Building a robust data pipeline with dbt, Airflow, and Great Expectations

PyLadies Berlin, March 4 2021

Sam Bail, Engineering Director, Partnerships
Superconductive / Great Expectations

Hi, I’m Sam!

I’m an engineering director at Superconductive, the team behind the Great Expectations open source project. I’m from Germany, but currently based in NYC. You can find me on Twitter @spbail

Q: What tools do you use for your data pipelines?

Post in the chat!

SUPERCONDUCTIVE

Welcome to the "dAG" Stack: dbt, Airflow, Great Expectations

1. What are all these tools?
2. How do they fit together?
3. Why should you test your data?

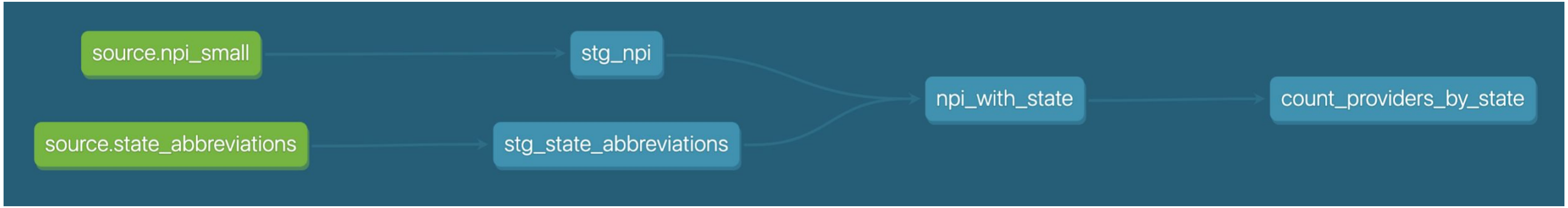
1: dbt (data build tool)

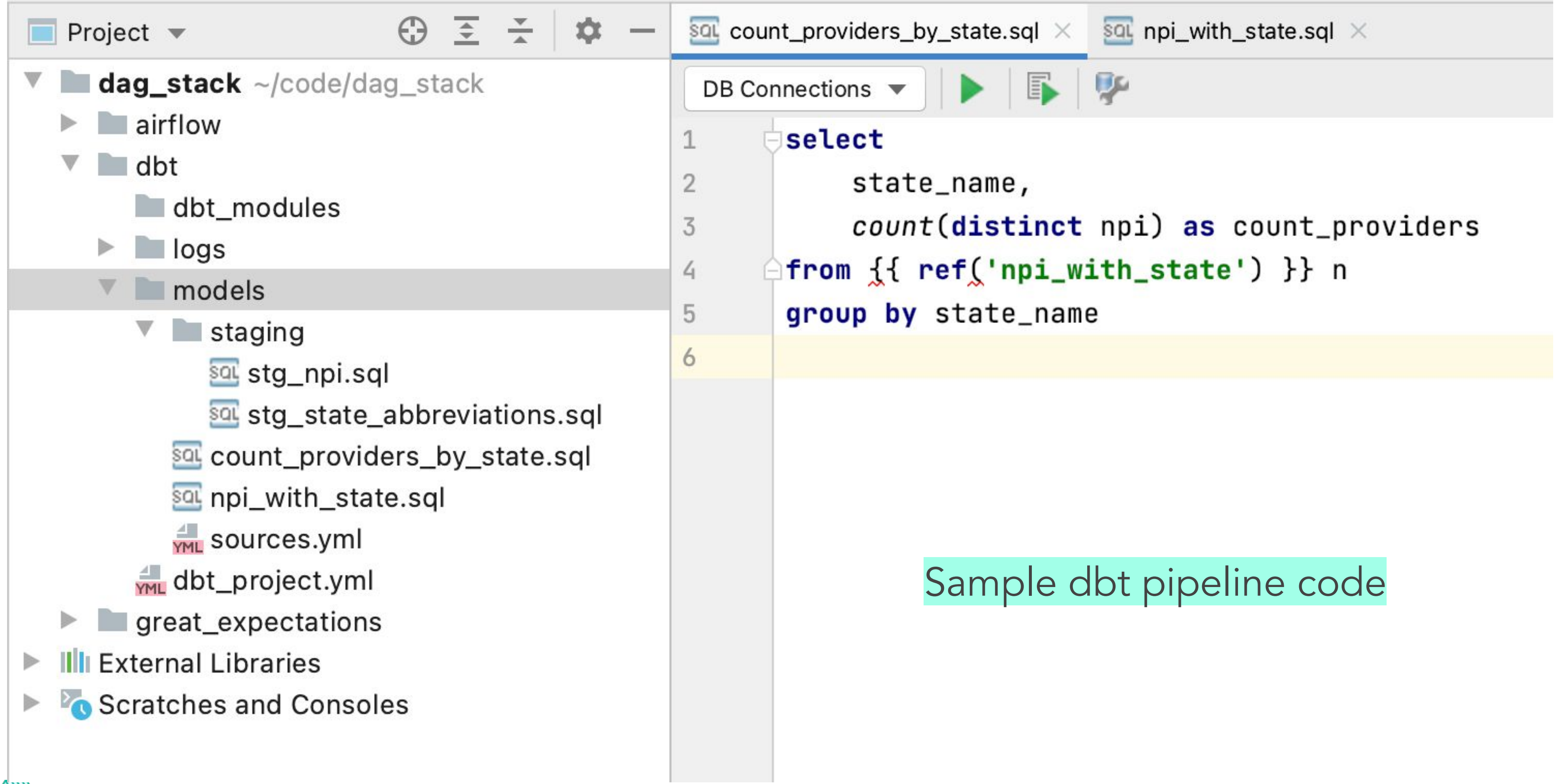
dbt

"The T in ELT"

Open source Python package

Write a data pipeline as a series of templated SQL nodes



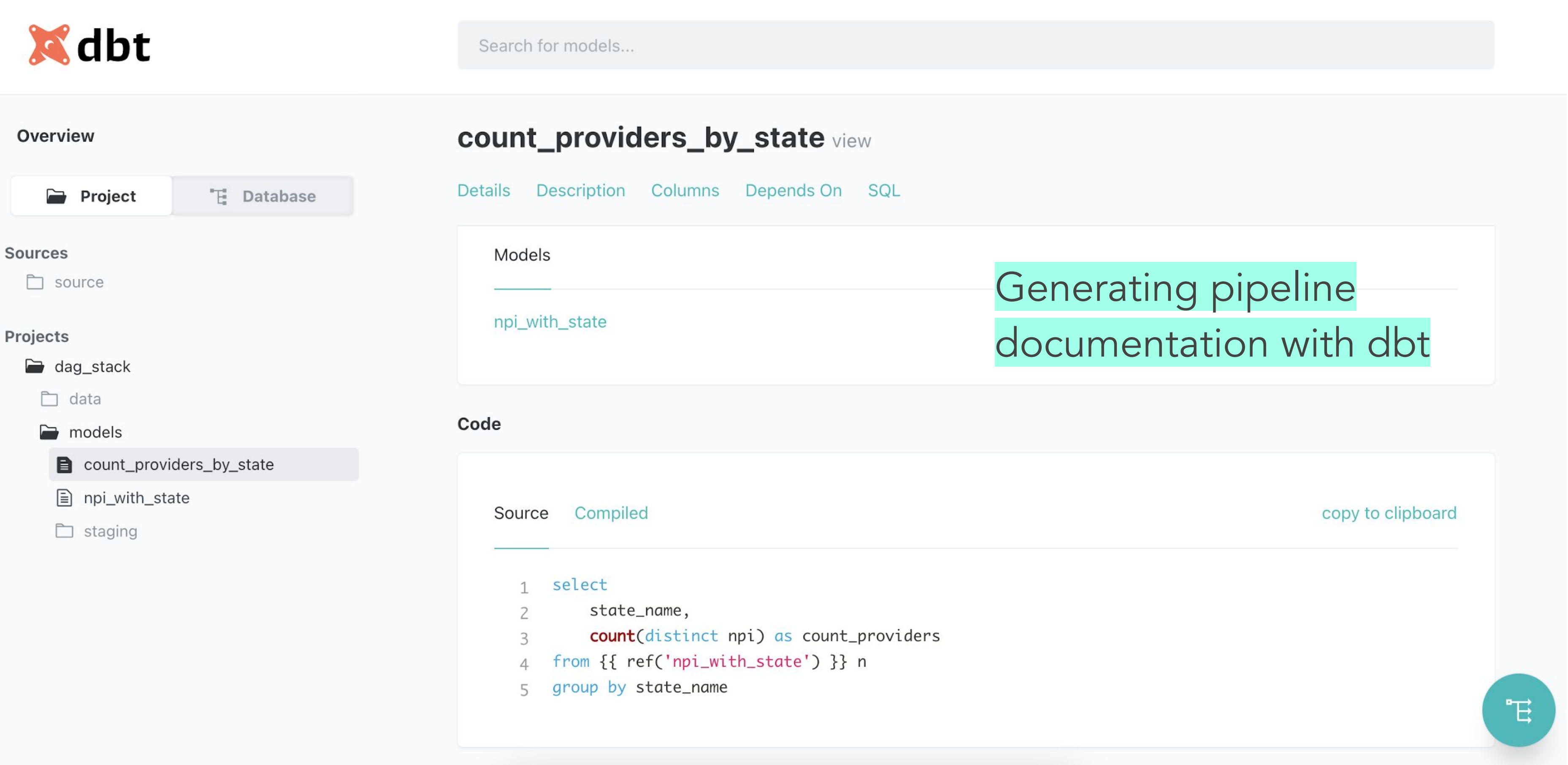


Sample dbt pipeline code

```
(base) sam@Sams-MacBook-Pro dbt % dbt run
Running with dbt=0.19.0
Found 4 models, 0 tests, 0 snapshots, 0 analyses, 138 macros, 0 operations, 2 seed files, 2 sources,
0 exposures

18:26:47 | Concurrency: 1 threads (target='dev')
18:26:47 |
18:26:47 | 1 of 4 START view model public.stg_npi.............................. [RUN]
18:26:48 | 1 of 4 OK created view model public.stg_npi......................... [CREATE VIEW in 1.00
s]
18:26:48 | 2 of 4 START view model public.stg_state_abbreviations.............. [RUN]
18:26:48 | 2 of 4 OK created view model public.stg_state_abbreviations......... [CREATE VIEW in 0.58
s]
18:26:48 | 3 of 4 START view model public.npi_with_state....................... [RUN]
18:26:49 | 3 of 4 OK created view model public.npi_with_state.................. [CREATE VIEW in 0.63
s]
18:26:49 | 4 of 4 START view model public.count_providers_by_state............. [RUN]
18:26:50 | 4 of 4 OK created view model public.count_providers_by_state........ [CREATE VIEW in 0.67
s]
18:26:50 |
18:26:50 | Finished running 4 view models in 4.70s.

Completed successfully

Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
```

Running a dbt pipeline

Generating pipeline documentation with dbt



```
(base) sam@Sams-MacBook-Pro dbt % dbt test
Running with dbt=0.19.0
Found 4 models, 2 tests, 0 snapshots, 0 analyses, 138 macros, 0 operations, 2 seed files, 2 sources,
0 exposures

19:11:48 | Concurrency: 1 threads (target='dev')
19:11:48 |
19:11:48 | 1 of 2 START test not_null_npi_with_state_npi..................... [RUN]
19:11:49 | 1 of 2 PASS not_null_npi_with_state_npi........................... [PASS in 0.52s]
19:11:49 | 2 of 2 START test unique_npi_with_state_npi....................... [RUN]
19:11:49 | 2 of 2 PASS unique_npi_with_state_npi............................. [PASS in 0.50s]
19:11:49 |
19:11:49 | Finished running 2 tests in 2.56s.

Completed successfully

Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
```

```yaml
models:
  - name: npi_with_state
    columns:
      - name: npi
        tests:
          - unique
          - not_null
```

Adding tests to a dbt pipeline

dbt does not have any built-in scheduling functionality

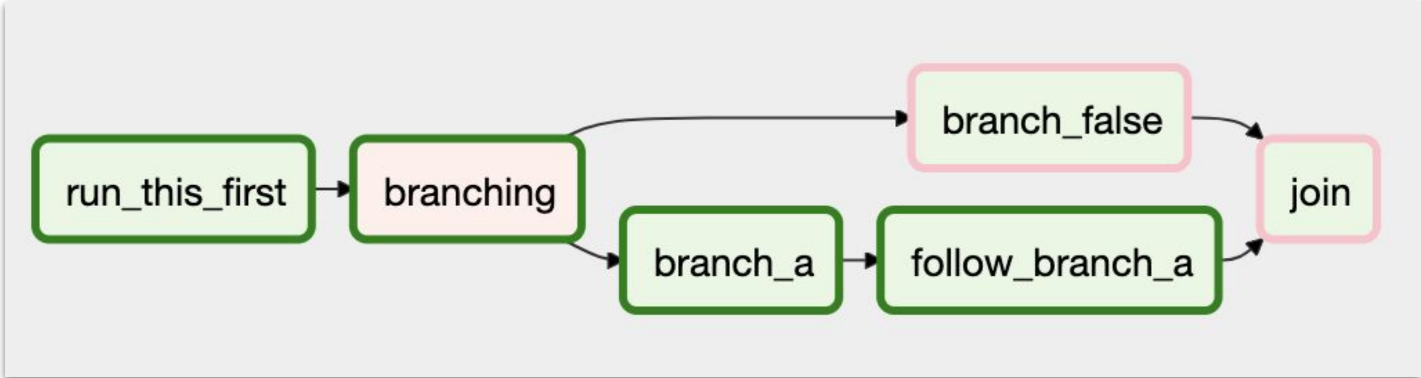
If you want to run a pipeline periodically, you'll have to schedule that with… something.

2: Apache Airflow

Workflow orchestration tool

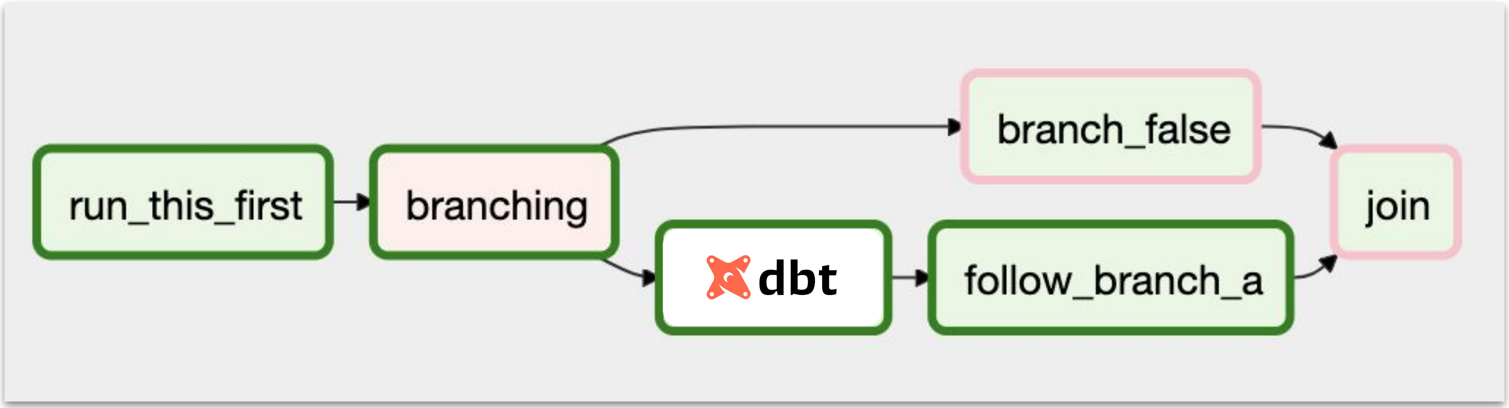
Another open source Python package :)

“Cron on steroids” - scheduling and more



Apache Airflow

We can run a dbt pipeline as a task in an Airflow DAG



```python
import airflow
from airflow import AirflowException, DAG
from airflow.operators.bash import BashOperator

dbt_dag_dir = '/Users/sam/code/dag_stack/dbt'
default_args = {
    "owner": "Airflow",
    "start_date": airflow.utils.dates.days_ago(1)
}

dag = DAG(
    dag_id='ge_tutorials_dag_with_ge',
    default_args=default_args,
    schedule_interval=None,
)

task_transform_data_in_db = BashOperator(
    task_id='run_dbt_dag',
    bash_command='dbt run --project-dir {}'.format(dbt_dag_dir),
    dag=dag
)
```

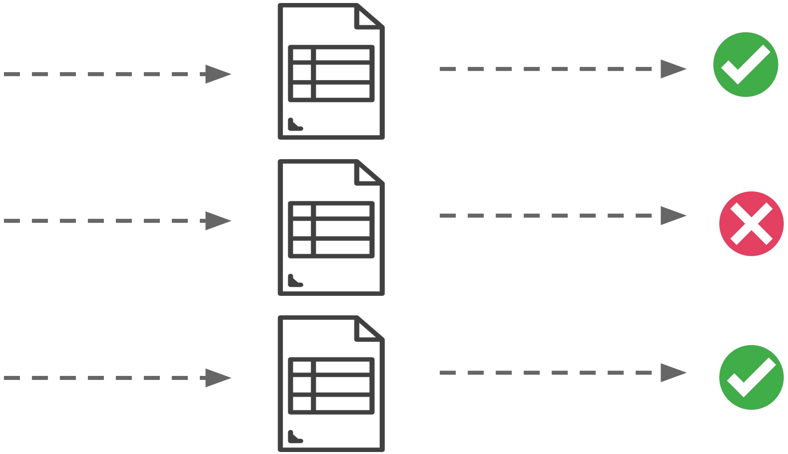
Running a dbt pipeline in an Airflow task

3: Great Expectations

Open source data validation and documentation tool

Let's 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

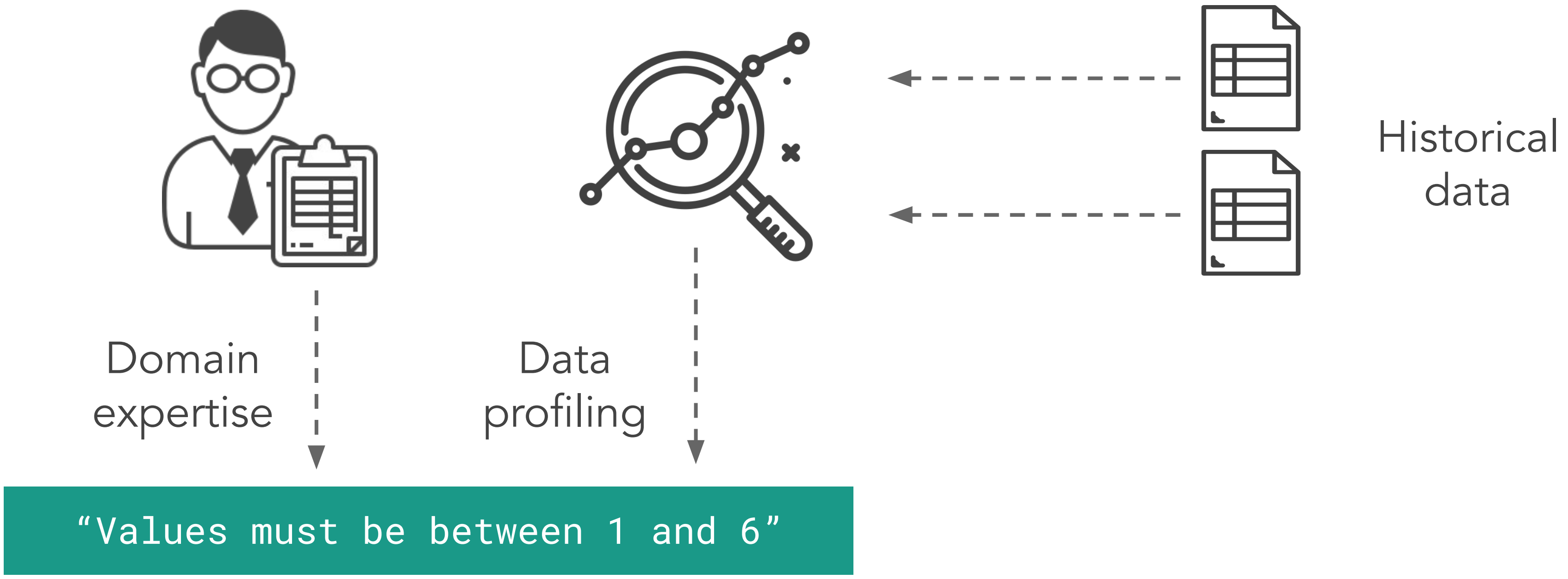

```
{
  "expectation_type": "expect_column_values_to_be_between",
      "kwargs": {
        "column": "passenger_count",
        "min_value": 1,
        "max_value": 6
      },
}
```


That is stored in JSON

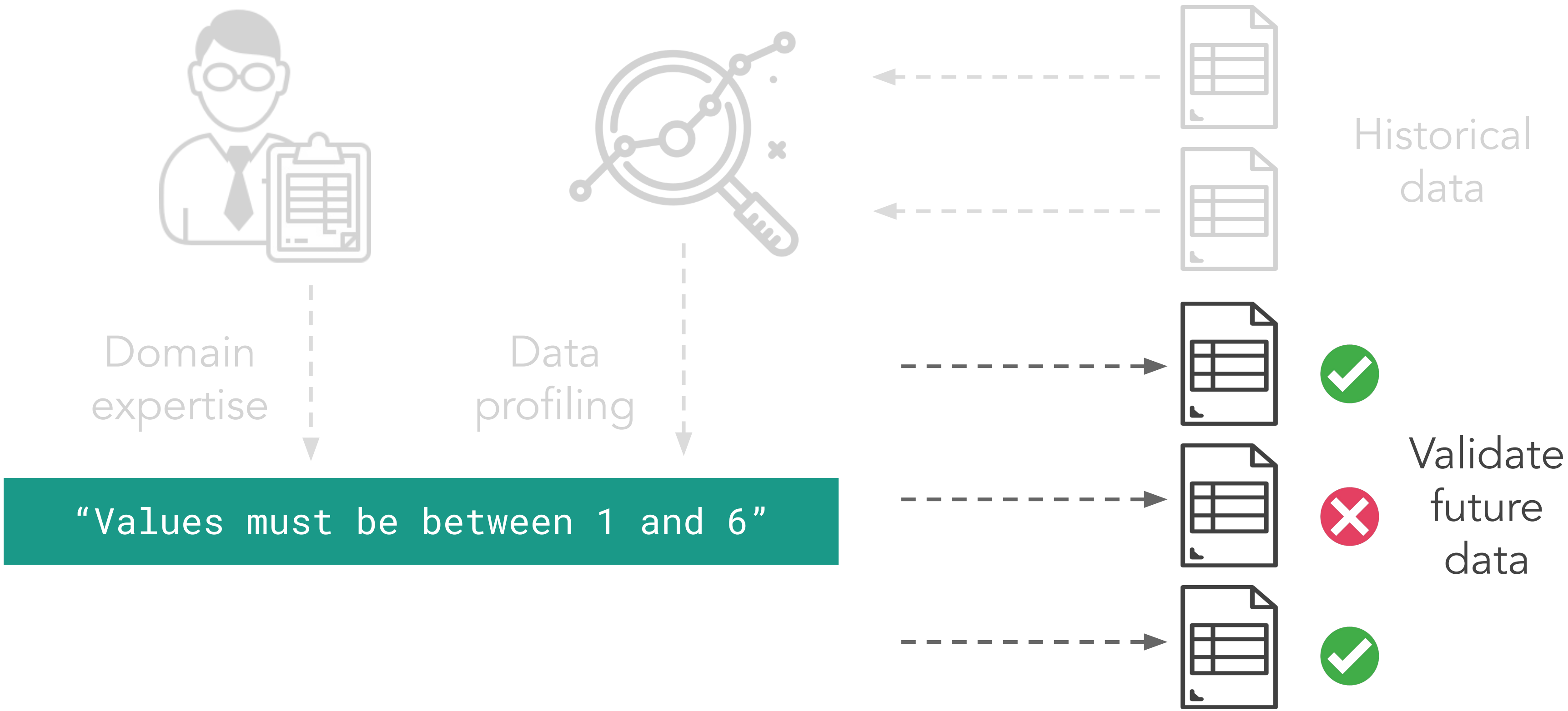
```
“Values in this column must be between 1 and 6”
```

And can be translated into a human-readable format

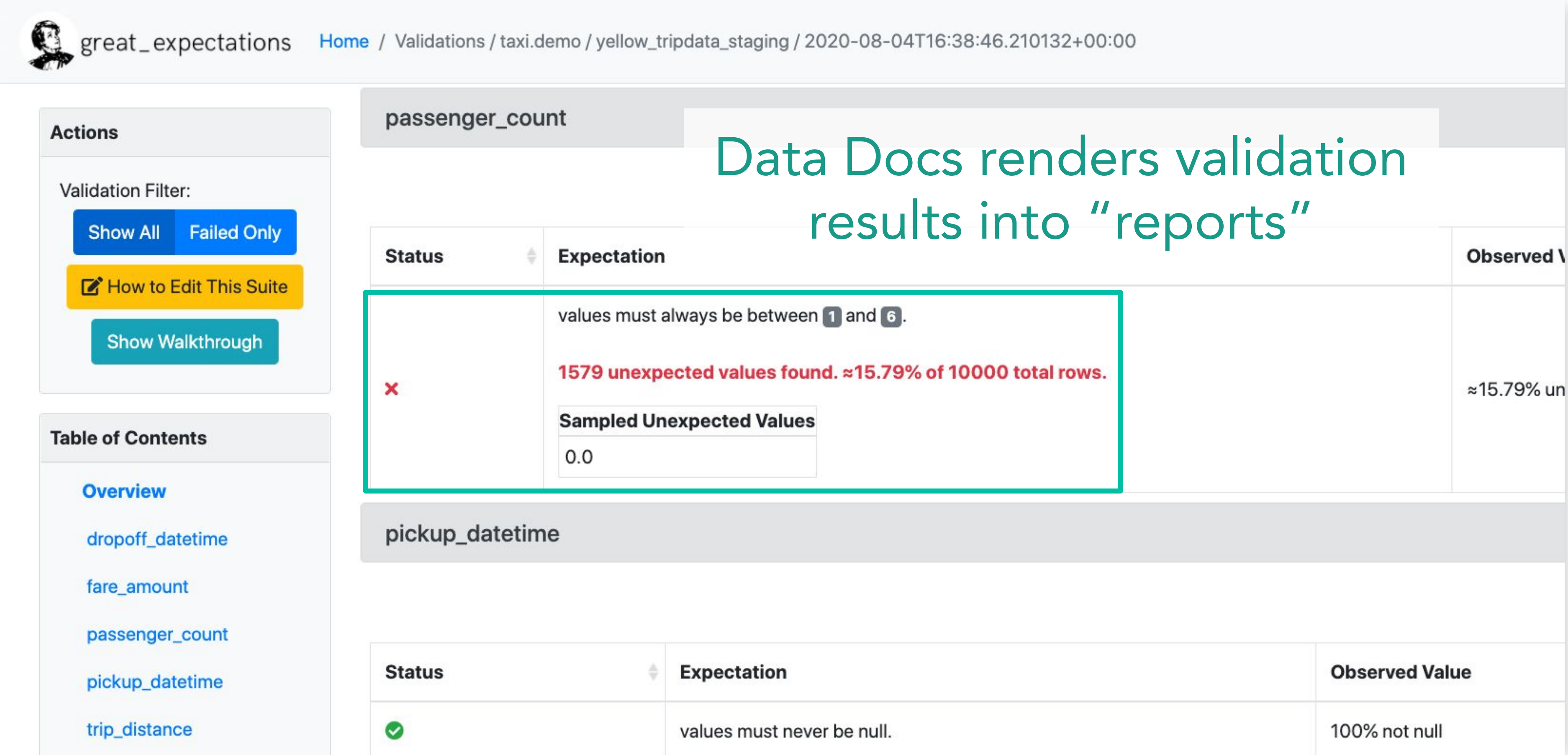
Automated profiling to “scaffold” Expectations



Validating your data



Data Docs renders validation results into “reports”



Quick live demo of Great Expectations!

SUPERCONDUCTIVE

Ok, now back to our stack: How does this all fit together?

SUPERCONDUCTIVE

1: You should test your data.

(No, really.)

Don't believe me?

"Our stakeholders would notice data issues before we did… which really eroded trust in the data and our team."

(A Great Expectations user)

"Re-running our pipelines after finding a data quality issue would incur actual costs for the compute environment."

(A Great Expectations user)

"Remember that one Thanksgiving where we worked all weekend to fix those data issues we only noticed at the last minute? Never again."

(That was me.True #datahorrorstory.)

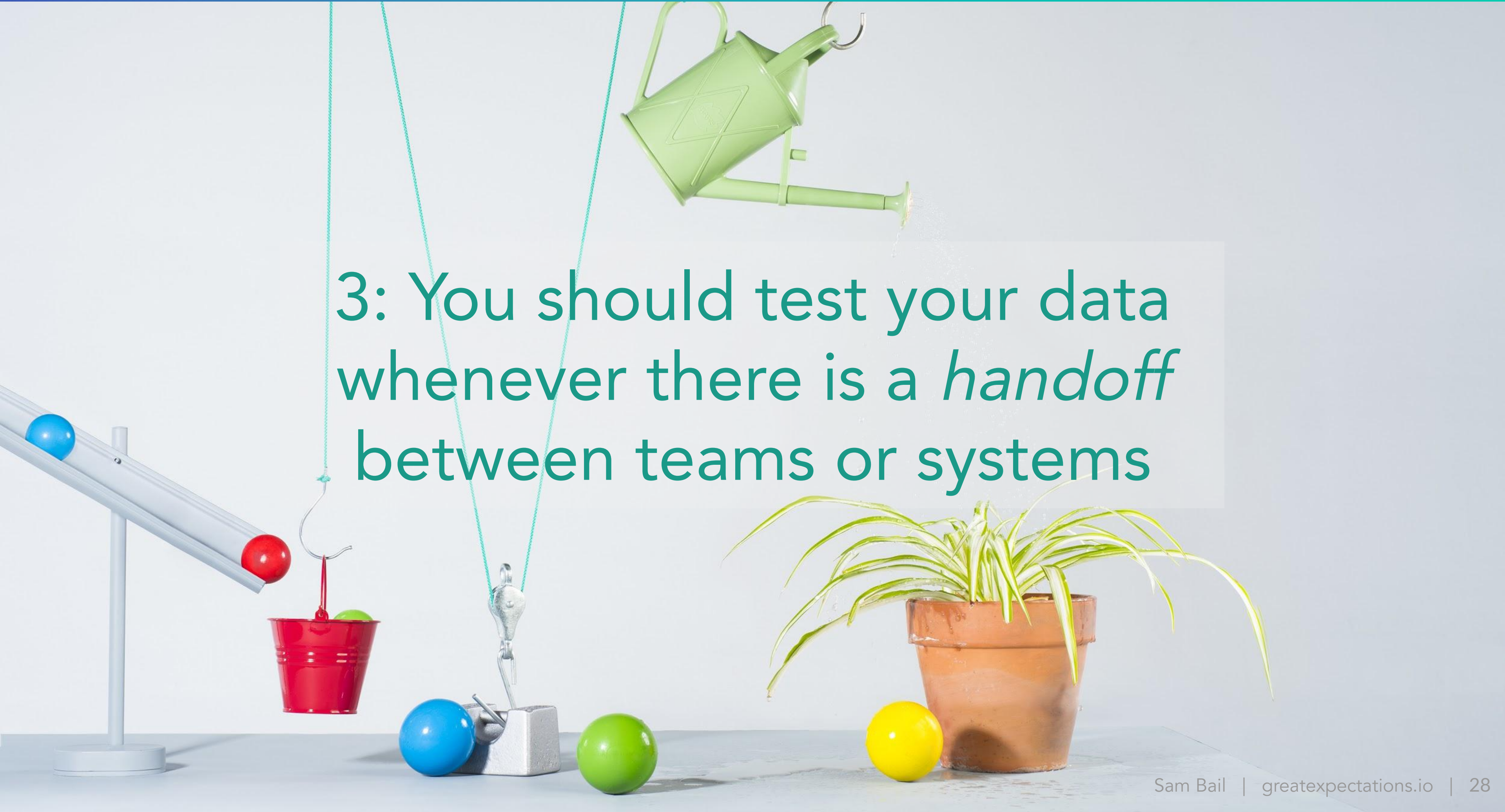
But… where do we start?

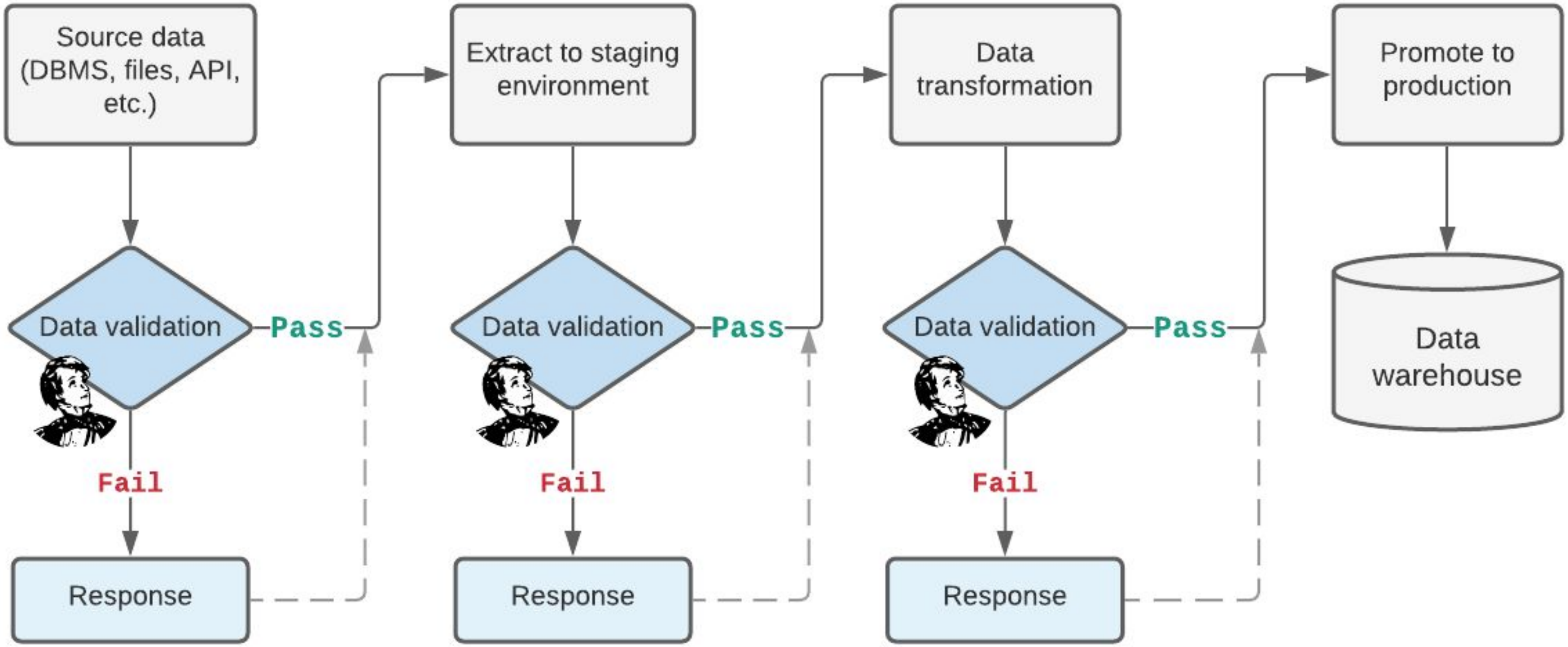
(Which tool should we use? How do we know we're testing the right thing? What do we do when tests fail? Who owns this? How do we keep them up to date? How do our stakeholders find out about the state of the data?)

2: Data testing is kinda hard.

(But I can show you how to get started...)

3: You should test your data whenever there is a *handoff* between teams or systems



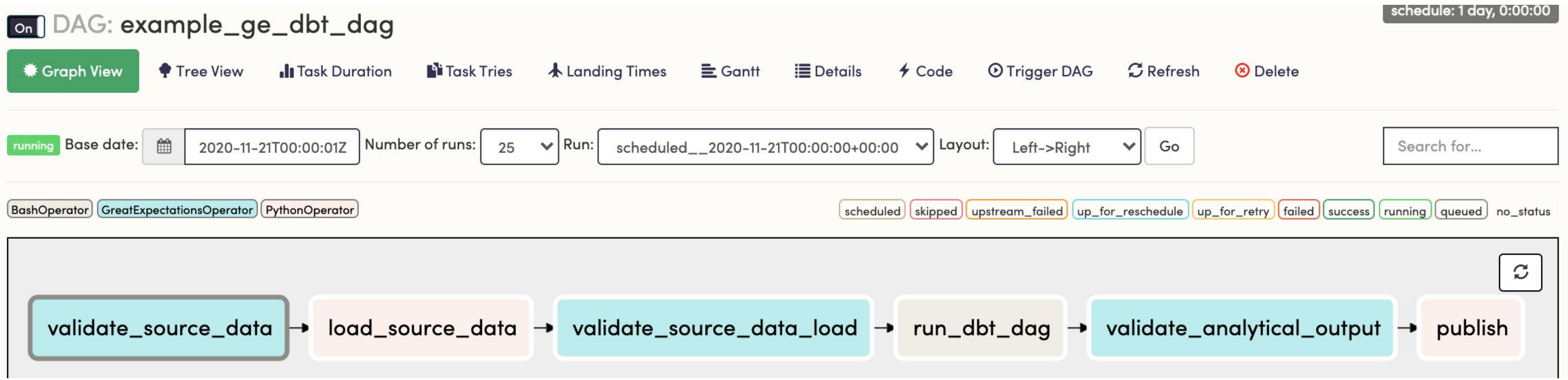

Source data (DBMS, files, API, etc.)
Extract to staging environment
Data transformation
Promote to production
Data validation
Pass
Data validation
Pass
Data validation
Pass
Data warehouse
Fail
Fail
Fail
Response
Response
Response

Ok, *now* we're back to the stack.

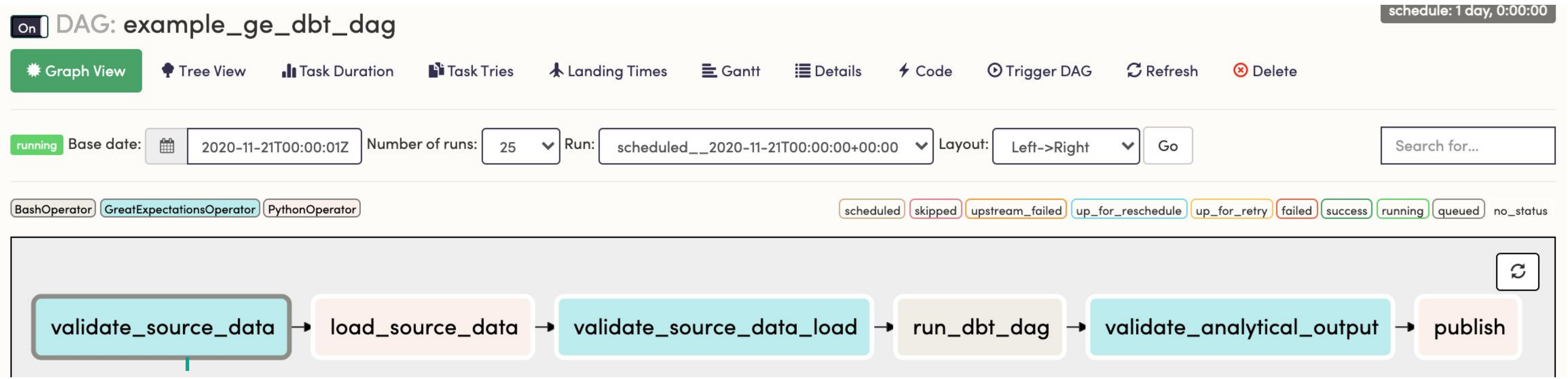
SUPERCONDUCTIVE

```
.
├── airflow
│   └── my_dag.py
├── dbt
│   ├── dbt_modules
│   ├── dbt_project.yml
│   ├── logs
│   └── models
└── great_expectations
    ├── expectations
    ├── great_expectations.yml
    ├── notebooks
    ├── plugins
    └── uncommitted
```

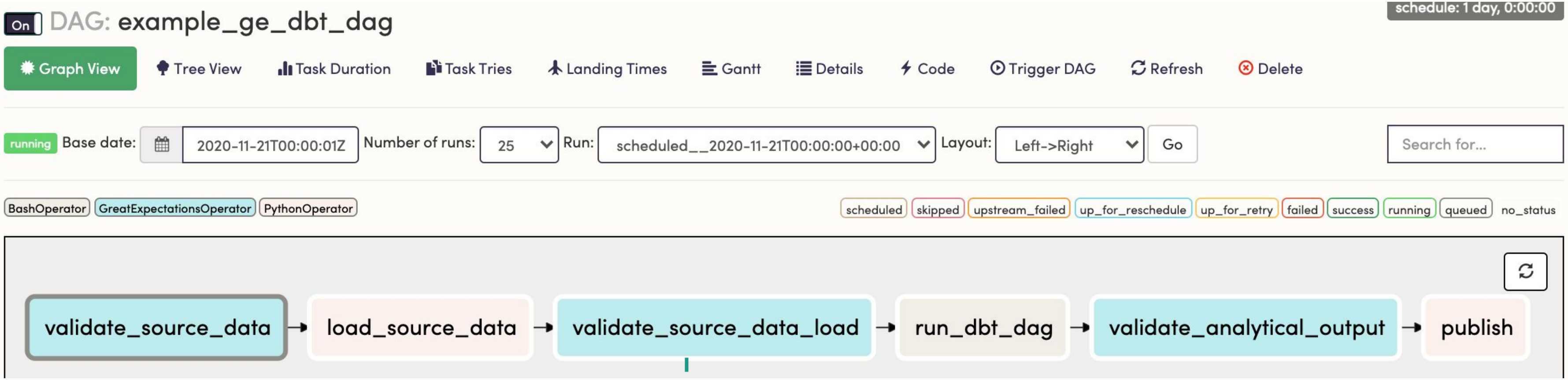
```
task_validate_source_data = GreatExpectationsOperator(...
)

task_load_files_into_db = PythonOperator(...
)

task_validate_source_data_load = GreatExpectationsOperator(...
)

task_transform_data_in_db = BashOperator(...
)

task_validate_analytical_output = GreatExpectationsOperator(...
)

task_publish = PythonOperator(...
)
```



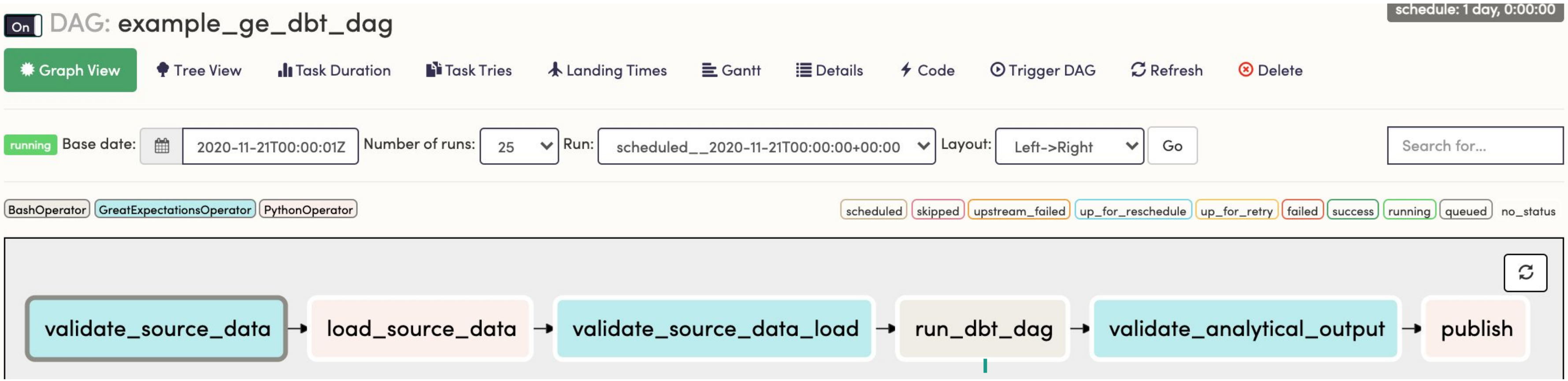
On
DAG: example_ge_dbt_dag
schedule: 1 day, 0:00:00
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Trigger DAG
Refresh
Delete
running
Base date:
2020-11-21T00:00:01Z
Number of runs:
25
Run:
scheduled__2020-11-21T00:00:00+00:00
Layout:
Left->Right
Go
Search for...
BashOperator
GreatExpectationsOperator
PythonOperator
scheduled
skipped
upstream_failed
up_for_reschedule
up_for_retry
failed
success
running
queued
no_status
validate_source_data
load_source_data
validate_source_data_load
run_dbt_dag
validate_analytical_output
publish



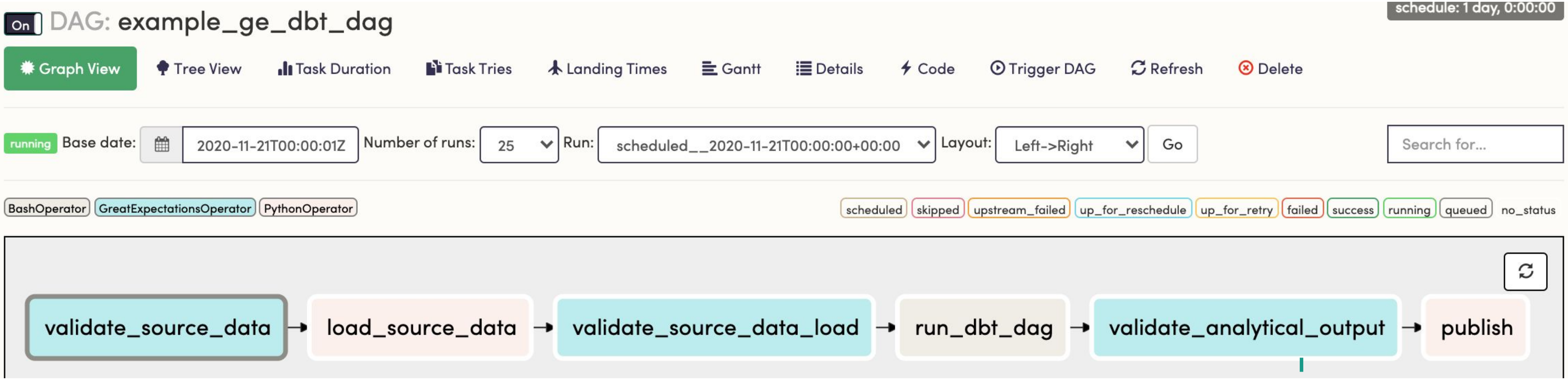
Test that source data matches expected format, e.g. correct number of columns, data types, row count "similar" to last month's, etc.



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!

Airflow, dbt, and Great Expectations = a modern data pipeline stack with testing capabilities

Test your data whenever there is a handoff between teams or systems

Use dbt or Great Expectations depending on what you test and the complexity of your tests

Thank you!