

# 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!

# 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?

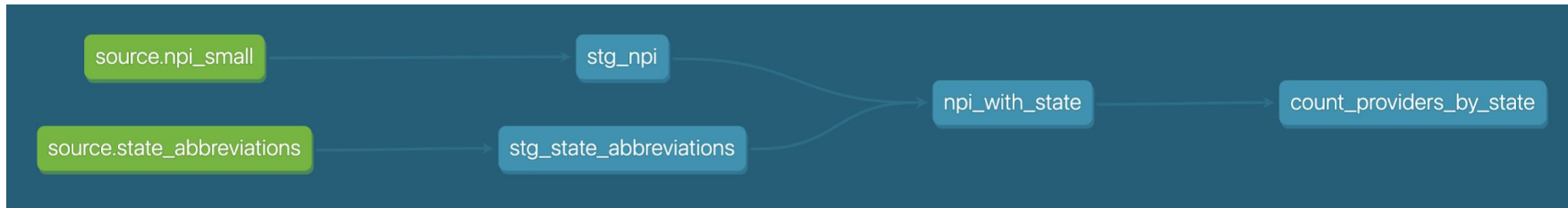


# 1: dbt (data build tool) **dbt**

“The T in ELT”

Open source Python package

Write a data pipeline as a series of templated SQL nodes



Project ▾

⊕

⊖

⊗

⚙

—

▼ dag\_stack ~/code/dag\_stack

▶ airflow

▼ dbt

▶ dbt\_modules

▶ logs

▼ models

▼ staging

sql stg\_npi.sql

sql stg\_state\_abbreviations.sql

sql count\_providers\_by\_state.sql

sql npi\_with\_state.sql

YML sources.yml

YML dbt\_project.yml

▶ great\_expectations

▶ External Libraries

▶ Scratches and Consoles

sql count\_providers\_by\_state.sql ×

sql npi\_with\_state.sql ×

DB Connections ▾

▶

📄

🔧

1

2

3

4

5

6

select

state\_name,

count(distinct npi) as count\_providers

from {{ ref('npi\_with\_state') }} n

group by state\_name

Sample dbt pipeline code

Sam Bail | greatexpectations.io | 6

```
(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

Running a dbt pipeline

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



Search for models...

## Overview

Project

Database

## Sources

source

## Projects

dag\_stack

data

models

count\_providers\_by\_state

npi\_with\_state

staging

## count\_providers\_by\_state view

Details Description Columns Depends On SQL

Models

npi\_with\_state

Generating pipeline  
documentation with dbt

## Code

Source Compiled

copy to clipboard

```
1 select
2     state_name,
3     count(distinct npi) as count_providers
4 from {{ ref('npi_with_state') }} n
5 group by state_name
```





```
(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
```

## Adding tests to a dbt pipeline

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

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.



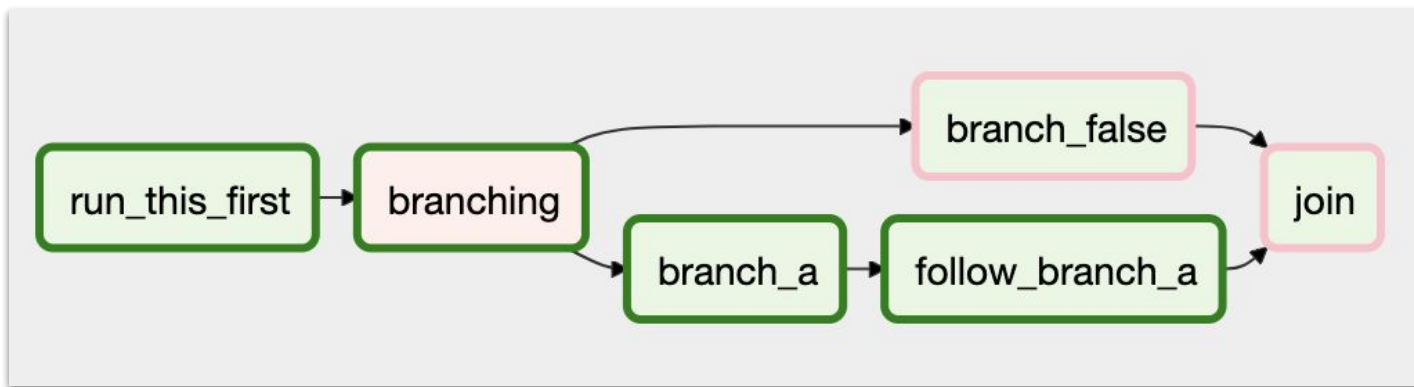
## 2: Apache Airflow



Workflow orchestration tool

Another open source Python package :)

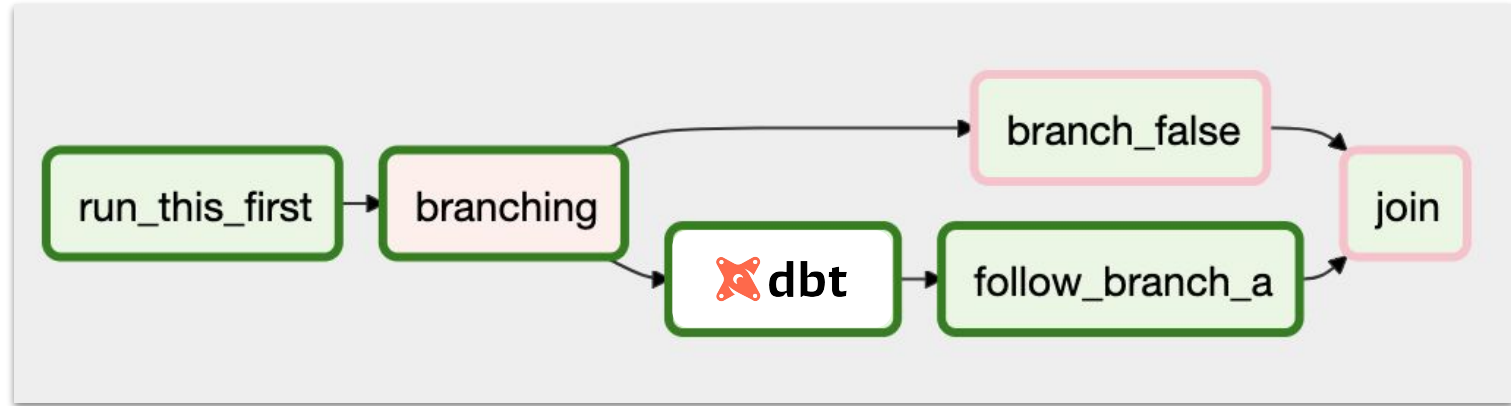
"Cron on steroids" - scheduling and more



# Apache Airflow



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



```
1 import airflow
2 from airflow import AirflowException, DAG
3 from airflow.operators.bash import BashOperator
4
5 dbt_dag_dir = '/Users/sam/code/dag_stack/dbt'
6 default_args = {
7     "owner": "Airflow",
8     "start_date": airflow.utils.dates.days_ago(1)
9 }
10
11 dag = DAG(
12     dag_id='ge_tutorials_dag_with_ge',
13     default_args=default_args,
14     schedule_interval=None,
15 )
16
17 task_transform_data_in_db = BashOperator(
18     task_id='run_dbt_dag',
19     bash_command='dbt run --project-dir {}'.format(dbt_dag_dir),
20     dag=dag
21 )
```

Running a dbt pipeline in  
an Airflow task

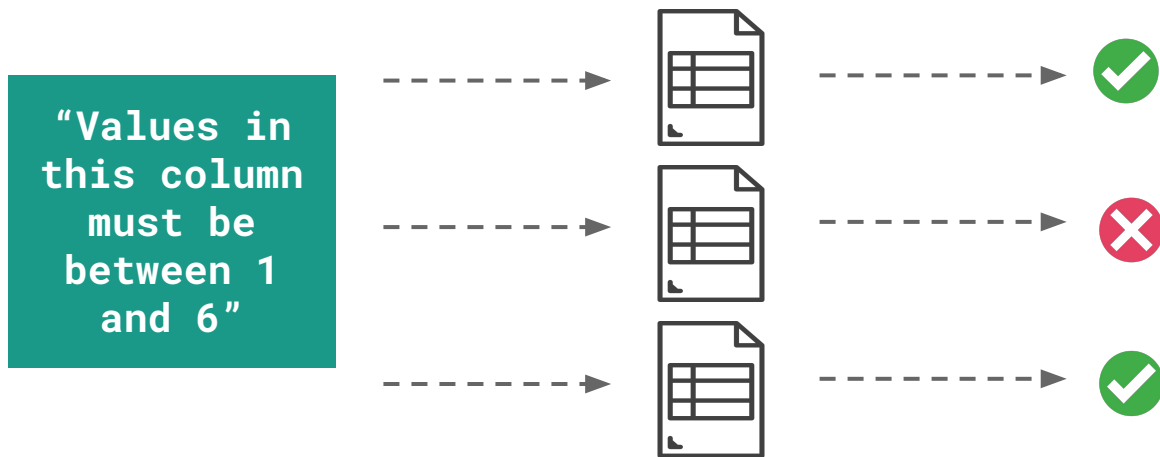


### 3: Great Expectations



Open source data validation and documentation tool

Let's you express what you *\*expect\** from your data (ha!)



# 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

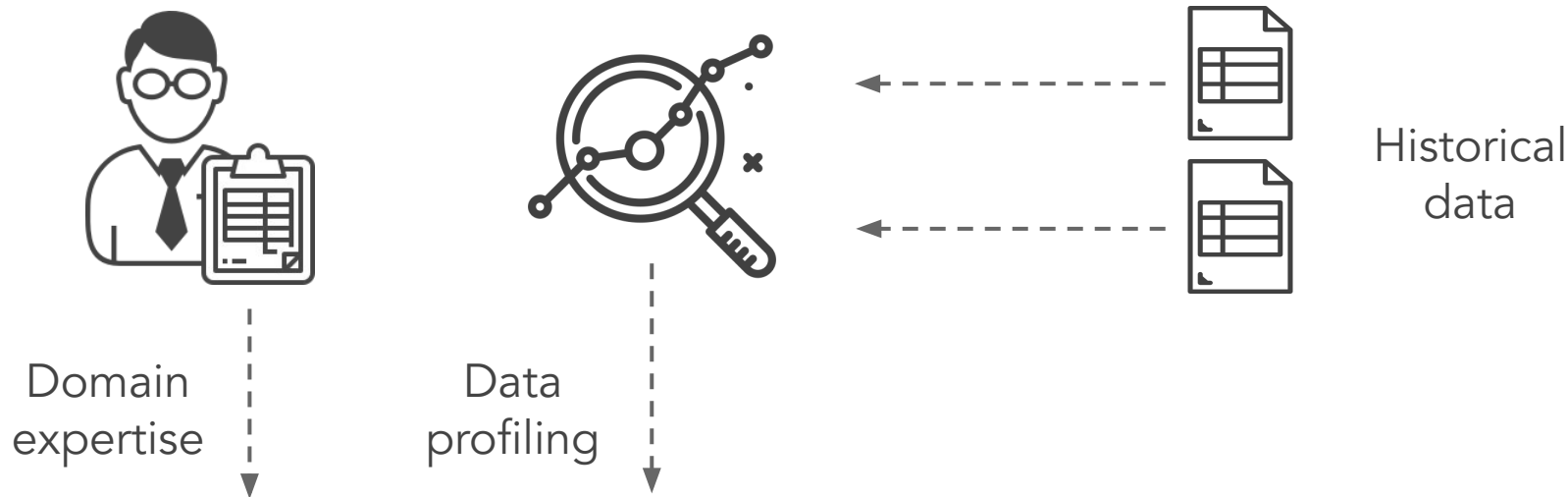
```
{  
  "expectation_type": "expect_column_values_to_be_between",  
  "kwargs": {  
    "column": "passenger_count",  
    "min_value": 1,  
    "max_value": 6  
  },  
}
```

That is stored in JSON

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

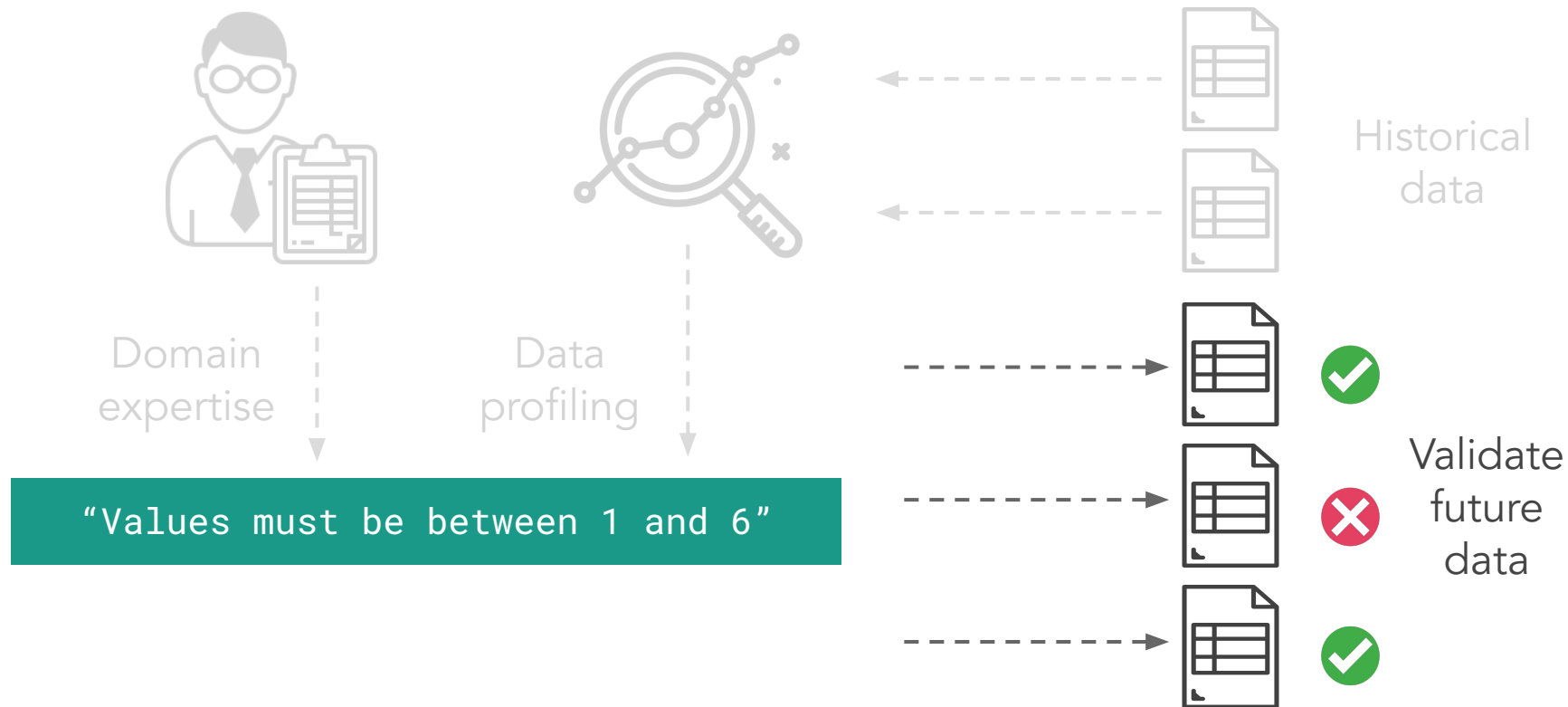
And can be translated into a human-readable format

# Automated profiling to “scaffold” Expectations





# Validating your data





## 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

# Data Docs renders validation results into "reports"

Results into Reports

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><b>Sampled Unexpected Values</b></div> <div>0.0</div>	≈15.79% un

## pickup\_datetime

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



## Quick live demo of Great Expectations!



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



# 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.”*

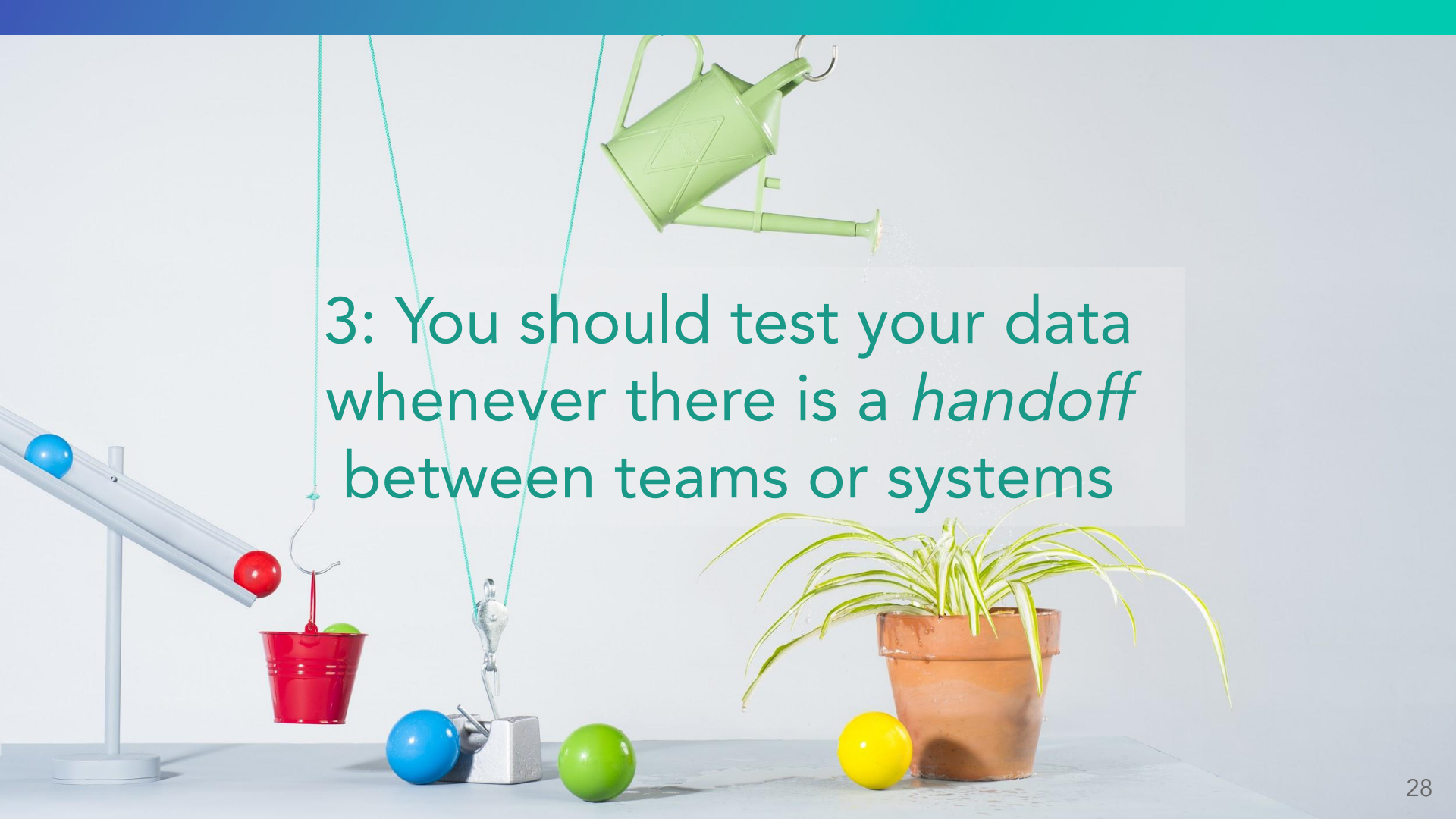
*(That was me. True #datahorrorstory.)*

# But... where do we start?

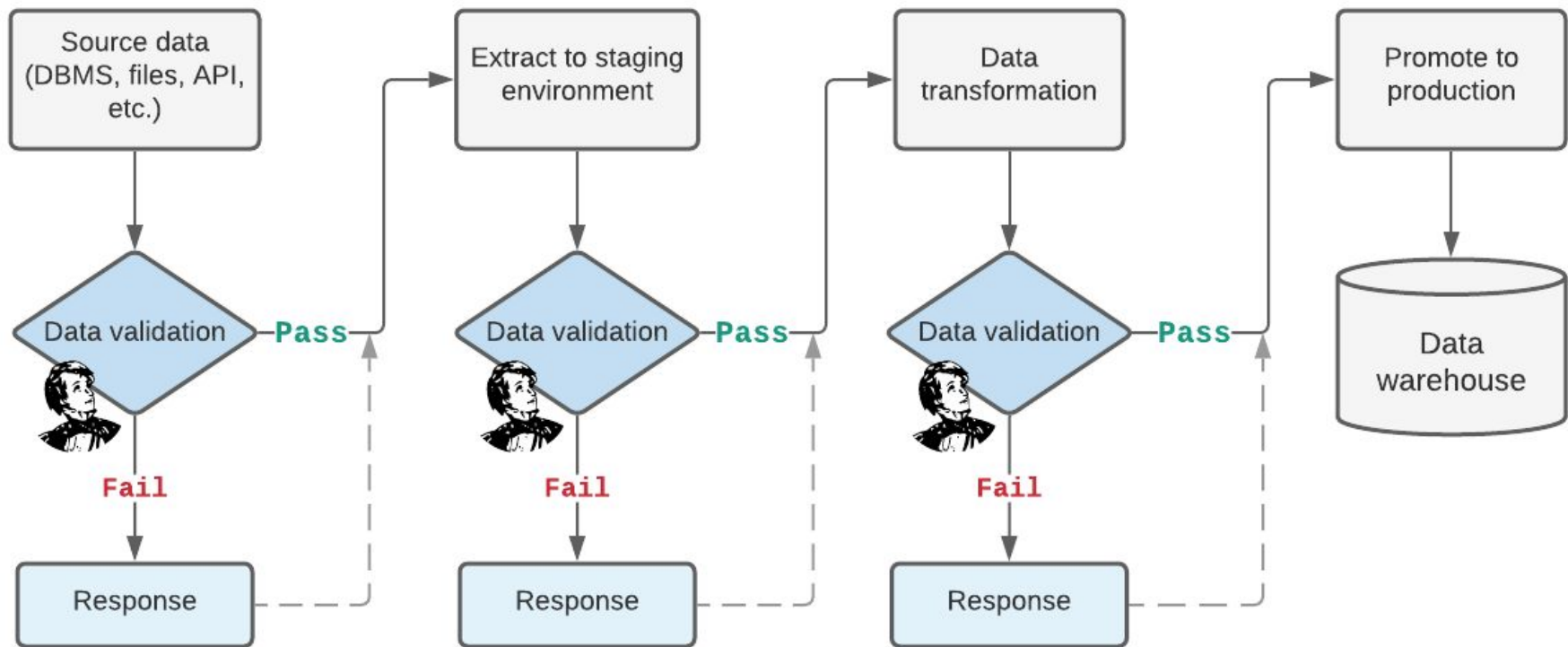
(Which tool should we use? How do we know we're testing the right thing? What do we do when tests fail? Who owns this? How do we keep them up to date? How do our stakeholders find out about the state of the data?)

## 2: Data testing is kinda hard.

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

A Rube Goldberg-style contraption is shown against a light blue background. At the top, a green watering can is suspended by a string and is tipping over, pouring water. Below it, a red bucket is suspended by a string and is also tipping over, pouring water. A pulley system is visible in the center, with a string running over a pulley and a small metal clip attached to it. To the left, a ramp with a blue ball and a red ball is shown. In the foreground, there are several colorful balls (blue, green, yellow) and a potted plant with long, thin leaves. The text "3: You should test your data whenever there is a *handoff* between teams or systems" is overlaid on the image.

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



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



```
.  
├── 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(  
)
```



# DAG: example\_ge\_dbt\_dag

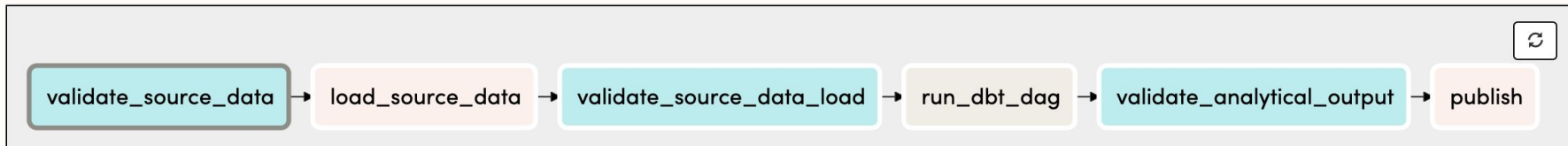
schedule: 1 day, 0:00:00

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

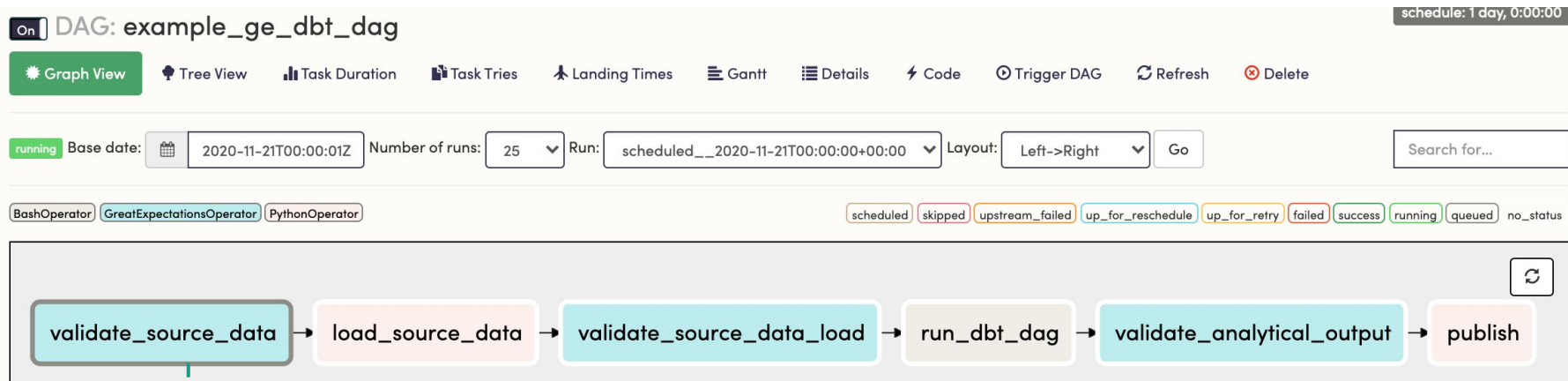
running Base date: 2020-11-21T00:00:01Z Number of runs: 25 Run: scheduled\_\_2020-11-21T00:00:00+00:00 Layout: Left->Right Go Search for...

BashOperator GreatExpectationsOperator PythonOperator

scheduled skipped upstream\_failed up\_for\_reschedule up\_for\_retry failed success running queued no\_status







Test that source data 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

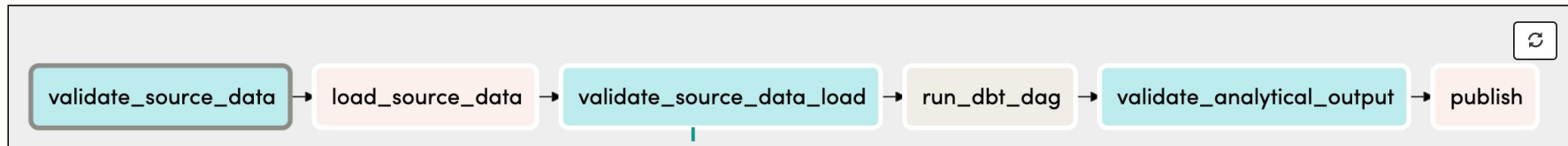
schedule: 1 day, 0:00:00

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

On DAG: example\_ge\_dbt\_dag

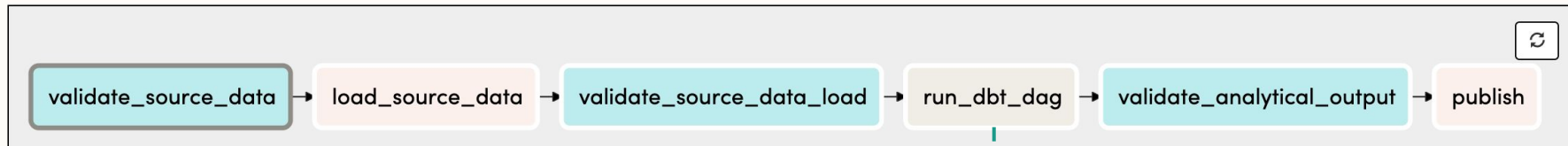
schedule: 1 day, 0:00:00


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



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

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

 **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!

