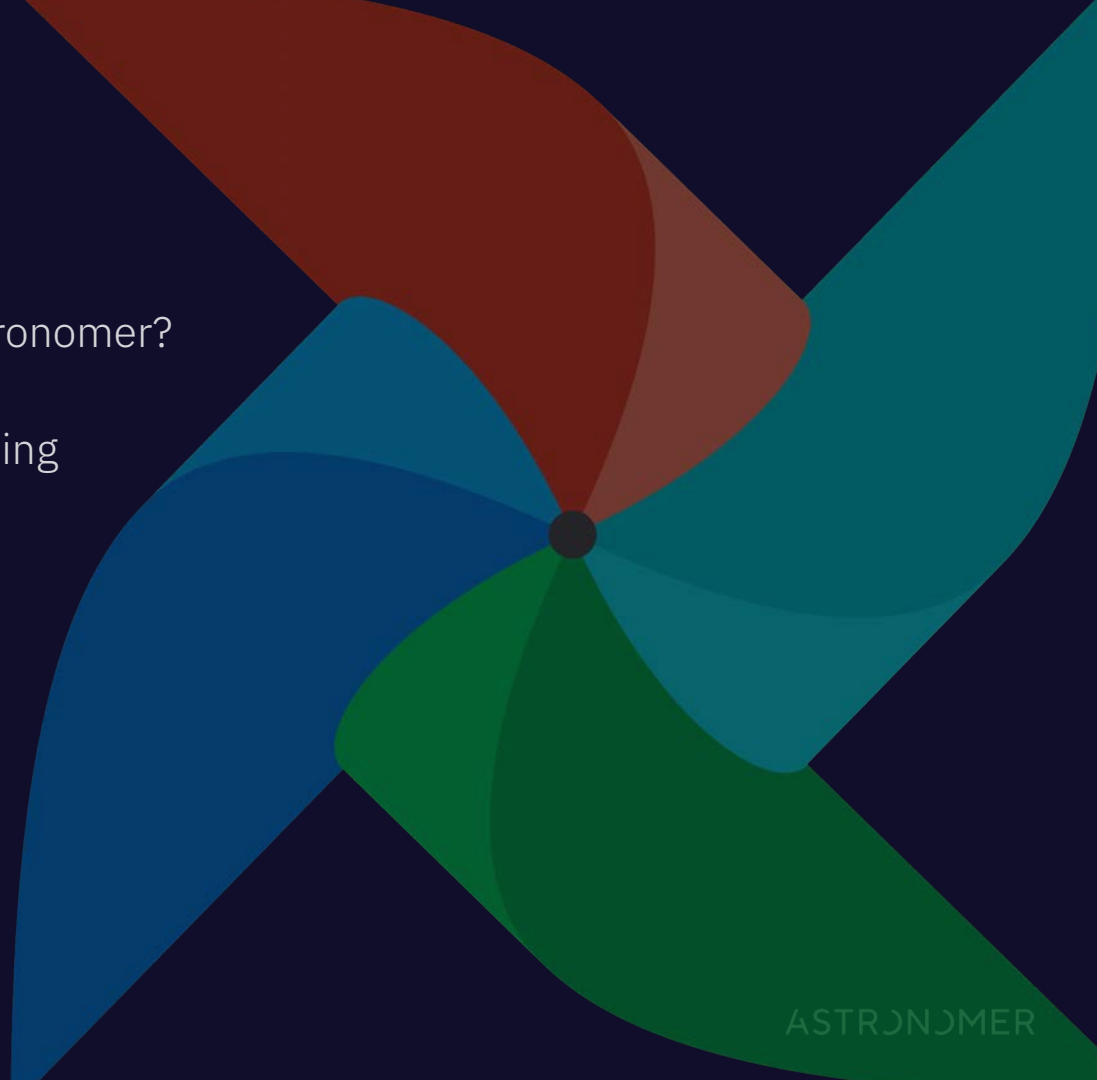
# How to use common Python frameworks to test Apache Airflow data pipelines

# Agenda

- What is Airflow? Who is Astronomer?
- Why test data pipelines?
- Local development and testing
- CI/CD and testing
- Demo



# What is Airflow?
# Who is Astronomer?

# **Airflow is** the open standard for **Workflow Management.**

**12M+**
monthly downloads

GROWTH

**42k+**
Slack members

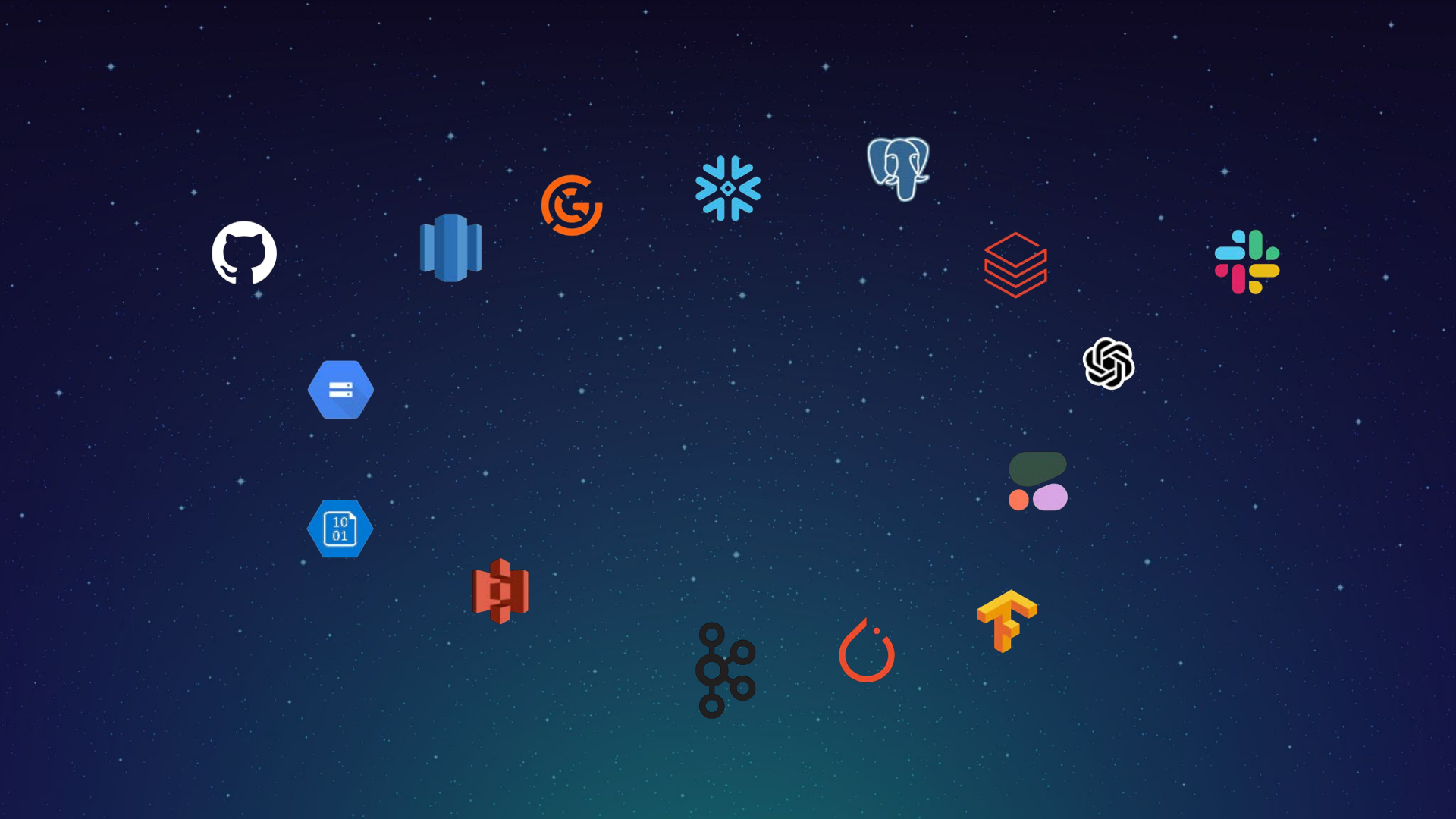
COMMUNITY

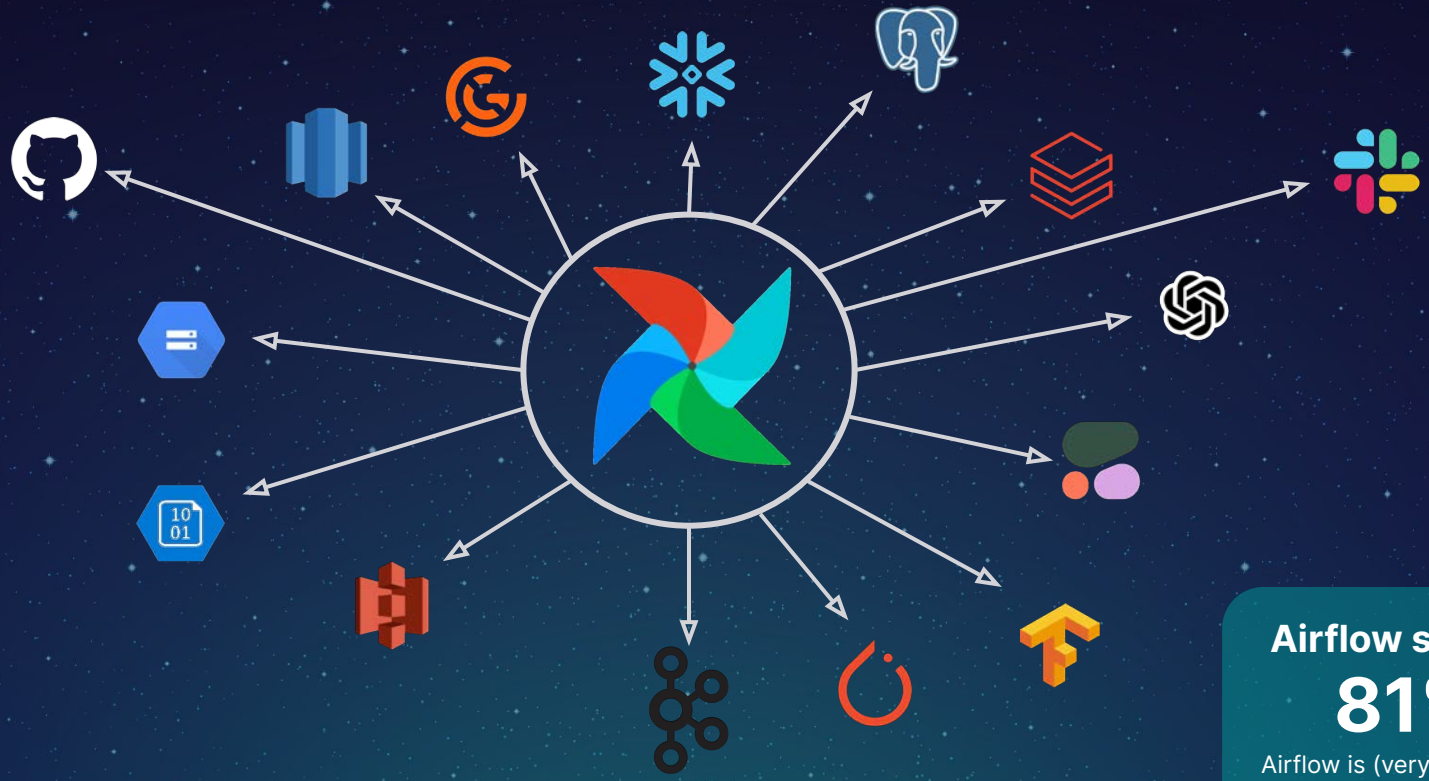
**2800+**
contributors

INNOVATION

**1600+**
building blocks

ECOSYSTEM


10
01

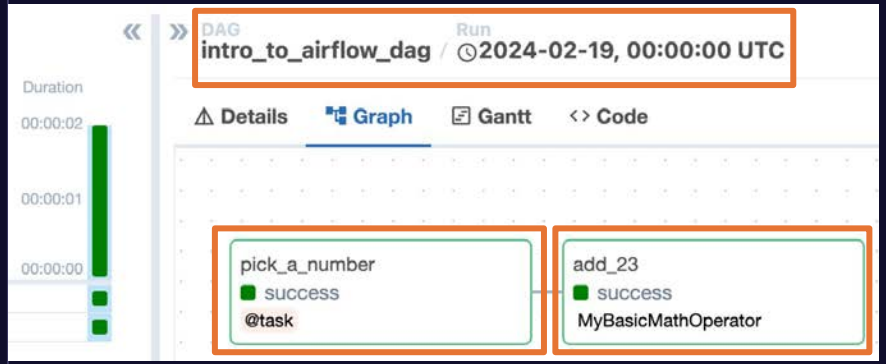


**Airflow survey:**

**81%**

Airflow is (very) important to our business! (n=806)

# Airflow 101




Pipeline in Airflow = DAG

A DAG contains tasks.

Tasks are defined in Python using operator classes and/or decorators.

DAGs run periodically, incrementally, automatically.

Tasks are idempotent, atomic and modular

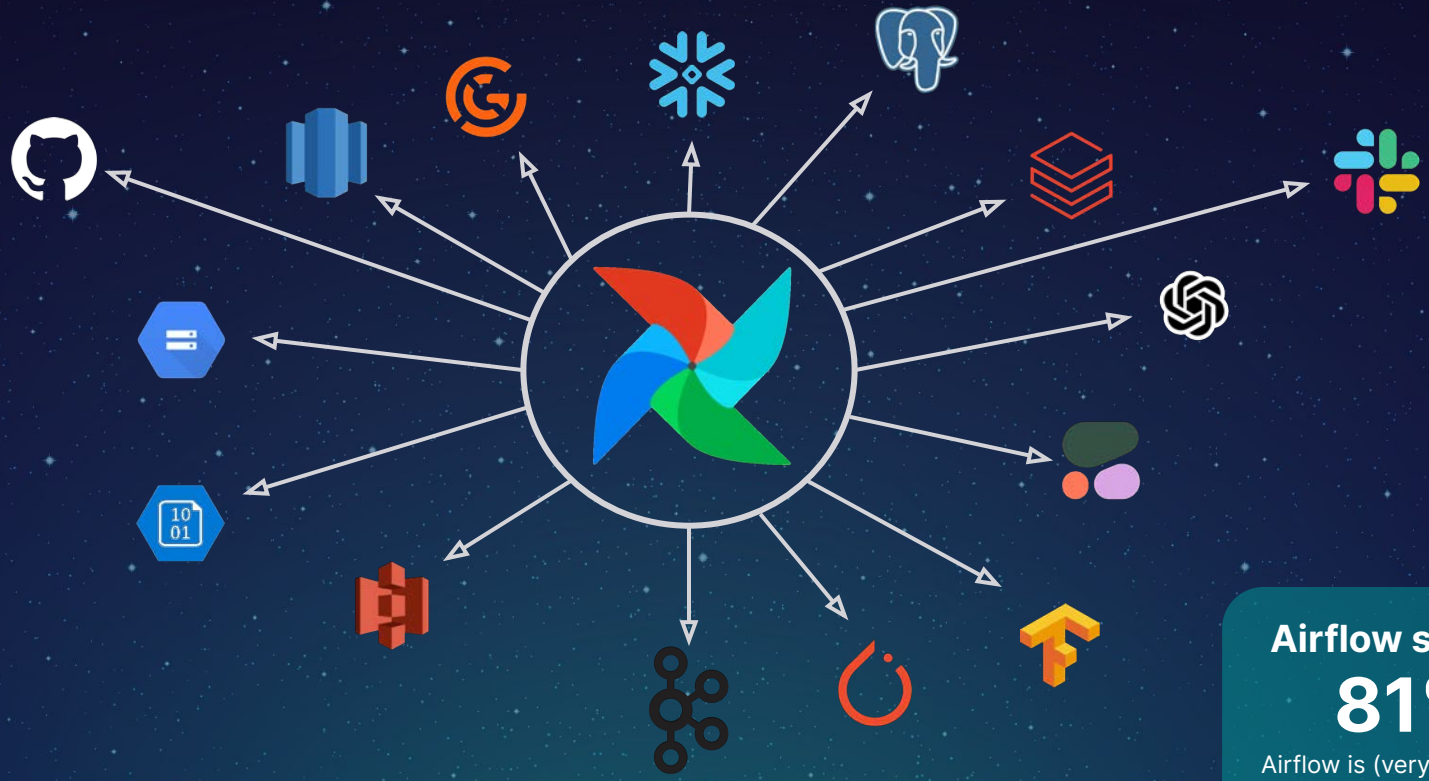
dags > intro_to_airflow_dag.py > ...

```python
from airflow.decorators import dag, task
from airflow.models.baseoperator import chain
from pendulum import datetime
from include.custom_operators import MyBasicMathOperator


@dag(
    start_date=datetime(2024, 1, 1),
    schedule="@daily",
    catchup=False,
)
def intro_to_airflow_dag():

    @task
    def pick_a_number() -> int:
        "Return a random number between 1 and 100."
        import random

        return random.randint(1, 100)

    pick_a_number_obj = pick_a_number()

    add_23 = MyBasicMathOperator(
        task_id="add_23",
        first_number=pick_a_number_obj,
        second_number=23,
        operation="+",
    )

    chain(pick_a_number_obj, add_23)


intro_to_airflow_dag()
```

Free trial of **Astro**:
astronomer.io/try-astro

# **Astronomer is** the best place to run Apache Airflow **in production.**

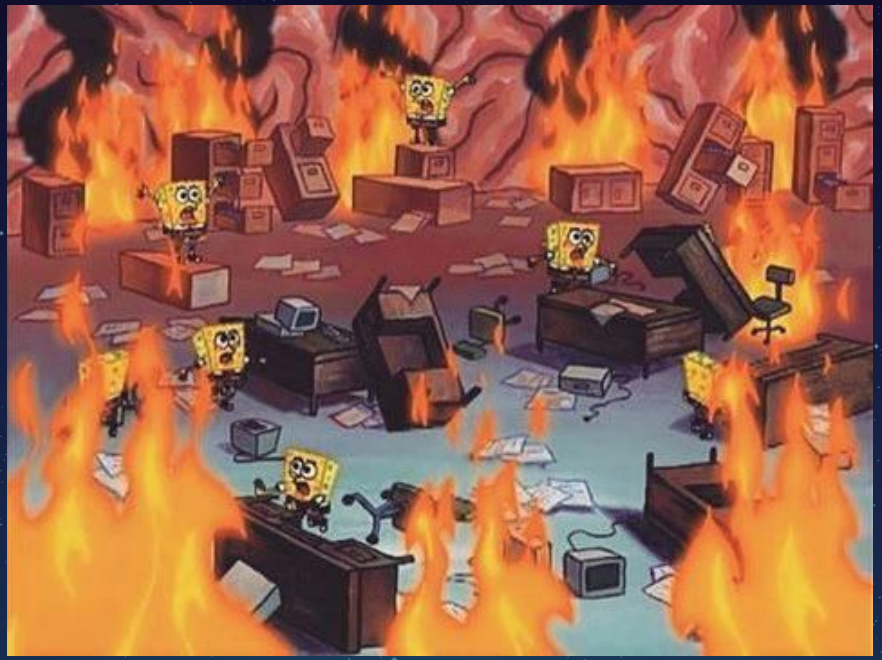
# Why test data pipelines?



**Airflow survey:**

**81%**

Airflow is (very) important to our business! (n=806)

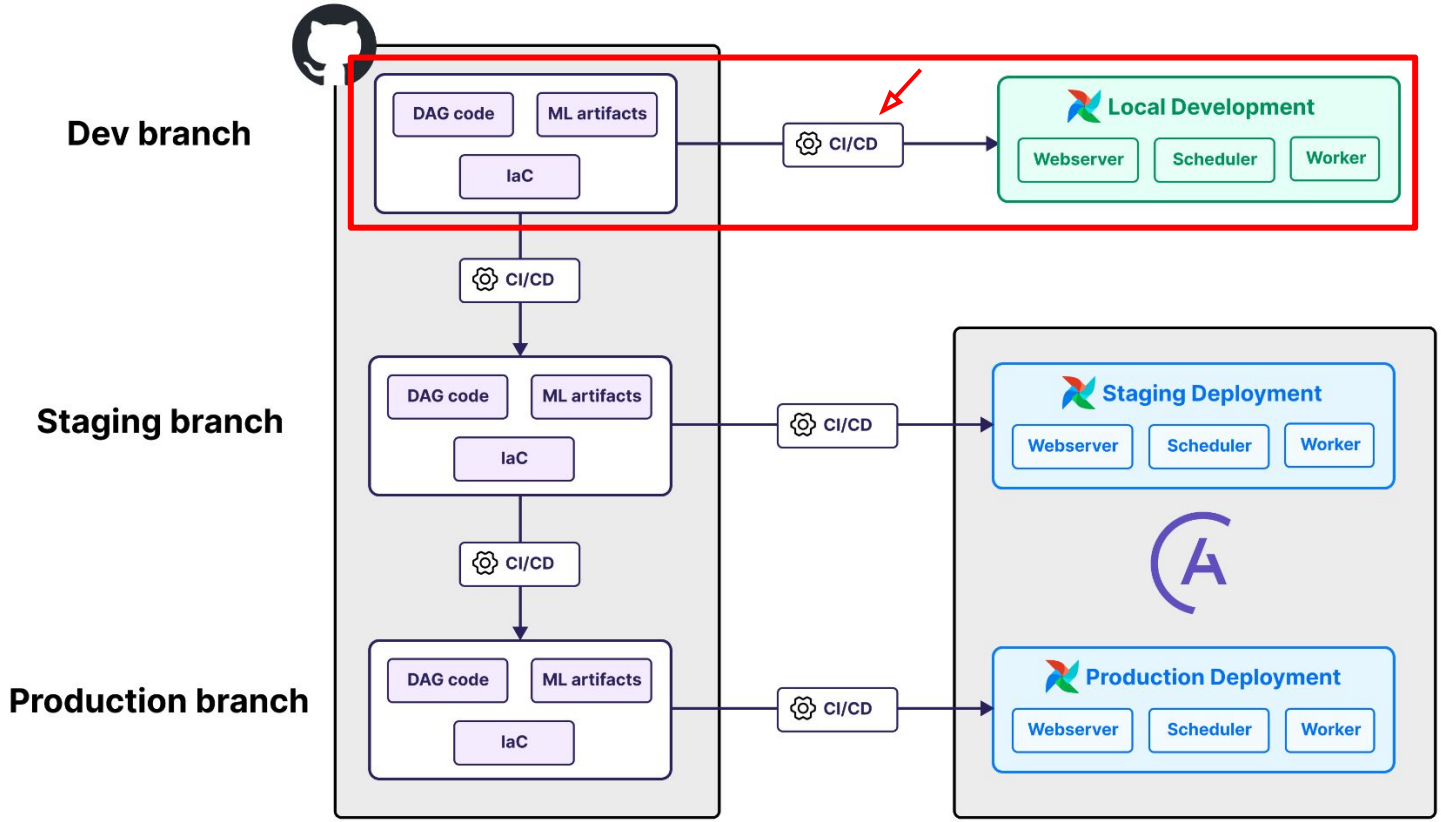


Testing your data pipelines prevents (some of) this!

Airflow is written in Python and Airflow pipelines are **just Python code**.

All software engineering and DevOps best practices apply, including testing and CI/CD!

# Local development and testing


Dev branch
Staging branch
Production branch
DAG code
ML artifacts
IaC
CI/CD
Local Development
Webserver
Scheduler
Worker
Staging Deployment
Production Deployment

# Local development and testing with the OSS **Astro CLI**

Reproducible local Airflow environment in **Docker**.

Easy to spin up.

Built-in testing features.

Install via Homebrew: `brew install astro`

```
0 23:10:45 2024_conf42_python_airflow_testing  % astro dev init
Initializing Astro project
Pulling Airflow development files from Astro Runtime 10.3.0
```

```
0 23:11:13 2024_conf42_python_airflow_testing  % astro dev start
[build -t 2024-conf-42-python-airflow-testing_ffeaba/airflow:latest -f Docke
[+] Building 0.6s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
```

```
Airflow Webserver: http://localhost:8080
Postgres Database: localhost:5432/postgres
The default Airflow UI credentials are: admin:admin
The default Postgres DB credentials are: postgres:postgres
```

https://docs.astronomer.io/astro/cli/install-cli

# Astro CLI and Airflow CLI testing features

## astro dev parse

Parses DAGs, ensures there are no import errors.

## astro dev pytest

Runs all tests in the /tests/ directory. Any Python framework!

## astro dev upgrade-test

Tests the environment against newer Airflow/Runtime versions.

## airflow dags test
**(astro dev run dags...)**

Executes a single DAGrun.

## airflow tasks test
**(astro dev run tasks...)**

Executes a single task.

# dag.test()

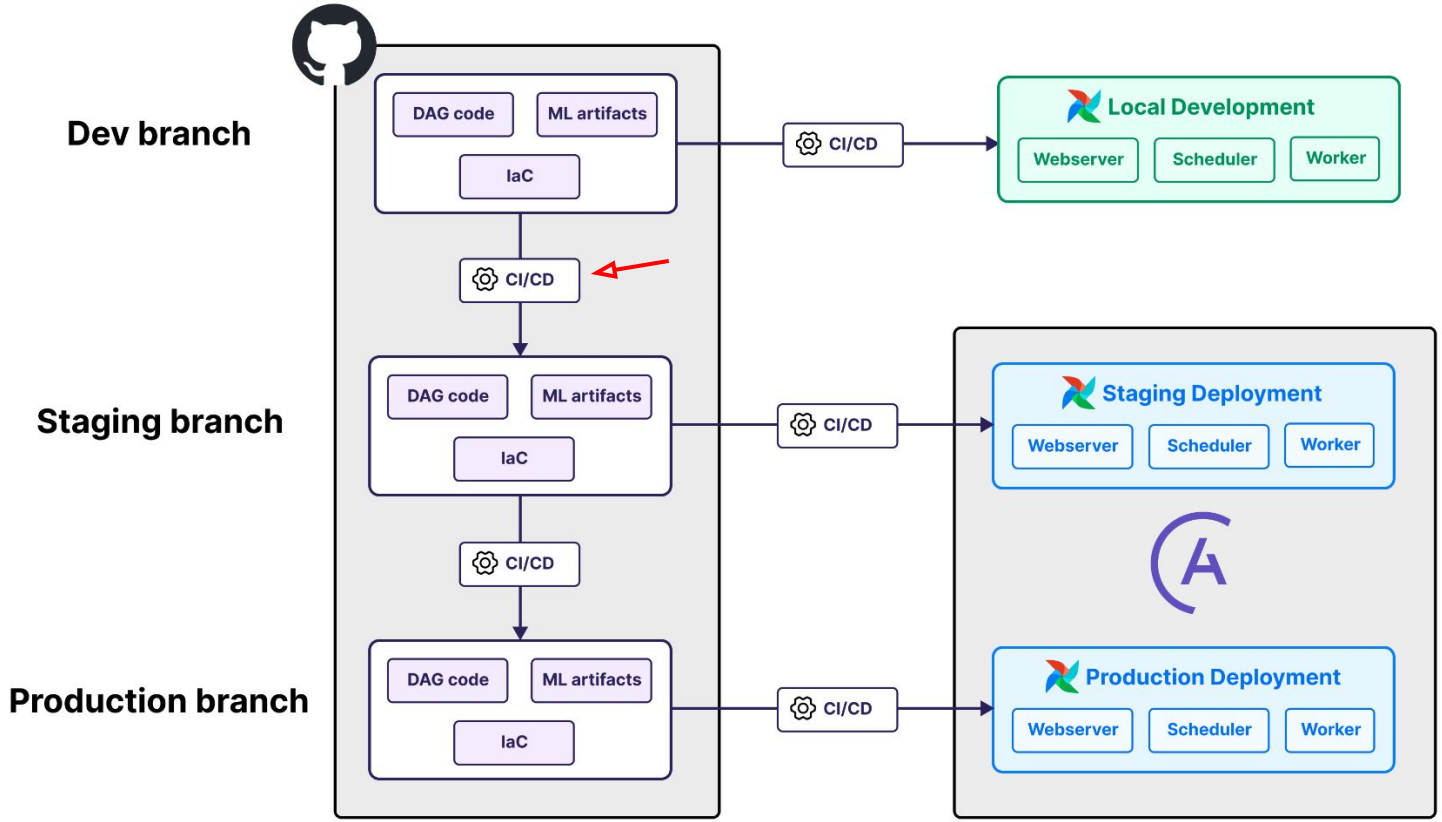
With dag.test you can test Airflow DAGs interactively with your favorite **Python debugging** tool.

Includes:

- Picking an execution date
- Using connections
- Using variables
- Using DAG conf

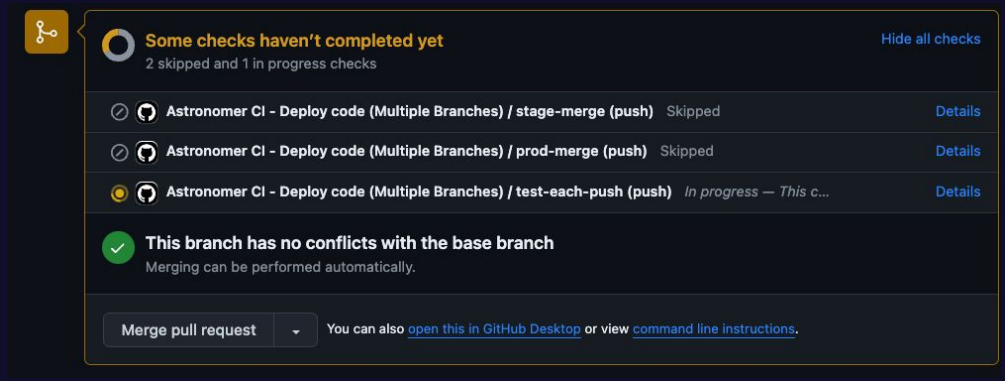
```python
if __name__ == "__main__":
    conn_path = "connections.yaml"
    variables_path = "variables.yaml"
    upper_limit = 50
    lower_limit = 10

    dag_obj.test(
        execution_date=datetime(2024, 2, 29),
        conn_file_path=conn_path,
        variable_file_path=variables_path,
        run_conf={"upper_limit": upper_limit, "lower_limit": lower_limit},
    )
```

# CI/CD and testing


Dev branch
Staging branch
Production branch
DAG code
ML artifacts
IaC
CI/CD
Local Development
Webserver
Scheduler
Worker
Staging Deployment
Production Deployment

# From dev to staging

- **PR** from the **dev** branch **to** the **staging** branch in your version control tool.
- `astro dev pytest` as part of CI/CD runs:
  - DAG validation tests
  - Unit Tests
  - Integration Tests
- Only merge to staging if tests pass!



## DAG validation tests

**Airflow specific!**

DAG validation tests test:

- DAG parsing = is this a valid Airflow DAG?
- Custom DAG rules, for example:
  - constraints on schedules, start_dates or tags
  - Only allow specific operators

You can use any Python test framework you like!

```
57  @pytest.mark.parametrize(
58      "dag_id,dag, fileloc", get_dags(), ids=[x[2] for x in get_dags()]
59  )
60  def test_dag_has_catchup_false(dag_id, dag, fileloc):
61      """
62      test if a DAG has catchup set to False
63      """
64      assert (
65          dag.catchup == False
66      ), f"{dag_id} in {fileloc} must have catchup set to False."
67
```

# Unit tests with Airflow

💡For modules from Airflow providers unit tests are already done!

registry.astronomer.io

Use unit tests to test custom Python code in:

- Custom hooks and operators
- Functions used in @task decorated tasks

You can use any Python test framework you like!

```python
class TestMyBasicMathOperator(unittest.TestCase):

    def test_addition(self):
        operator = MyBasicMathOperator(
            task_id="basic_math_op", first_number=2, second_number=3, operation="+"
        )
        result = operator.execute(None)
        self.assertEqual(result, 5)
```

# Integration tests with Airflow

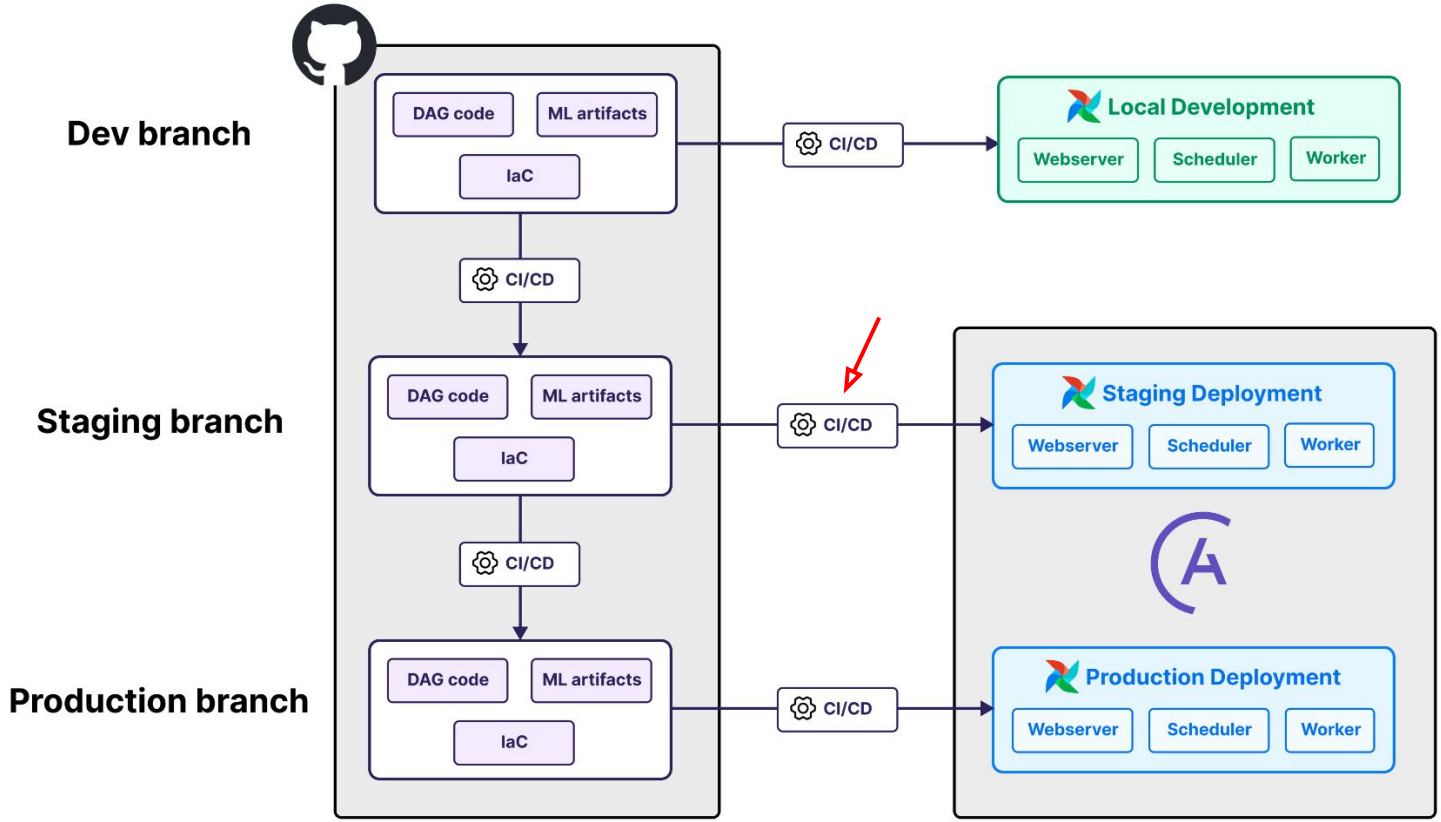
Integration tests to test API calls and connections in custom code:

- Custom hooks and operators
- Functions used in @task decorated tasks

You can use any Python test framework you like!

Careful: Can incur cost and take a lot of time, for example with LLM calls.

```python
from include.utils import get_random_number_from_api

def test_get_random_number_from_api():
    result = get_random_number_from_api(min=1, max=100, count=1)
    assert 1 <= result <= 100

```

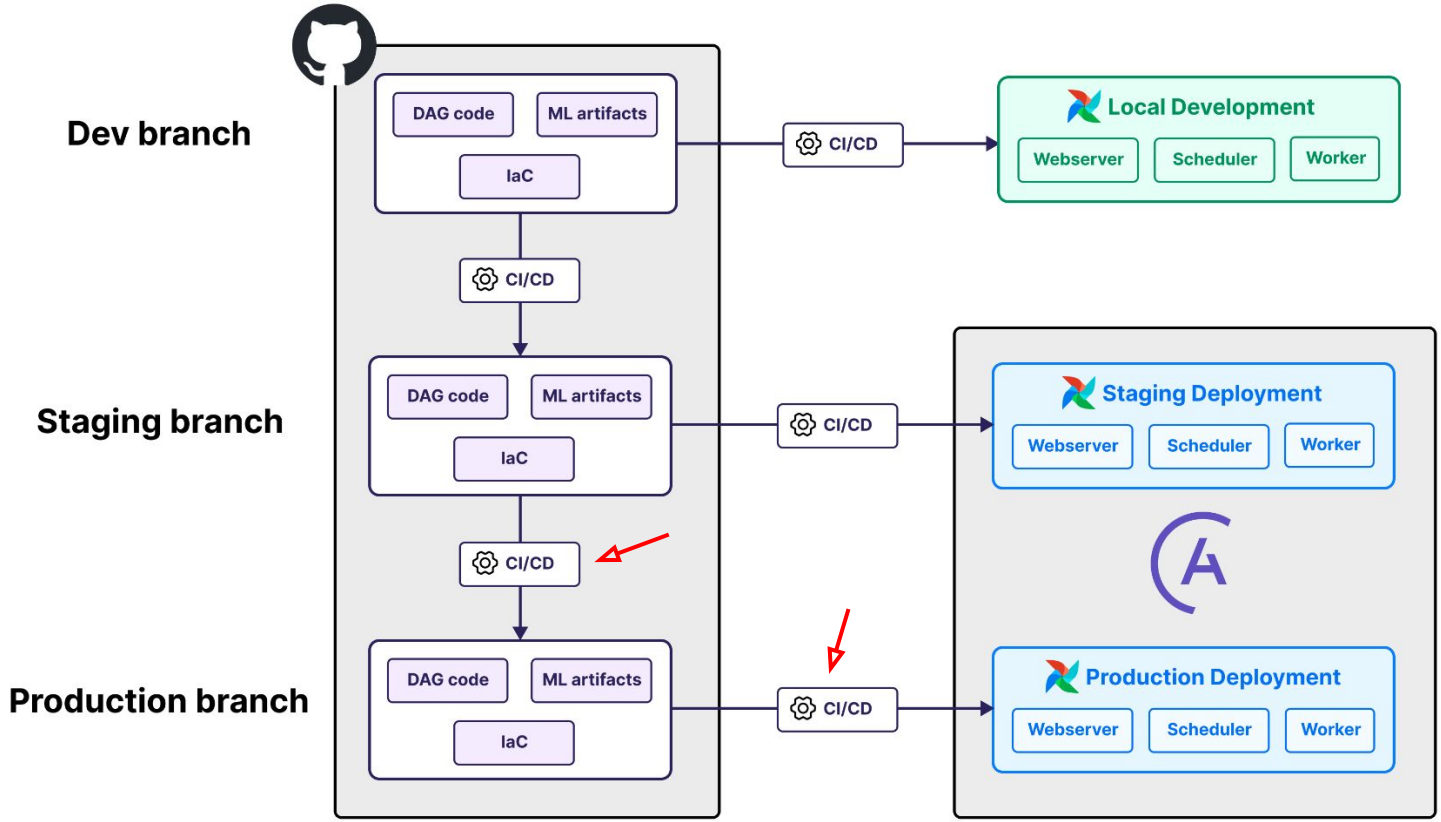

Dev branch
Staging branch
Production branch
DAG code
ML artifacts
IaC
CI/CD
Local Development
Webserver
Scheduler
Worker
Staging Deployment
Production Deployment

# The CD in CI/CD - our code on the way to the ☁️

Your CI/CD script should **run all tests** in your /tests/ folder. Afterwards the code is **automatically deployed** to the staging deployment.

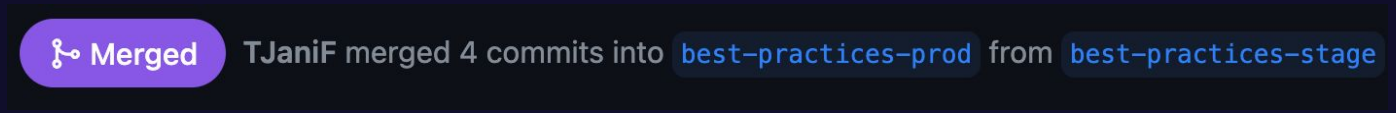
- You can use any CI/CD tool that you like.
- Example scripts are available

Consider having automated creation of your deployment and your **infrastructure management as part of your CI/CD**. (For Astronomer customers: Astro API)


Dev branch
Staging branch
Production branch
DAG code
ML artifacts
IaC
CI/CD
Local Development
Webserver
Scheduler
Worker
Staging Deployment
Production Deployment

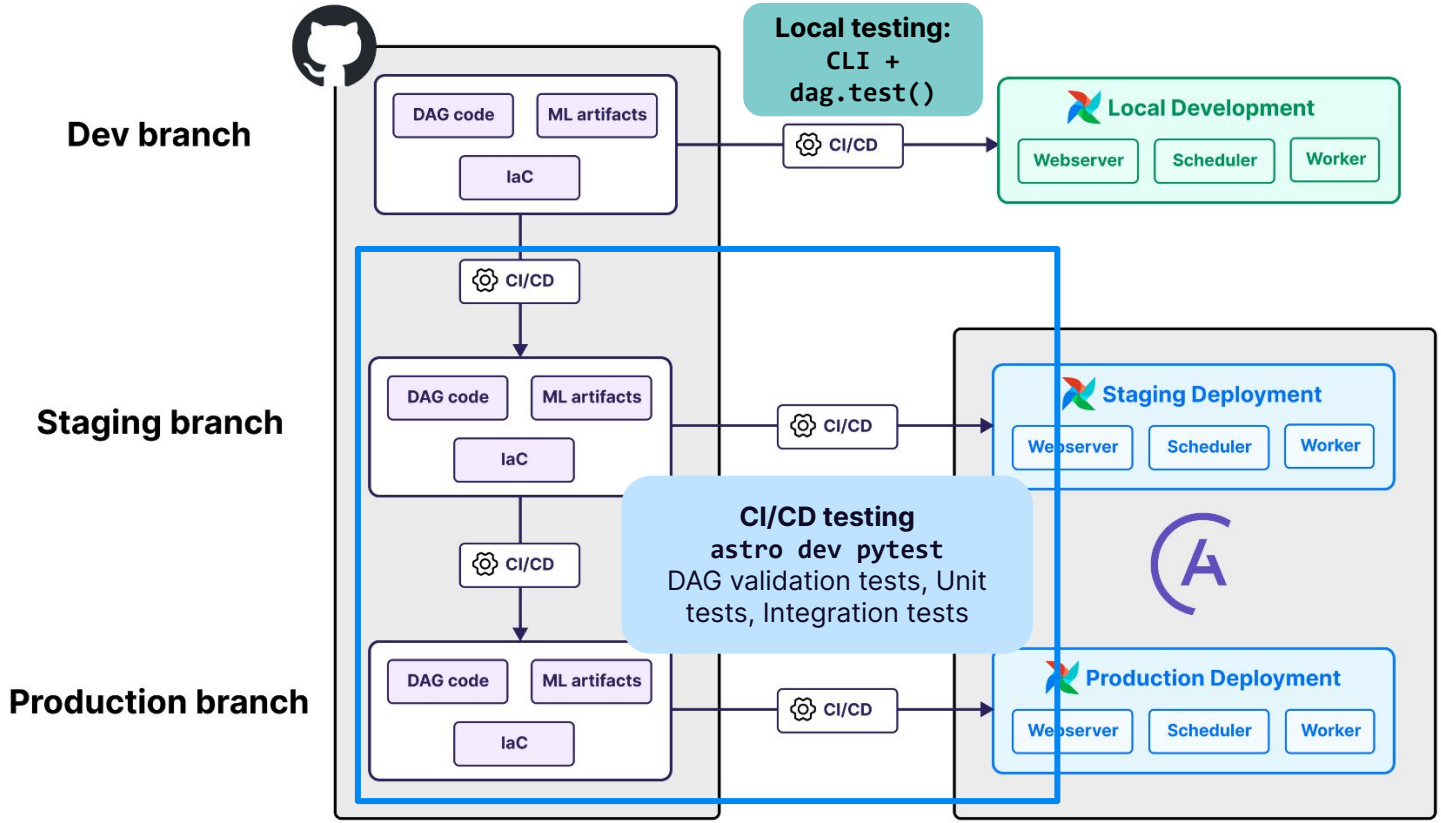
# Code getting promoted to prod

- Recommended: Run your code a few days in your staging deployment as an end-to-end test.
- Bundled PRs Staging to Prod.
- As with the Staging PR, tests run again and deployment is automatic.



# Demo

https://github.com/astronomer/external-talk-demos/tree/2024-conf42-python-airflow-testing-prod


Dev branch
Staging branch
Production branch
DAG code
ML artifacts
IaC
CI/CD
Local testing:
CLI +
dag.test()
Local Development
Webserver
Scheduler
Worker
Staging Deployment
Webserver
Scheduler
Worker
CI/CD testing
astro dev pytest
DAG validation tests, Unit tests, Integration tests
Production Deployment
Webserver
Scheduler
Worker

## Take Home Message:

Airflow is written in Python and Airflow pipelines are **just Python code**.
All software engineering and DevOps best practices apply, including testing and CI/CD!