Airflow in the Cloud: Lessons from the field
Airflow Summit, May 2022

Rafal Biegacz (he/him)
Sr. Engineering Manager, Google
rbiegacz@/twitter

Filip Knapik
Group Product Manager, Google
FilipKnapik1@/twitter
BIO

Rafal Biegacz

- **Cloud Composer** Sr. Eng Manager
- Has been working on Airflow for ~3 years
- Holds MSc degree in the field of Teleinformatics from Gdansk University of Technology
- Delivers Google Cloud Platform and cloud computing lectures to students of University of Warsaw and Technical University of Warsaw
Filip Knapik

- **Cloud Composer** Group Product Manager
- Working with Airflow for ~3 years
- 18+ years of IT management experience
- MSc in Computer Networks and Services at AGH University of Science and Technology in Cracow, Poland
Super-quick intro on Composer
Airflow and Composer

Cloud Composer

100% open source compatible

Cloud Composer
What is Composer used for?

Orchestrate work across **Google Cloud**, external **SaaS services** and **proprietary APIs**
Cloud Composer as a managed Airflow

- **Composer 2**
  - **Access Control**: Cloud IAM
  - **DAGs & Plugins**: Cloud Storage
  - **Logs**: Cloud Logging
  - **Metrics & Alerts**: Cloud Monitoring
  - **Airflow UI**: Cloud Composer

- **Compute layer**: Kubernetes Engine
- **Schedulers**
- **Workers**: autoscaled Task queue
- **Airflow Database**: Cloud SQL
Cloud Composer’s value

- Simple deployment
- Enterprise Security Features
- Managed infrastructure
- Robust Monitoring & Logging
- Technical Support
- DAG code portability
Airflow IS AWESOME!

Airflow Community is at the heart of Airflow
Customers appreciate Airflow

- Customers value community contributions
- Airflow 2 is awesome
- Richness of Airflow operators
- Extensibility

New users every day
Learning #1

The power of Airflow is in its extensibility
Off-the-shelf Airflow Providers don’t cover all needs

Customers orchestrate work in their own API’s through:
- Python Operator
- Bash Operator
- KubernetesPodOperator and GKEPodOperator
- Custom Operators/Sensors

More than 60% of tasks run within Composer are based on the above.

💡 Extensibility of Airflow is the fundamental power of Airflow.
As a community, we need to keep investing in this area.
Learning #2

Less Airflow-savvy users start using Airflow
“Just want to focus on running my DAGs, not Airflow management”
User quote: “I just want to run my DAG”

Some users assume that everything just works automatically. Ideally, Airflow tunes itself, Airflow DB retention/pruning happens automatically.

Additional quotes from users:
- I don’t want to actively manage Airflow environments
- I care about stability more than new Airflow features
- I don’t need to move to newer Airflow version. Why would I fix what’s not broken?

💡 All “auto-healing” capabilities in Airflow are super important.
Proper use of underlying infrastructure (e.g. Kubernetes) helps.
Great out-of-the-box experience is important for new or less Airflow-savvy users
Airflow has 300+ parameters...

...and they vary by version

...and some are interdependent

... current and deprecated coexisting for some time
With great power comes great ... complexity

- Airflow configurations are great for experts
- Most customers mis-configure Airflow
- Adjusting Celery, Worker and Scheduler params requires experience & learning

💡 Airflow configuration autotuning/recommender would be a massive benefit for less experienced users

source: https://giphy.com/
Learning #3

The better Airflow becomes, the higher users’ expectations
“Nobody [in the organization] has an incentive to keep upgrading what’s working well”
Implications of growing maturity

- Maturity
  - Quality
  - Stability
  - Features

- Criticality of Airflow workloads

- Interest in upgrades

💡 Upgrades between Airflow versions as smooth as possible. Ideally no or minimal changes to DAGs.
Deprecations... wrrr.

Users hardly read Release Notes and they are not eager to change their code to adjust.
Deprecations prevent new versions adoption

Challenging deprecations
● Operators
● Airflow configuration parameters
● Airflow connection names

Deprecation messages litter the logs

Airflow 1 vs 2 incompatibility holds customers in Airflow 1

💡 Feature deprecations need to be smooth, especially requiring DAG code changes
Users don’t like retries
Retries take time and money

- Recommending DAG or task Retries doesn’t always work

- Tasks can
  - cost hundreds of $, ₹, PLN, €
  - take hours
  - play important role in business processes

💡 Need to aim to make Airflow fault-tolerant and reduce the risk of task retries

Users expectations is that DAGs executions in 100% successful.
Learning #4

DAG and User Isolation Wanted
User Isolation, User Code separation... Multi-tenancy

- **Access Control in UI** and **Per-folder Role auto-registration** are very popular.

- User code still can impact parsing and scheduling.
- User code still can mess with the content of the database.
- Users can step on each other toes (e.g. 2 different DAGs with the same name)

Better isolation between DAG Processor and Scheduler is needed. 
AIP-43 DAG Processor separation and AIP-44 Airflow Internal API can improve the separation significantly.
Airflow is **amazing**!

It attracts new users every day and their satisfaction continuously increases.
Customers’ critical business processes rely more and more on Airflow.
As an Airflow Community, we should aim to support Airflow’s flexibility and hide the complexity (where possible)
Let’s keep the momentum of Airflow and work towards
○ Improved autohealing & fault tolerance
○ Simplified Airflow configuration
○ Feature deprecation control
○ DAG and User Isolation
Thank you!

Rafal Biegacz (he/him)
Sr. Engineering Manager, Google
rbiegacz@

Filip Knapik
Group Product Manager, Google
FilipKnapik1@