

Building a robust data pipeline with the dAG stack: dbt, Airflow, and Great Expectations

Sam Bail @spbail
Airflow Summit 2021



Hi, I'm Sam!

- I'm a "data person" & consultant based in NYC
- I've worked for a few data-centric startups in healthcare and data infrastructure (Flatiron Health, Superconductive / Great Expectations)
- I run, bike, podcast @blogcastpod, and organize workshops @NYCPyLadies



Agenda

The dAG stack
components:

Quick recap of
dbt, Airflow, Great
Expectations

Choose your
own DAG pt 1:

Integrating dbt
and Airflow

Choose your
own DAG pt 2:

Testing with dbt
and Great
Expectations

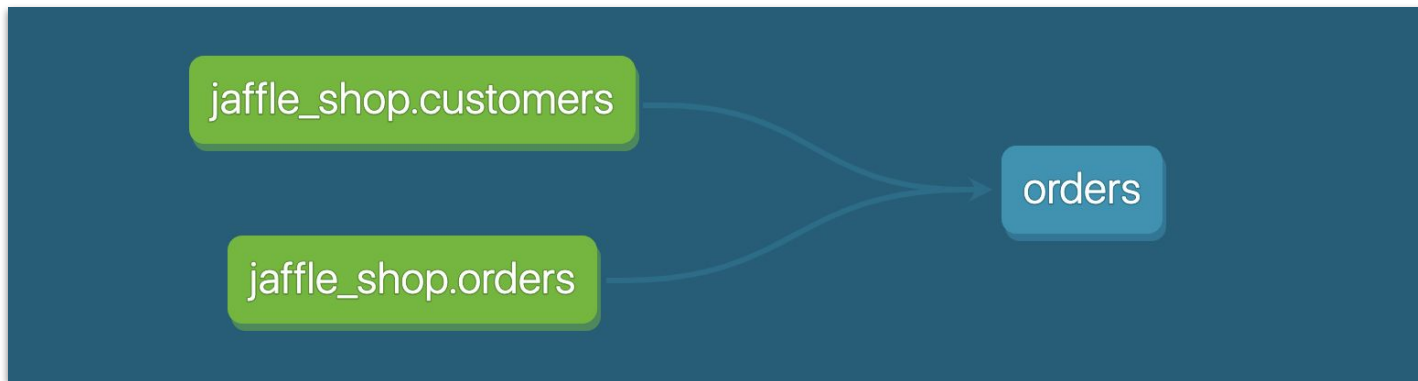
The dAG stack components

(Quick recap)



"The T in ELT"

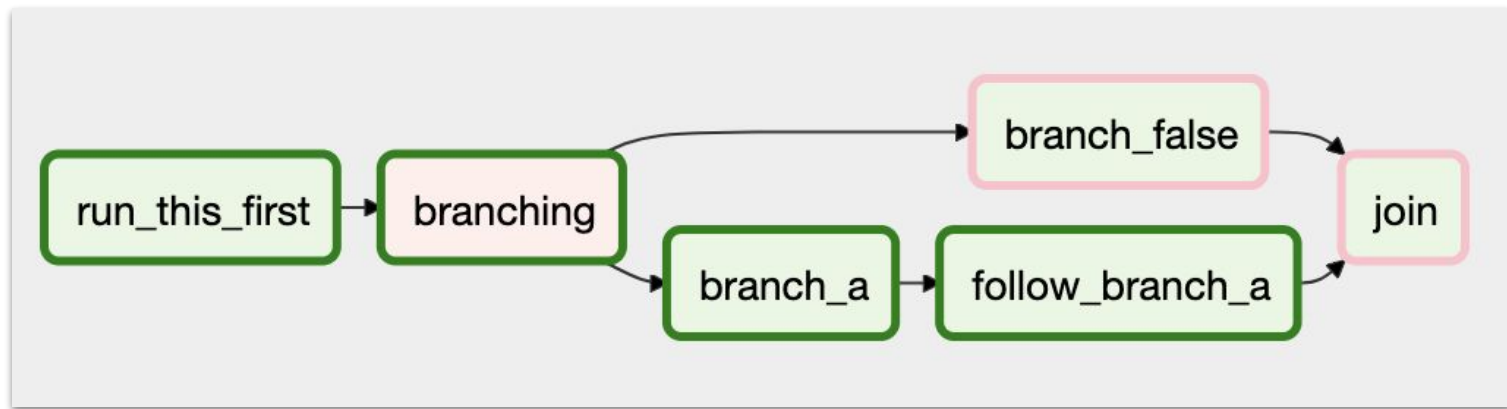
Lets you construct a data transformation pipeline using templated SQL queries



Apache Airflow



Workflow orchestration tool
... you know this already...

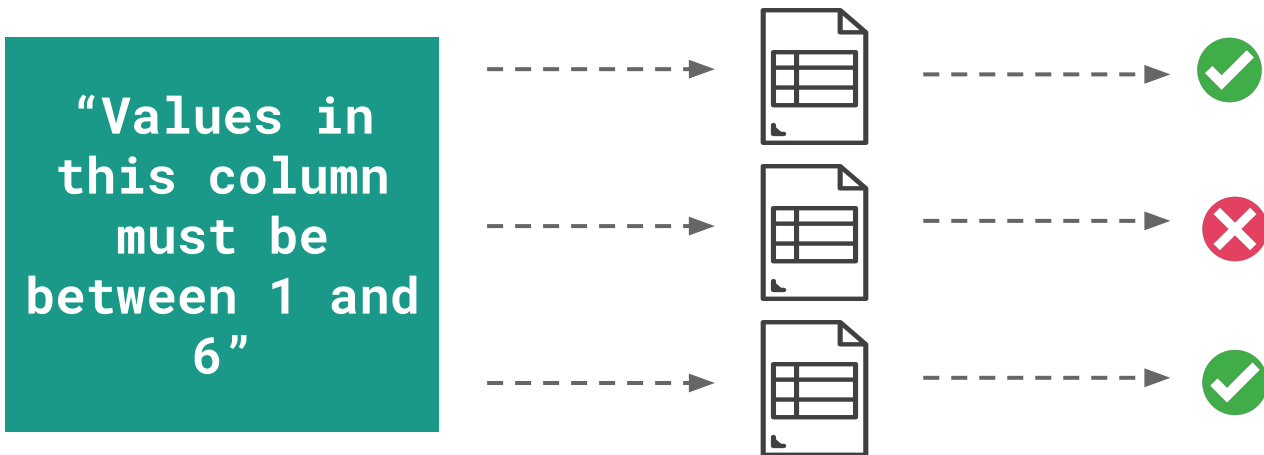


Great Expectations



Open source data validation and documentation tool

Lets you express what you **expect** from your data (ha!)



What is an Expectation?



```
expect_column_values_to_be_between(  
    column='passenger_count',  
    min_value=1,  
    max_value=6  
)
```

A statement about what we expect from our data, that can be expressed in code...

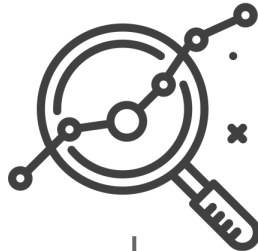
“Values in this column must be between 1 and 6”

... and translated into a human-readable format

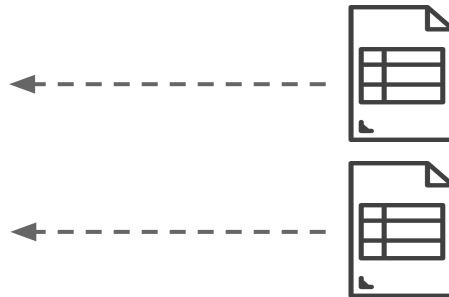
Create Expectations from profiled data...



Domain
expertise



Data
profiling



Historical
data

“Values must be between 1 and 6”

... and validate new data

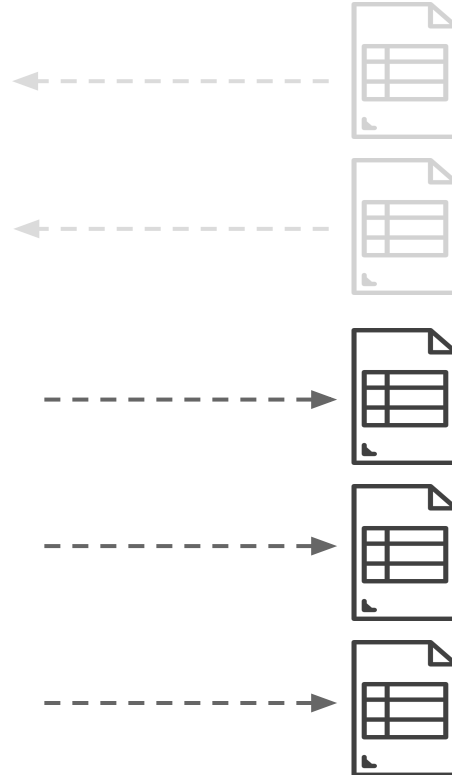


Domain
expertise



Data
profiling

“Values must be between 1 and 6”



Historical
data



Validate
future
data

Data Docs = built-in data quality reports



great_expectations

[Home](#) / [Validations](#) / [taxi.demo](#) / [yellow_tripdata_staging](#) / 2020-08-04T16:38:46.210132+00:00

Actions

Validation Filter:

Show All Failed Only

How to Edit This Suite

Show Walkthrough

Table of Contents

Overview

[dropoff_datetime](#)

[fare_amount](#)

[passenger_count](#)

[pickup_datetime](#)

[trip_distance](#)

passenger_count

Status	Expectation	Observed Value
✗	<p>values must always be between 1 and 6.</p> <p>1579 unexpected values found. ≈15.79% of 10000 total rows.</p> <div>Sampled Unexpected Values 0.0</div>	≈15.79% un

pickup_datetime

Status	Expectation	Observed Value
✓	values must never be null.	100% not null

Choose your own dAG stack pt 1: Integrating dbt and Airflow

Different approaches

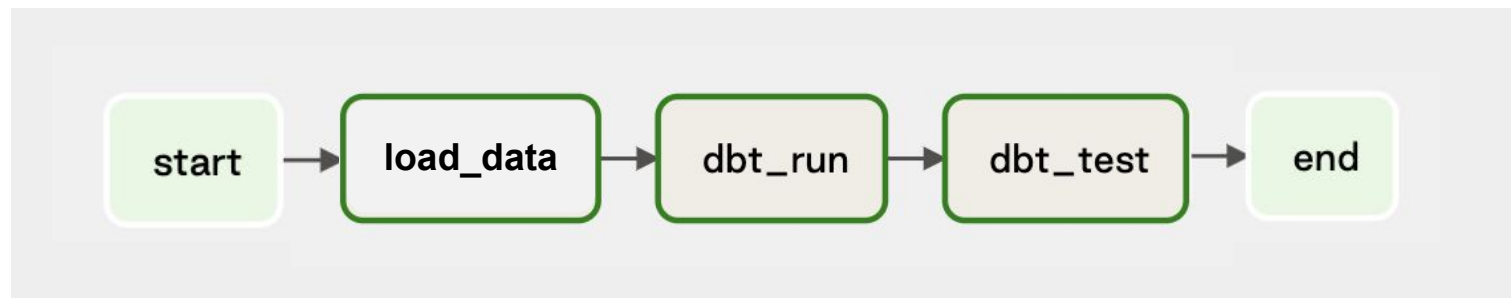
dbt DAG = 1 task

- entire dbt DAG run is triggered by single Airflow task
- straightforward approach, can use dbt operator
- dbt run is a “black box”

1 dbt model = 1 task

- maps each model to an individual Airflow task by parsing the dbt manifest
- consider added complexity and parse time per model
- fine-grained control over tasks (failure, reruns, etc)

Entire dbt DAG = 1 Airflow task



github.com/astronomer/airflow-dbt-demo

Entire dbt DAG = 1 Airflow task

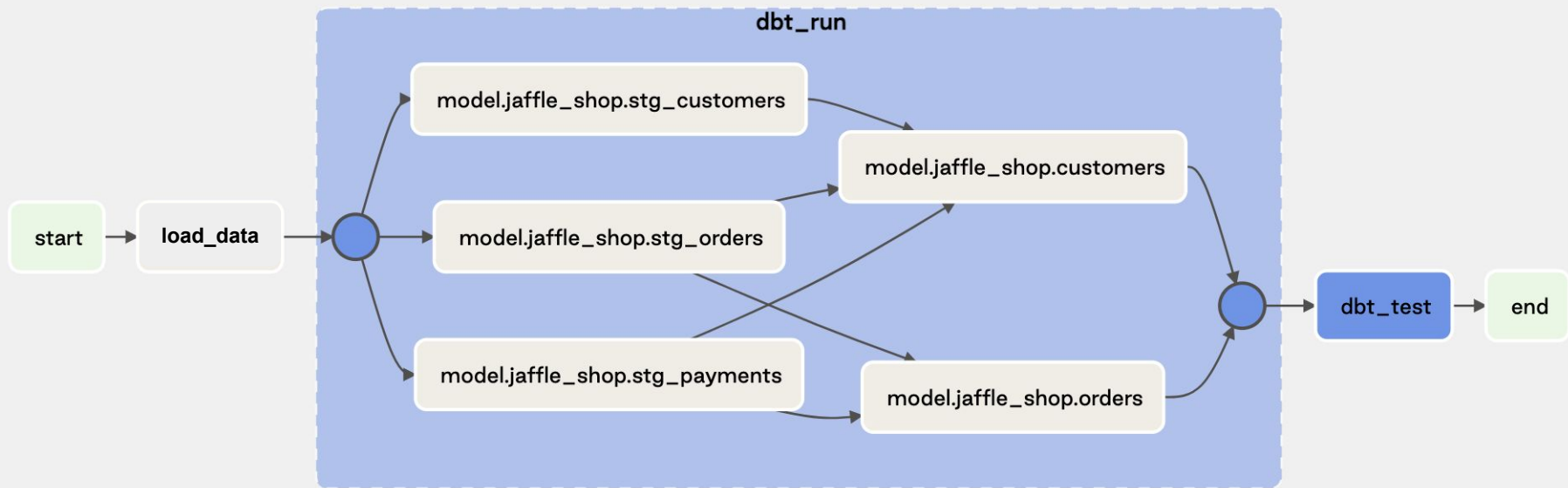
```
with dag:
    dbt_seed = BashOperator(
        task_id="dbt_seed",
        bash_command=f"dbt seed --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}"
    )

    dbt_run = BashOperator(
        task_id="dbt_run",
        bash_command=f"dbt run --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}"
    )

    dbt_test = BashOperator(
        task_id="dbt_test",
        bash_command=f"dbt test --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}"
    )

    dbt_seed >> dbt_run >> dbt_test
```

1 dbt model = 1 Airflow task



github.com/astronomer/airflow-dbt-demo

1 dbt model = 1 Airflow task

```
with dag:

    start_dummy = DummyOperator(task_id='start')
    # We're using the dbt seed command here to populate the database for the purpose of this demo
    dbt_seed = BashOperator(
        task_id='dbt_seed',
        bash_command=f'dbt {DBT_GLOBAL_CLI_FLAGS} seed --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}'
    )
    end_dummy = DummyOperator(task_id='end')

    # The parser parses out a dbt manifest.json file and dynamically creates tasks for "dbt run" and "dbt test"
    # commands for each individual model. It groups them into task groups which we can retrieve and use in the DAG.
    dag_parser = DbtDagParser(dag=dag,
                              dbt_global_cli_flags=DBT_GLOBAL_CLI_FLAGS,
                              dbt_project_dir=DBT_PROJECT_DIR,
                              dbt_profiles_dir=DBT_PROJECT_DIR,
                              dbt_target=DBT_TARGET
                              )
    dbt_run_group = dag_parser.get_dbt_run_group()
    dbt_test_group = dag_parser.get_dbt_test_group()

    start_dummy >> dbt_seed >> dbt_run_group >> dbt_test_group >> end_dummy
```

Choose your own dAG stack pt 2: Testing with dbt and Great Expectations

Let's compare...

dbt

- tests supported out of the box
- tests operate on data in database
- comes with certain built-in tests and allows writing custom tests in SQL

Great Expectations

- requires additional packages and config
- can test any type of data asset (file, database, in-memory...)
- comes with complex built-in tests & custom tests in Python

On DAG: example_ge_dbt_dag schedule: 1 day, 0:00:00

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code Trigger DAG Refresh Delete

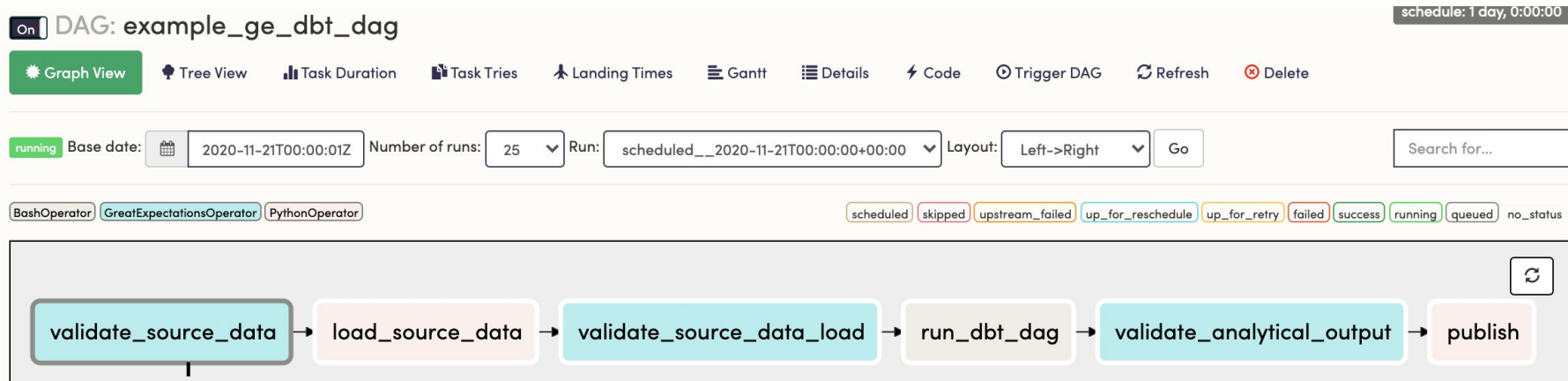
running Base date: 2020-11-21T00:00:01Z Number of runs: 25 Run: scheduled__2020-11-21T00:00:00+00:00 Layout: Left->Right Go Search for...

BashOperator GreatExpectationsOperator PythonOperator

scheduled skipped upstream_failed up_for_reschedule up_for_retry failed success running queued no_status

```
graph LR; validate_source_data --> load_source_data; load_source_data --> validate_source_data_load; validate_source_data_load --> run_dbt_dag; run_dbt_dag --> validate_analytical_output; validate_analytical_output --> publish;
```

github.com/spbail/dag-stack



Test that source data matches expected format, e.g. correct number of columns, data types, row count “similar” to last month’s, etc.

On DAG: example_ge_dbt_dag

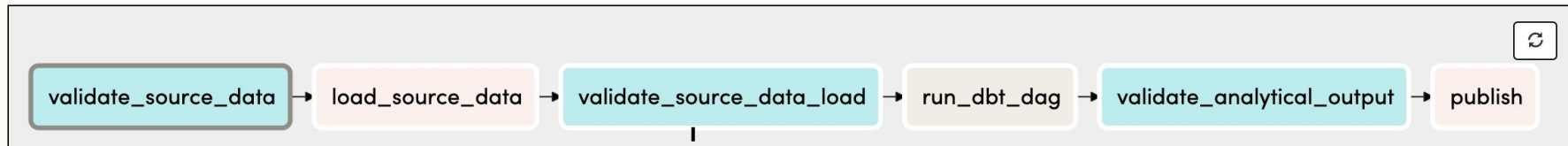
schedule: 1 day, 0:00:00

Graph View Tree View Task Duration Task Tries Landing Times Gantt Details Code Trigger DAG Refresh Delete

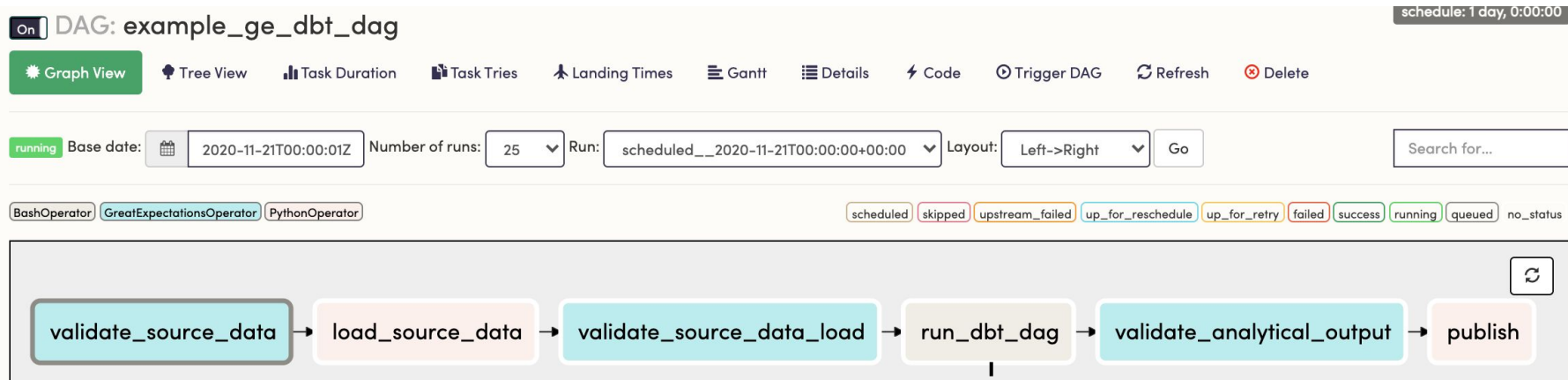
running Base date: 2020-11-21T00:00:01Z Number of runs: 25 Run: scheduled__2020-11-21T00:00:00+00:00 Layout: Left->Right Go Search for...


BashOperator GreatExpectationsOperator PythonOperator

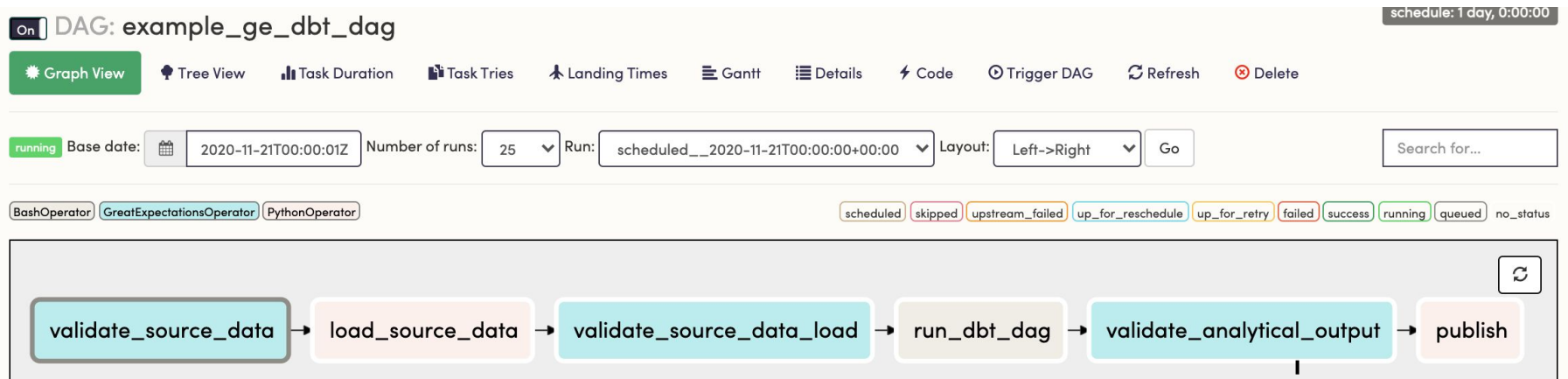
scheduled skipped upstream_failed up_for_reschedule up_for_retry failed success running queued no_status




Test that source data load was successful, e.g. no rows lost compared to source



 **dbt** Run tests during DAG *development* to check for integrity of transformations



 **dbt** Test integrity of transformations, e.g. no fan-out joins, no NULL columns, etc.



Use off-the-shelf methods for complex tests, e.g. distributions of values - and generate Data Docs

Wrap-up

- Choose your own dAG stack based on your needs
- Consider different dbt integration models and trade-offs
- Take advantage of dbt and Great Expectations for testing at different points in the pipeline
- Sample projects are both linked at github.com/spbail/dag-stack

Thank you!

Ping me @spbail or in the Airflow Slack

