Automated Testing and Deployment of DAGs

Austin Bennett

Automation and Testing Foundations

For Python/AirFlow

Best Practices: Foundations

There are some “Best Practices”:

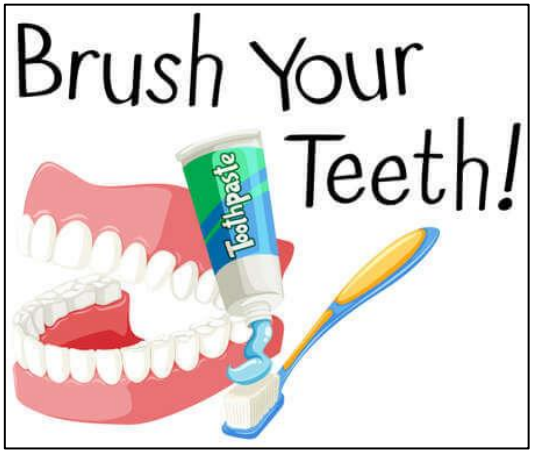
https://airflow.apache.org/docs/apache-airflow/stable/best-practices.html

But, let’s talk about the FOUNDATIONS/BASICS

P.S. I take for Git use as a given; though, I think much of the talk will be applicable even if not(?)

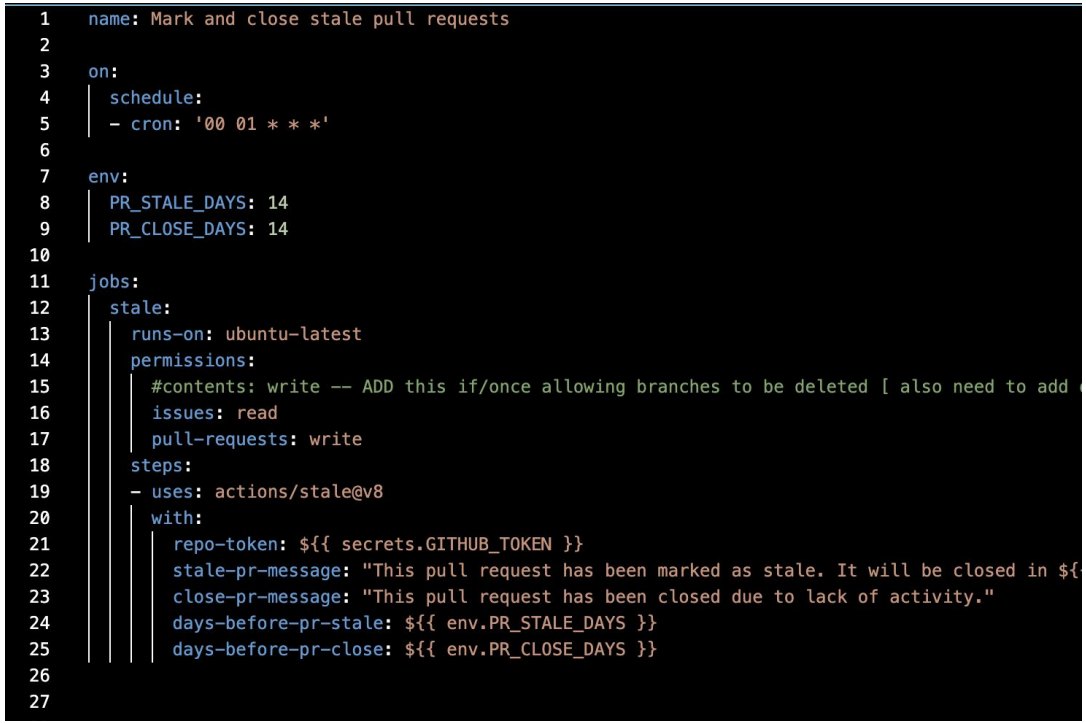
Code Hygiene

- Might not seem fun, or even a distraction …
- BUT …



StaleBot

https://github.com/actions/stale



```yaml
name: Mark and close stale pull requests

on:
  schedule:
  - cron: '00 01 * * *'

env:
  PR_STALE_DAYS: 14
  PR_CLOSE_DAYS: 14

jobs:
  stale:
    runs-on: ubuntu-latest
    permissions:
      #contents: write -- ADD this if/once allowing branches to be deleted [ also need to add
      issues: read
      pull-requests: write
    steps:
    - uses: actions/stale@v8
      with:
        repo-token: ${{ secrets.GITHUB_TOKEN }}
        stale-pr-message: "This pull request has been marked as stale. It will be closed in ${
        close-pr-message: "This pull request has been closed due to lack of activity."
        days-before-pr-stale: ${{ env.PR_STALE_DAYS }}
        days-before-pr-close: ${{ env.PR_CLOSE_DAYS }}
```

Consistency, Commonalities

Where to begin?

- We are talking airflow, so esp. Python code in:

  - **$AIRFLOW_HOME/plugins**
  - **$AIRFLOW_HOME/dags**

Some Tools

- CONSISTENT ENVIRONMENTS
  - Poetry
  - Nix
  - Devcontainers
- pre-commit
- GH Actions
- PYTHON
  - Black
  - Ruff / Flake8
  - Type-checking(?!)
- SQL
  - SQLFluff
-

CONSISTENCY Across Team

Define and Pin Versions/Dependencies to save debug headaches!!

https://python-poetry.org/

At least a requirements.txt

Poetry is great if sticking to ONLY Python

BUT, for environments not tied to python, see:

- NIX
- DevContainers

Wider than Python Ecosystem: NIX

https://nixos.org/

Pros/Cons.

Cool Design.

Works.

Not Common.

Plays well with poetry:

https://github.com/nix-community/poetry2nix

Tutorial: https://determinate.systems/posts/zero-to-nix/

DevContainers

https://containers.dev/

Can run locally while developing, and beyond

Well supported by some major IDEs

Can be ‘just’ Dockerfiles

Pre-Commit

Run things locally “before” the commit

Official supported hooks:

- https://pre-commit.com/hooks.html

https://pre-commit.com/

Also, can just be a ‘hook’/trigger based on the action of attempting to commit

Airflow Repo uses Pre-Commit:

- https://github.com/apache/airflow/blob/main/.pre-commit-config.yaml

^^ is currently 1346 lines on master

Pre-Commit: Starter

```yaml
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v4.4.0
    hooks:
      - id: check-yaml
      - id: end-of-file-fixer
      - id: trailing-whitespace
      - id: check-toml
      - id: check-json
```

A good starting place for Python

- Black is the uncompromising Python code formatter. By using it, you agree to cede control over minutiae of hand-formatting. In return, Black gives you speed, determinism, and freedom from pycodestyle nagging about formatting. You will save time and mental energy for more important matters.

Pre-Commit: Black [“custom”]

```yaml
repos:
  - repo: local
    hooks:
      - id: black
        name: black
        language: system
        entry: poetry run black --check
        types: [python]
```

GitHub Action: Black [off-the-shelf]

```yaml
name: Lint

on: [push, pull_request]

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: psf/black@stable
```

GitHub Action: Black [“custom”]

This can even be improved

Ex: find changed files

- jitterbit/get-changed-files

Or custom code …

Then only check the ‘new’/updated

Again, pros/cons

```yaml
name: Black

on: [pull_request]

jobs:
  black:
    runs-on: ubuntu-latest
    container:
      image: ${{ vars.CONTAINER }}
      options: --platform linux/amd64
      credentials:
        username: ${{ github.actor }}
        password: ${{ secrets.github_token }}

    steps:
      - uses: actions/checkout@v4

        # included to aid debuggin
      - name: Echo value for container build
        run: echo "${{ vars.CONTAINER }}"

      - name: Run Black
        shell: 'bash'
        run: poetry run black --check .

```

AIRFLOW RULES

- Ex: https://github.com/BasPH/pylint-airflow [needs updated]
- Also: https://github.com/feluelle/airflint [says not production ready]

The current codes are:

Code	Symbol	Description
C8300	different-operator-varname-taskid	For consistency assign the same variable name and task_id to operators.
C8301	match-callable-taskid	For consistency name the callable function '_[task_id]', e.g. PythonOperator(task_id='mytask', python_callable=_mytask).
C8302	mixed-dependency-directions	For consistency don't mix directions in a single statement, instead split over multiple statements.
C8303	task-no-dependencies	Sometimes a task without any dependency is desired, however often it is the result of a forgotten dependency.
C8304	task-context-argname	Indicate you expect Airflow task context variables in the **kwargs argument by renaming to **context.
C8305	task-context-separate-arg	To avoid unpacking kwargs from the Airflow task context in a function, you can set the needed variables as arguments in the function.
C8306	match-dagid-filename	For consistency match the DAG filename with the dag_id.
R8300	unused-xcom	Return values from a python_callable function or execute() method are automatically pushed as XCom.
W8300	basehook-top-level	Airflow executes DAG scripts periodically and anything at the top level of a script is executed. Therefore, move BaseHook calls into functions/hooks/operators.
E8300	duplicate-dag-name	DAG name should be unique.
E8301	duplicate-task-name	Task name within a DAG should be unique.
E8302	duplicate-dependency	Task dependencies can be defined only once.
E8303	dag-with-cycles	A DAG is acyclic and cannot contain cycles.
E8304	task-no-dag	A task must know a DAG instance to run.

AIRFLOW RULES: Ruff!

See: https://github.com/astral-sh/ruff/issues/4421

Currently ‘just’ one rule:

- “task variable name should be same as task_id”
  - https://github.com/astral-sh/ruff/pull/4687

Room for more. Community is open for contributions!

RUFF

Extremely FAST

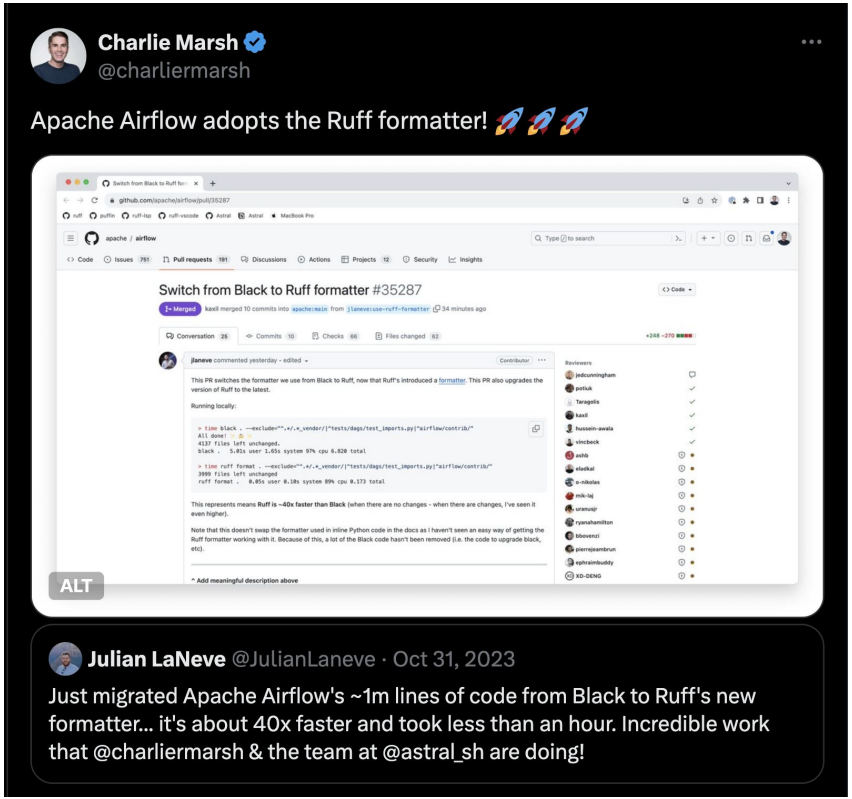
Ruff re-implements some of the most popular Flake8 plugins and related code quality tools, including:

Compare to Flake-8

AND MOAR

RUFF

Has been in use for awhile



https://x.com/charliermarsh/status/1719496146815422536

Ruff: Pre-Commit

```yaml
- repo: https://github.com/astral-sh/ruff-pre-commit
  # Ruff version.
  rev: v0.6.4
  hooks:
    # Run the linter.
    - id: ruff
      args: [ --fix ]
    # Run the formatter.
    - id: ruff-format
```

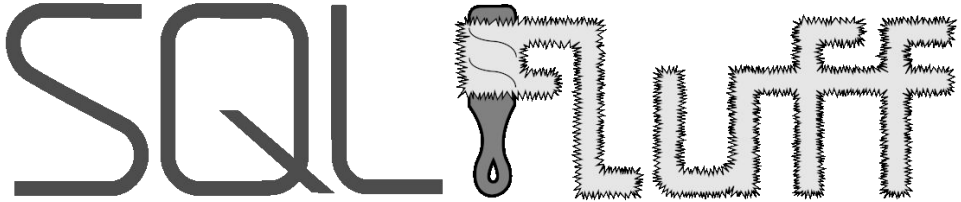
https://github.com/astral-sh/ruff-pre-commit

Ruff GitHub Action

```yaml
name: Ruff
on: [push, pull_request]
jobs:
  ruff:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: chartboost/ruff-action@v1
```

SQLFluff

DAGs orchestrate ALOT of SQL



📜 The SQL Linter for humans.

Lint 🧻

SQLFluff finds issues with your SQL code and reports them back to you (and your team) automatically so that your code reviews can be more about function and less about form.

Fix 🛠️

SQLFluff saves time by fixing linting issues found in your code to save you time, and make it easy to have consistent and legible SQL.

Parse 📜

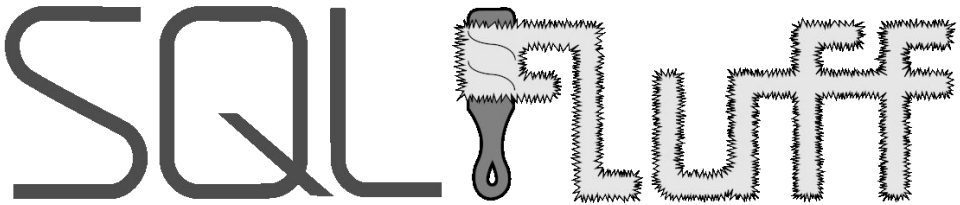
SQLFluff parses your SQL to catch a range of syntax issues without needing access to the database, so you can catch mistakes earlier in your development process.

Configure ⚙️

SQLFluff is configurable to work with a range of SQL dialects and style choices. It has opinionated defaults, so you can get going easily, but a range of flexible configuration options to fit your local style.

SQLFluff

DIALECTS:
ANSI
Athena
BigQuery
ClickHouse
Databricks
Db2
DuckDB
Exasol
Greenplum
Hive
Materialize
MariaDB
MySQL
Oracle
PostgreSQL
Redshift
Snowflake
SOQL
SparkSQL
SQLite
T-SQL
Teradata
Trino
Vertica



📜 The SQL Linter for humans.

See Contribution guide if need more than this →

https://github.com/sqlfluff/sqlfluff/wiki/Contributing-Dialect-Changes

Templating [env/project] variables

https://docs.sqlfluff.com/en/2.1.3/developingplugins.html

Manual Deployment

Manual LOG IN
To deploy !?!

$AIRFLOW_HOME/dags

Auto-Deploy

With Safeguards, once the code is OK, deploy it…

BUT … “PROTECT THE REPO”

DEPLOY

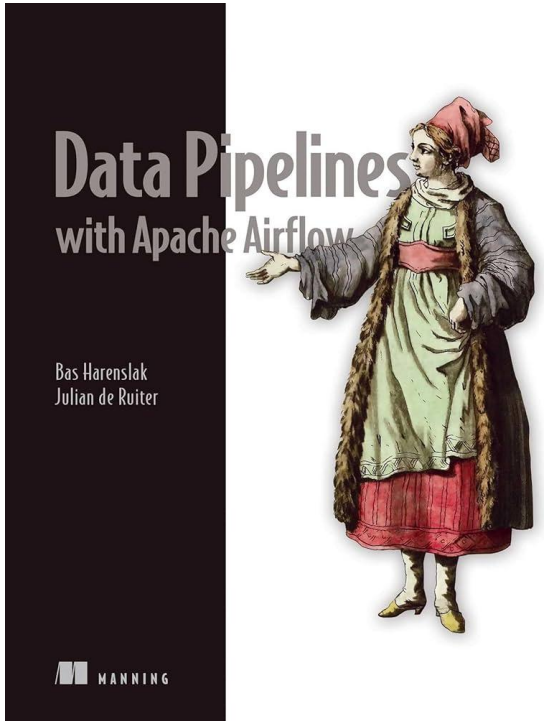
```yaml
name: Deploy Airflow DAGs

on:
  push:
    branches:
      - master

permissions:
  id-token: write
  contents: read

jobs:
  airflow-dags:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - id: 'auth'
        uses: 'google-github-actions/auth@v2'
        with:
          workload_identity_provider: ${{ vars.PROD_WIP }}
          service_account: ${{ vars.PROD_SA }}
      - name: 'Set up Cloud SDK'
        uses: 'google-github-actions/setup-gcloud@v2'
      - name: 'sync DAG files'
        run: 'gsutil -m rsync -d -r $REPO_PATH/dags $AF_ENV/dags'

```

Airflow Rules!

9 Testing

This chapter covers

- Testing Airflow tasks in a CI/CD pipeline
- Structuring a project for testing with pytest
- Mimicking a DAG run to test tasks that apply templating
- Faking external system events with mocking
- Testing behavior in external systems with containers



MOR

- ActionLint :-p
  - Lint yor GH Actions
- Types … MyPy, PyRight, PyType
  - !!
- PyTest and more
- Commit/extend RUFF!
- Don’t forget to read/implement the `Data Pipelines with Airflow` book
- With all of these you’ll be in solid shape