**Architecture**

bt (data build tool) is an analytics engineering tool that enables data analysts and engineers to transform data in their warehouse more effectively. The architecture of dbt primarily revolves around simplifying data transformation workflows by adopting a modular, SQL-based approach. Here’s a breakdown of dbt’s architecture:

**1. Project Structure**

* dbt operates around **projects**, which are structured sets of files where you organize models, tests, snapshots, documentation, and configurations.
* Within a dbt project, you'll have directories for:
  + **Models**: SQL files defining transformations.
  + **Snapshots**: Representations of data at a certain point in time.
  + **Tests**: SQL files for quality checks (uniqueness, not-null constraints).
  + **Macros**: Jinja-based functions to help with code reuse.

**2. Data Models**

* **Model Files**: dbt operates primarily through **models**—SQL files that define transformations to be executed on the data.
* Models are categorized in stages, typically raw (stage models), intermediate, and final, to structure transformation.
* **Ref Function**: dbt uses ref() to manage model dependencies by specifying which model another model depends on, allowing dbt to construct a Directed Acyclic Graph (DAG).

**3. Dependency Management (Directed Acyclic Graph - DAG)**

* dbt automatically detects dependencies between models through the ref() function.
* The **DAG** is the heart of dbt’s execution. When you run dbt, it builds and runs models according to their dependencies within the DAG, ensuring that models load in the correct order.
* The DAG also allows dbt to optimize execution by parallelizing independent model runs, improving efficiency.

**4. Execution Layer**

* dbt leverages the database’s compute power to execute SQL code directly on the data warehouse. It connects to popular cloud data warehouses like Snowflake, BigQuery, Redshift, and Databricks.
* dbt runs models in the order determined by the DAG, executing transformations directly within the warehouse, making it highly performant for data manipulation.

**5. Macros and Jinja Templating**

* dbt uses **Jinja**, a Python templating language, to enable dynamic SQL generation.
* **Macros**: Functions written in Jinja and SQL to create reusable code, promoting code modularity and avoiding repetition.
* Macros allow users to create standardized SQL transformations that can be reused across models, improving maintainability and scalability.

**6. Testing and Documentation**

* dbt supports **data testing** directly within models, allowing users to define tests for data quality, such as ensuring values are unique, not null, or within specific ranges.
* **Documentation**: dbt can auto-generate documentation based on model descriptions, tests, and DAGs. It integrates with data catalogs, making it easy for data teams to share details about their transformations and dependencies.
* **Sources**: dbt allows users to define external data sources, so teams can trace lineage back to the raw data.

**7. Snapshots**

* Snapshots allow dbt to store records of data at specific intervals, creating a history of data changes.
* Snapshots are useful for **slowly changing dimensions** (SCDs), allowing users to track changes in source data over time without altering the original data.

**8. Deployment and Orchestration**

* dbt Cloud (the managed service for dbt) provides a web interface for managing, scheduling, and deploying dbt jobs.
* Alternatively, dbt integrates seamlessly with orchestration tools like **Airflow** for scheduling and executing transformation pipelines, supporting continuous integration and deployment (CI/CD) workflows.

**9. CLI and dbt Cloud Interface**

* **dbt CLI**: A command-line interface that developers use for running models, testing, building documentation, and interacting with their dbt projects.
* **dbt Cloud**: A web-based IDE and managed environment for scheduling jobs, hosting documentation, and monitoring performance, especially useful for larger teams.

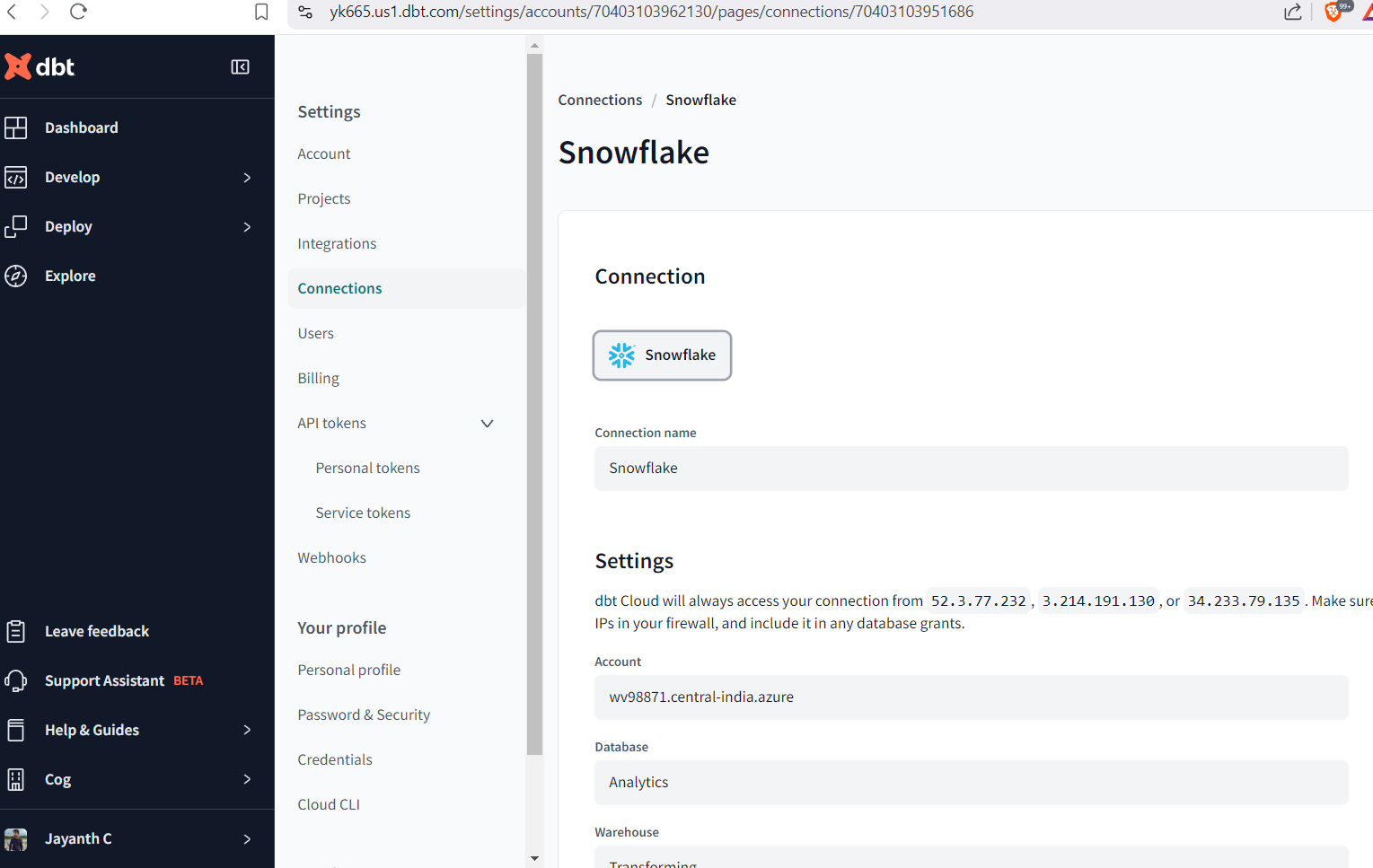
**Project**

In the below screen shot as we can in projects section a Analytics project is present which I have created.

A screenshot of a computer

Description automatically generated

While creating project itself we have to create the connection for warehouse. There will be many options to connection warehouses like Snowflake, Bigquery , Redshift. I have created the connection for snowflake warehouse account and you should have that particular warehouse account before connecting.



**Creating SQL models for gold and silver layers**

If we want to create the models, macros etc for a particular project , from the left top go to **Develop**->**Cloud IDE.**  
Here you can on project and create the models , macros, seeds, tests etc.  
I have created gold layer, silver layer directories with the models .sql file in each directory of model directory.

A screenshot of a computer

Description automatically generated

**Automate transformation**

In dbt, **automating transformations** means setting up a workflow where data transformations run on a regular schedule or are triggered by events, often without manual intervention. This is typically achieved by orchestrating dbt runs with scheduling tools or platforms like **dbt Cloud**, **Airflow**, or **Cron**. Here are several ways to implement automatic transformations in dbt:

**Using dbt Cloud's Scheduler**

dbt Cloud provides built-in scheduling to run dbt jobs, making it straightforward to automate transformations.

* **Create a Job**: In dbt Cloud, create a new job with the transformation steps you want to run (e.g., dbt run, dbt test, etc.).
* **Configure the Schedule**: Set up the schedule using the dbt Cloud interface. You can specify the frequency (e.g., hourly, daily, weekly) and customize the timing to meet your needs.

A screenshot of a computer

Description automatically generated