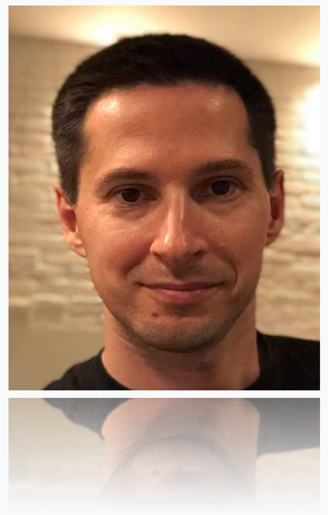
Airflow in the Cloud: Lessons from the field

Airflow Summit, May 2022

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



Filip Knapik
Group Product Manager, Google
FilipKnapik1@

Fill in Airflow Community Survey: https://bit.ly/AirflowSurvey22

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

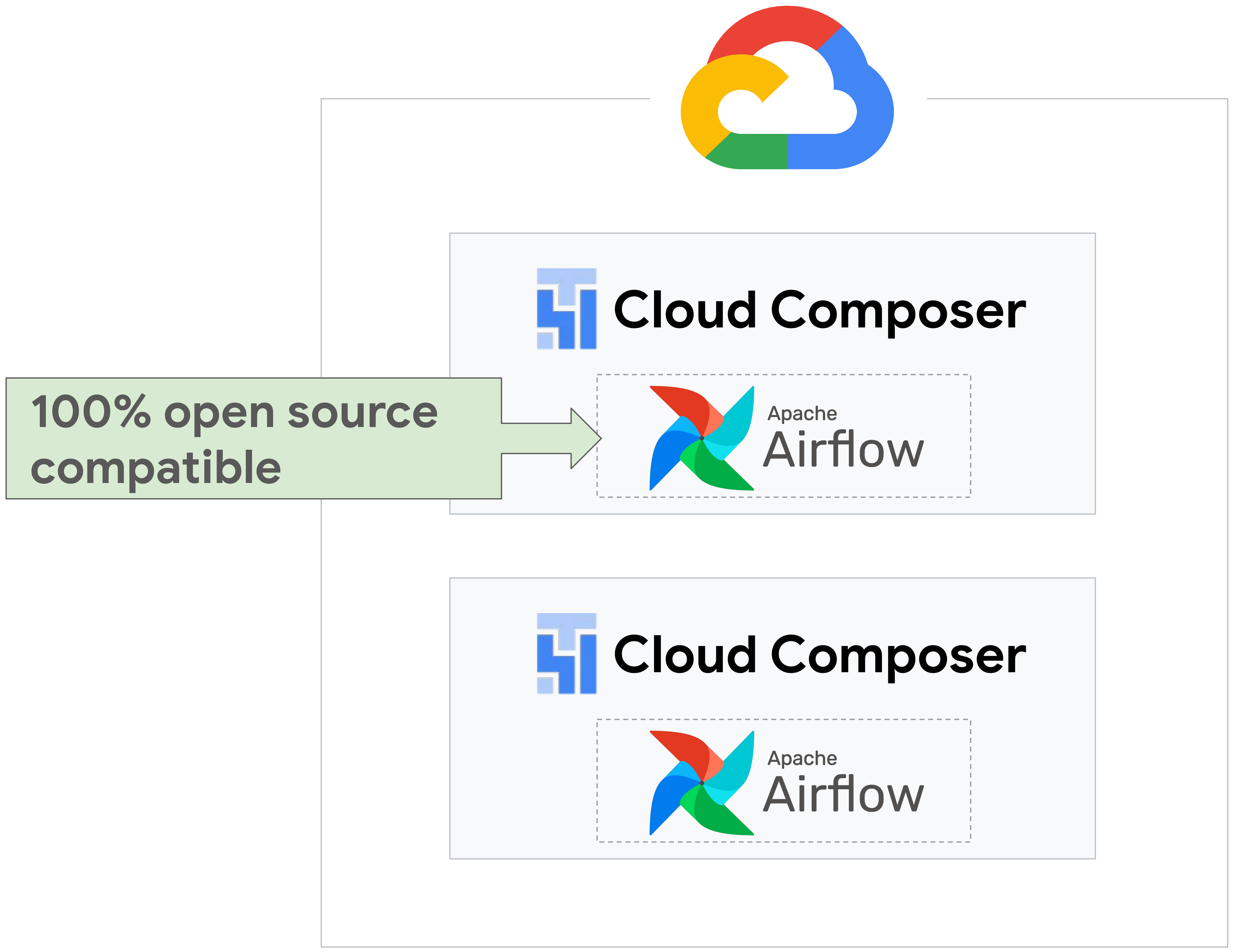
BIO

Filip Knapik

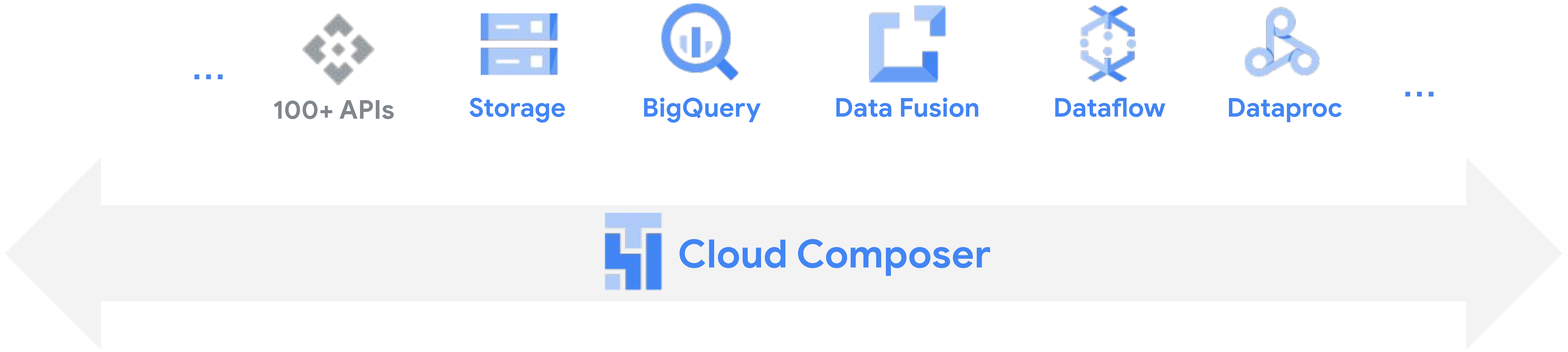
- Cloud Composer Group Product Manager
- Working with Airflow for ~3 years
- 18+ years of IT management experience
- MSc in Computer Networks and Services at AGH University of Science and Technology in Cracow, Poland

Super-quick intro on Composer

Airflow and Composer

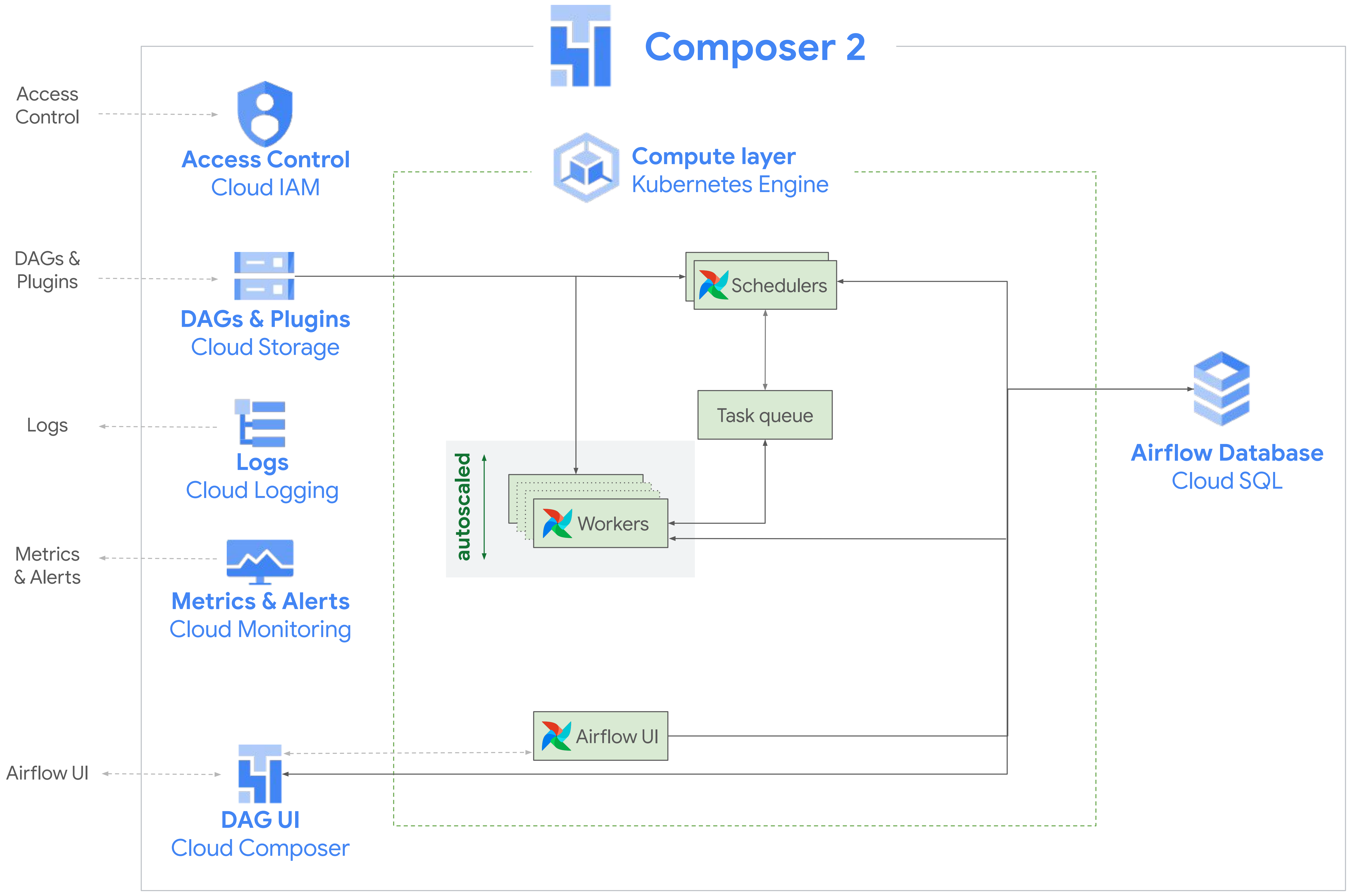


What is Composer used for?



Orchestrate work across **Google Cloud**, external **SaaS services** and **proprietary APIs**

Cloud Composer as a managed Airflow



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

Google

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...

AIRFLOW__API__FALLBACK_PAGE_LIMIT
AIRFLOW__SMTP__SMTP_RETRY_LIMIT
AIRFLOW__API__ACCESS_CONTROL_ALLOW_HEADERS
AIRFLOW__WEBSERVER__PAGE_SIZE
AIRFLOW__CELERY__TASK_TRACK_STARTED
AIRFLOW__CORE__ENABLE_XCOM_PICKLING
AIRFLOW__DATABASE__SQL_ALCHEMY_SCHEMA
AIRFLOW__CORE__DEFAULT_TIMEZONE
AIRFLOW__KUBERNETES__TCP_KEEP_INTVL
AIRFLOW__OPERATORS__DEFAULT_CPUS
AIRFLOW__SCHEDULER__PRINT_STATS_INTERVAL
AIRFLOW__KUBERNETES__DELETE_WORKER_PODS
AIRFLOW__SCHEDULER__ORPHANED_TASKS_CHECK_INTERVAL
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN
AIRFLOW__CORE__MAX_ACTIVE_RUNS_PER_DAG
AIRFLOW__LOGGING__GOOGLE_KEY_PATH
AIRFLOW__EMAIL__DEFAULT_EMAIL_ON_FAILURE
AIRFLOW__CORE__FERNET_KEY
AIRFLOW__KUBERNETES__NAMESPACE
AIRFLOW__CLI__API_CLIENT
AIRFLOW__CORE__DAGBAG_IMPORT_ERROR_TRACEBACK_DEPTH
AIRFLOW__WEBSERVER__AUTO_REFRESH_INTERVAL
AIRFLOW__WEBSERVER__DAG_ORIENTATION
AIRFLOW__WEBSERVER__INSTANCE_NAME
AIRFLOW__WEBSERVER__ERROR_LOGFILE
AIRFLOW__LOGGING__COLORED_LOG_FORMAT
AIRFLOW__LOGGING__REMOTE_BASE_LOG_FOLDER
AIRFLOW__WEBSERVER__HIDE_PAUSED_DAGS_BY_DEFAULT
AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__VISIBILITY_TIMEOUT
AIRFLOW__CELERY__RESULT_BACKEND_CMD
AIRFLOW__CORE__MAX_NUM_RENDERED_TI_FIELDS_PER_TASK
AIRFLOW__EMAIL__SUBJECT_TEMPLATE
AIRFLOW__WEBSERVER__BASE_URL
AIRFLOW__CELERY__CELERY_CONFIG_OPTIONS
AIRFLOW__WEBSERVER__WEB_SERVER_PORT
AIRFLOW__SCHEDULER__SCHEDULER_IDLE_SLEEP_TIME
AIRFLOW__WEBSERVER__PROXY_FIX_X_FOR
AIRFLOW__SMTP__SMTP_USER
AIRFLOW__SCHEDULER__USE_ROW_LEVEL_LOCKING
AIRFLOW__DATABASE__SQL_ALCHEMY_POOL_SIZE
AIRFLOW__CELERY__WORKER_UMASK
AIRFLOW__CELERY__FLOWER_BASIC_AUTH
AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL
AIRFLOW__KUBERNETES__TCP_KEEP_CNT
AIRFLOW__WEBSERVER__COOKIE_SECURE
AIRFLOW__CELERY__TASK_PUBLISH_MAX_RETRIES
AIRFLOW__LOGGING__COLORED_CONSOLE_LOG
AIRFLOW__WEBSERVER__NAVBAR_COLOR
AIRFLOW__CELERY__WORKER_AUTOSCALE
AIRFLOW__SCHEDULER__SCHEDULER_HEALTH_CHECK_THRESHOLD
AIRFLOW__EMAIL__HTML_CONTENT_TEMPLATE
AIRFLOW__GITHUB_ENTERPRISE__API_REV
AIRFLOW__LOGGING__LOG_FILENAME_TEMPLATE
AIRFLOW__CELERY__RESULT_BACKEND_SECRET
AIRFLOW__WEBSERVER__WORKERS
AIRFLOW__WEBSERVER__SECRET_KEY_SECRET
AIRFLOW__CELERY__SSL_CERT
AIRFLOW__SMTP__SMTP_HOST
AIRFLOW__WEBSERVER__ANALYTICS_TOOL
AIRFLOW__KUBERNETES__WORKER_PODS_CREATION_BATCH_SIZE
AIRFLOW__SCHEDULER__ALLOW_TRIGGER_IN_FUTURE
AIRFLOW__SCHEDULER__TRIGGER_TIMEOUT_CHECK_INTERVAL
AIRFLOW__LOGGING__DAG_PROCESSOR_MANAGER_LOG_LOCATION
AIRFLOW__LOGGING__LOGGING_CONFIG_CLASS
AIRFLOW__WEBSERVER__WORKER_REFRESH_INTERVAL
AIRFLOW__DATABASE__SQL_ALCHEMY_POOL_RECYCLE
AIRFLOW__WEBSERVER__DEFAULT_DAG_RUN_DISPLAY_NUMBER
AIRFLOW__SCHEDULER__MAX_DAGRUNS_TO_CREATE_PER_LOOP
AIRFLOW__SCHEDULER__IGNORE_FIRST_DEPENDS_ON_PAST_BY_DEFAULT
AIRFLOW__CORE__DEFAULT_POOL_TASK_SLOT_COUNT
AIRFLOW__WEBSERVER__ANALYTICS_ID
AIRFLOW__DEBUG__FAIL_FAST
AIRFLOW__DATABASE__SQL_ALCHEMY_MAX_OVERFLOW
AIRFLOW__CORE__HIDE_SENSITIVE_VAR_CONN_FIELDS
AIRFLOW__WEBSERVER__LOG_FETCH_TIMEOUT_SEC
AIRFLOW__WEBSERVER__SESSION_BACKEND
AIRFLOW__CELERY__WORKER_CONCURRENCY
AIRFLOW__CORE__PLUGINS_FOLDER
AIRFLOW__SMTP__SMTP_SSL
AIRFLOW__CORE__TASK_RUNNER
AIRFLOW__CORE__SECURITY
Made In wordc.net

...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

source: https://giphy.com/

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

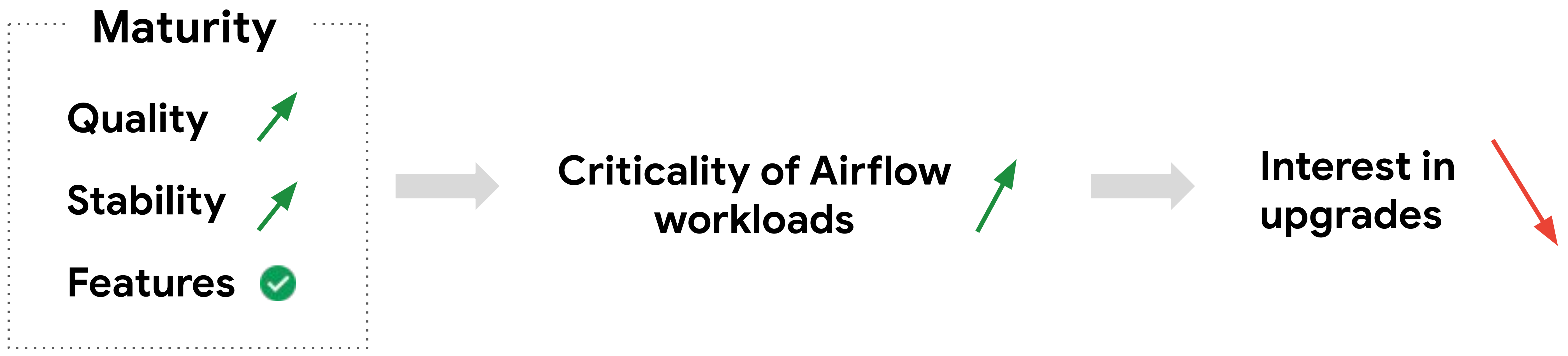
Google

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



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.

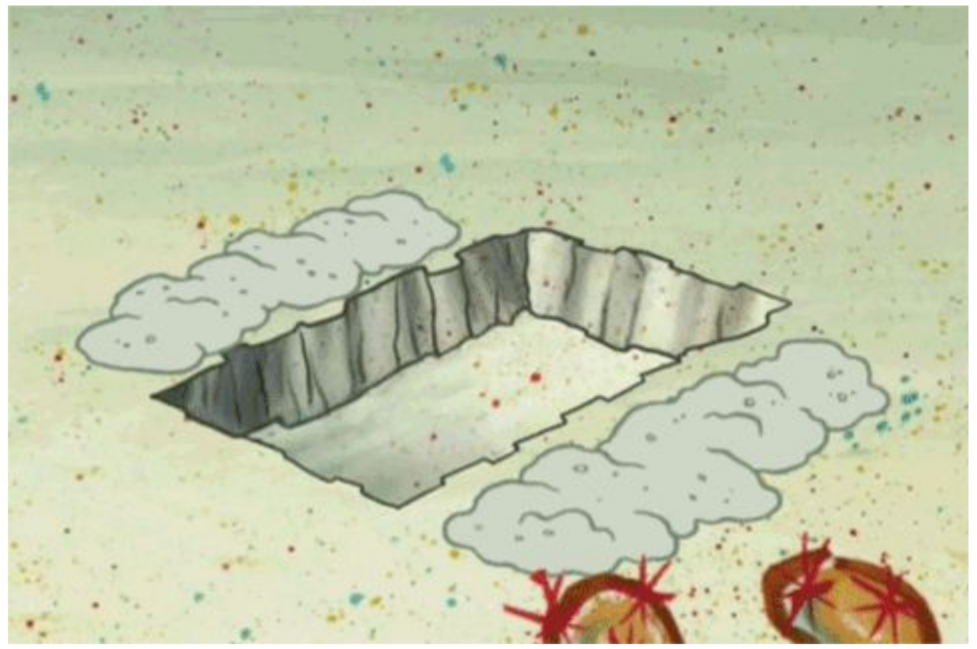
Deprecations prevent new versions adoption

Challenging deprecations

- Operators
- Airflow configuration parameters
- Airflow connection names

Deprecation messages litter the logs

Airflow 1 vs 2 incompatibility holds customers in Airflow 1


source: https://giphy.com/

Feature deprecations need to be smooth, especially requiring DAG code changes

Google

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

Business impact

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.

Google

Airflow is **amazing!**

It attracts new users every day and their satisfaction continuously increases.

Customers' critical business processes rely more and more on Airflow.

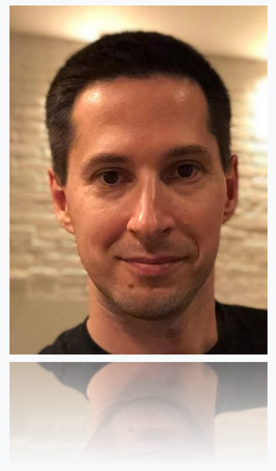
As an Airflow Community, we should aim to support Airflow’s flexibility and hide the complexity (where possible)

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@