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

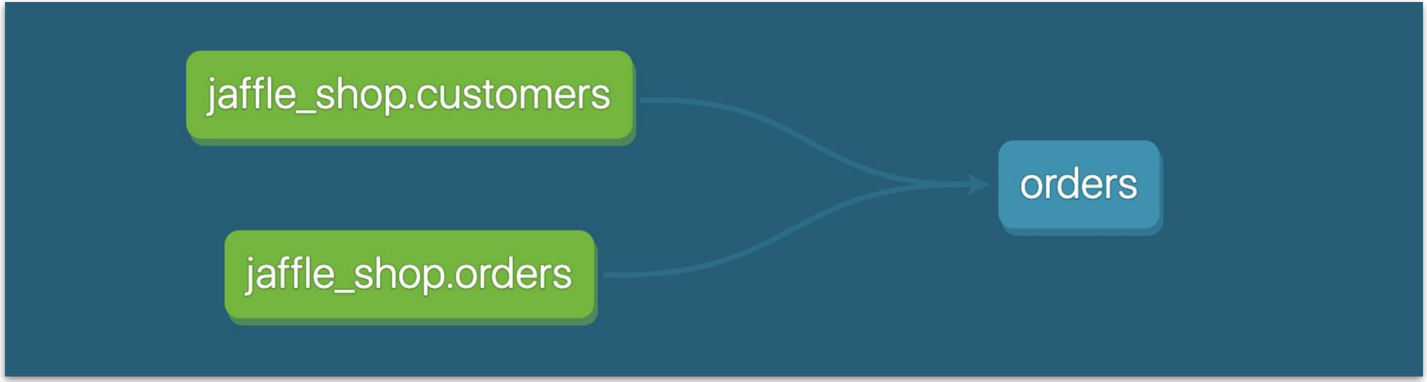
The dAG stack components

(Quick recap)

dbt

“The T in ELT”

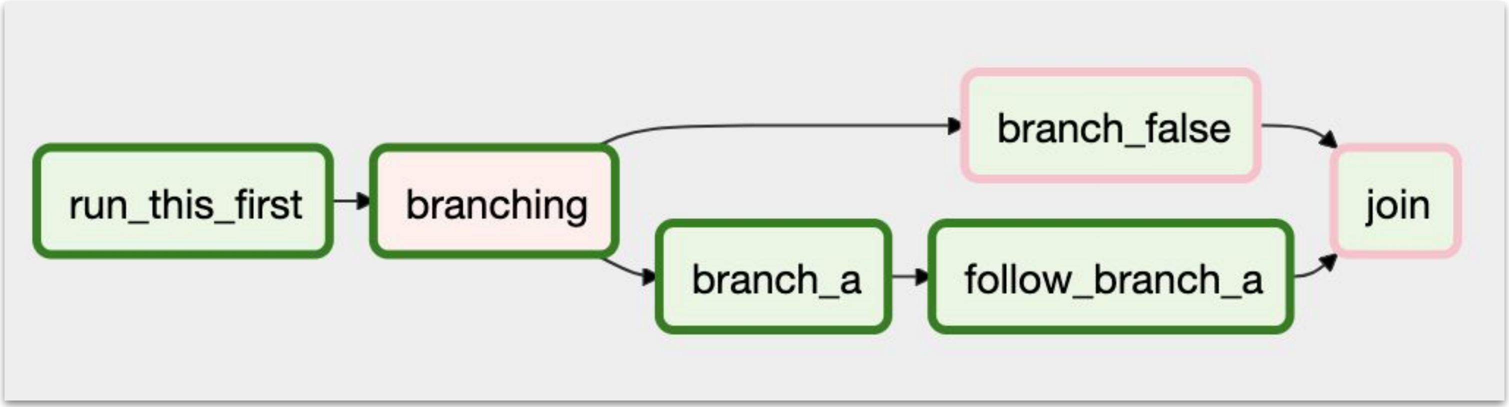
Lets you construct a data transformation pipeline using templated SQL queries



Apache Airflow

Workflow orchestration tool

… you know this already…

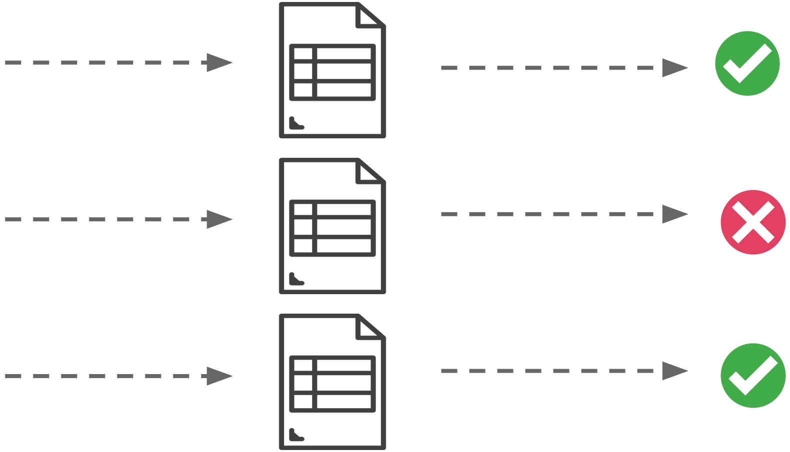


Great Expectations

Open source data validation and documentation tool

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

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



What is an Expectation?

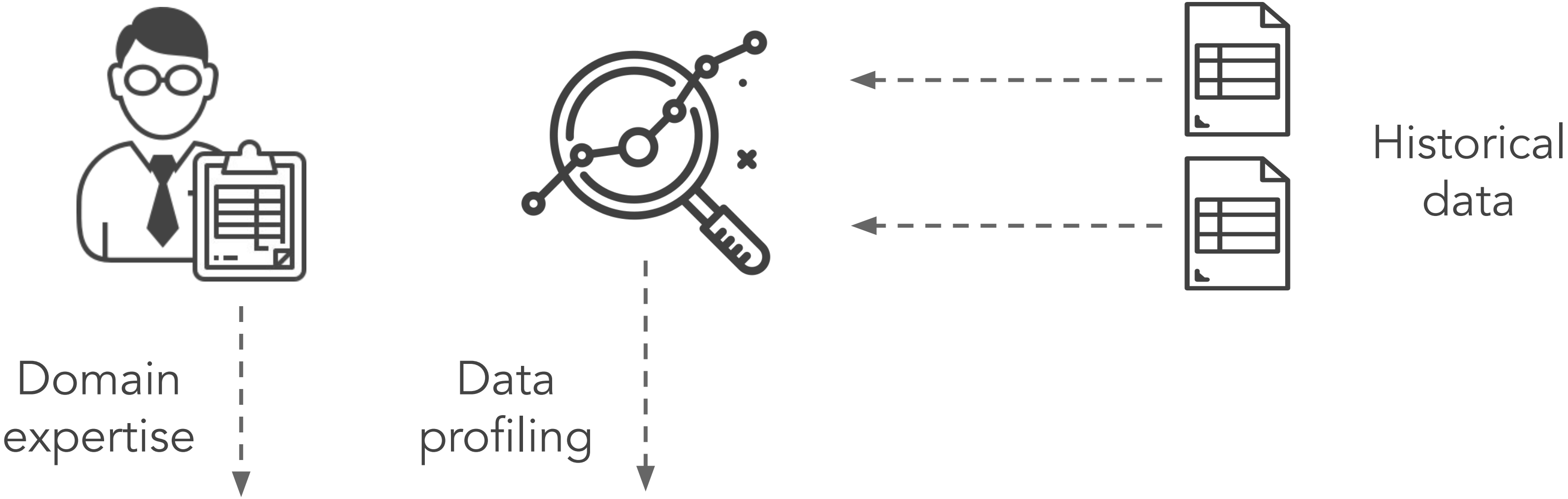
```
expect_column_values_to_be_between(
    column=’passenger_count’,
    min_value=1,
    max_value=6
)
```

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”

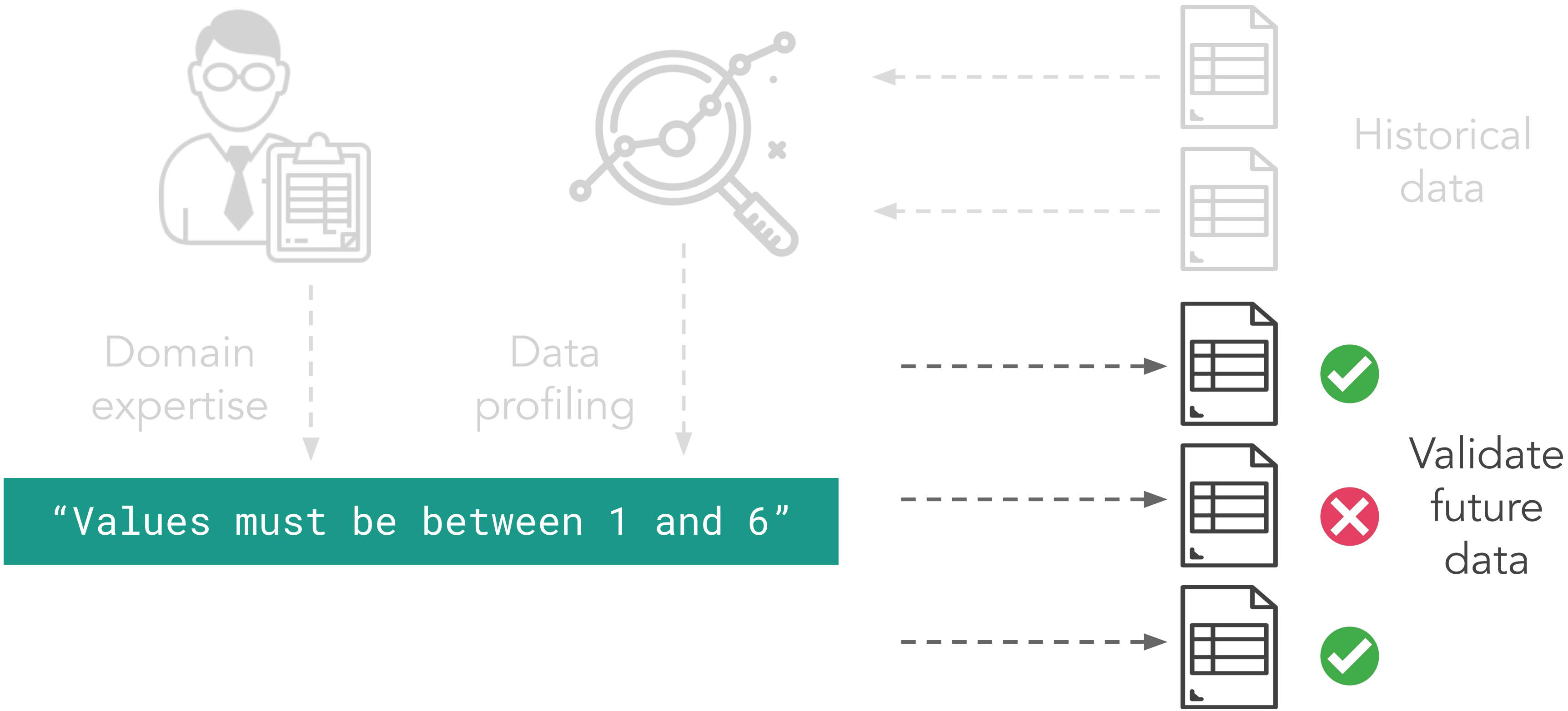
… and translated into a human-readable format

Create Expectations from profiled data...

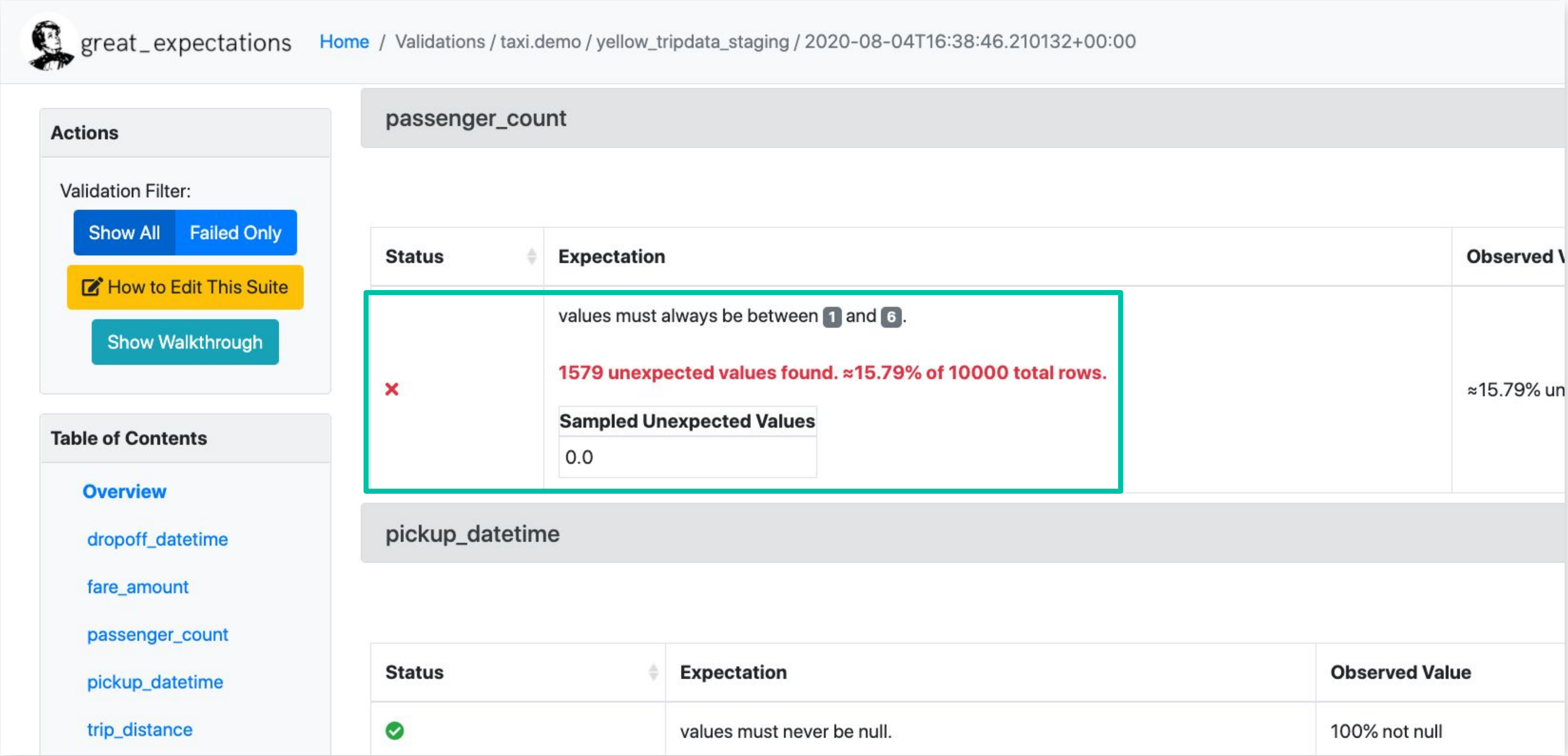


“Values must be between 1 and 6”

… and validate new data



Data Docs = built-in data quality reports



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

Different approaches

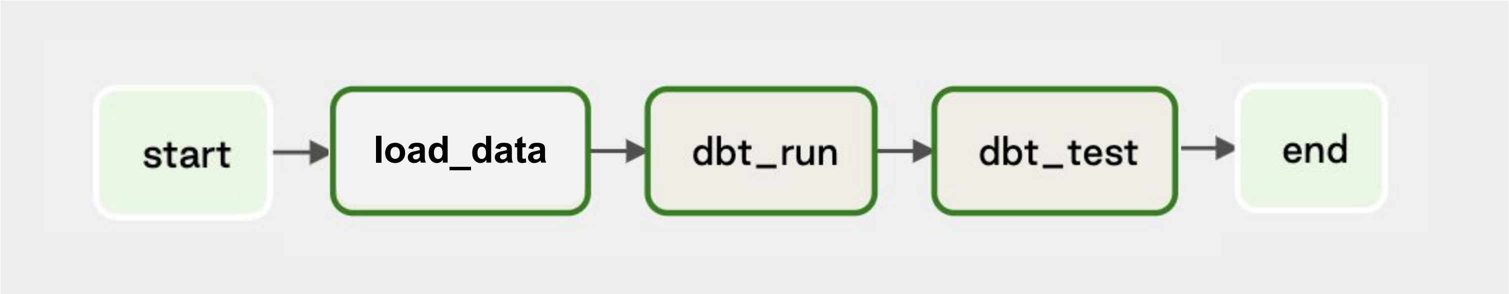
dbt DAG = 1 task

- 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

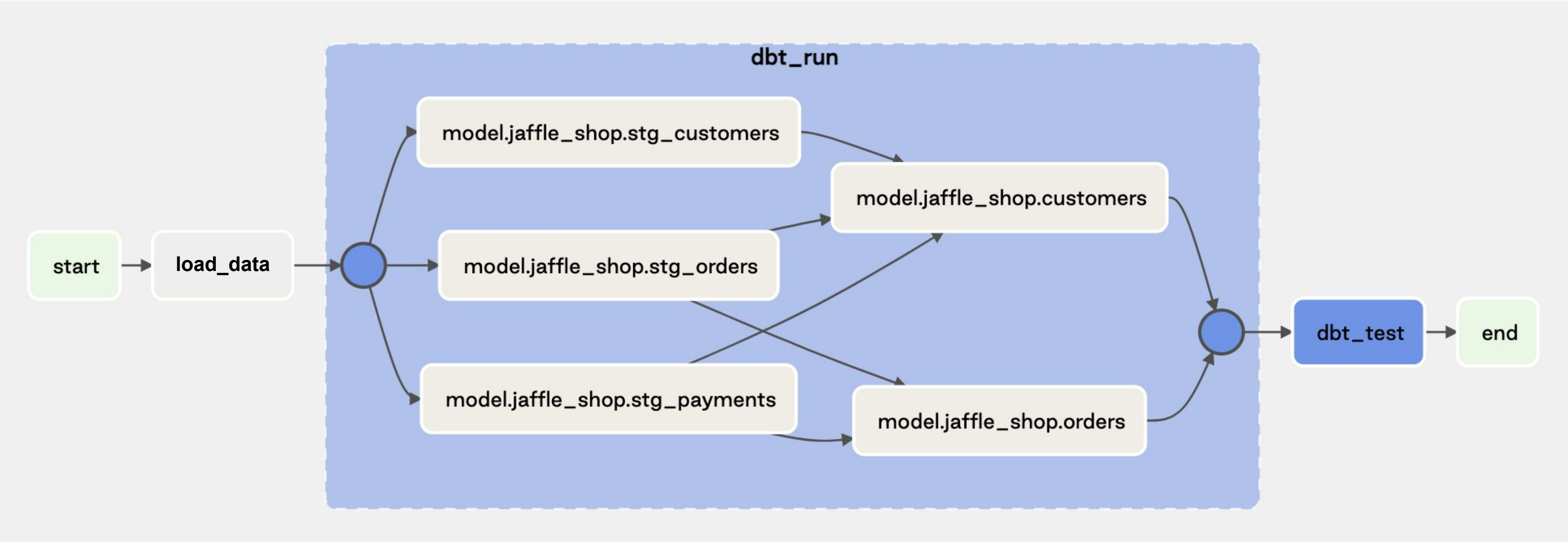
```python
with dag:
    dbt_seed = BashOperator(
        task_id="dbt_seed",
        bash_command=f"dbt seed --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}"
    )

    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

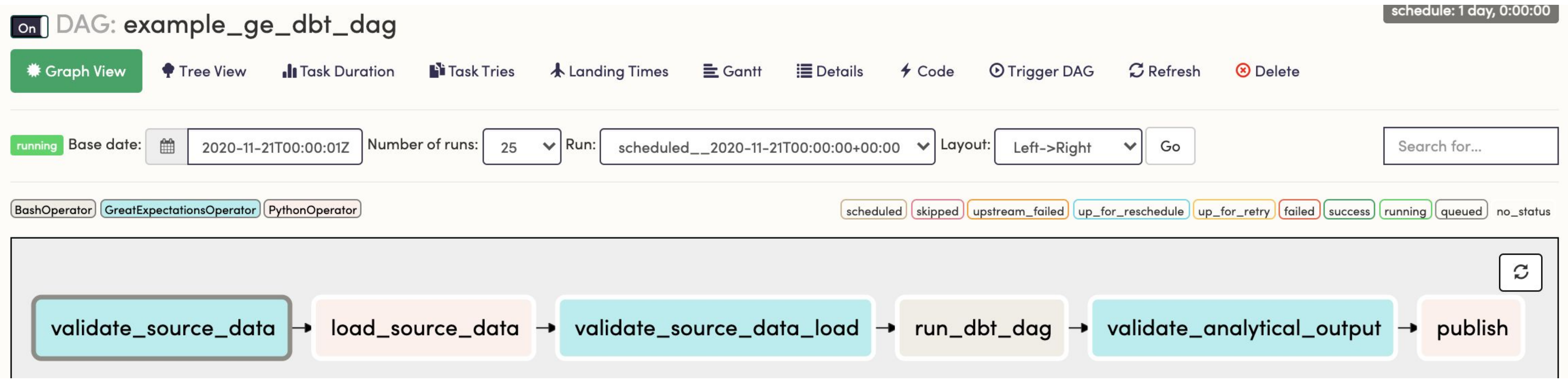
Let’s compare...

dbt

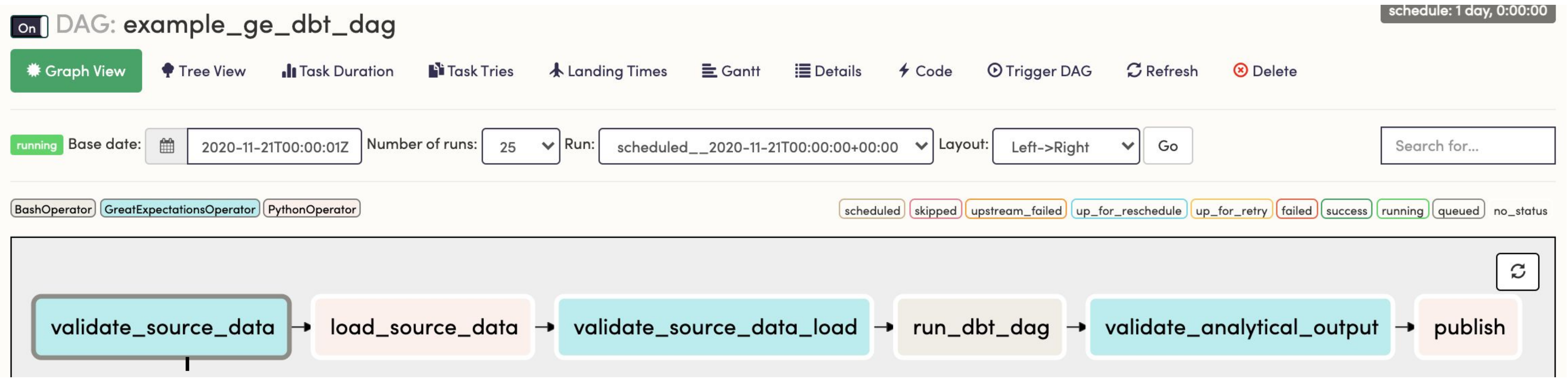
- tests supported out of the box
- tests operate on data in database
- comes with certain built-in tests and allows writing custom tests in SQL

Great Expectations

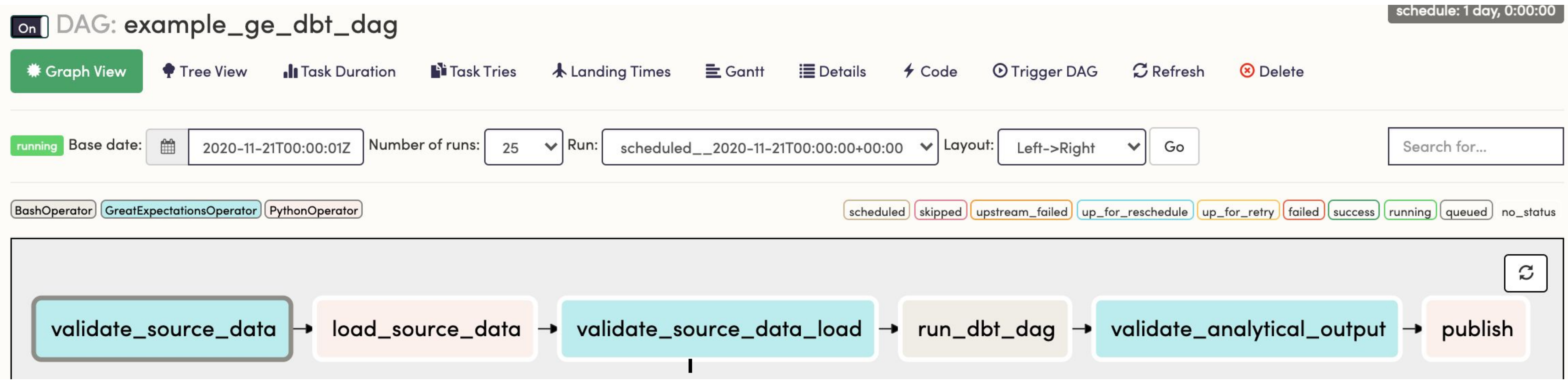
- requires additional packages and config
- can test any type of data asset (file, database, in-memory…)
- comes with complex built-in tests & custom tests in Python



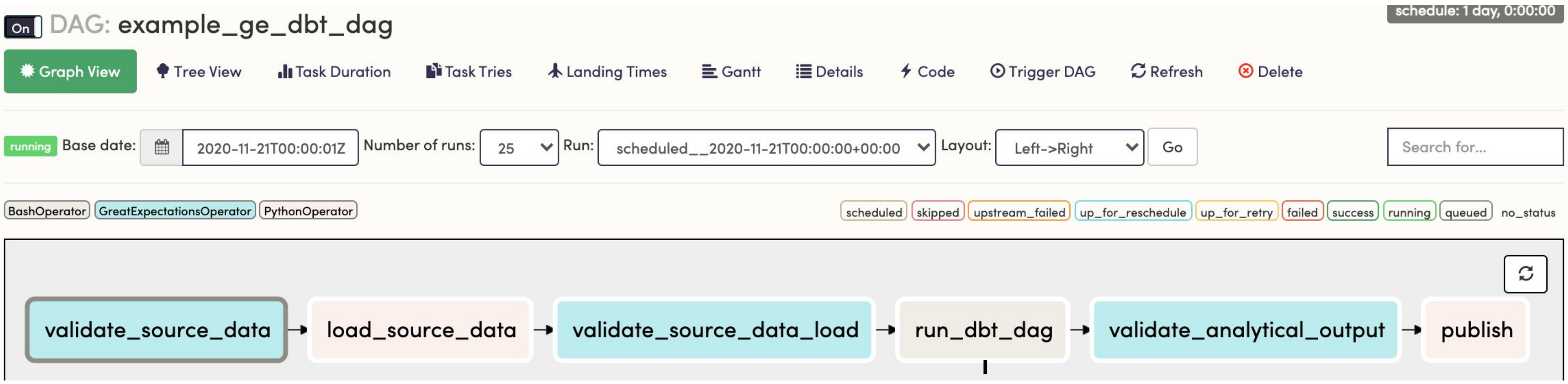
github.com/spbail/dag-stack



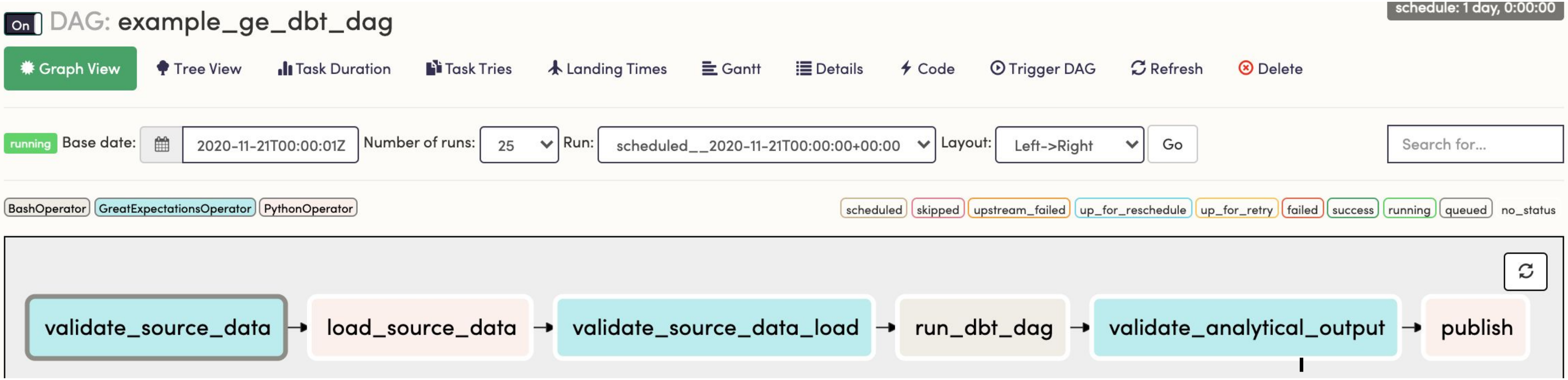
 Test that source data matches expected format, e.g. correct number of columns, data types, row count "similar" to last month's, etc.



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