Snap's Airflow Story

Zhengyi Liu, Han Gan, Yuri Desyatnik, Nanxi Chen

Airflow Summit
Let's flow together
September 19-21, 2023,
Toronto, Canada

Who are we

Zhengyi Liu

Engineering Manager @ Snap

Han Gan

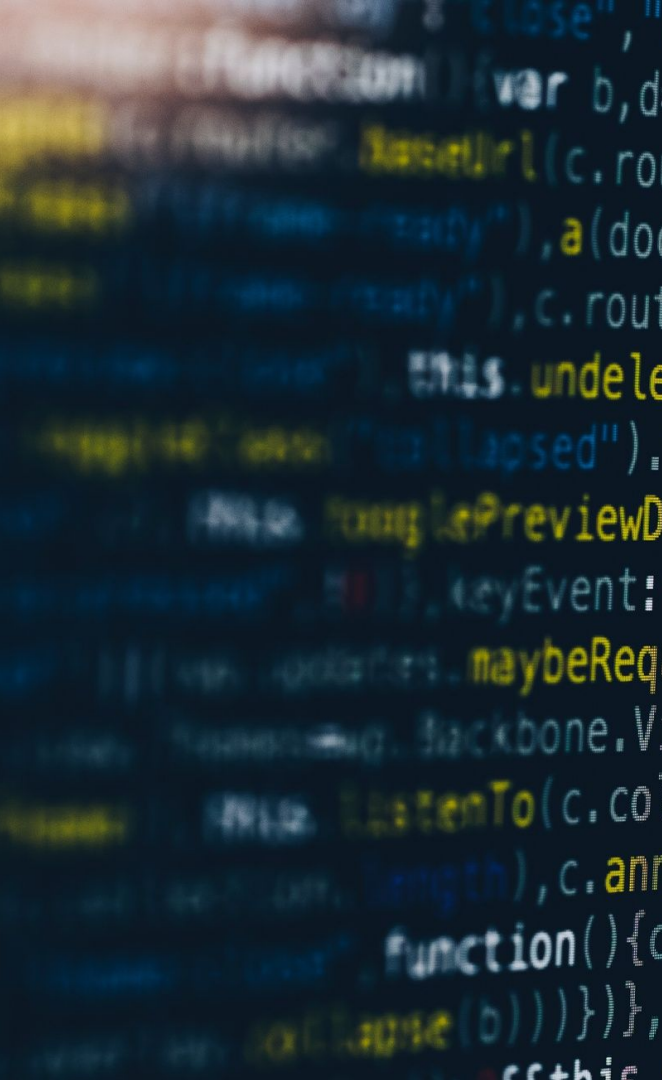
Software Engineer @ Snap

Yuri Desyatnik

Sr Security TPM @ Snap

Nanxi Chen

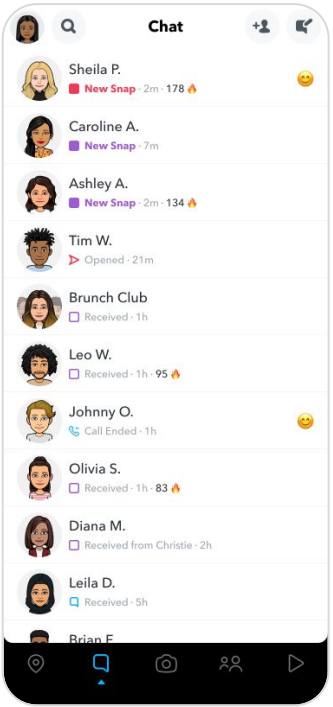
Privacy Engineer @ Snap



What we will cover today

1. Introduction - our story
2. Architecture choices
3. Securing our Airflow deployment
4. Tough part - migration!

Who is Snap Inc.?



Scale of Airflow @ Snap

3000 DAGs

330K Task Instances / Day

200+ Operators

1000+ Active Users

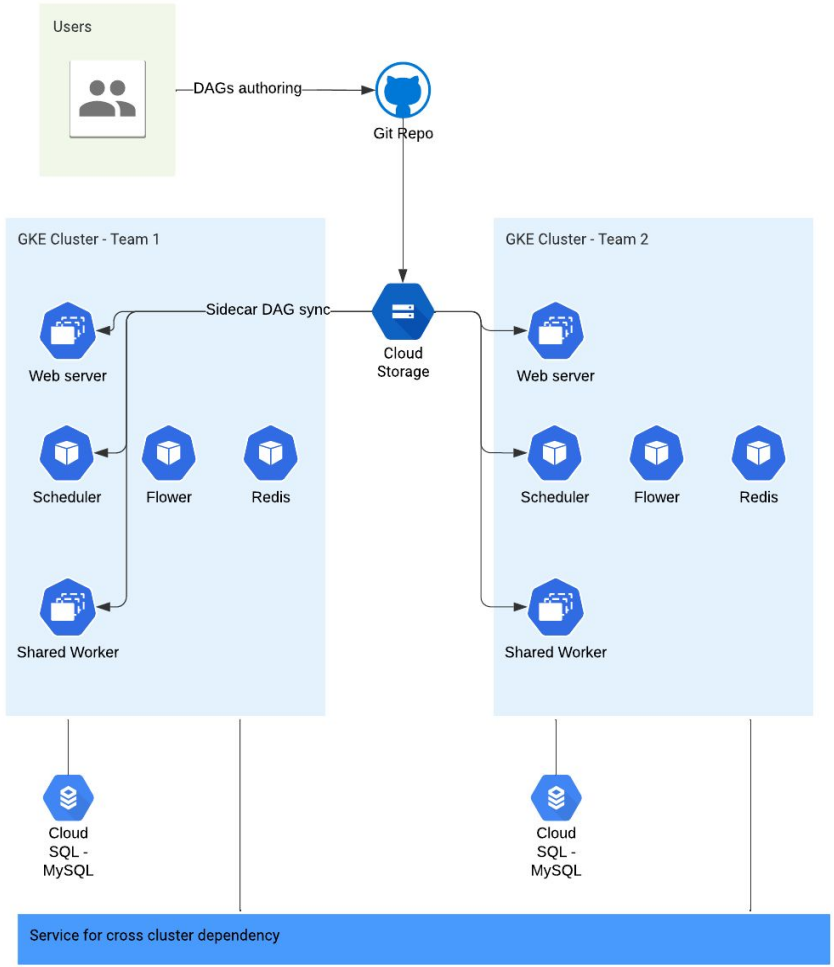

START
2016
Built the first Airflow deployment with slightly less than 100 DAGs
2018
Multiple Airflow deployments on GKE for isolations. It soon grows to 50+! Very hard to manage with a lean team.
2019
Built a task level access control model with code integration. DAG count grew from few hundreds to 2000+, managing task level permissions was painful.
Challenges
At this moment, there are multiple challenges regarding infra/software maintenance, permission management, discoverability, etc.
2022
Launch Airflow 2 side by side with brand new security model and toolings.
2023
Embraced Airflow 2+ and migrated teams over

Architecture Choices

	Single cluster	Multiple single-tenant clusters	Multi-tenant cluster	Multiple multi-tenant clusters
Maintainability	⭐⭐⭐	⭐	⭐⭐⭐	⭐
Scalability	⭐⭐	⭐⭐⭐	⭐⭐	⭐⭐⭐
Isolation/Security	⭐	⭐⭐	⭐⭐⭐	⭐⭐⭐

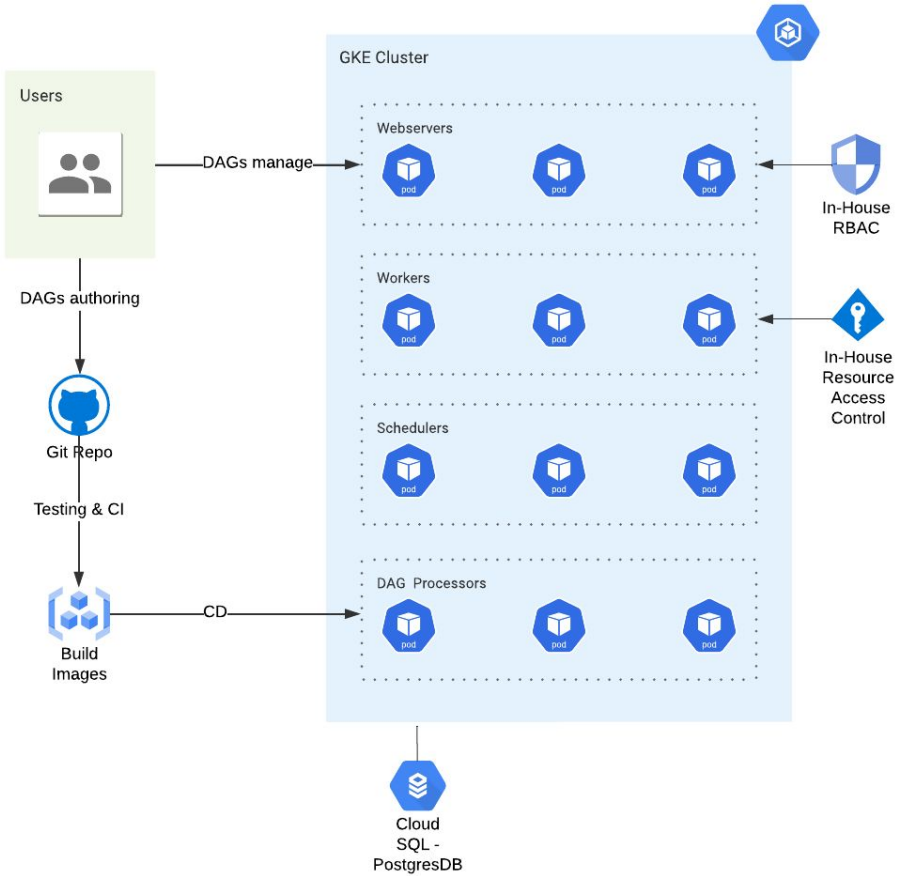
Previous Architecture

- Multiple clusters - **poor DAG discoverability and extra service for cross cluster dependencies**
- Sidecars to sync DAG code from GCS - **sometime inconsistent and difficult to track deployment**
- Celery executor with shared worker - **no flexibility in scale and runtime environment**



Current Architecture

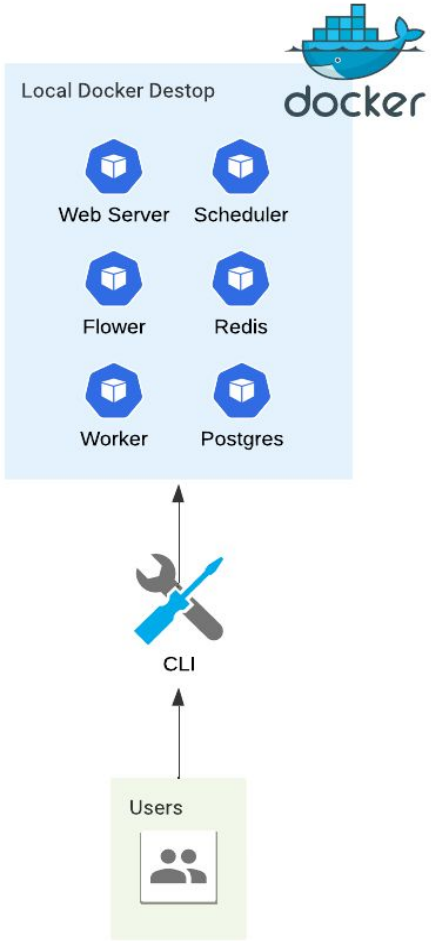
- Heavily leveraging Kubernetes and embrace Airflow Kubernetes executor
- Enforced team level RBAC & pod level resource access control
- Number of tenants increased from 65+ to 125+



From Local to Remote Dev

Docker Desktop to host Airflow server in local is convenient, but..

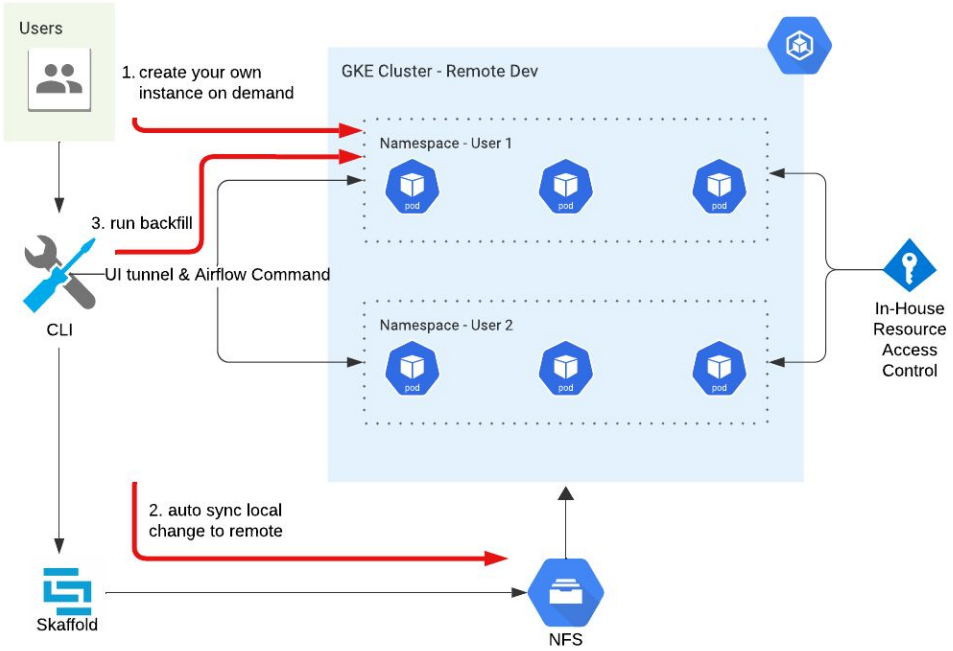
- slow due to limited laptop resource
- hard to manage resource access permission
- inconsistent behavior with production



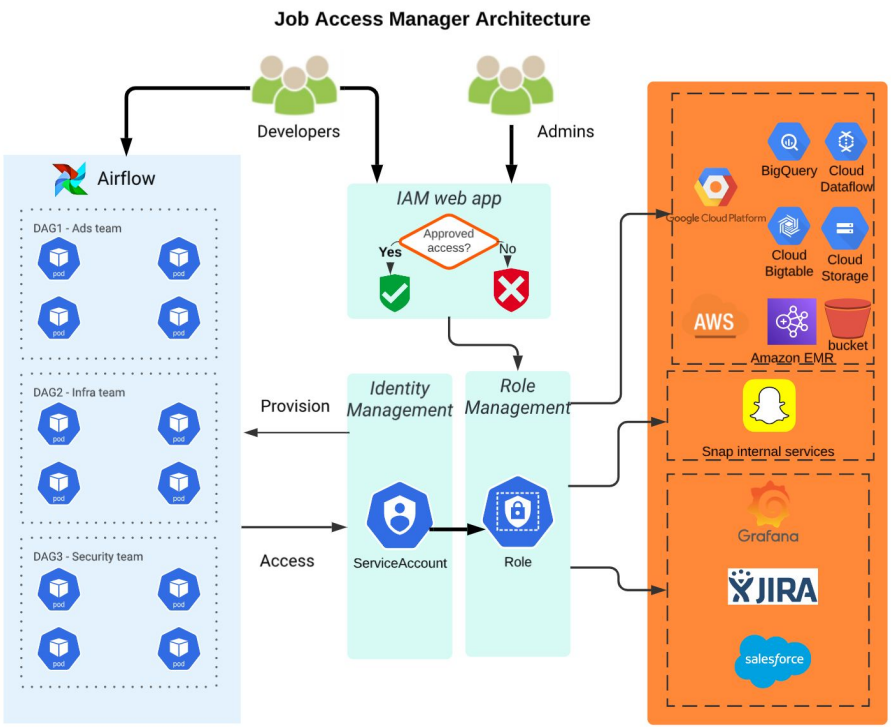
Remote Dev

- Leverage Skaffold for faster dev iteration in remote GKE - auto sync local files change to remote NFS on the fly

- Manage resource access with the same in-house tooling



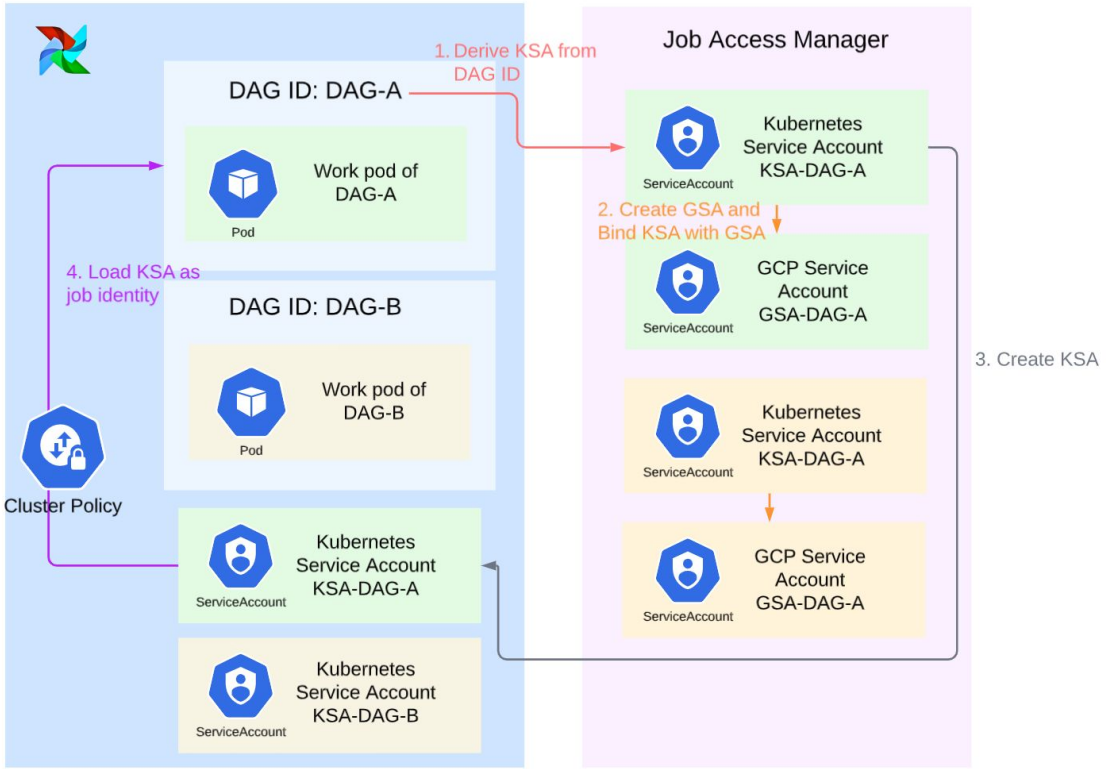
Job Access Manager Architecture



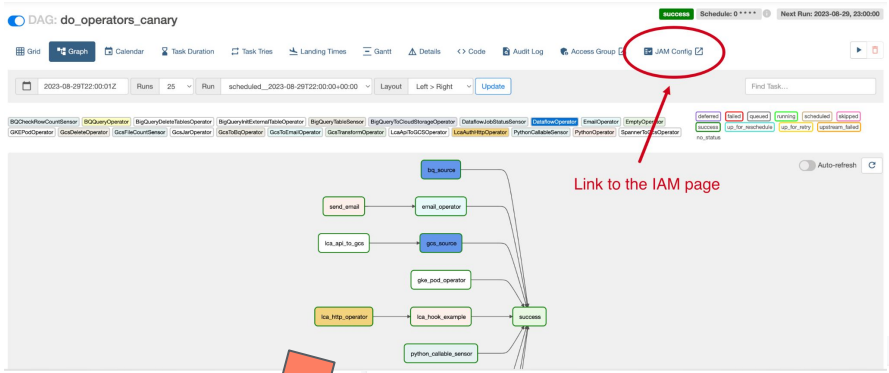
- One service account per DAG
- One-stop access management: cloud resources & internal/external services
- Job profile
- ACL management
- Access review

Workload Identity

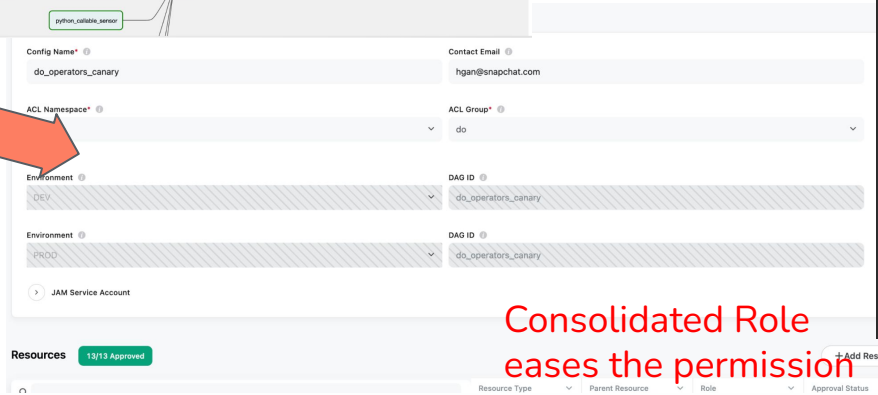
- Leverage workload identity to isolate permission footprint on each worker pod
- No credentials / keys store on disk nor in the Airflow database



Streamline access management



Deep integration with cloud IAM



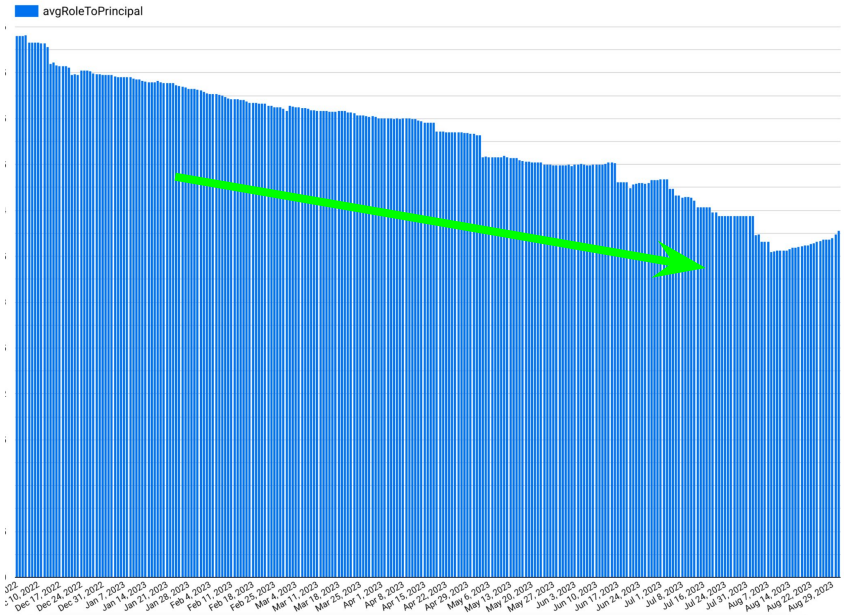

```
extendedList.add(
    new IamService.IamPolicy(
        iamPolicy.principal,
        RoleConstants.DATAFLOW_ADMIN_ROLE,
        iamPolicy.resource.resourceId,
        iamPolicy.resource.parentResource,
        iamPolicy.resource.resourceScope));
extendedList.add(
    new IamService.IamPolicy(
        iamPolicy.principal,
        RoleConstants.DATAFLOW_WORKER_ROLE,
        iamPolicy.resource.resourceId,
        iamPolicy.resource.parentResource,
        iamPolicy.resource.resourceScope));
// Allow cross project service account to be dataflow worker service account
extendedList.add(
    new IamService.IamPolicy(
        iamPolicy.principal,
        RoleConstants.SERVICE_ACCOUNT_USER_ROLE,
        iamPolicy.principal,
        Project.SC_JAM,
        ResourceScope.GCP_SA));
extendedList.add(
    new IamService.IamPolicy(
        computeAgent,
        RoleConstants.SERVICE_ACCOUNT_TOKEN_CREATOR_ROLE,
        iamPolicy.principal,
        Project.SC_JAM,
        ResourceScope.GCP_SA));
extendedList.add(
    new IamService.IamPolicy(
        computeAgent,
        RoleConstants.SERVICE_ACCOUNT_USER_ROLE,
        iamPolicy.principal,
        Project.SC_JAM,
        ResourceScope.GCP_SA));
extendedList.add(
    new IamService.IamPolicy(
        dataflowAgent,
        RoleConstants.SERVICE_ACCOUNT_TOKEN_CREATOR_ROLE,
        iamPolicy.principal,
        Project.SC_JAM,
        ResourceScope.GCP_SA));
```

Consolidated Role eases the permission management

PRJ	blizzard-operation	blizzard-operation	roles/dataflow.jam	Approved	View Admins	Delete
PRJ	blizzard-operation	blizzard-operation	roles/dataflow.jam	Approved	View Admins	Delete
PRJ	flowrida-staging	flowrida-staging	roles/bigquery.dataOwner	Approved	View Admins	Delete

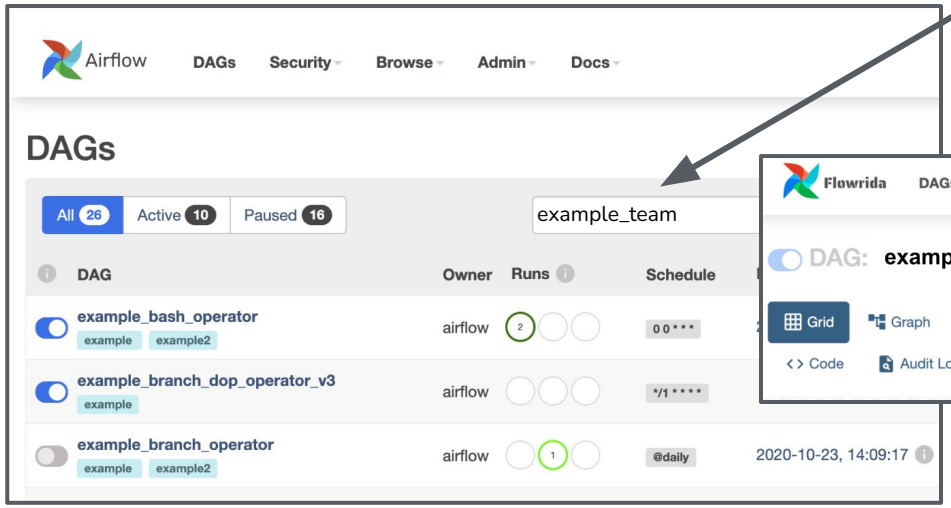
Permission reduction

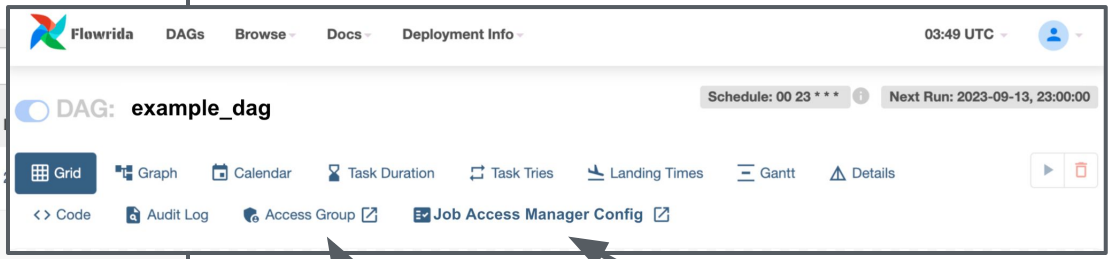
- To enforce Least Privilege Principle
- DAGs are isolated by using exclusive SA for each DAG
- DAGs are periodic. Permission is high likely not needed if it is unused after several DAG run intervals.
- Very helpful after migration. Earlier permissions are over-provisioned for one DAG as SA is shared by multiple DAGs.



Security - RBAC

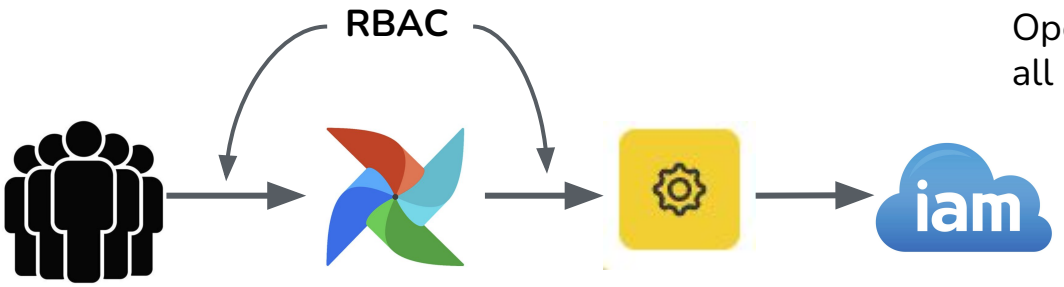
Access group name shows only DAGs they own





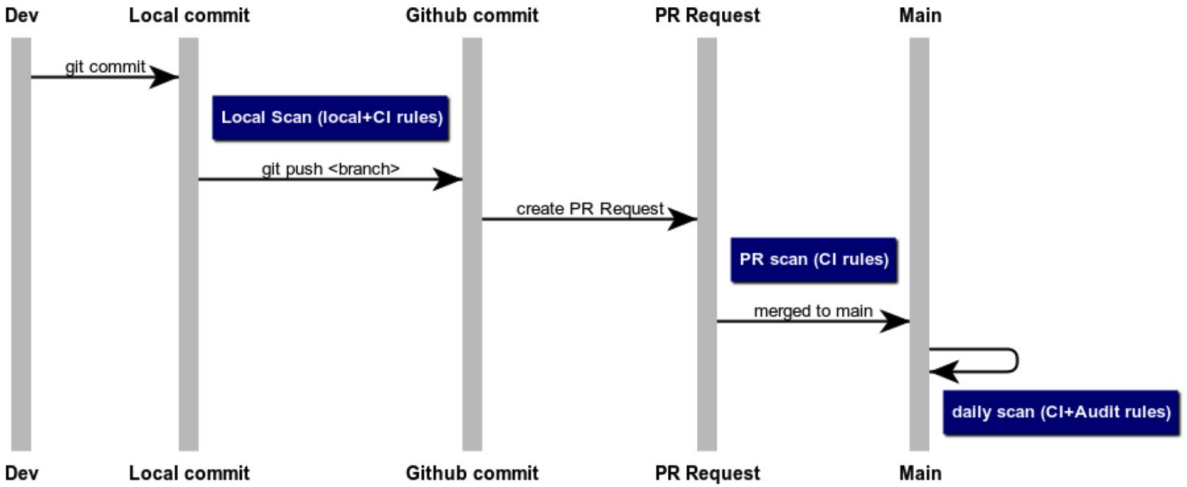
Opens IAM UI with all group members

Opens Job Access Manager UI with all resources owned by access group



DAG code security

CI/CD checks ensure DAGs are free from common security vulnerabilities



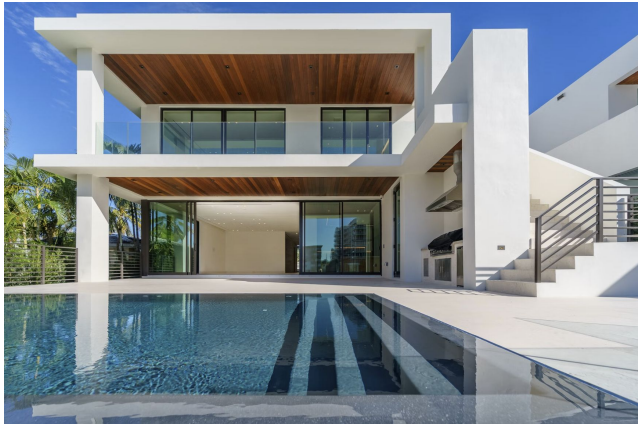
- Pre-commit scan - for monitoring branch commits
- PR scan - for monitoring commits to main
- Daily scan of main - to prevent vulnerabilities introduced outside CI/CD

Migration Challenges

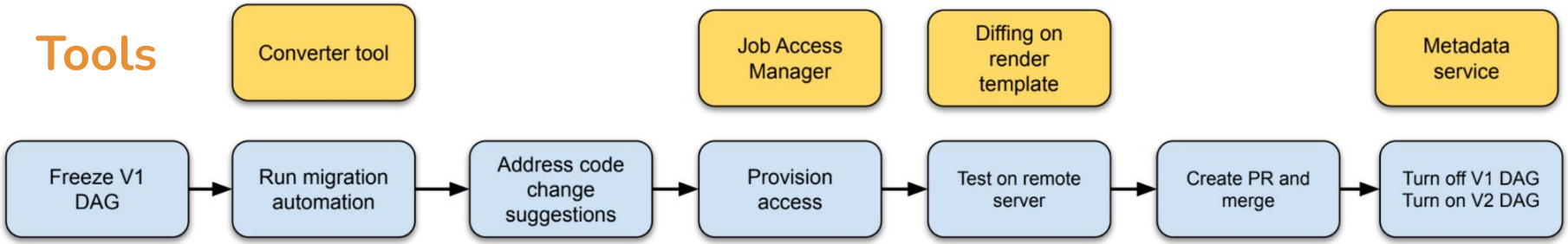
- Engineering resources
  - DAG owners are busy people
  - How to entice Airflow customers to move?

- Operator availability
  - New secure operators have to be created
  - It’s hard to make some operators secure (e.g. GKEPodOperator)

- Migration efficiency
  - How to make migration simple, fast and error-free?
  - How to organize, engage and facilitate customer team migrations?



Migration Flow



Goals for migration process:

- Ease of migration / user experience
- Customer feels supported
- Zero negative production outcomes

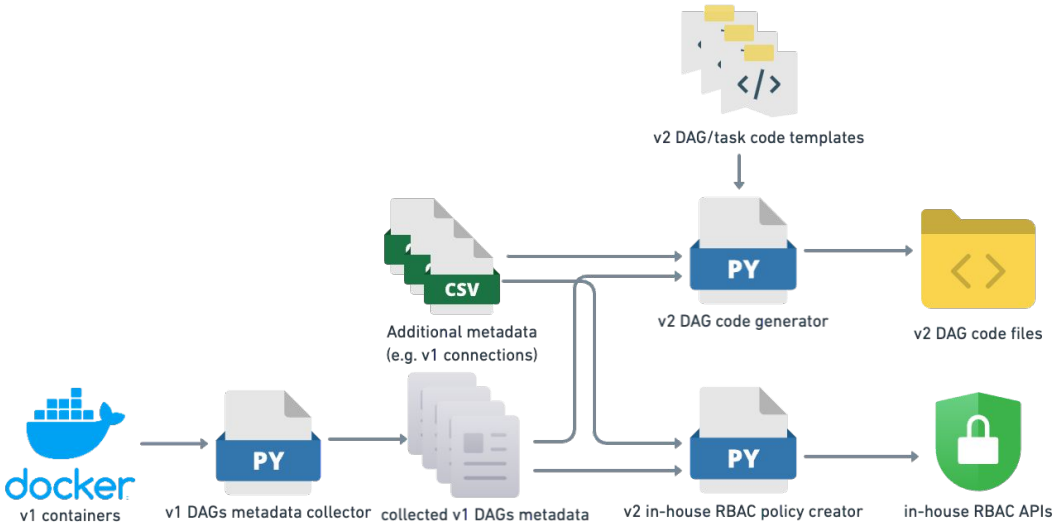
Tools

- Converter - code changes
- Job Access Manager - add permissions/roles base on old service account
- Diffing on render template to confirm new DAG works
- Metadata service - allow Airflow v1 and v2 external task sensors to poke across for clusters dependencies

DAG Generation from Metadata

- Collect metadata from old DAG to generate v2 code and permissions
- Work great for operators with limited custom logic

Worked for ~40% of DAGs



Takeaways

- Infrastructure
  - Multi-tenant cluster
  - Remote server for testing and backfill
- Security
  - One service account per DAG
  - Mapped to workload identity of execution pod
  - RBAC for UI and service account access
  - DAG code CI/CD scanning
- Migration
  - Maximum automation
  - Positive customer engagement
  - Flexibility with approach to different customers
  - Executive support

Airflow 2.x

Managed K8s

RBAC

Job access isolation

DAG static analysis

Questions?

Optionally share some contact info like email, blog or social media handles