Airflow CI/CD: Github to Composer (easy as 1, 2, 3)

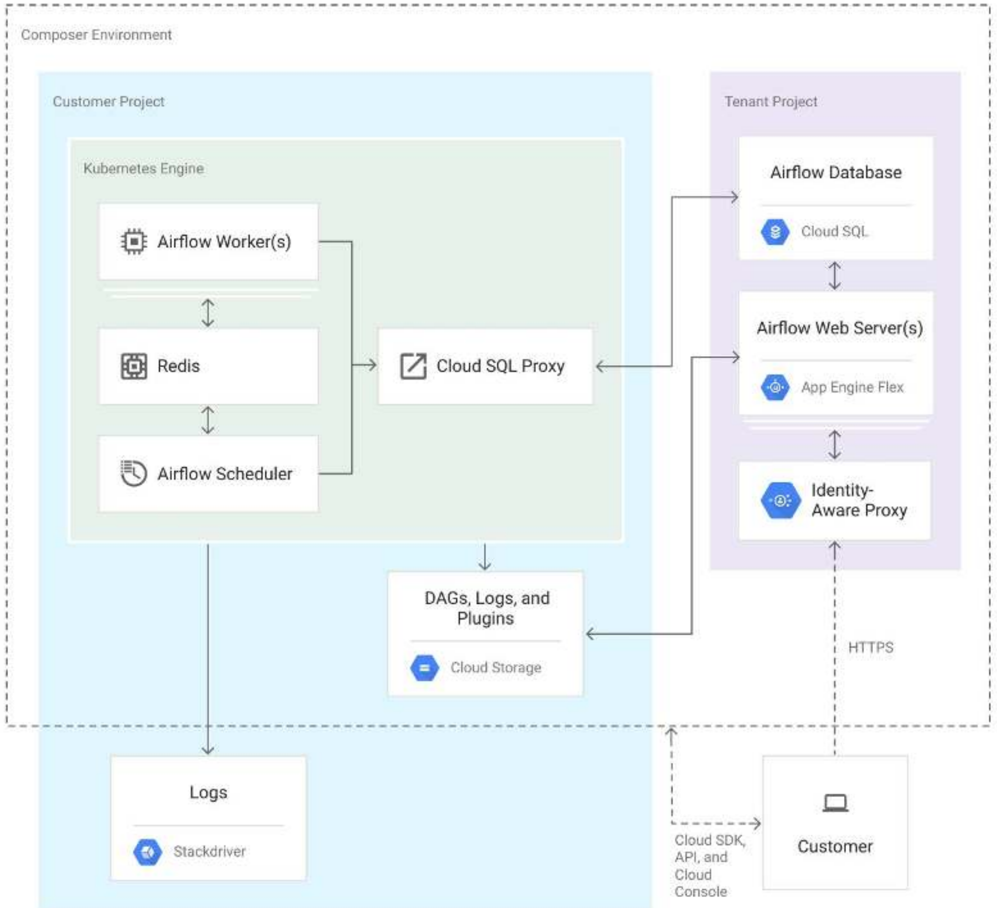
Google Cloud

Speaker: Jake Ferriero Email: jferriero@google.com
Github: jaketf@
Source:
https://github.com/jaketf/ci-cd-for-data-processing-workflow
July 2020

Composer Basics

Airflow Architecture

- Storage (GCS)
  - Code artifacts
- Kubernetes (GKE)
  - Workers
  - Scheduler
  - Redis (Celery Queue)
- AppEngine (GAE)
  - Webserver / UI
- Cloud SQL
  - Airflow Metadata Database



Google Cloud

GCS Directory mappings

GCS “folder”	Mapped Local Directory	Usage	Sync type
gs://{composer-bucket}/dags	/home/airflow/gcs/dags	DAGs (SQL Queries)	Periodic 1-way rsync (workers / web-server)
gs://{composer-bucket}/plugins	/home/airflow/gcs/plugins	Airflow plugins (Custom Operators / Hooks etc.)	Periodic 1-way rsync (workers / web-server)
gs://{composer-bucket}/data	/home/airflow/gcs/data	Workflow-related data	GCSFUSE (workers only)
gs://{composer-bucket}/logs	/home/airflow/gcs/logs	Airflow task logs (should only read)	GCSFUSE (workers only)

Google Cloud

Testing Pipelines

1

CI/CD for Composer == CI/CD for everything it Orchestrates

- Often Airflow is used to manage a series of tasks that themselves need a CI/CD Process
    - ELT Jobs: BigQuery
        - dry run your SQL, unit test your UDFs
        - deploy SQL to dags folder so parseable by workers and webserver
    - ETL Jobs: Dataflow / Dataproc Jobs
        - run unit tests and integration tests with a build tool like maven.
        - deploy artifacts (JARs) to GCS

Google Cloud

DAG Sanity Checks

- Python Static Analysis (flake8)
- Unit / Integration tests on custom operators
- Unit test that runs on all DAGs to assert best practices / auditability across your team.
- Example Source `test_dag_validation.py`:
    - DAGs parse w/o errors
        - catches a plethora of common “referencing things that don’t exist errors” e.g. files, Variables, Connections, modules, etc.
    - DAG Parsing < threshold (2 seconds)
    - No dags in running_dags.txt missing or ignored
    - (opinion) Filename == Dag ID for tracability
    - (opinion) All DAGs have an owners email with your domain name.

Inspired by: “Testing in Airflow Part 1 — DAG Validation Tests, DAG Definition Tests and Unit Tests” - Chandu Kavar

Google Cloud

Integration Testing with Composer

- A popular failure mode for a DAG is referring to something in the target environment that does not exist:
  - Airflow Variable
  - Environment Variable
  - Connection ID
  - Airflow Plugin
  - pip dependency
  - SQL / config file expected on workers' / webserver's filesystem
- Most of these can be caught by staging DAGs in some directory and running `list_dags`
  - In Composer we can leverage the fact that the `data/` path on GCS is synced to the workers' local file system

```
$ gsutil -m cp ./dags \
    gs://<composer-bucket>/data/test-dags/<build-id>


$ gcloud composer environments run \
    <environment> \
    list_dags -- -sd \
    /home/airflow/gcs/data/test-dags/<build-id>/
```

Google Cloud

Deploying DAGs to Composer

2

Google Cloud

Deploying a DAG to Composer: High-Level

1. Stage all artifacts required by the DAG
   a. JARs for Dataflow jobs to known location GCS
   b. SQL queries for BigQuery jobs (somewhere under `dags/` folder and ignored by `.airflowignore`)
   c. Set Airflow Variables referenced by your DAG
2. (Optional) delete old (versions of) DAGs
   a. This should be less of a problem in an airflow 2.0 world with DAG versioning!
3. Copy DAG(s) to GCS `dags/` folder
4. Unpause DAG(s) (assuming best practice of `dags_paused_on_creation=True`)
   a. New Challenge: But now I have to unpause each DAG which sounds exhausting if deploying many DAGs at once
   b. This may require a few retries during the GCS -> GKE worker sync. Enter `deploydags` application...

Deploying a DAG to Composer: deploydags app

A simple golang application to orchestrate the deployment and sunsetting of DAGs by taking the following steps:

= airflow CLI
* = Need for concurrency

1. list_dags
2. compare to a running_dags.txt config file of what "should be running"
   a. Allows you to keep a DAG in VCS you don't wish to
3. validate that running DAGs match source code in VCS
   a. GCS filehash comparison
   b. (Optional) -replace Stop and redeploy new DAG with same name
4. * Stop DAGs
   a. pause
   b. delete source code from GCS
   c. * delete_dag
5. * Start DAGs
   a. Copy DAG definition file to GCS
   b. * unpause

Need to concurrency to stop / deploy many DAGs quickly

Need to be retried (for minutes not seconds) until successful due to GCS -> worker rsync process

Google Cloud

Stitching it all together with Cloud Build

3

Google Cloud

Cloud Build is not perfect!

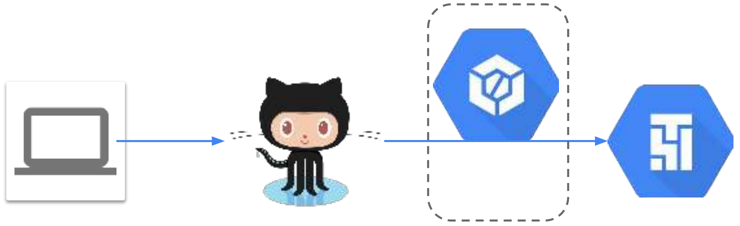
- Most of the tooling built for this talk is not Cloud Build specific :) bring it into your favorite CI tooling

- Cloud Build is great
  - Managed / no-ops / serverless (easy to get started / maintain compared to more advanced tooling like Jenkins / Spinnaker etc.)
  - Better than nothing
  - No need to contract w/ another vendor

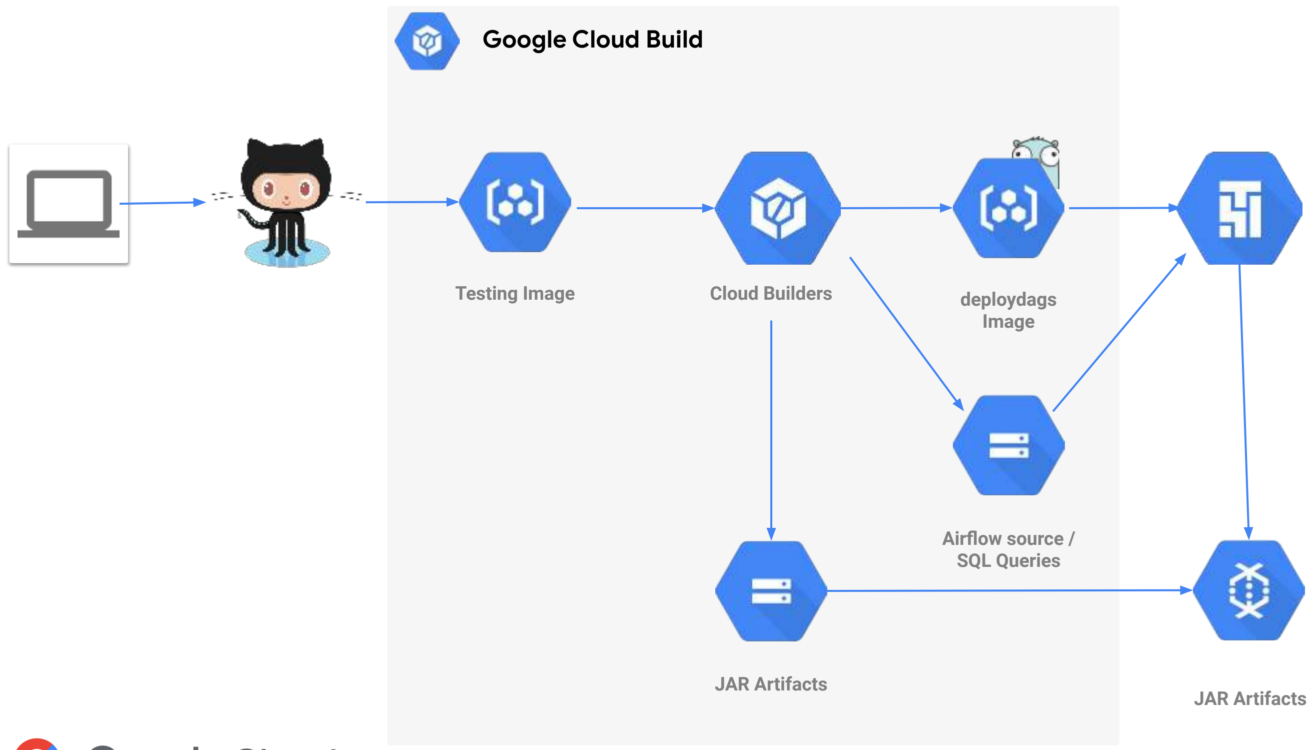
- Cloud Build has painful limitations for being a full CI solution:
  - Only /gcbrun triggers
    - not easy to have multiple test suites gated on different reviewer commands
  - No out of the box advanced queueing mechanics for preventing parallel builds
  - Does not have advanced features around “rolling back” (though you can always revert to old commit and run the build again)
  - Does not run in your network so need some public access to Airflow infrastructure (e.g. public GKE master or through bastion host)

Cloud Build with Github Triggers

- Github Triggers allow you to easily run integration tests on a PR branch
  - Optionally gated with “/gcbrun” comment from a maintainer.
    - Pre-commit automatically runs
    - Post-commit comment gated
- Cloud Build has convenient Cloud Builders for
  - Building artifacts
    - Running mvn commands
    - Building Docker containers
  - Publishing Artifacts to GCS / GCR
    - JARs, SQL files, DAGs, config files
  - Running gcloud commands
  - Running tests or applications like deploydags in containers

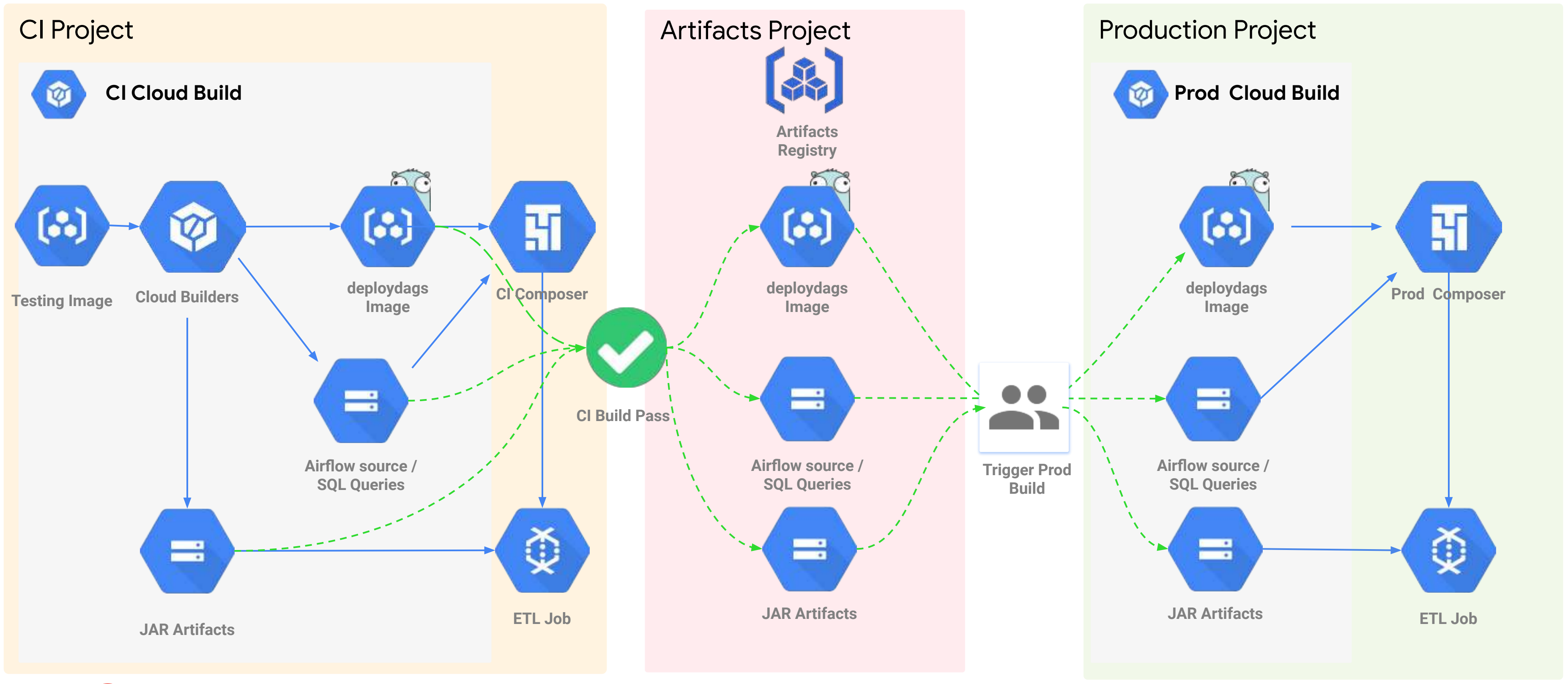


Cloud Build with Github Triggers for CI



Google Cloud

Isolating Artifacts and Push to Prod



Google Cloud

Cloud Build Demo

- Let’s validate a PR to Deploy N new DAGs that orchestrate BigQuery jobs and Dataflow jobs
  - Static Checks (runs over whole repo)
  - Unit tests (defined in precommit_cloudbuild.yaml in each dir which is run by `run_relevant_cloudbuilds.sh` if any files in this dir were touched)
  - Deploy necessary artifacts to GCS / GCR
  - DAG parsing tests (w/o error and speed)
  - Integration tests against target Composer Environment
  - Deploy to CI Composer Environment
- This similar cloudbuild.yaml could be invoked with substitutions for the production environment values for deploy to prod (pulling the artifacts from the artifact registry project).
- Source: https://github.com/jaketf/ci-cd-for-data-processing-workflow

Google Cloud

Future Work

3+

Google Cloud

Future Work

- CI Composer shouldn't cost this much and we need to Isolate CI tests
  - Ephemeral composer CI environments per test (SLOW)
    - Working hours CI environments though… :)
  - Acquire a "Lock" on the CI environment and queue ITs so they don't stomp on each other
    - Require a "wipeout CI environment" automation to reset the CI environment
- Security
  - Support deployments with only Private IP
  - Add support for managing airflow connections with CI/CD
- Portability
  - Generalize deploydags to run airflow cli commands with go client k8s exec to make this useful for non-composer deployments
- Examples
  - Different DAGs in different environments w/ multiple running_dags.txt configs (or one yaml)
  - Support "DAGs to Trigger" for DAGs that run system tests and poll to assert success
  - BigQuery EDW DAGs
  - Publish Solutions Page & Migrate repo to Google Cloud Platform GitHub Org

Contributions and Suggestions Welcome! Join the conversation in GitHub Issues
And join the community conversation on the new #airflow-ci-cd Slack Channel!

Thank you!

Special thanks to:

1. Google Cloud Professional Services for enabling me to work on cool things like this
2. Ben White for requirements and initial feedback
3. Iniyavan Sathiamurthi for his collaboration on POC implementation of similar concepts @ OpenX (check out his blog)
4. Airflow community leaders Jarek and Kamil for getting me excited about OSS contributions
5. My partner, Janelle for constant love and support

Google Cloud