It’s a Breeze to develop Airflow

Jarek Potiuk - jarek.potiuk@polidea.com

Apache Airflow

Airflow is a platform to programmatically author, schedule and monitor workflows.

Dynamic/Elegant

Extensible

Scalable

Team @ Polidea

Hi!

Jarek Potiuk

Principal Software Engineer @Polidea

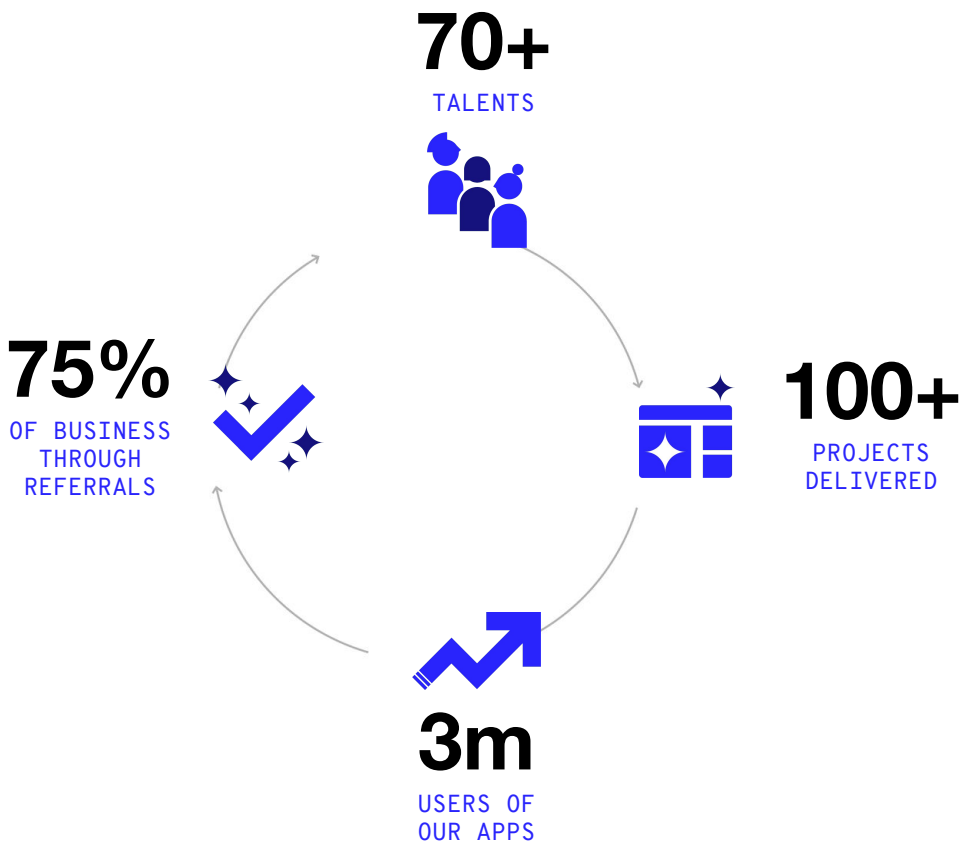
Apache Airflow PMC member

Certified GCP Architect

ex-Googler, ex-CTO, ex-choir member

@higrys

Team @Polidea



All-time Apache Airflow team at Polidea

Jarek Potiuk

Kamil Breguła

Tomasz Urbaszek

Karolina Rosół

Tobiasz Kędzierski

Michał Słowikowski

Dariusz Aniszewski

Szymon Przedwojski

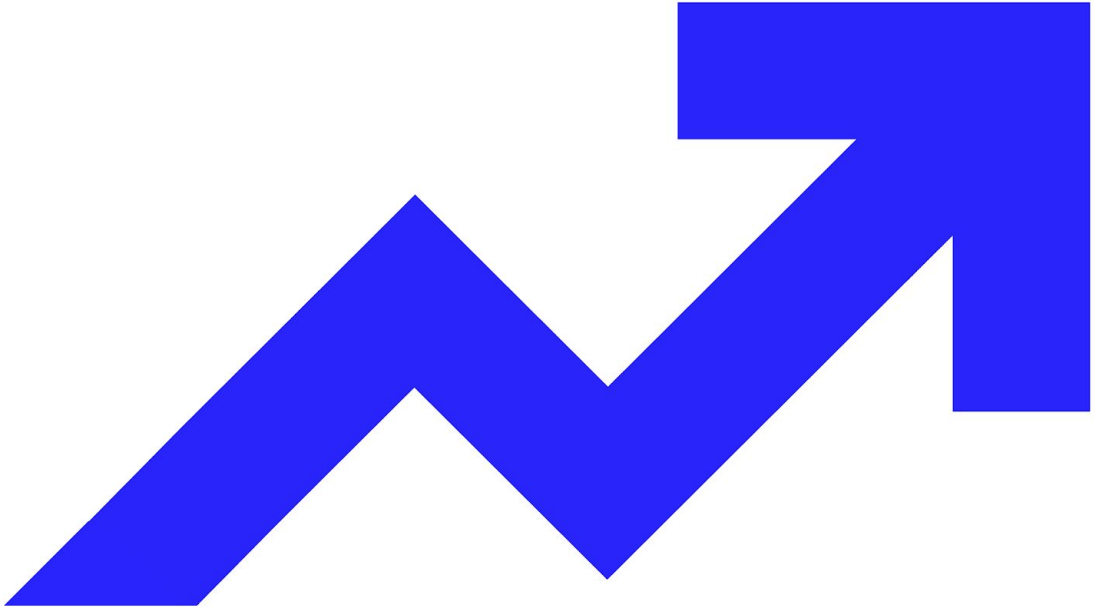
Antoni Smoliński

Polidea & Apache Airflow

Timeline

September 2019

6 (9) people



August 2018

2 people

Our tasks

- 100+ operators
- 18+ GCP services
- Oozie-To-Airflow

It’s a Breeze to develop [https://polidea.com]

polidea.com/blog/its-a-breeze-to-develop-apache-airflow/

Polidea

Our work | Services | Blog | About us | Careers | Get in touch

Engineering

10min read

#open-source #cloud

Share

It’s a "Breeze" to Develop Apache Airflow

Table of contents

- My journey to developer productivity
- What do I mean by productivity
- The Apache Airflow project’s setup
- Optimizing the process
- Open-sourcing our environment
- Bringing the environment to the Airflow community
- Learning from the experts
- Working with the community
- It’s a Breeze

What we delivered extra

- 1 Apache Airflow committer, 1 PMC member
- Documentation improvements
- Breeze - improved development environment
- Py2 -> Py3
- Pylint compatibility
- CI environment reimplemented
- Operator scaffolding

Integration Test Challenges

Integration tests on Travis CI

Pre-test			6 min 5 sec
✓	# 25131.1	Static checks (no pylint, no licence check)	4 min 14 sec
✓	# 25131.2	Check licence compliance for Apache	1 min 18 sec
✓	# 25131.3	Pylint checks	5 min 55 sec
✓	# 25131.4	Build documentation	5 min 9 sec
Test			28 min 22 sec
✓	# 25131.5	Tests postgres python 3.6	25 min 22 sec
✓	# 25131.6	Tests sqlite python 3.5	25 min 42 sec
✓	# 25131.7	Tests mysql python 3.7	28 min 22 sec
✓	# 25131.8	Tests postgres kubernetes python 3.6 (persistent)	15 min 4 sec
✓	# 25131.9	Tests postgres kubernetes python 3.6 (git)	16 min 57 sec

Integration tests challenges

- Multiple backends: postgres, mysql, sqlite
- Multiple python versions (2.7) - 3.5, 3.6. 3.7
- Multiple executors: Local/Sequential/Kubernetes
- Automated static code analysis
- Automated documentation building

The problems with Integration Tests

- Long time to set it up
- Frustrations of fresh developer experience
- High friction/learning curve for Airflow development environment
- Slow iteration speed
- Complicated Development Environment

Original CI environment

- Scripts only designed for CI, not local environment
- Dependencies installed every time you start the environment
- Always full database reset
- Minutes to run one test
- No guidance how to iterate over tests

Ash’s “Hacking on Airflow”

Challenge accepted

The Goal

- Focus on developer productivity
- Faster development cycle
- Decrease developer frustration
- Improve the teamwork
- Easy for ad-hoc contributors to code & test

Improvements

- **AIP-10**: Multi-layered and multi-stage official Airflow image
- **AIP-7**: Simplified Development Workflow
- **AIP-26**: Production-ready Airflow Docker Image and helm chart
- **AIP-23:** Migrate out of Travis CI
- **AIP-4**: Support for System Tests for external systems

The environments

- Local virtualenv
- Own Travis CI fork
- Docker compose (Travis CI equivalent)

Previous testing experience

- Total time: 7 minutes
- Running one test only
- Failure at the end (!)
- Re-run - 10-20 seconds for DB
- Re-enter - **same time (!)**
- No bash history

Improved Integration Tests

Step 1 - Multi-stage, multi-layered Docker image

- Docker images built from master automatically (DockerHub)
- Local images use cached images
- Tests and static checks run using Docker Compose/Docker environment
- Can be run on Kubernetes Cluster (Docker-In-Docker)
- CI system - independent
- Base to build production image

Step 2 - local scripts to manage the environment

- Entering the environment
  - `PYTHON_VERSION=3.5 BACKEND=postgres ENV=docker ./scripts/ci/local_ci_enter_environment.sh`
- Static checks run in Docker
  - Mypy: `./scripts/ci/ci_mypy.sh`
  - Pylint main: `./scripts/ci/ci_pylint_main.sh`,
  - Pylint tests: `./scripts/ci/ci_pylint_test.sh`
  - Flake8: `./scripts/ci/ci_flake8.sh`
  - Licence check: `./scripts/ci/ci_check_licence.sh`
  - Documentation build: `./scripts/ci/ci_docs.sh`
- Run static checks on individual files/packages
  - `./scripts/ci/ci_pylint.sh ./airflow/stats.py`
- Update images
  - `./scripts/ci/local_ci_build.sh`
  - `./scripts/ci/pull_and_build.sh`

Step 3 - Easy way of running tests

- works out-of-the-box
- initializes DB when needed
- environment variables set
- sub-second test overhead
- ipdb debugging
- verbose output

```
Usage: run-tests [FLAGS] [TESTS_TO_RUN] -- <EXTRA_NOSETEST_ARGS>

Runs tests specified (or all tests if no tests are specified)

Flags:

-h, --help
        Shows this help message.

-i, --with-db-init
        Forces database initialization before tests

-s, --nocapture
        Don't capture stdout when running the tests. This is useful if you are
        debugging with ipdb and want to drop into console with it
        by adding this line to source code:

            import ipdb; ipdb.set_trace()

-v, --verbose
        Verbose output showing coloured output of tests being run and summary
        of the tests - in a manner similar to the tests run in the CI environment.
```

Breeze

The ideal workflow

- entering the environment: **./breeze --backend sqlite --python 3.5**
- re-entering the environment: **./breeze**
- automated image management
- autocomplete of options
- sub-second test execution overhead
- host sources mounted to Docker container
- ports forwarded
- hints for ad-hoc developers

Extras - nice to haves

- **run-tests** tests.core**<TAB><TAB>** autocomplete
- bash history across sessions
- run static checks with Breeze
- easy debugging (including debugging with IDE)
- pre-commit checks

Feel the Breeze

AIRFLOW
BREEZE

Breeze goodies

Additional sources

Breeze features

- Docker images management
- Pre-commit checks (almost all merged)
- Run-tests with DB initialisation
- Travis CI integration
- Comprehensive documentation - **Google Season of Docs YAY!**

Breeze Documentation

Table of Contents

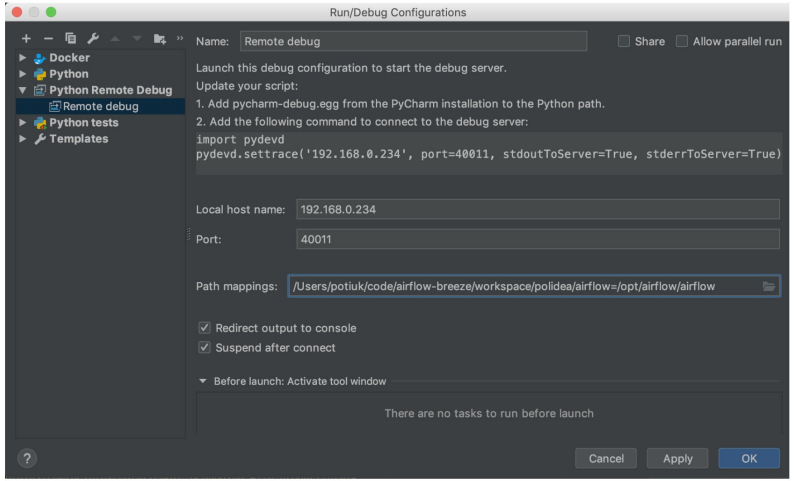
- Airflow Breeze
- Installation
- Setting up autocomplete
- *Using the Airflow Breeze environment*
  - Entering the environment
  - Running tests in Airflow Breeze
  - Debugging with ipdb
  - Airflow directory structure in Docker
  - Port forwarding
- *Using your host IDE*
  - Configuring local virtualenv
  - Running unit tests via IDE
  - Debugging Airflow Breeze Tests in IDE
- *Running commands via Airflow Breeze*
  - Running static code checks
  - Building the documentation
  - Running tests
  - Running commands inside Docker
  - Running Docker Compose commands
  - Convenience scripts
- *Keeping images up-to-date*
  - Updating dependencies
  - Pulling the images
- Airflow Breeze flags

Debugging Airflow Breeze Tests in IDE

When you run example DAGs, even if you run them using UnitTests from within IDE, they are run in a separate container. This makes it a little harder to use with IDE built-in debuggers. Fortunately for IntelliJ/PyCharm it is fairly easy using remote debugging feature (note that remote debugging is only available in paid versions of IntelliJ/PyCharm).

You can read general description about remote debugging

You can setup your remote debug session as follows:



Breeze follow-ups

Pre-commit checks

- easy to use
  - `pre-commit install`
  - `pre-commit run`
  - `pre-commit run mypy`
  - `pre-commit run --all-files`
- run only for changed files (fast)
- catches errors early
- make committers time efficient
- promotes good practices

```
Check if image build is needed.....................................Passed
Check if licences are OK for Apache...............................Skipped
No-tabs checker....................................................Passed
Add licence for all SQL files......................................Passed
Add licence for all other files....................................Passed
Add licence for all rst files......................................Passed
Add licence for all JS files.......................................Passed
Add licence for shell files........................................Passed
Add licence for all XML files......................................Passed
Add licence for yaml files.........................................Passed
Add licence for all md files.......................................Passed
Add TOC for md files...............................................Passed
Check hooks apply to the repository................................Passed
Check for merge conflicts..........................................Passed
Detect Private Key.................................................Passed
Fix End of Files...................................................Passed
Mixed line ending..................................................Passed
Check that executables have shebangs...............................Passed
Check Xml..........................................................Passed
Check yaml files with yamllint.....................................Passed
Check Shell scripts syntax correctness.............................Passed
Lint dockerfile....................................................Passed
Run mypy...........................................................Passed
Run pylint for main sources........................................Skipped
Run pylint for tests...............................................Skipped
Run flake8.........................................................Passed
```

Example errors with pre-commit

```
Lint dockerfile.....................................................Passed
Run mypy............................................................Passed
Run pylint for main sources.........................................Skipped
Run pylint for tests................................................Skipped
Run flake8..........................................................Failed
hookid: flake8

tests/gcp/operators/test_mlengine.py:23:1: F811 redefinition of unused 'ANY' from line 21
tests/gcp/operators/test_mlengine.py:23:1: F811 redefinition of unused 'patch' from line 21
tests/gcp/operators/test_mlengine_utils.py:23:1: F811 redefinition of unused 'ANY' from line 20
tests/gcp/operators/test_mlengine_utils.py:24:1: F811 redefinition of unused 'patch' from line 21
There were some flake8 errors. Exiting


There were some flake8 errors. Exiting

The command "./scripts/ci/ci_run_all_static_tests_except_pylint_licence.sh" exited with 1.
```

What’s next ?

- Migrating out of Travis CI
  - GitLab CI (only CI) or GitHub Actions
  - Kubernetes Cluster on Google Kubernetes Engine (Thanks Google!)
- Automation of Performance Tests
- Automation of Release Tests

Workshop for first time contributors to Apache Airflow

It's a Breeze to contribute to Apache Airflow

AIRFLOW BREEZE

Google Cloud

contentful

Polidea

Apache Airflow

APACHECON

It’s a Breeze to contribute to Airflow

http://bit.ly/35NrOie

Thanks!

Polidea

hello@polidea.com