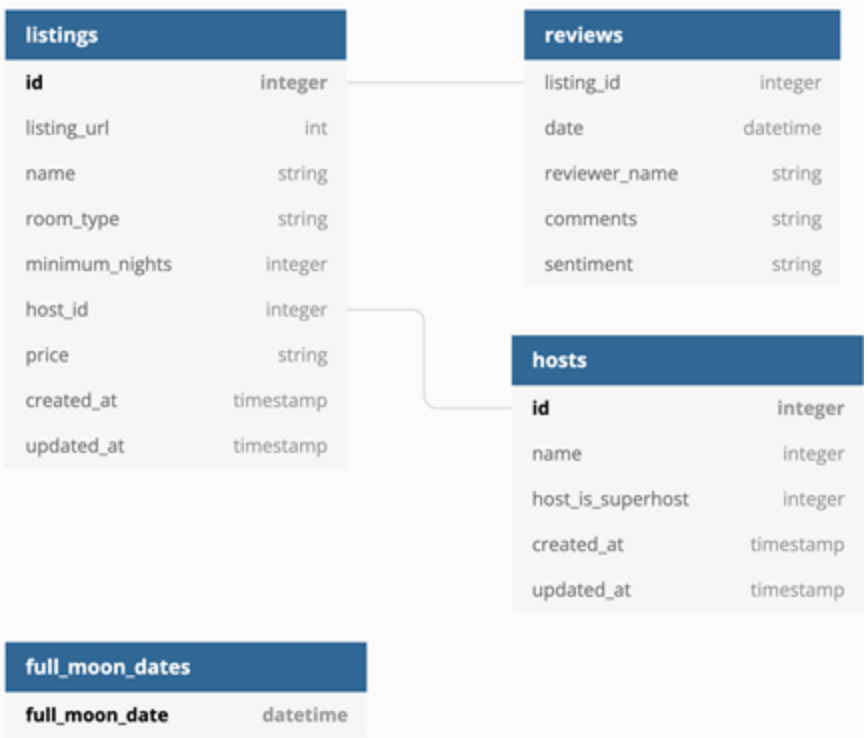
Airbnb DBT Project

By: Adrian Chavez-Loya

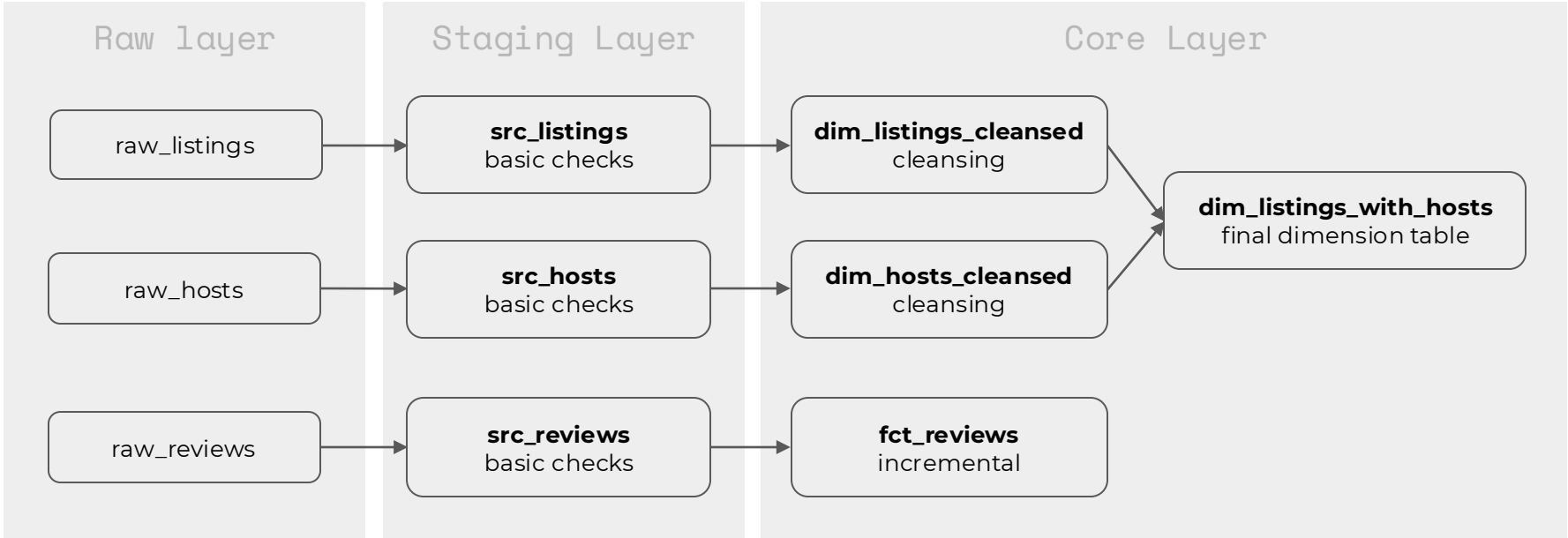
Source data:
Comes from official Airbnb open-source data:
https://insideairbnb.com/get-the-data/

The historical data we used is for listing and host details from Berlin from the years 2015-2021. This data was stored in AWS.

Input source data we took from AWS S3



Our Data Stream in Snowflake we will implement



DB/Schema Creation

airbnb_raw_setup_and_impo ··· × | airbnb_user_role_creation.sql | operations.sql

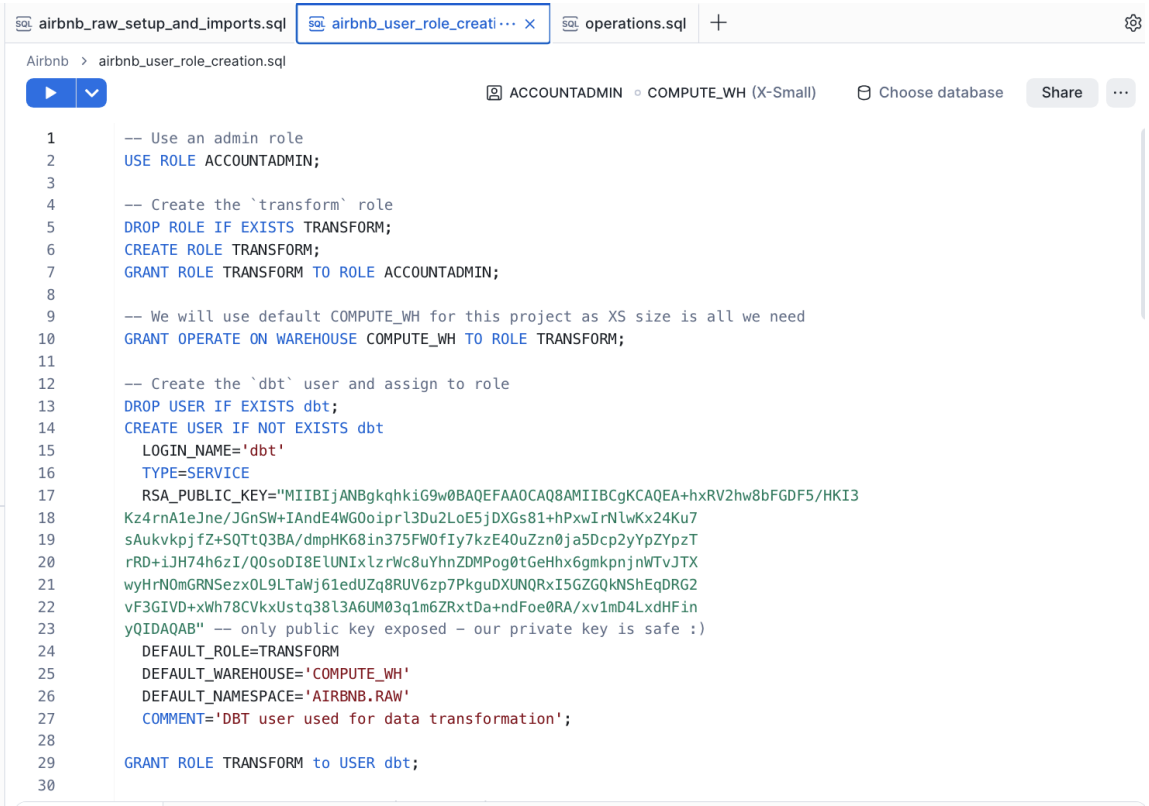
Airbnb > airbnb_raw_setup_and_imports.sql

ACCOUNTADMIN ◦ COMPUTE_WH (X-Small)

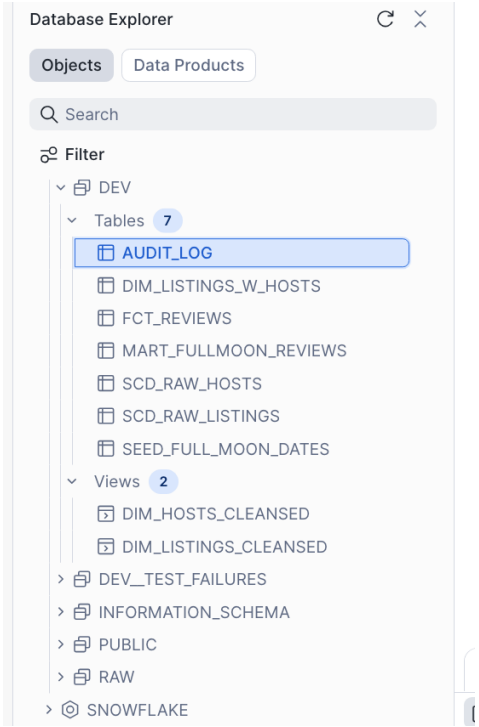
```
1  -- Set up the defaults
2  CREATE WAREHOUSE IF NOT EXISTS COMPUTE_WH;
3  USE WAREHOUSE COMPUTE_WH;
4
5  DROP DATABASE IF EXISTS AIRBNB CASCADE;
6
7  CREATE DATABASE AIRBNB;
8
9  CREATE SCHEMA IF NOT EXISTS AIRBNB.RAW;
10 CREATE SCHEMA IF NOT EXISTS AIRBNB.DEV;
11
12 USE DATABASE AIRBNB;
13 USE SCHEMA RAW;
14
15 -- Create our three tables and import the data from S3
16 CREATE OR REPLACE TABLE raw_listings
17                     (id integer,
18                      listing_url string,
19                      name string,
20                      room_type string,
21                      minimum_nights integer,
22                      host_id integer,
23                      price string,
24                      created_at datetime,
25                      updated_at datetime);
26
27 COPY INTO raw_listings (id,
28                          listing_url,
29                          name,
30                          room_type,
```

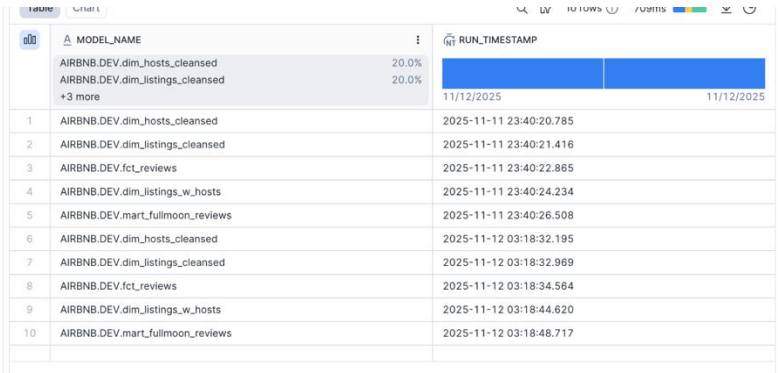
```
CREATE OR REPLACE TABLE raw_reviews
                    (listing_id integer,
                     date datetime,
                     reviewer_name string,
                     comments string,
                     sentiment string);

COPY INTO raw_reviews (listing_id, date, reviewer_name, comments, sentiment)
                   from 's3://dbt-datasets/reviews.csv'
                    FILE_FORMAT = (type = 'CSV' skip_header = 1
                    FIELD_OPTIONALLY_ENCLOSED_BY = '"');


CREATE OR REPLACE TABLE raw_hosts
                    (id integer,
                     name string,
                     is_superhost string,
                     created_at datetime,
                     updated_at datetime);

COPY INTO raw_hosts (id, name, is_superhost, created_at, updated_at)
                   from 's3://dbt-datasets/hosts.csv'
                    FILE_FORMAT = (type = 'CSV' skip_header = 1
                    FIELD_OPTIONALLY_ENCLOSED_BY = '"');
```

Created roles/warehouse, and dbt access user (private key is safe don't worry)



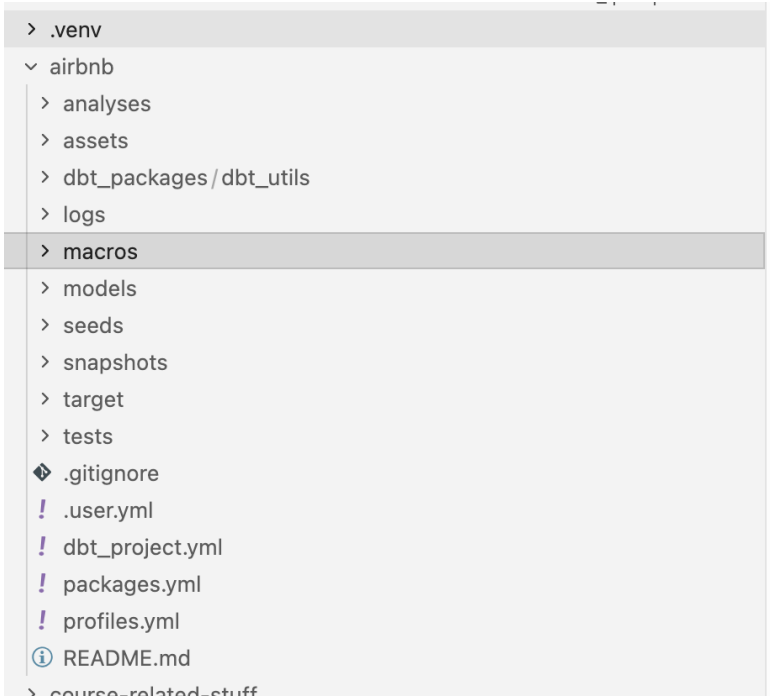
Final view of our data warehouse in the end. Includes audit_log which provides logs of model runs (made in dbt)

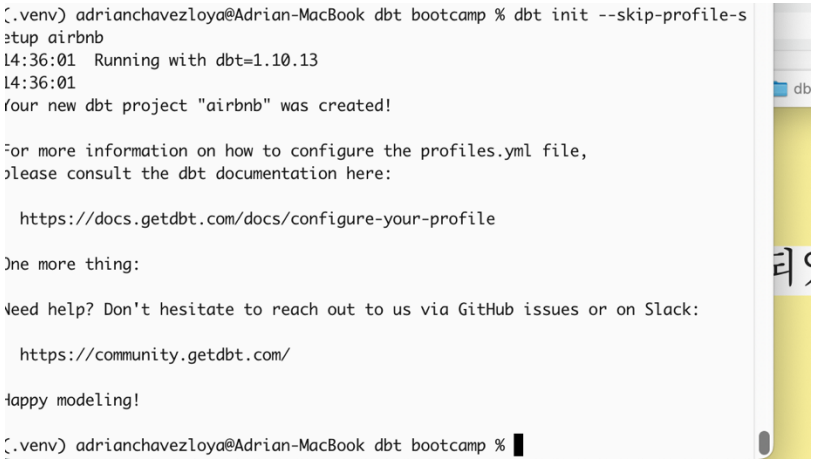




	MODEL_NAME	RUN_TIMESTAMP
	AIRBNB.DEV.dim_hosts_cleansed 20.0% AIRBNB.DEV.dim_listings_cleansed 20.0% +3 more	11/12/2025 – 11/12/2025
1	AIRBNB.DEV.dim_hosts_cleansed	2025-11-11 23:40:20.785
2	AIRBNB.DEV.dim_listings_cleansed	2025-11-11 23:40:21.416
3	AIRBNB.DEV.fct_reviews	2025-11-11 23:40:22.865
4	AIRBNB.DEV.dim_listings_w_hosts	2025-11-11 23:40:24.234
5	AIRBNB.DEV.mart_fullmoon_reviews	2025-11-11 23:40:26.508
6	AIRBNB.DEV.dim_hosts_cleansed	2025-11-12 03:18:32.195
7	AIRBNB.DEV.dim_listings_cleansed	2025-11-12 03:18:32.969
8	AIRBNB.DEV.fct_reviews	2025-11-12 03:18:34.564
9	AIRBNB.DEV.dim_listings_w_hosts	2025-11-12 03:18:44.620
10	AIRBNB.DEV.mart_fullmoon_reviews	2025-11-12 03:18:48.717

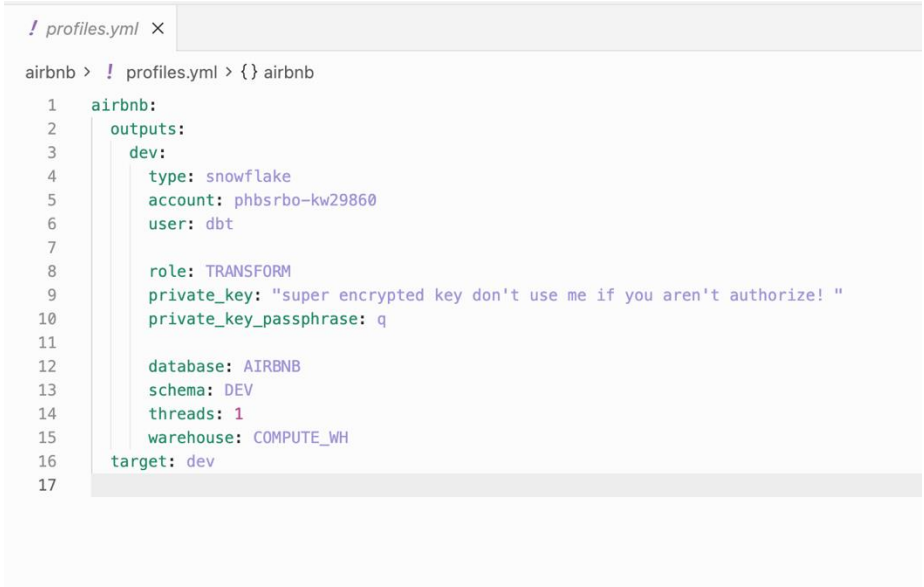
We initialized and isolated dependencies with a virtual environment. On the left is the final view of the dbt project.





We also generated a public/private key for secure access to Snowflake (user/password method is going to be discontinued in Snowflake). You saw the public key earlier.

We have our profiles.yml connecting dbt to our data warehouse



dbt_project.yml defines our targets and configurations for our folders. In this, we practiced usings grants for roles, hooks, and set our types of default materialization settings we wanted for our data layers.

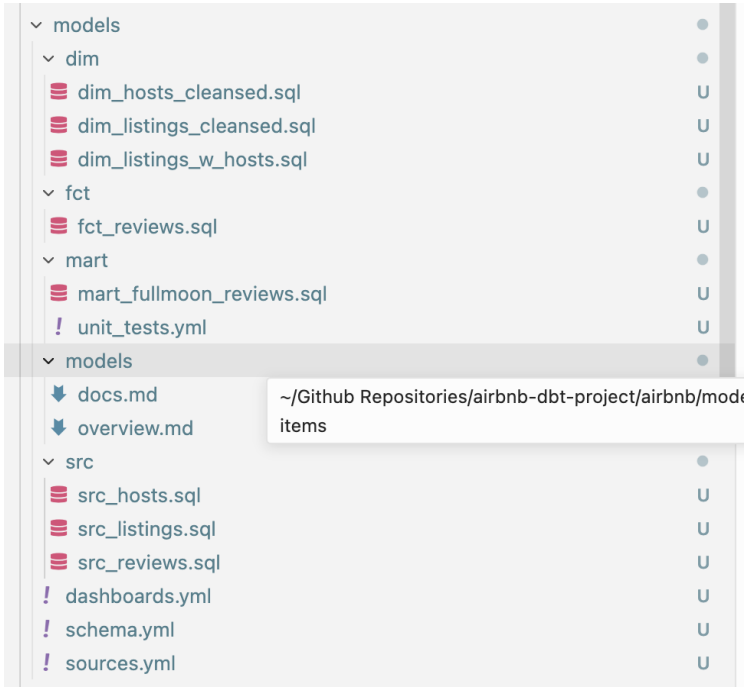
```yaml
# Name your project! Project names should contain only lowercase characters
# and underscores. A good package name should reflect your organization's
# name or the intended use of these models
name: 'airbnb'
version: '1.0.0'

# This setting configures which "profile" dbt uses for this project.
profile: 'airbnb'

# These configurations specify where dbt should look for different types of files.
# The `model-path
# found in the "m
model-paths: ["mo
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]
asset-paths: ["assets"]

clean-targets:         # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"
```

dbt Project
Source: dbt_project-latest.json

tes that models in this project can be
ably won't need to change these!

```yaml
on-run-start:
  - "CREATE TABLE IF NOT EXISTS {{ target.schema }}.audit_log (
      model_name STRING,
      run_timestamp TIMESTAMP
    )"

models:
  airbnb:
    +grants:
      select: ["transform", "reporter"] # grants to our roles
    +post-hook:
      - "INSERT INTO {{ target.schema }}.audit_log VALUES ('{{ this }}', CURRENT_TIMESTAMP)"
    +materialized: view
    dim:
      +materialized: table
    src:
      +materialized: ephemeral ## Source views no longer needed



data_tests:
  +store_failures: true
  +schema: _test_failures
```

Models folder has all our models. Look at them all in detail in the github repository



Our source layer (staging) tables! I just adjusted some columns from our raw layer.

package-lock.yml U | src_hosts.sql U

airbnb > models > src > src_hosts.sql

```sql
--all sources defaulting to views
WITH raw_hosts AS (
    SELECT
        *
    FROM
        {{ source('airbnb', 'hosts') }}
)
SELECT
    id AS host_id,
    NAME AS host_name,
    is_superhost,
    created_at,
    updated_at
FROM
    raw_hosts
```

package-lock.yml U | src_listings.sql U

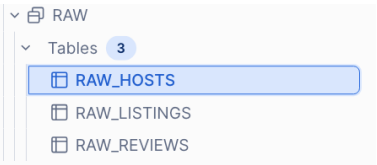
airbnb > models > src > src_listings.sql

```sql
WITH raw_listings AS (
    SELECT
        *
    FROM
        {{ source('airbnb', 'listings') }}
)
SELECT
    id AS listing_id,
    name AS listing_name,
    listing_url,
    room_type,
    minimum_nights,
    host_id,
    price AS price_str,
    created_at,
    updated_at
FROM
    raw_listings
```

package-lock.yml U | src_reviews.sql U

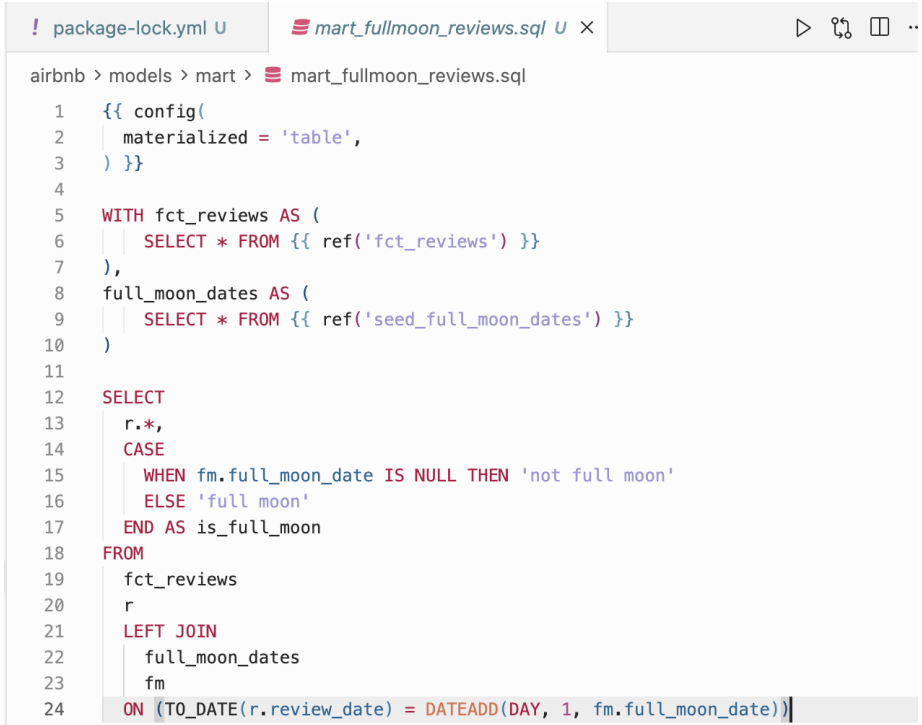
airbnb > models > src > src_reviews.sql

```sql
WITH raw_reviews AS (
    SELECT
        *
    FROM
        {{ source('airbnb', 'reviews') }}
)
SELECT
    listing_id,
    date AS review_date,
    reviewer_name,
    comments AS review_text,
    sentiment AS review_sentiment
FROM
    raw_reviews
```



Raw tables

This is a mart table we created for fun to view reviews to see if people are more moody and perhaps give worse reviews.



Dimension tables for listings and hosts are made and combined for better analysis. We mainly use dim_listings_w_hosts for our dashboard later.

dim_hosts_cleansed.sql U

airbnb > models > dim > dim_hosts_cleansed.sql

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


WITH src_hosts AS (
    SELECT
        *
    FROM
        {{ ref('src_hosts') }}
)

SELECT
    host_id,
    COALESCE(host_name, 'Anonymous')::TEXT as host_name,
    is_superhost,
    created_at,
    updated_at
FROM
    src_hosts
```

dim_listings_cleansed.sql U

airbnb > models > dim > dim_listings_cleansed.sql

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


WITH src_listings as (
    SELECT * FROM {{ ref('src_listings') }}

)


SELECT
    listing_id,
    listing_name,
    room_type,
    CASE
        WHEN minimum_nights = 0 THEN 1 -- min zero nights means one night actuall
        ELSE minimum_nights
    END AS minimum_nights,
    host_id,
    REPLACE(price_str, '$', '') :: NUMBER(10,2) AS price, -- remove '$' and cast
    created_at,
    updated_at
FROM
    src_listings
```

dim_listings_w_hosts.sql U

airbnb > models > dim > dim_listings_w_hosts.sql

```sql
-- Will be initialized as a table
WITH
l AS (
    SELECT
        *
    FROM
        {{ ref('dim_listings_cleansed') }}
),
h AS (
    SELECT *
    FROM {{ ref('dim_hosts_cleansed') }}
)

SELECT
    l.listing_id,
    l.listing_name,
    l.room_type,
    l.minimum_nights,
    l.price,
    l.host_id,
    h.host_name,
    h.is_superhost as host_is_superhost,
    l.created_at,
    GREATEST(l.updated_at, h.updated_at) as updated_at
FROM l
LEFT JOIN h ON (h.host_id = l.host_id)
```

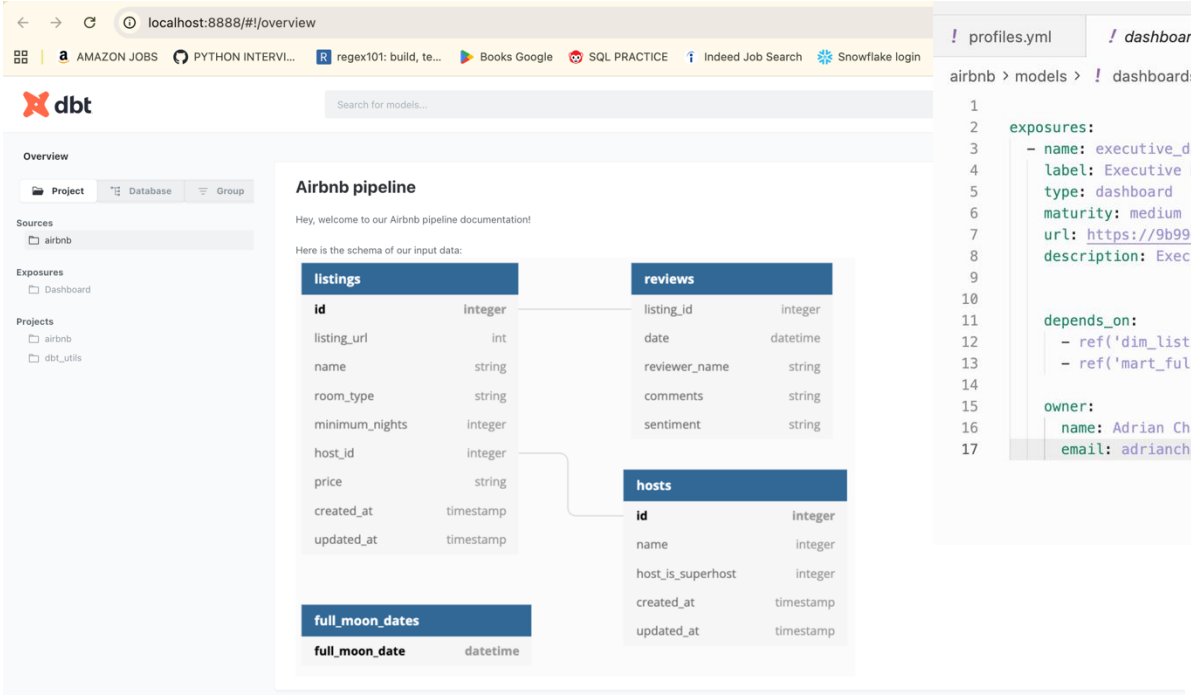
We created a fact table for reviews but doesn't include the typical metrics a typical fact table would

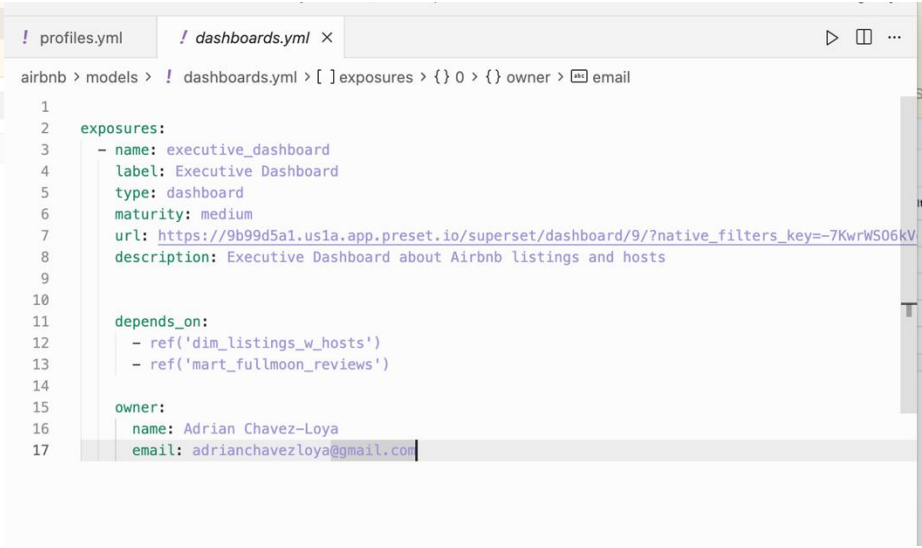
fct_reviews.sql U

airbnb > models > fct > fct_reviews.sql

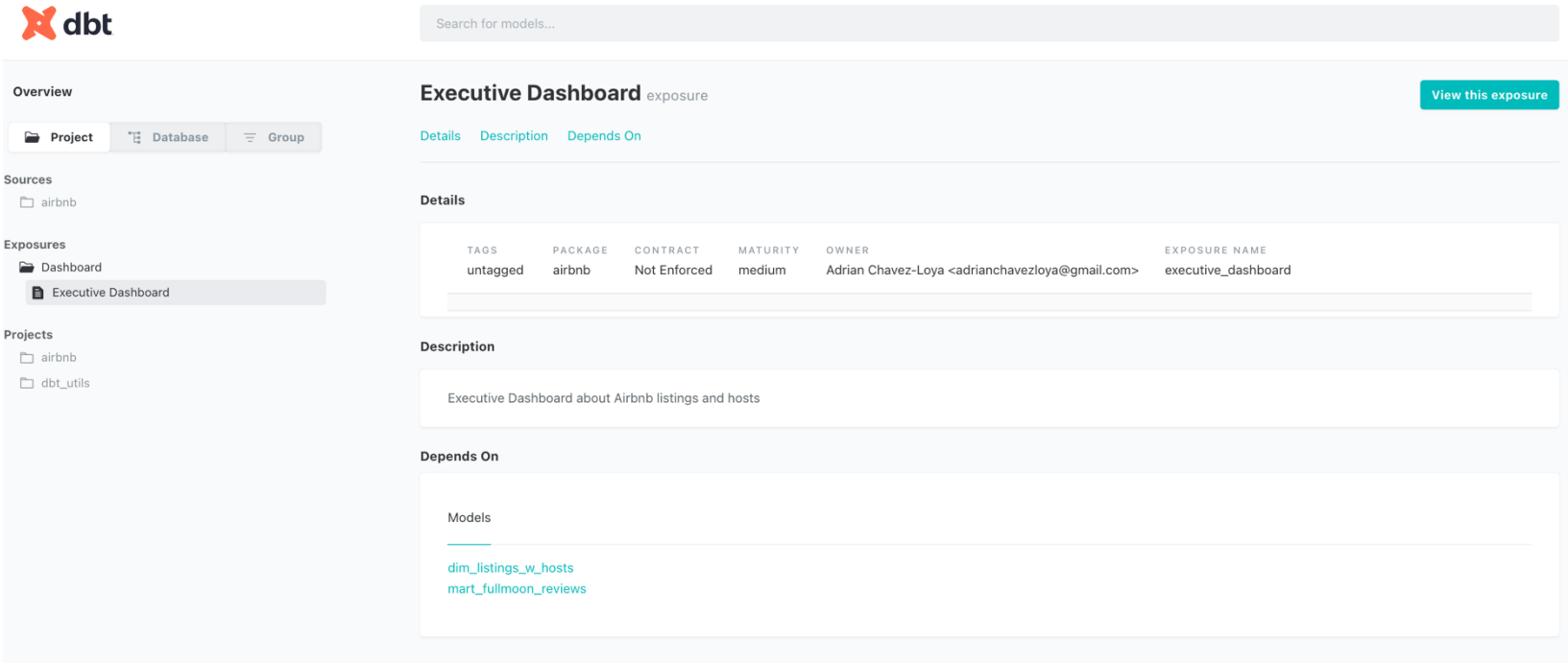
```sql
{{
  config(
    materialized = 'incremental',
    on_schema_change='fail'
    )
}}
WITH src_reviews AS (
  SELECT * FROM {{ ref('src_reviews') }}
)
SELECT
  {{ dbt_utils.generate_surrogate_key(['listing_id', 'review_date', 'reviewer_name', 'review_text']) }}
    AS review_id,
  *
  FROM src_reviews
WHERE review_text is not null
{% if is_incremental() %}
  AND review_date > (select max(review_date) from {{ this }})
{% endif %}
```

We created easy markdown files to create our UI menu which includes our input data and some text.

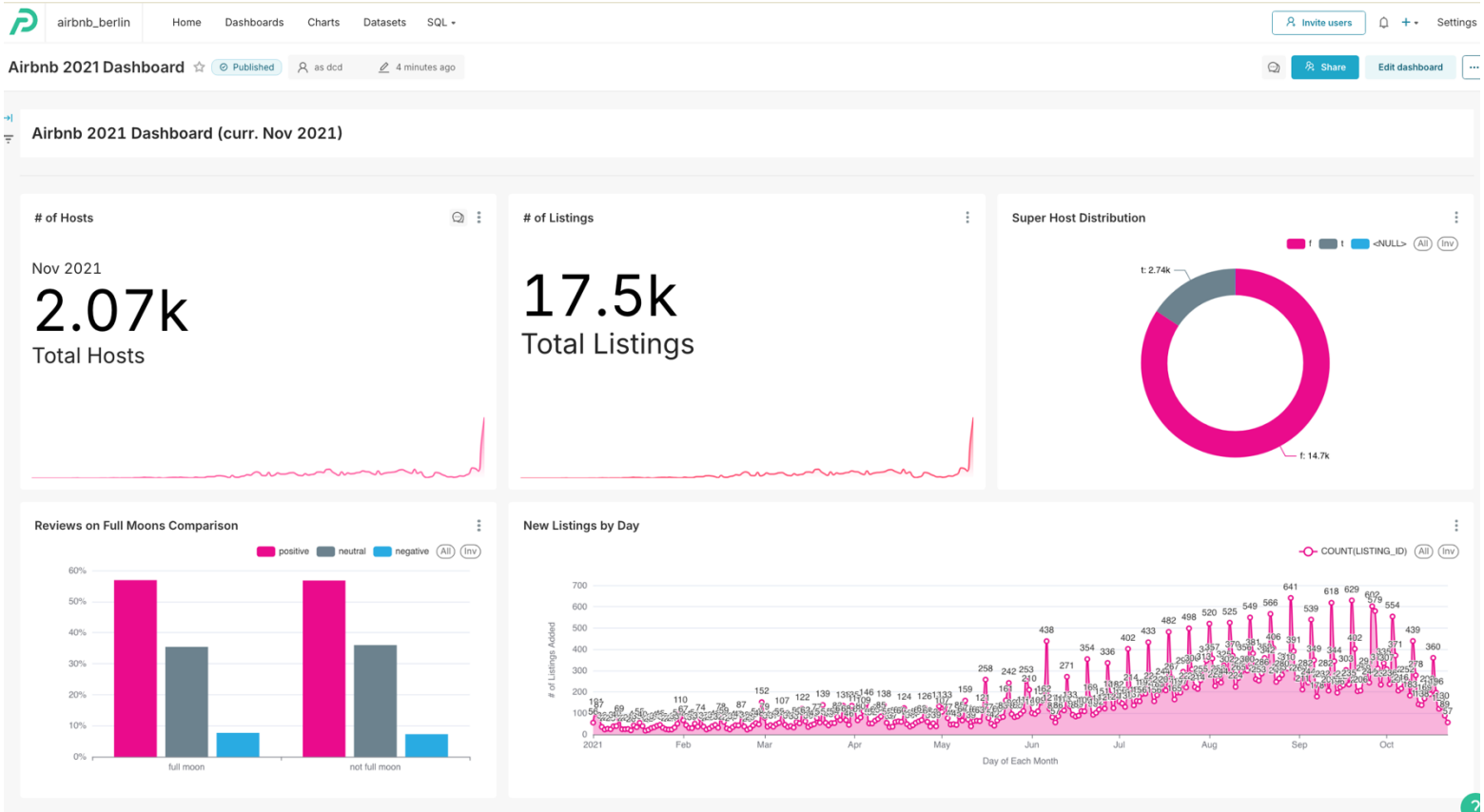




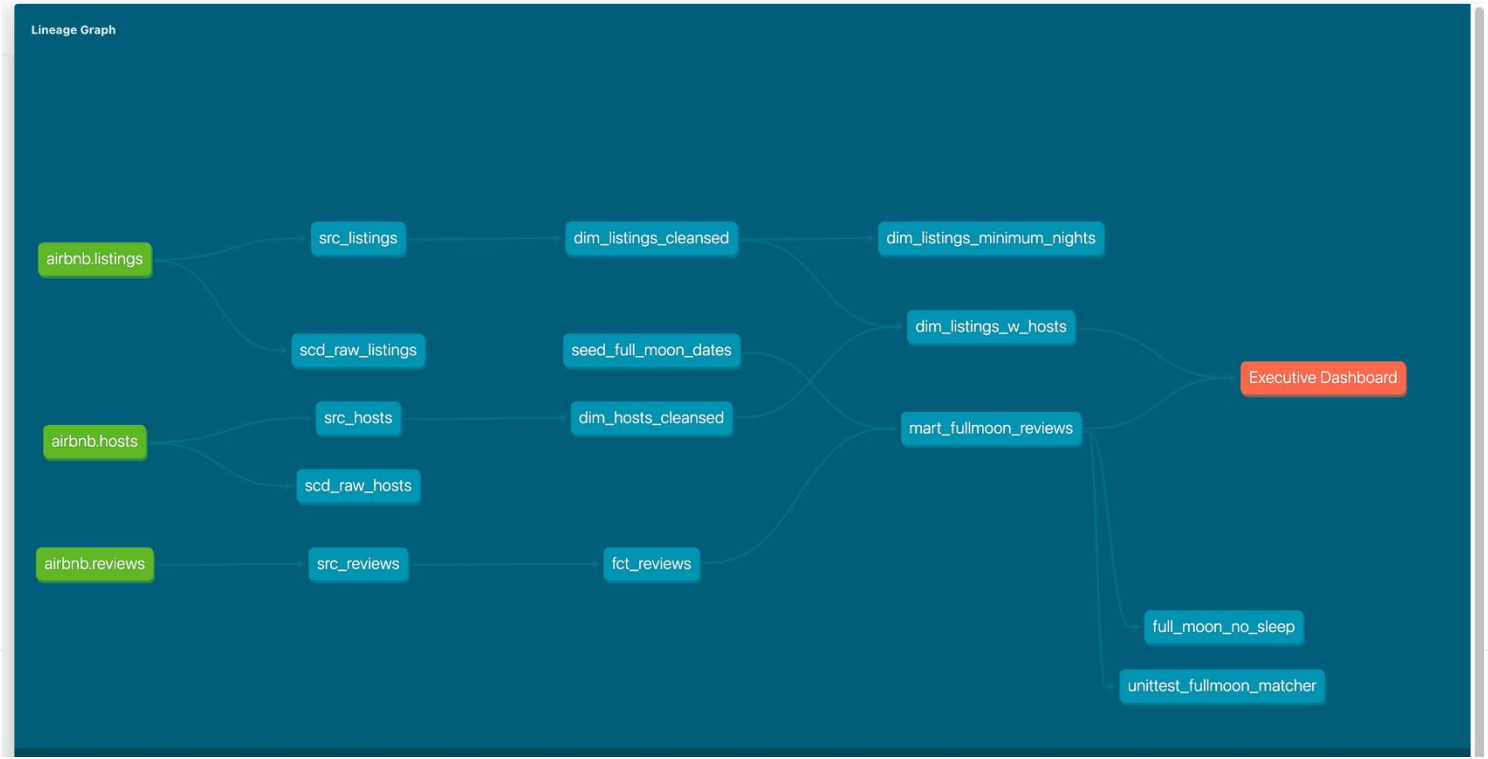
We also have a direct link to our simple dashboard for this project



Our dashboard we connected to Snowflake and created with Preset (Side Note: turns our people are unaffected by full moons when reviewing)



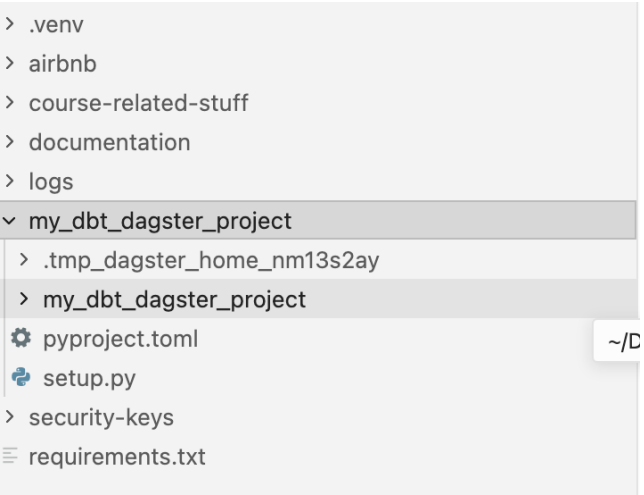
Here is our lineage graph made with dbt



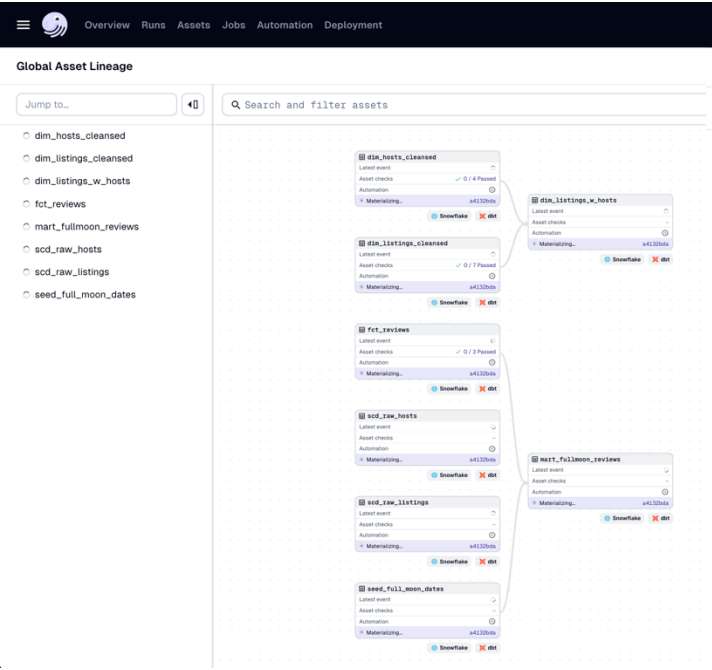
We did a number of other exercises which can be found in the repository. This includes:

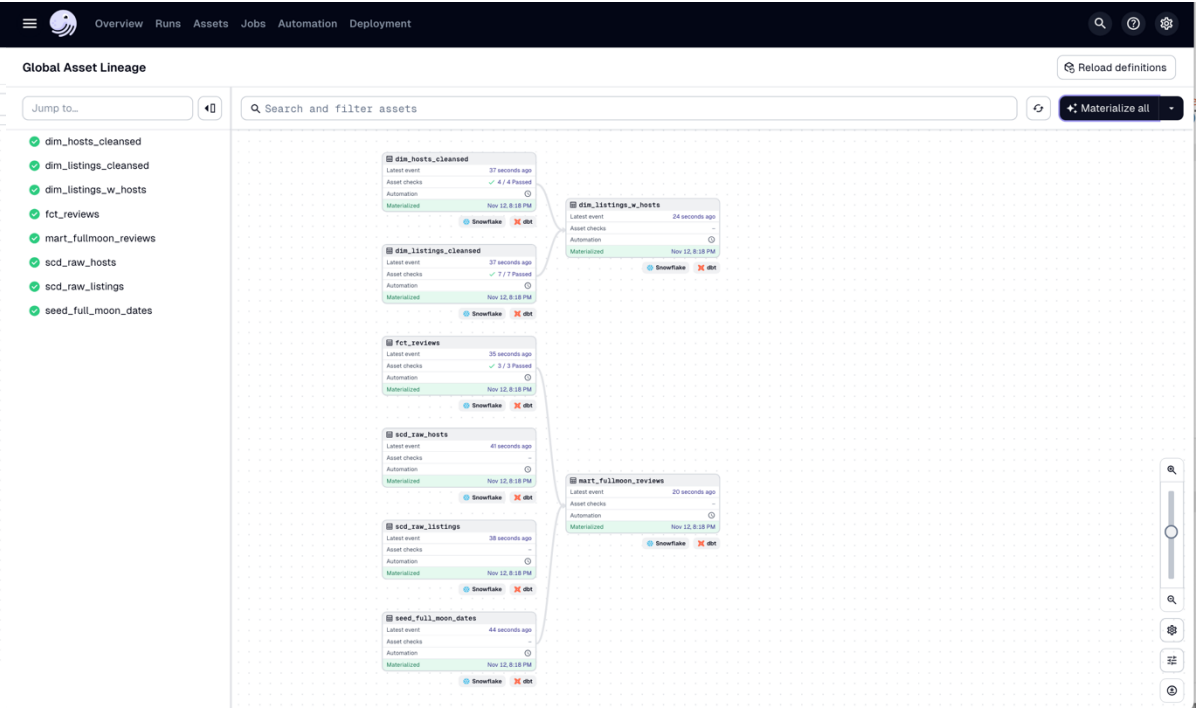
- Implementing Jinja2 macros
- Created snapshots
- Made sample validation tests
- Experimented with dagster orchestration

I will show you some screenshots of my initialized UI interface for dagster which I used to also play around with its scheduler and see how it can be used for orchestration much like Airflow. We were able to scaffold dagster into our dbt project.

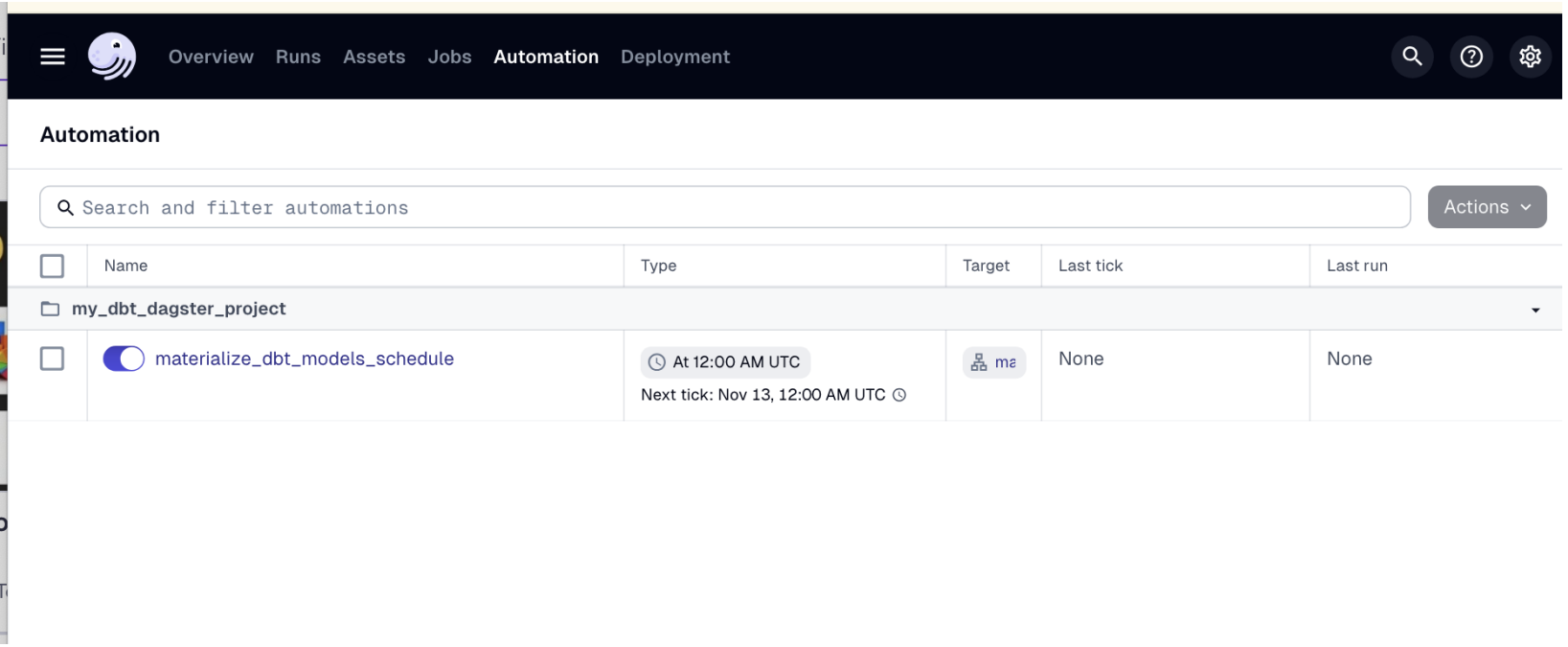


Here is asset lineage within Dagster





If we wanted to continually run our dbt models on schedule we could with Dagster!



Ultimately through this project, we were able to learn some of the following:

- How to implement data models and perform ELT for Snowflake using dbt
- How to tune and tweak different settings to implement different table/view types at different stages
- Learned about Jira macros, seeds, snapshots, hooks exposures, and so much more!
- Practiced using a creating a data warehouse that was usable for analysis