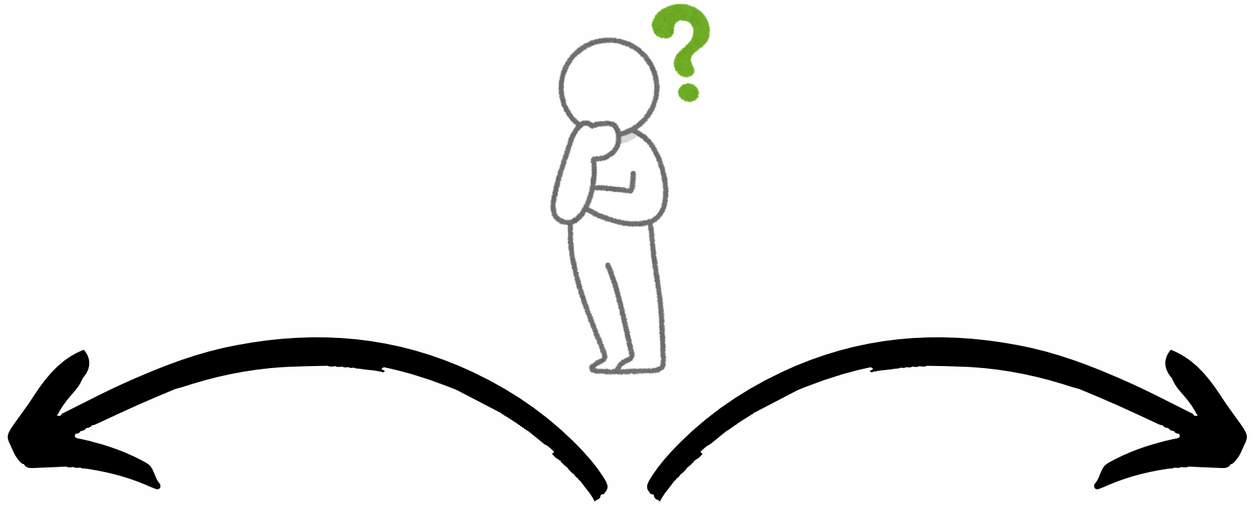
7-Deployment and Project Optimization

Deploying DBT in production and optimizing systems

Who Should Use DBT?



What is DBT?
And Why Should You Care



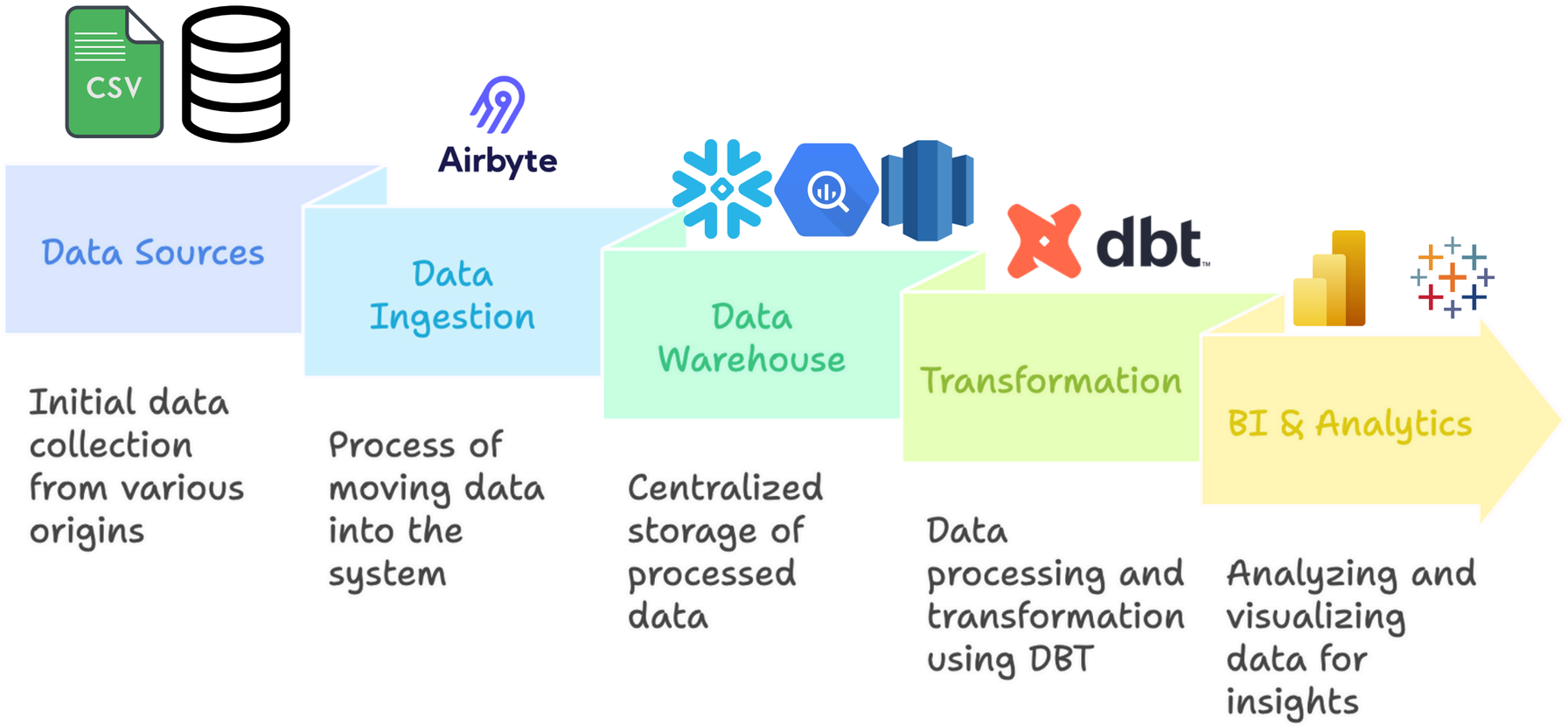
ETL

Extract >> Transform>>Load

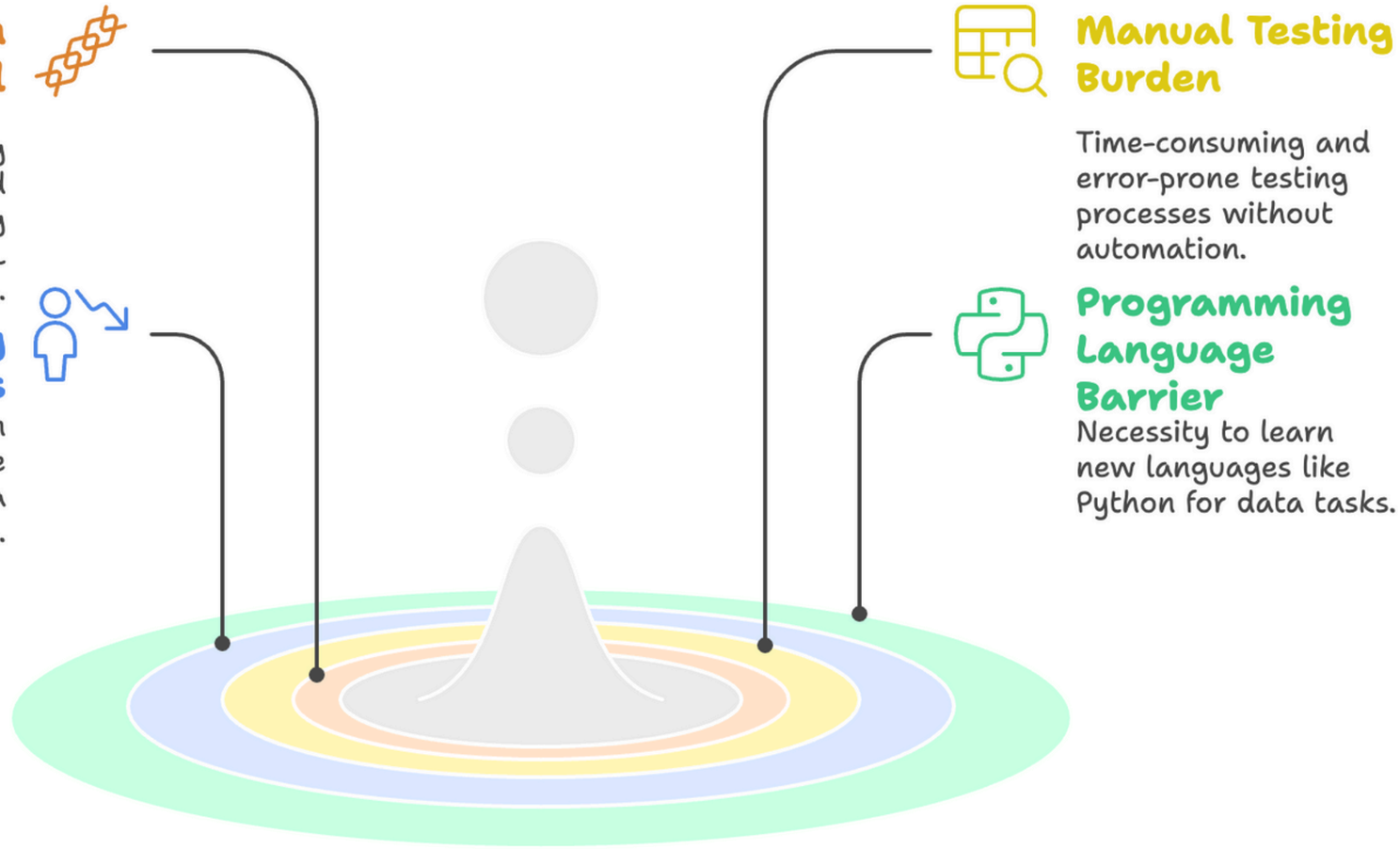
ELT

Extract >>Load>>Transform

The Modern Data Stack Diagram

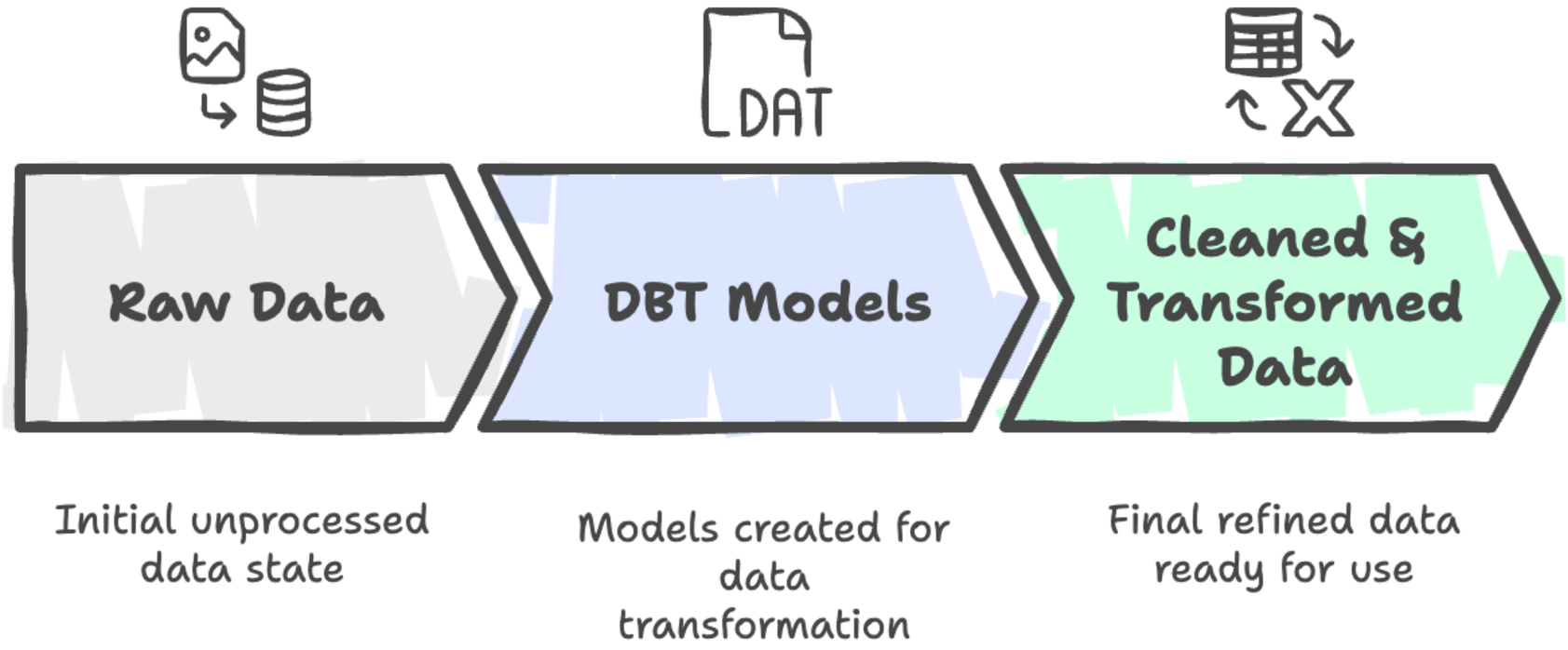


Challenges in Traditional Data Transformation



What is dbt?

- dbt (data build tool) is an open source tool that allows data analysts and engineers to transform, test, and document data in the cloud data warehouse using SQL.



The Real Power of DBT

Modularity: No More Copy-Pasting SQL

- Before DBT: You manually write and maintain long SQL queries every time you need a new report.

- With DBT: You break down SQL logic into reusable models (SQL files).
- Example: Instead of writing the same JOIN logic in 5 reports, DBT lets you define it once in a model and reuse it.

Jinja

- Using Jinja turns your dbt project into a programming environment for SQL, giving you the ability to do things that aren't normally possible in SQL.
- Use control structures (e.g. if statements and for loops) in SQL
- Use environment variables in your dbt project for production deployments

Jinja

```
{% set payment_methods = ["bank_transfer", "credit_card", "gift_card"] %}

select
    order_id,
    {% for payment_method in payment_methods %}
    sum(case when payment_method = '{{payment_method}}' then amount end) as {{payment_method}}_amount,
    {% endfor %}
    sum(amount) as total_amount
from app_data.payments
group by 1
```

This query will get compiled to:

```
select
    order_id,
    sum(case when payment_method = 'bank_transfer' then amount end) as bank_transfer_amount,
    sum(case when payment_method = 'credit_card' then amount end) as credit_card_amount,
    sum(case when payment_method = 'gift_card' then amount end) as gift_card_amount,
    sum(amount) as total_amount
from app_data.payments
group by 1
```

Macros

- Macros in Jinja are pieces of code that can be reused multiple times – they are analogous to "functions" in other programming languages, and are extremely useful if you find yourself repeating code across multiple models. Macros are defined in .sql files, typically in your macros directory (docs).

Macros

```
{% macro get_payment_amount(payment_method) %}
    SUM(CASE WHEN payment_method = '{{ payment_method }}' THEN amount END) AS {{ payment_method }}_amount
{% endmacro %}
```

This query will get compiled to:

```
SELECT
    order_id,
    {{ get_payment_amount('bank_transfer') }},
    {{ get_payment_amount('credit_card') }},
    {{ get_payment_amount('gift_card') }},
    SUM(amount) AS total_amount
FROM app_data.payments
GROUP BY 1;
```

Automated Data Quality Testing

- Before DBT: You only notice broken data when dashboards show incorrect numbers.
- With DBT: You can test your data automatically!

```yaml
models:
  - name: stg_orders
    tests:
      - unique: order_id
      - not_null: order_id
```

Version Control & Collaboration (Git)

- Before DBT:
  - SQL files are scattered across different team members' folders.
  - No easy way to track changes or revert mistakes.
- With DBT:
  - Every SQL change is version-controlled with Git.
  - Teams can collaborate without overwriting each other’s work.

Build Data Lineage & Documentation

- Before DBT: No clear way to see how data flows from raw to final tables.

- With DBT: You get automatic lineage graphs!
- Example: You can visually track how monthly_revenue is calculated from raw orders → staging → transformed tables.

Performance Optimization

- Before DBT:
  - Queries can become slow as data grows.

- With DBT:
  - DBT lets you materialize tables as views, tables, or incremental models.
  - Example: Instead of reprocessing all data, DBT can update only new rows (incremental models), speeding up transformations.

What are the benefits of DBT?

Version Control

DBT integrates with Git for effective version control and collaboration.

Modular & Scalable

Its modular design allows for scalability as data needs grow.

Automated Testing

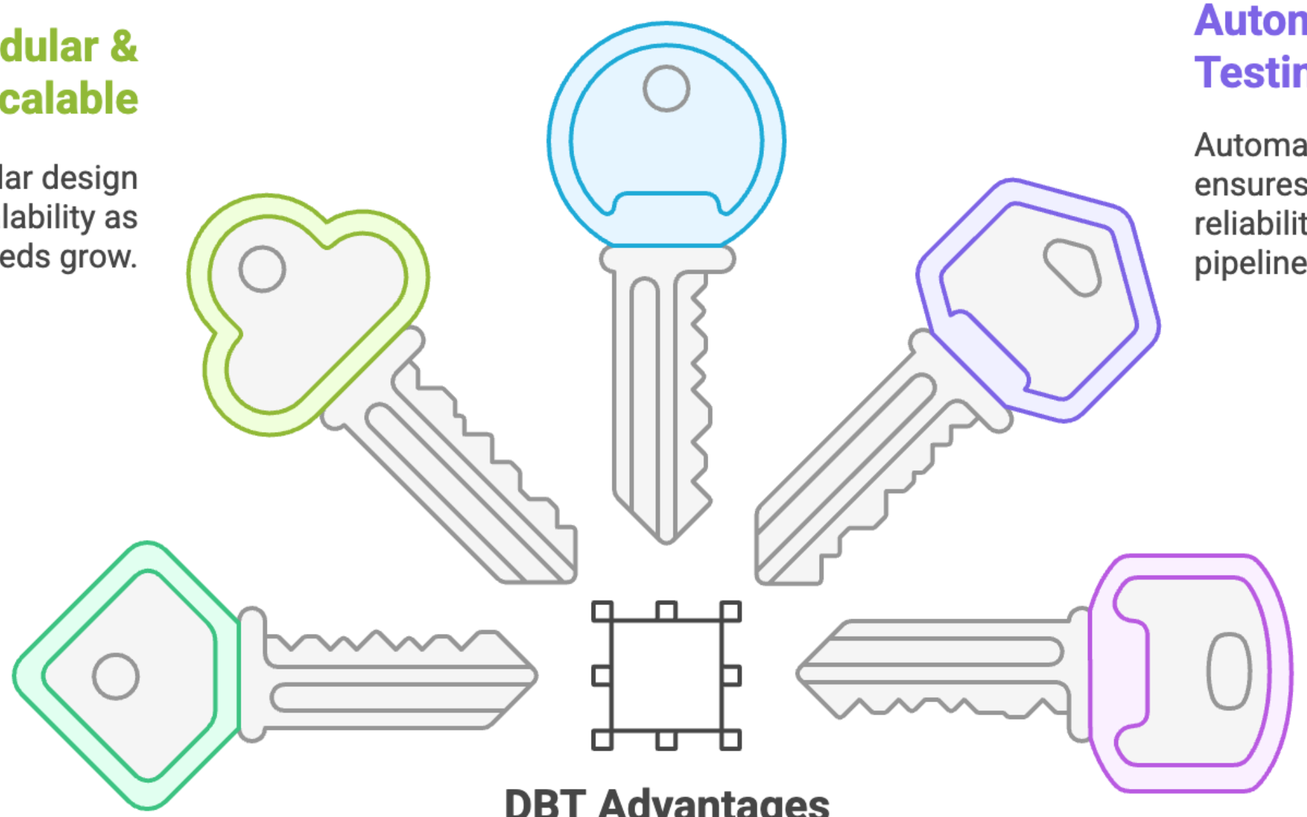
Automated data testing ensures accuracy and reliability in data pipelines.

SQL-Based

DBT's SQL-based approach makes it easy to learn and use for data professionals.

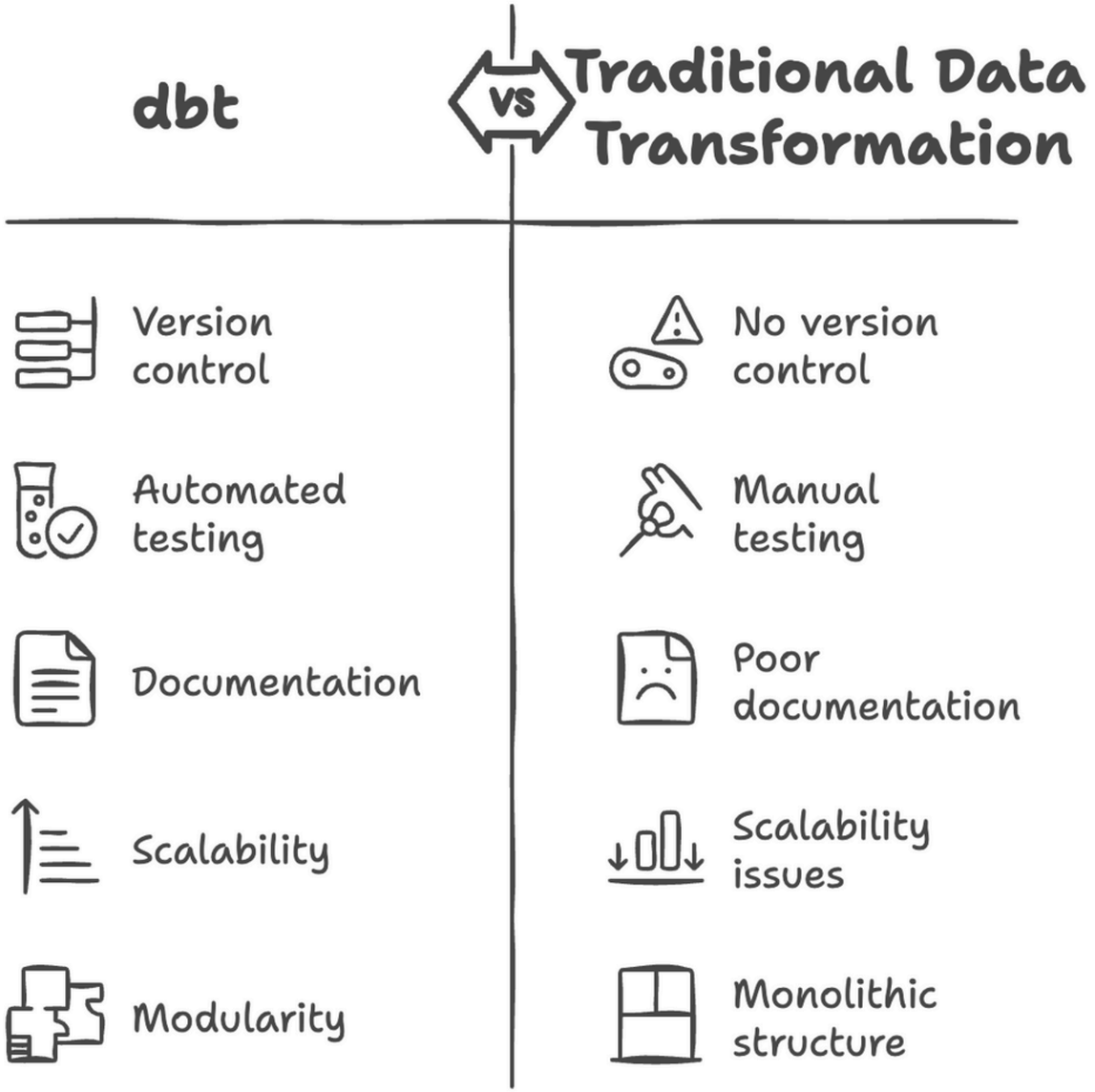
Documentation & Lineage

Comprehensive documentation and lineage tracking enhance data governance.

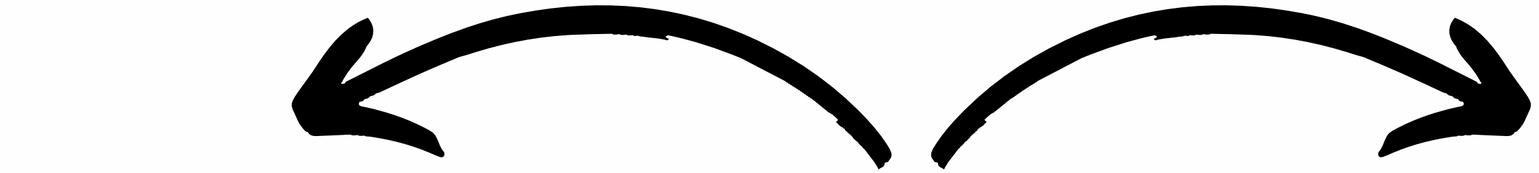


DBT Advantages

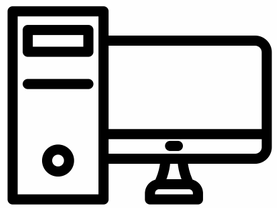
Traditional Data Transformation vs. dbt



dbt products



dbt core

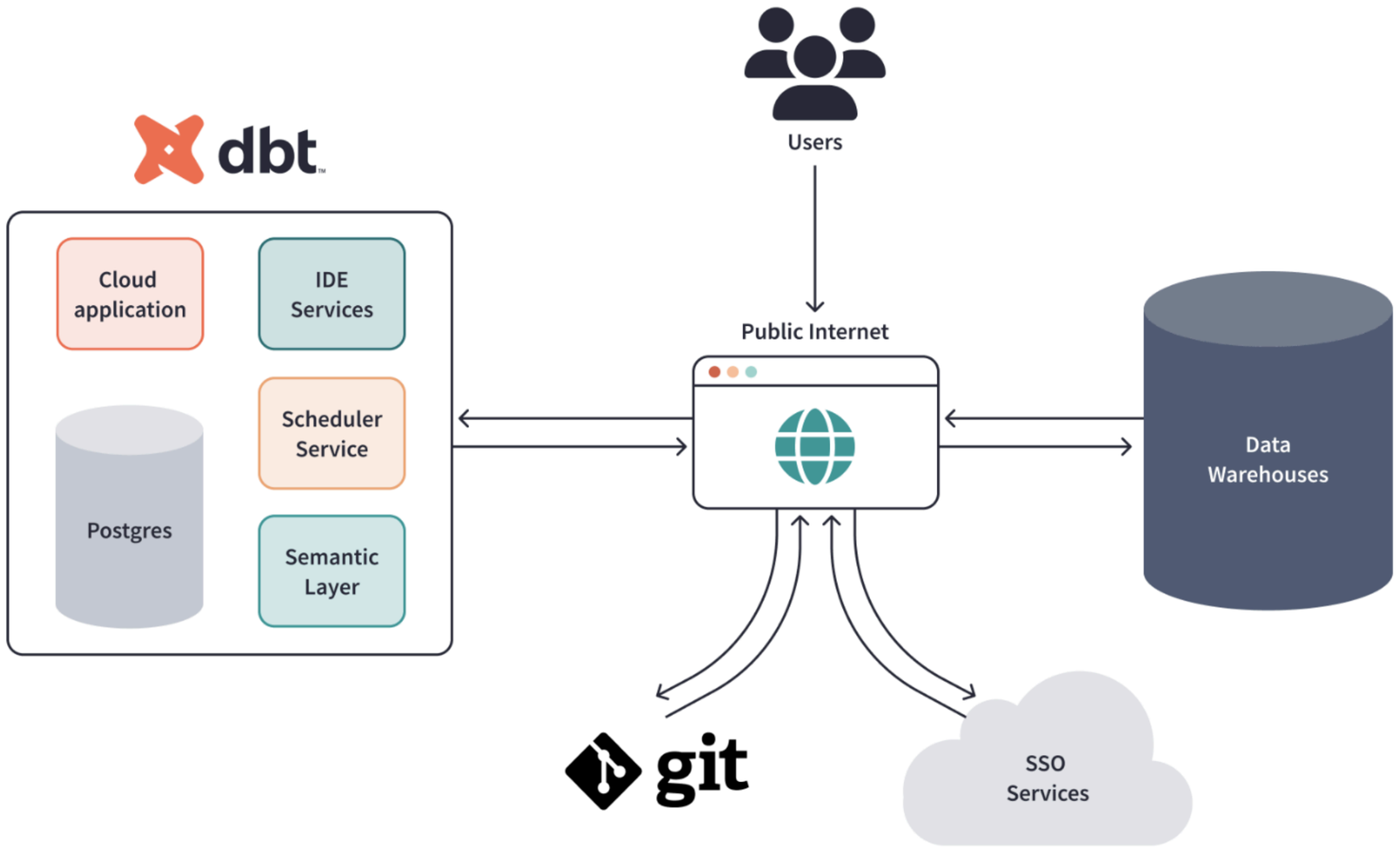


dbt cloud

Difference Between dbt Cloud and dbt Core

Feature	DBT Cloud	DBT Core
Interface	Web UI	CLI-based
Execution	Managed in the cloud	Runs locally or on a server
Scheduling	Built-in scheduler	External schedulers needed (Airflow, Prefect, Cron)
Authentication	SSO Integration	Handled manually
Version Control	Integrated with Git	Git-based but manually configured
Cost	Paid service	Free & open-source

Dbt Cloud Architecture



Thank You

Share

Subscribed

Home

Latest articles /* widget: Post Blocks */ @keyframes uc_post_blocks_elementor_d8668db3__item-animation{0%{filter: blur(4px); opacity: 0;}100%...

DevBlogIt / Jul 18, 2024

Devblogit بالعربي

Share your videos with friends, family, and the world

YouTube