Data lineage via Apache Airflow

Short bio Matteo

Matteo Senardi

* 2020-ongoing Head of Data in Docsity

* 2019-2020 Senior Data Engineer in Mediaset

* 2016-2019 Innovation Engineer in Webranking

* email: matteo.s@docsity.com

SCAN ME
https://github.com/pualien

Short bio Irene

Irene Soligno

* 2020-ongoing Data Scientist in Docsity

* 2019-2020 Data Scientist in Enginium

* 2015-2019 PhD and research fellow in POLITO (environmental engineering)

* email: irene.s@docsity.com

* ORCID: https://orcid.org/0000-0001-9884-5316

Founded in Italy in 2011 - Now present in 70 countries, with offices in Turin, Rome and Sao Paulo (Brazil).

Docsity is an EdTech company operating on an international scale that distributes and produces digital education content to help university students prepare for their exams.

docsity

Docsity also collaborates with over 150 universities and business schools from around the world, by promoting their educational programs through the Docsity student community (Business To Business - B2B).

Docsity has today 500K new registered users each month and 12M visitors per month.

Full-service approach: implement and optimize the whole student funnel.

WHY AIRFLOW?
part I

Let’s imagine - a very simple use case

01

PROBLEM

Decide if batch job should run today and how long to backfill

02

PROCESS

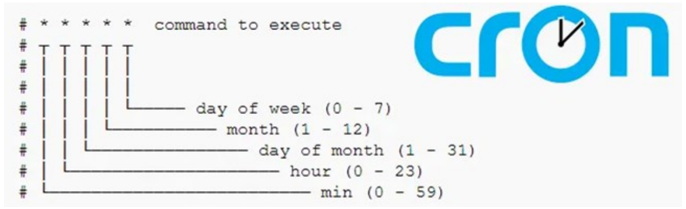
Data sources,
that need to be passed through data lake

03

TARGET

When all files are in place, submit the pre-defined job.

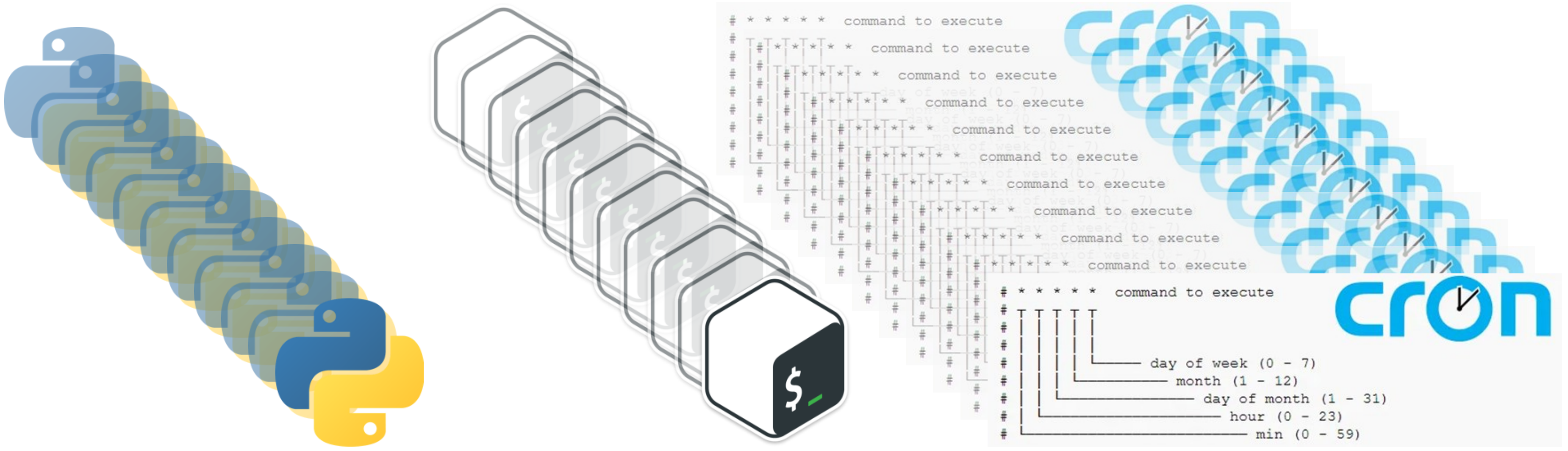
Let’s imagine - a very simple use case



```
# * * * * *  command to execute
# ┬ ┬ ┬ ┬ ┬
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ └───── day of week (0 - 7)
# │ │ │ └────────── month (1 - 12)
# │ │ └─────────────── day of month (1 - 31)
# │ └──────────────────── hour (0 - 23)
# └───────────────────────── min (0 - 59)
```

Scripting + Cron could do!

Let’s imagine - a very simple use case



```
# * * * * *  command to execute
# ┬ ┬ ┬ ┬ ┬
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ └───── day of week (0 - 7)
# │ │ │ └────────── month (1 - 12)
# │ │ └─────────────── day of month (1 - 31)
# │ └──────────────────── hour (0 - 23)
# └───────────────────────── min (0 - 59)
```

What if **hundreds** of workflows to be managed?

Let’s imagine - a very ~~simple~~ production use case

- Which features **are required** to manage production workflows?

MANAGE
Manage scripts and crontab for hundred of workflows

ENVIRONMENT
Jobs could have different requirements and environments

MONITOR
Track performance and completion of each step

EXECUTION
Consider performance in parallel

CONNECTION
Operational DBs, APIs, Cloud Services, ssh tunneling, all with their own configuration

RETRY
Re-run a specific step in case of failures or whatever

Apache Airflow

born in 2014 from airbnb

The technology to **automate** & **orchestrate** data pipelines

~1700 GITHUB CONTRIBUTORS

PYTHON BASED

MONITORING

DAGS AS CODE

FRIENDLY INTERFACE

SECURITY BY DESIGN

DISTRIBUTED EXECUTION

DATA LINEAGE

AIRFLOW DEPLOYMENT

Airflow Docsity solution selection

MANAGED

Google Cloud Composer

Amazon Managed Workflows for Apache Airflow

ASTRONOMER

CUSTOM CONTAINER

docker

Google Compute Engine

AWS ECS

Airflow Docsity solution selection

MANAGED

* KEYS IN HAND

* MAINTENANCE BY PROVIDER

* MANAGED SCALABILITY

* PREDEFINED AIRFLOW VERSIONS

CUSTOM CONTAINER

* REPRODUCIBILITY

* TESTING

* ISOLATION

* PORTABILITY

* PRICING

* CUSTOM AIRFLOW VERSIONS

Airflow Docsity solution

MANAGED

~~* KEYS IN HAND~~

~~* MAINTENANCE BY PROVIDER~~

~~* MANAGED SCALABILITY~~

~~* PREDEFINED AIRFLOW VERSIONS~~

CUSTOM CONTAINER

✓

* REPRODUCIBILITY

* TESTING

* ISOLATION

* PORTABILITY

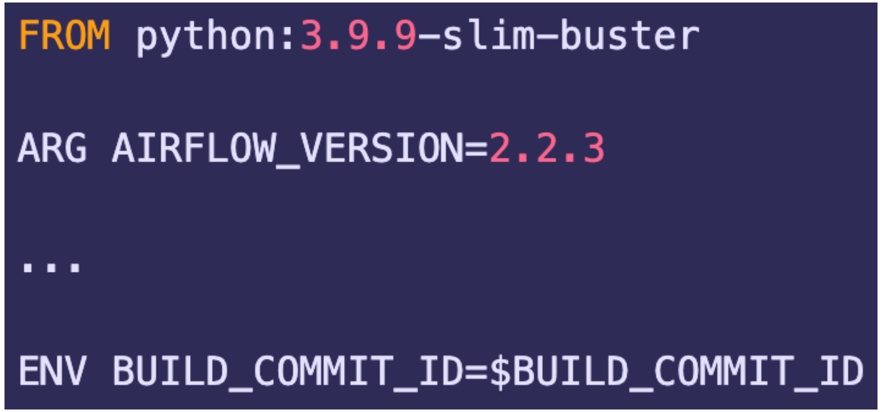
* PRICING

* CUSTOM AIRFLOW VERSIONS

Dockerfile

Python base image
size: 340MB
time: ~6 minutes

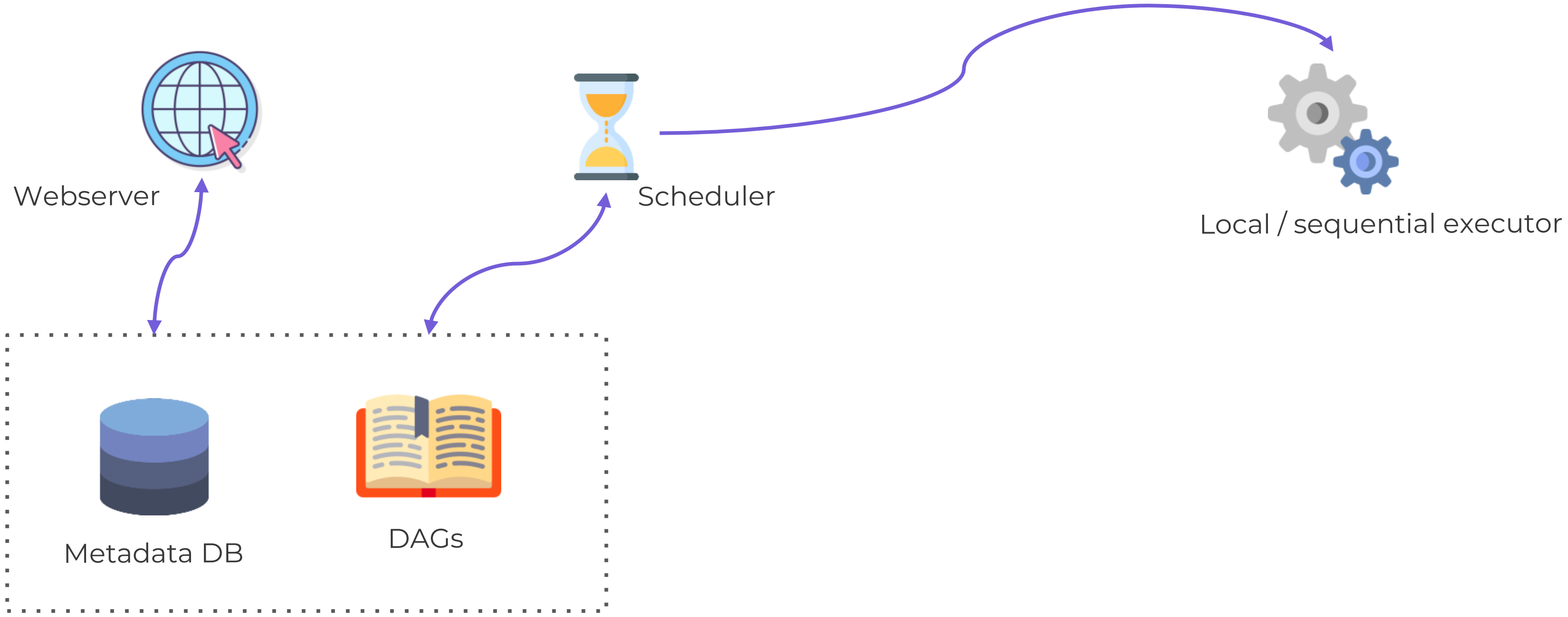
Dockerfile



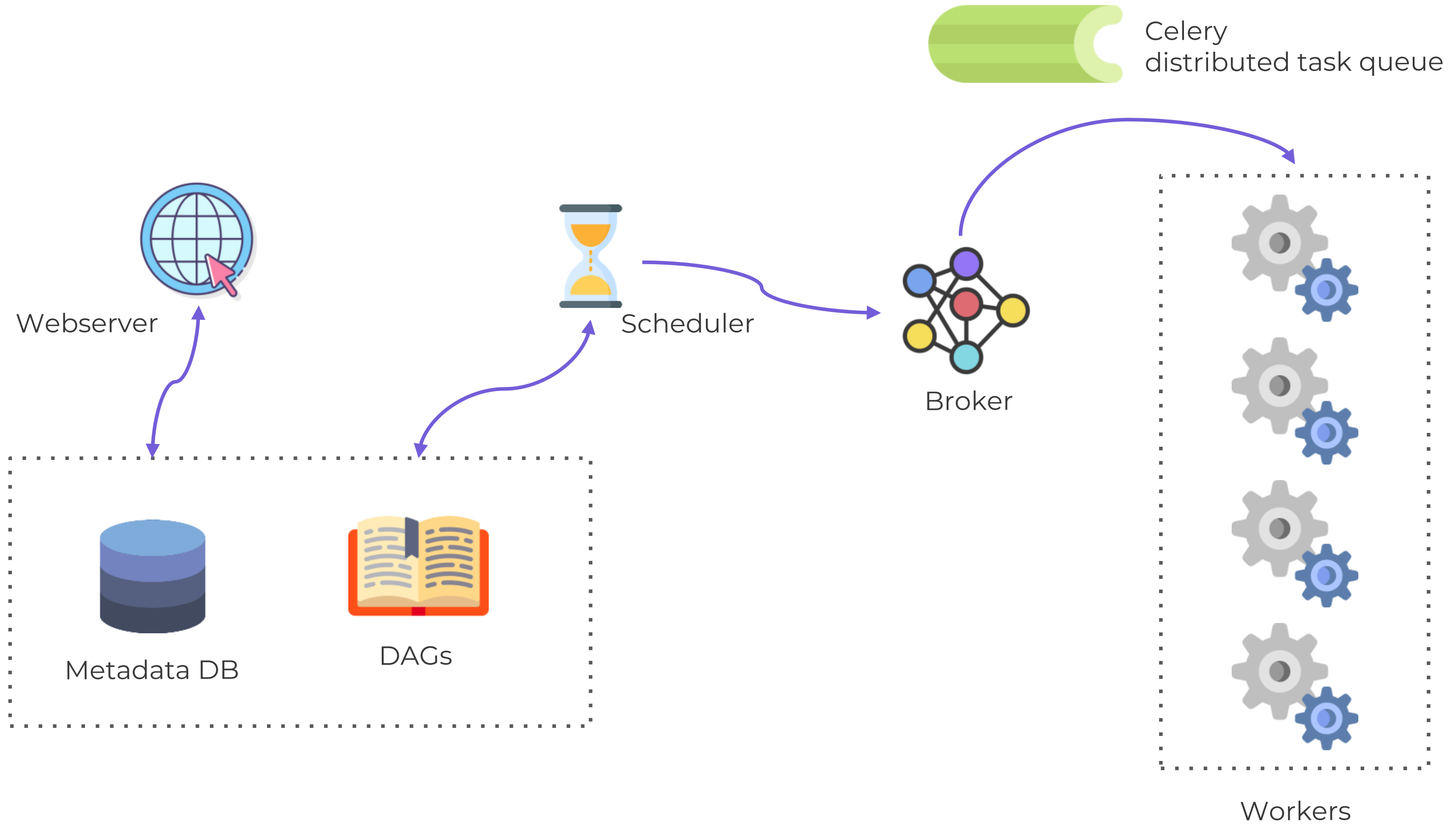
```
FROM python:3.9.9-slim-buster

ARG AIRFLOW_VERSION=2.2.3

...

ENV BUILD_COMMIT_ID=$BUILD_COMMIT_ID
```

Python slim-buster
size: 60MB
time: 4 minutes

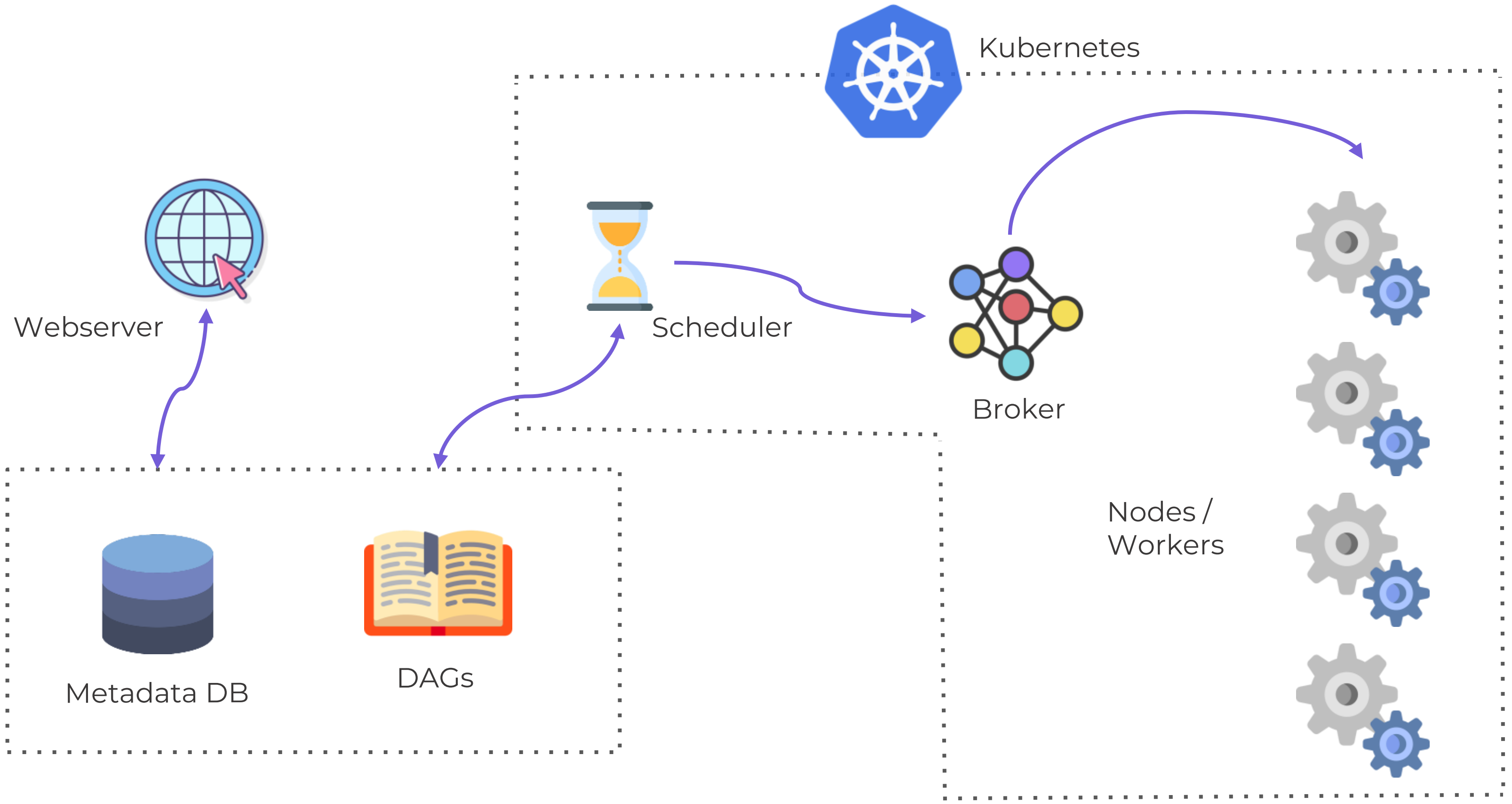
Airflow basic architecture



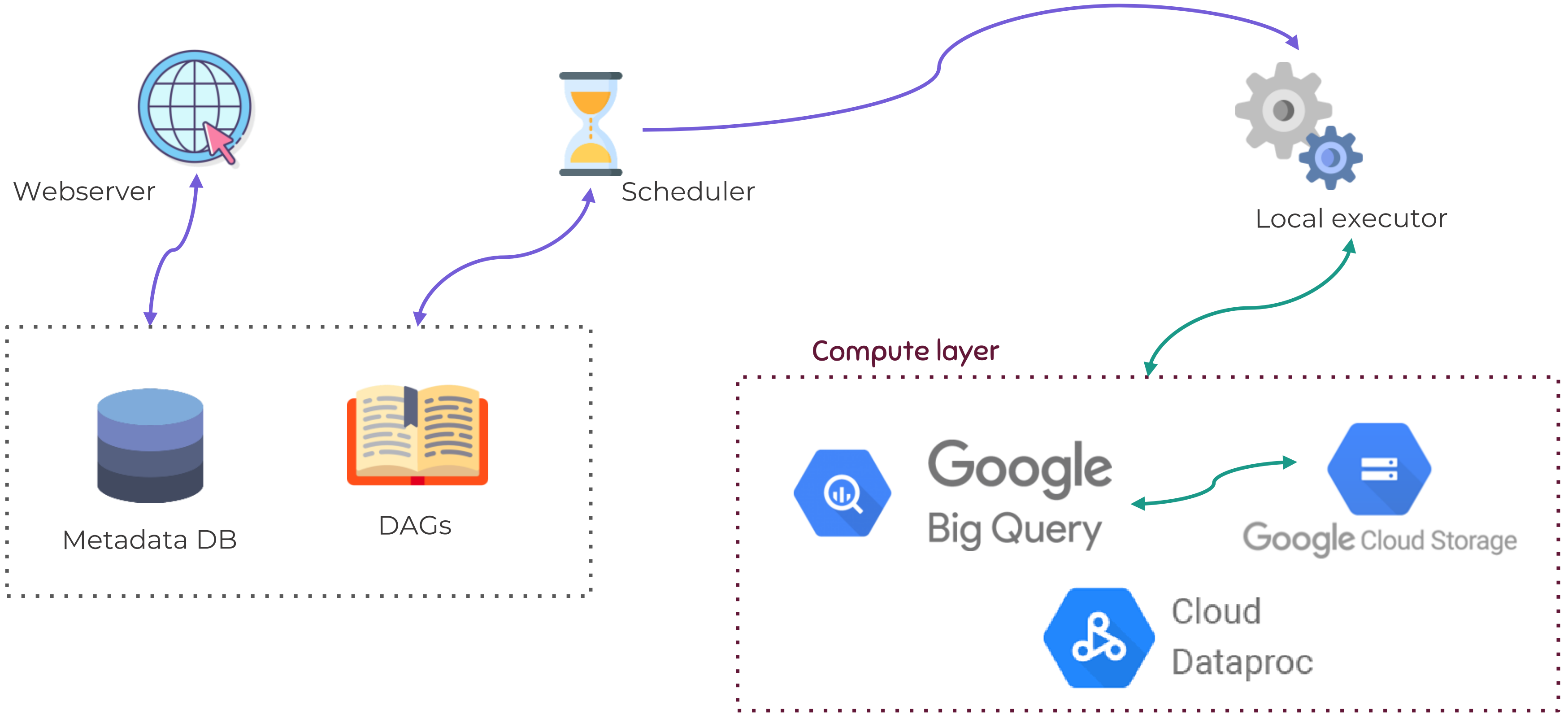
Airflow distributed process



Airflow distributed engine



Production architecture



CI/CD pipeline

Google Cloud Build

PREPARE & TRACK TO PRODUCTION

DECOUPLED STAGES

FASTER RELEASES

REPEATABLE

MORE ROBUST RELEASES

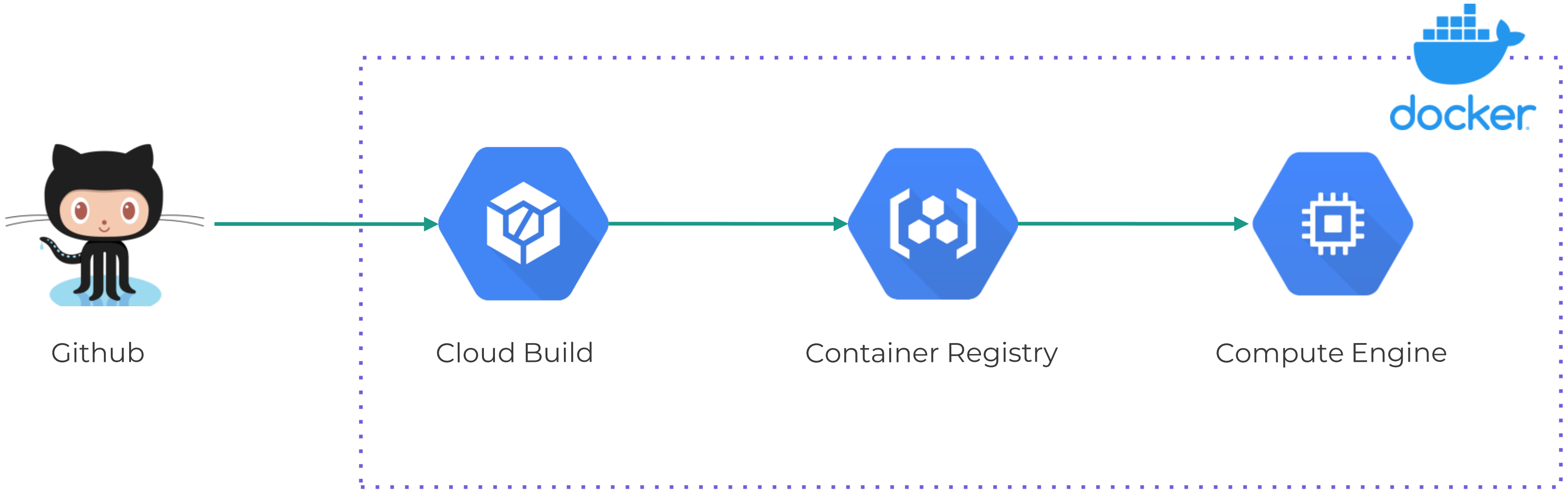
FAILING FAST

BETTER VISIBILITY ON CHANGE

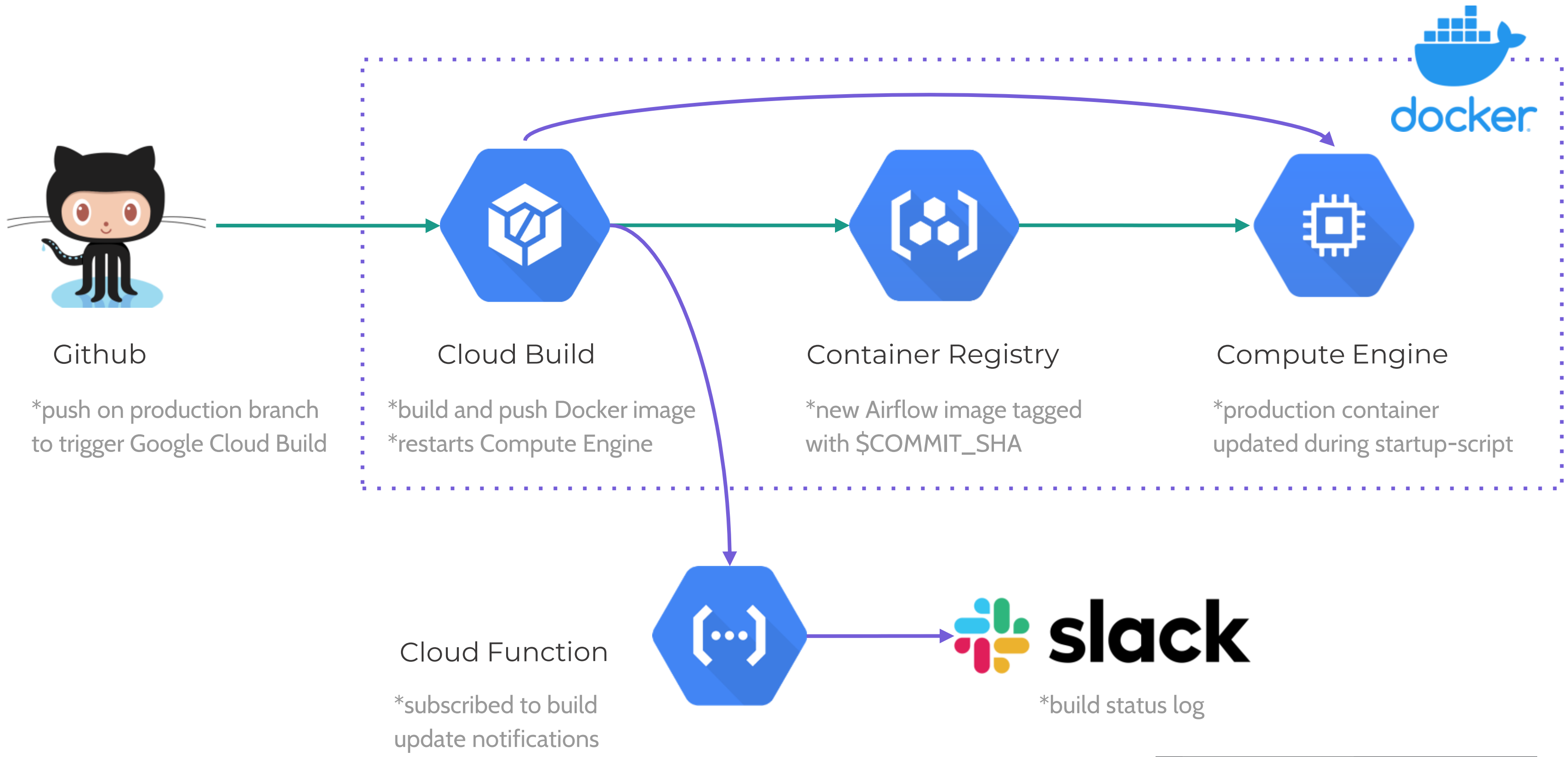
CENTRALISED ARTIFACTS

via cloudbuild.yaml

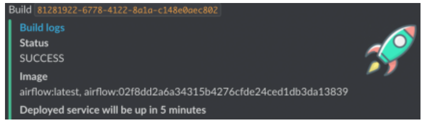
CI/CD pipeline



CI/CD flow



docker
Github
*push on production branch to trigger Google Cloud Build
Cloud Build
*build and push Docker image
*restarts Compute Engine
Container Registry
*new Airflow image tagged with $COMMIT_SHA
Compute Engine
*production container updated during startup-script
Cloud Function
*subscribed to build update notifications
slack
*build status log



```
Build 81281922-6778-4122-8a1a-c148e0aec802
Build logs
Status
SUCCESS
Image
airflow:latest, airflow:02f8dd2a6a34315b4276cfde24ced1db3da13839
Deployed service will be up in 5 minutes
```

CI/CD pipeline

cloudbuild.yaml

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/airflow:$COMMIT_SHA',
         '-t', 'gcr.io/$PROJECT_ID/airflow',
         '--build-arg', 'BUILD_COMMIT_ID=$COMMIT_SHA',
         'src/airflow-prod']
- name: gcr.io/cloud-builders/gcloud
  args: [ compute, instances, stop, **-airflow-engine, --zone=europe-***-*]
- name: gcr.io/cloud-builders/gcloud
  args: [ compute, instances, start, **-airflow-engine, --zone=europe-***-*]


substitutions:
  _AIRFLOW_VERSION: 2.0.2

images:
- 'gcr.io/$PROJECT_ID/airflow:latest'
- 'gcr.io/$PROJECT_ID/airflow:$COMMIT_SHA'
```

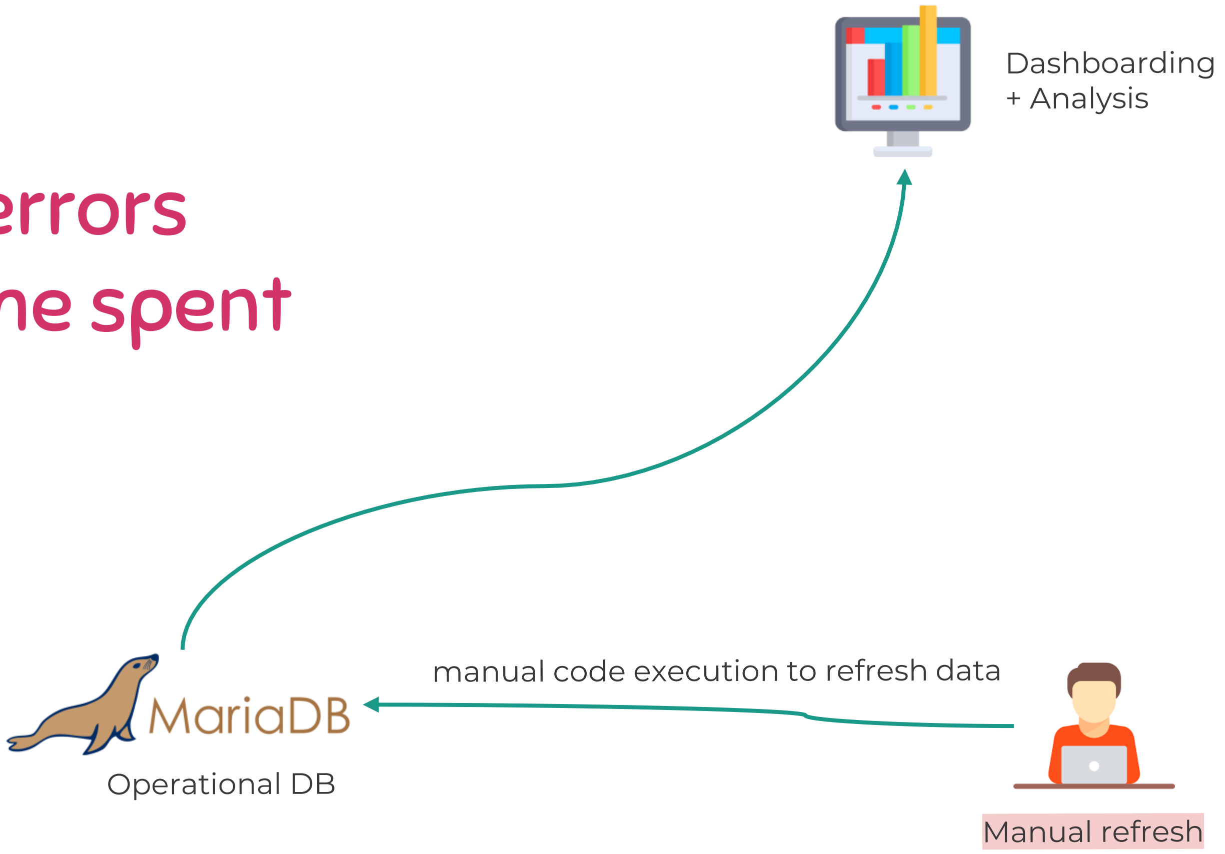
Github

Cloud Build

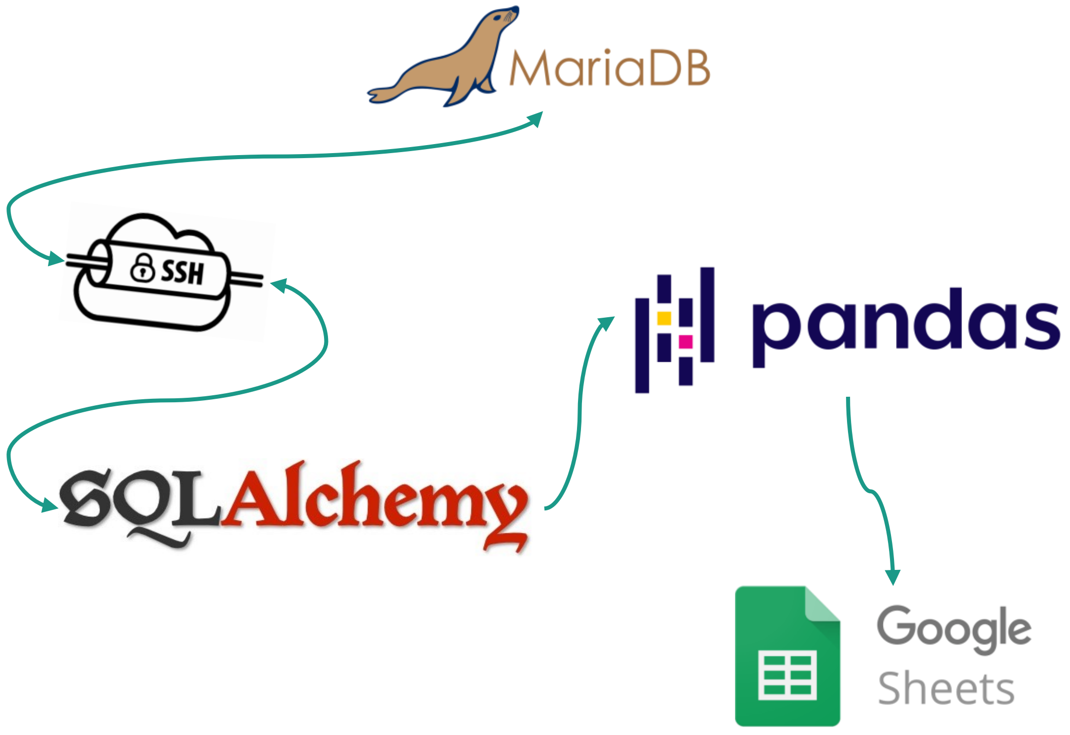
WHY AIRFLOW?
part II

PRE-2020

Lot of errors
and time spent

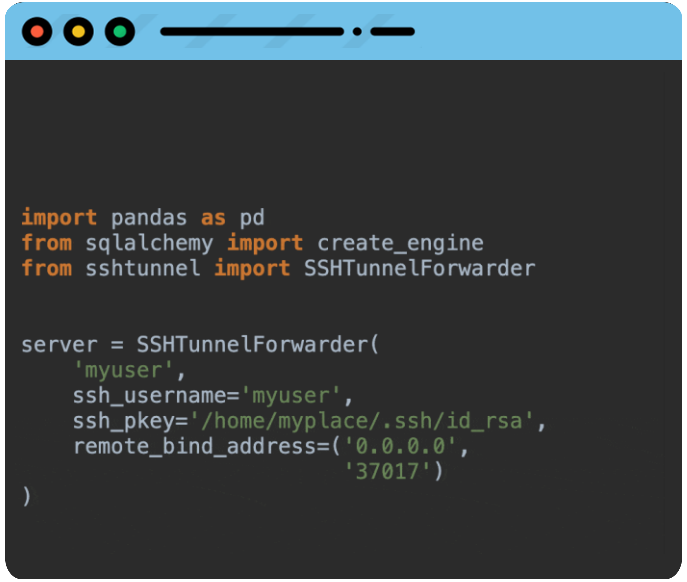


Pandas + SQLAlchemy



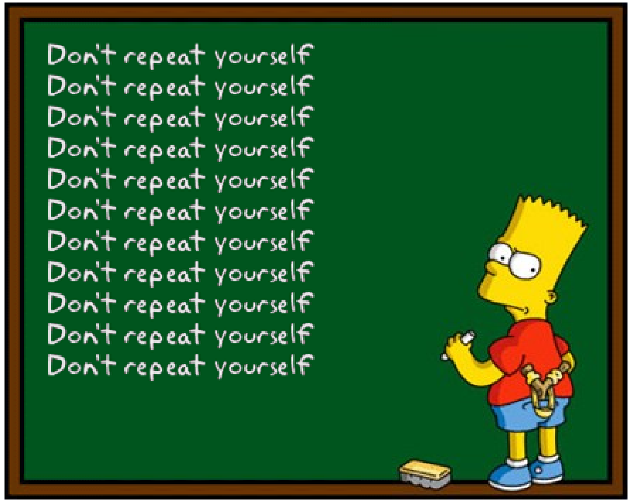
via df2gspread and gspread-pandas

Pandas + SQLAlchemy is not DRY



```python
import pandas as pd
from sqlalchemy import create_engine
from sshtunnel import SSHTunnelForwarder


server = SSHTunnelForwarder(
    'myuser',
    ssh_username='myuser',
    ssh_pkey='/home/myplace/.ssh/id_rsa',
    remote_bind_address=('0.0.0.0',
                         '37017')
)
```

~15 lines of code per query/table



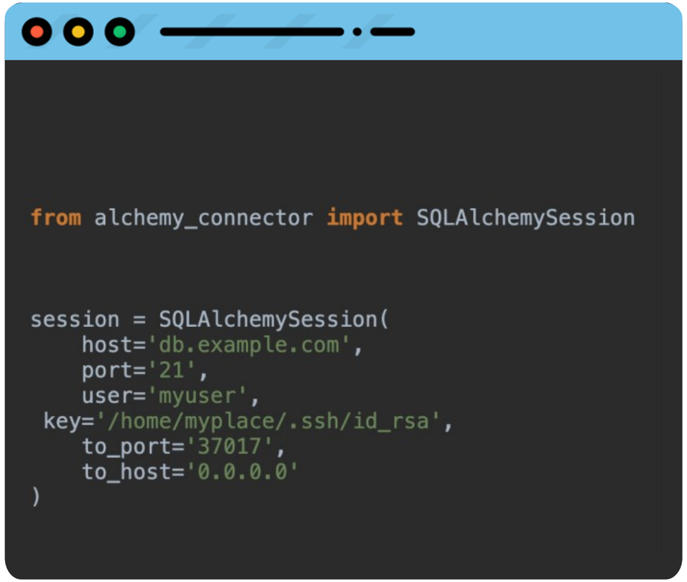
sqlalchemy-connector

only 4 lines of code!

```
pip install sqlalchemy-connector
```

INSTALL ME

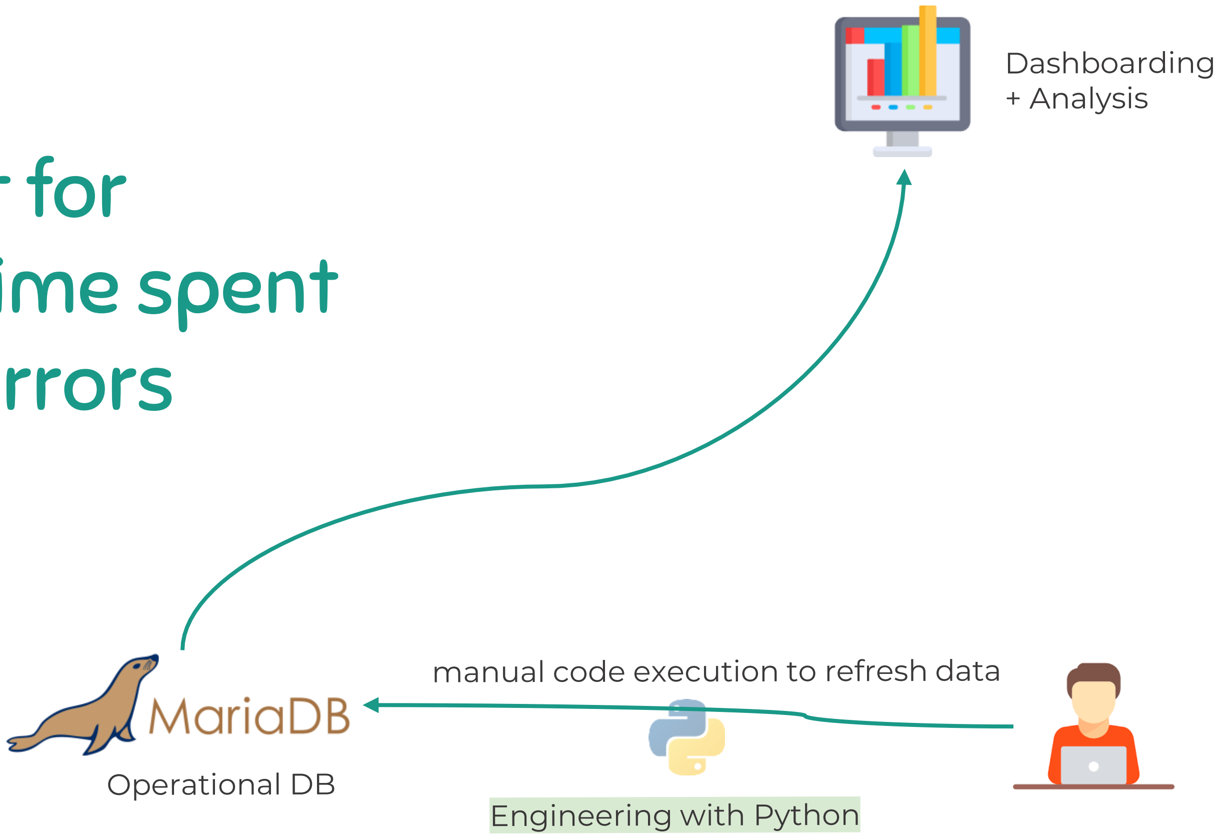
https://pypi.org/project/sqlalchemy-connector/



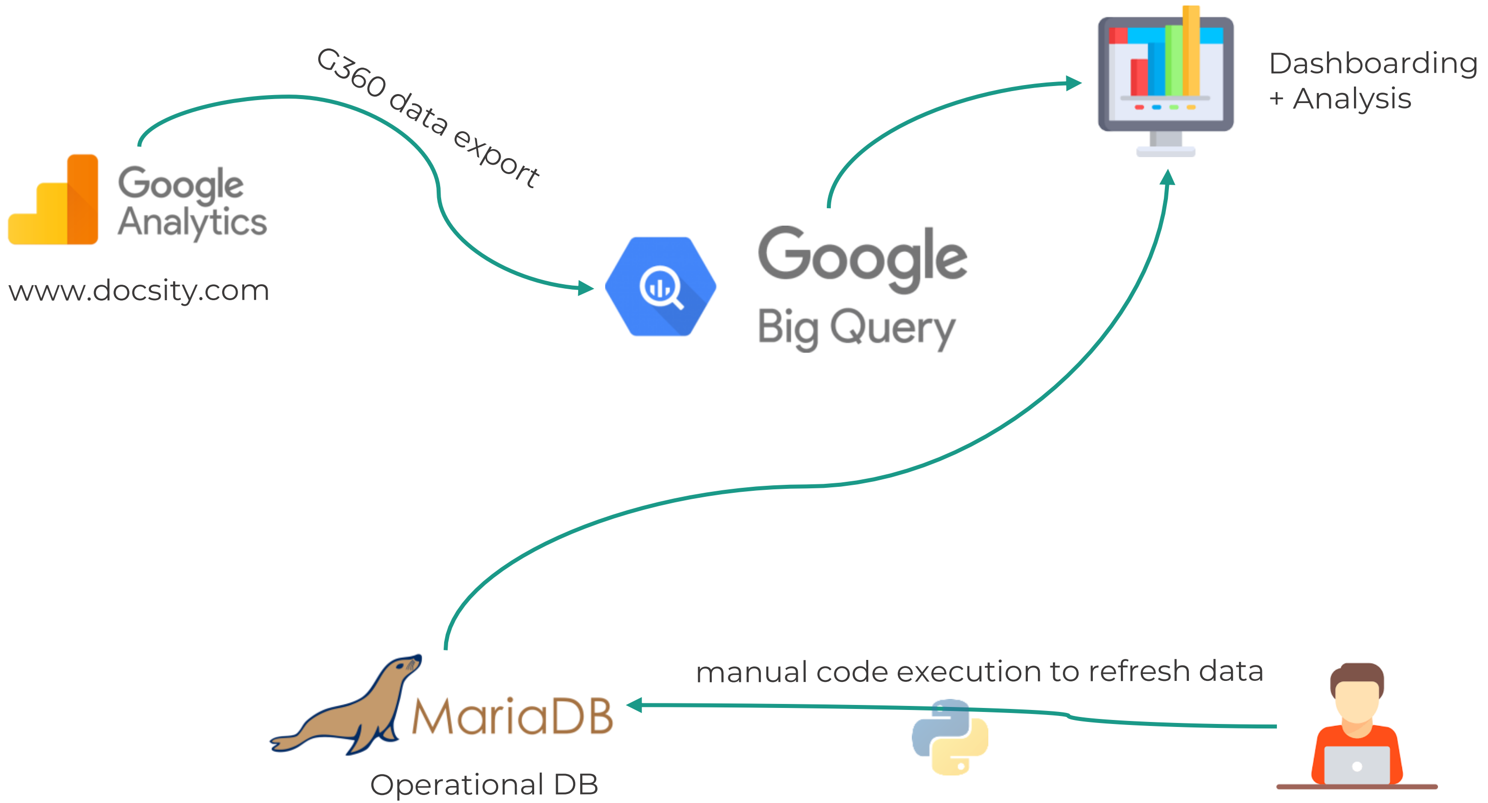
```
from alchemy_connector import SQLAlchemySession


session = SQLAlchemySession(
    host='db.example.com',
    port='21',
    user='myuser',
 key='/home/myplace/.ssh/id_rsa',
    to_port='37017',
    to_host='0.0.0.0'
)
```

2020 Q0

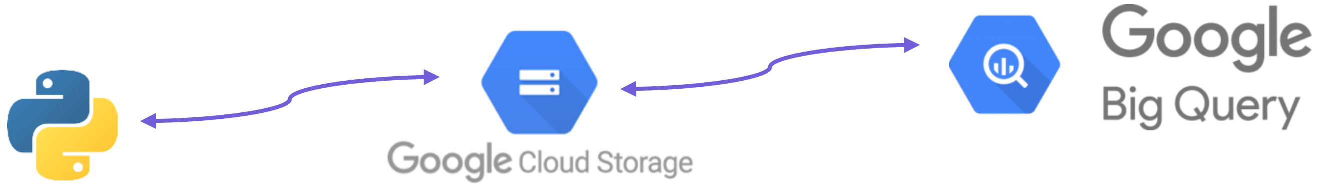
Boost for
less time spent
and errors



2020 Q1 data lake start



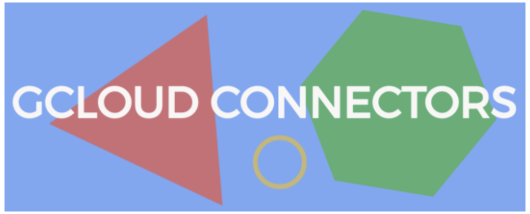
Automate BigQuery tables from Cloud Storage



Pandas + Google Cloud
Storage API

Google BigQuery API

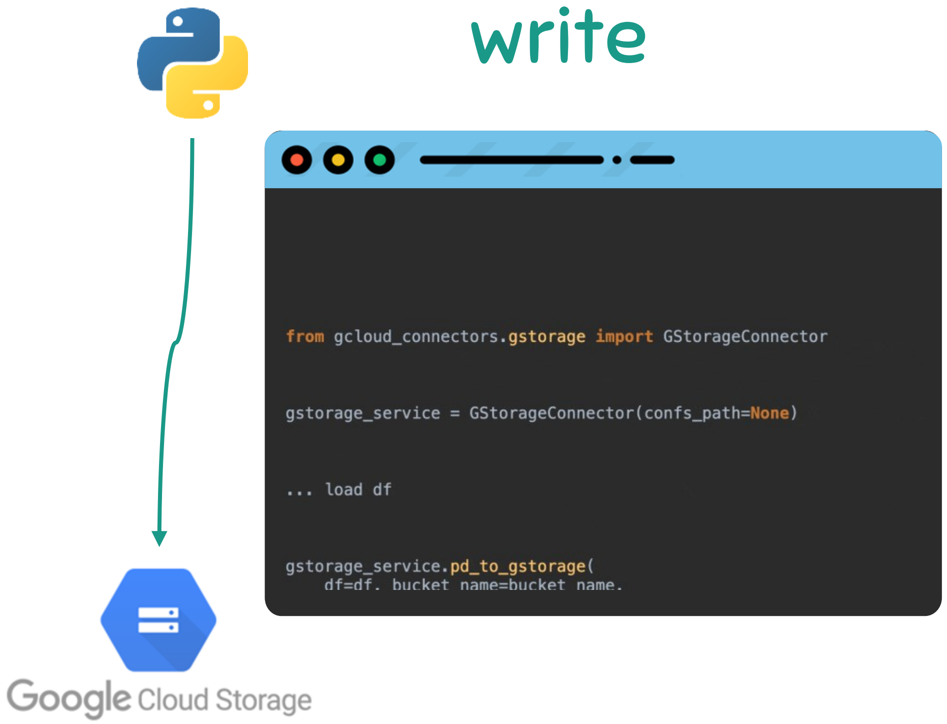
gcloud-connectors



```
pip install gcloud-connectors
```

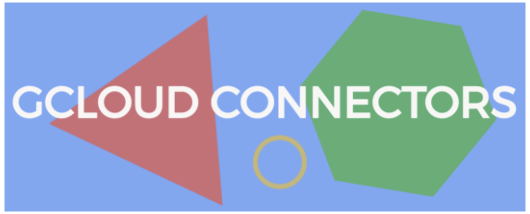
INSTALL ME
https://github.com/pualien/py-gcloud-connectors

write



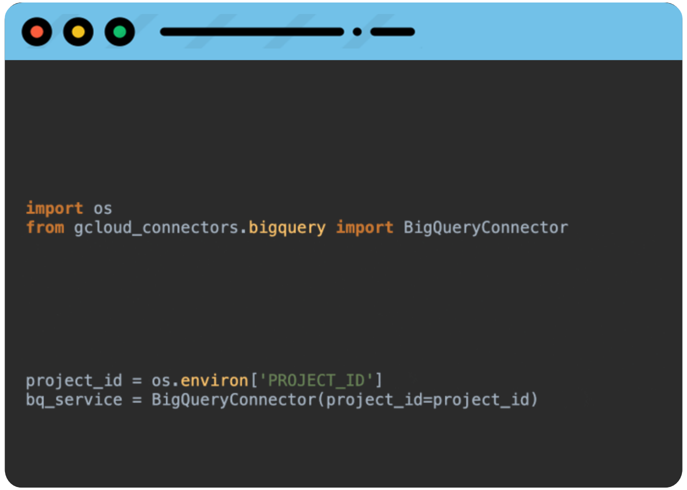
```
from gcloud_connectors.gstorage import GStorageConnector

gstorage_service = GStorageConnector(confs_path=None)

... load df

gstorage_service.pd_to_gstorage(
    df=df, bucket_name=bucket_name,
```

gcloud-connectors



```
pip install gcloud-connectors
```

Google
Big Query
read



```
import os
from gcloud_connectors.bigquery import BigQueryConnector



project_id = os.environ['PROJECT_ID']
bq_service = BigQueryConnector(project_id=project_id)
```

Extended gcloud-connectors

Google Analytics

Google Sheets

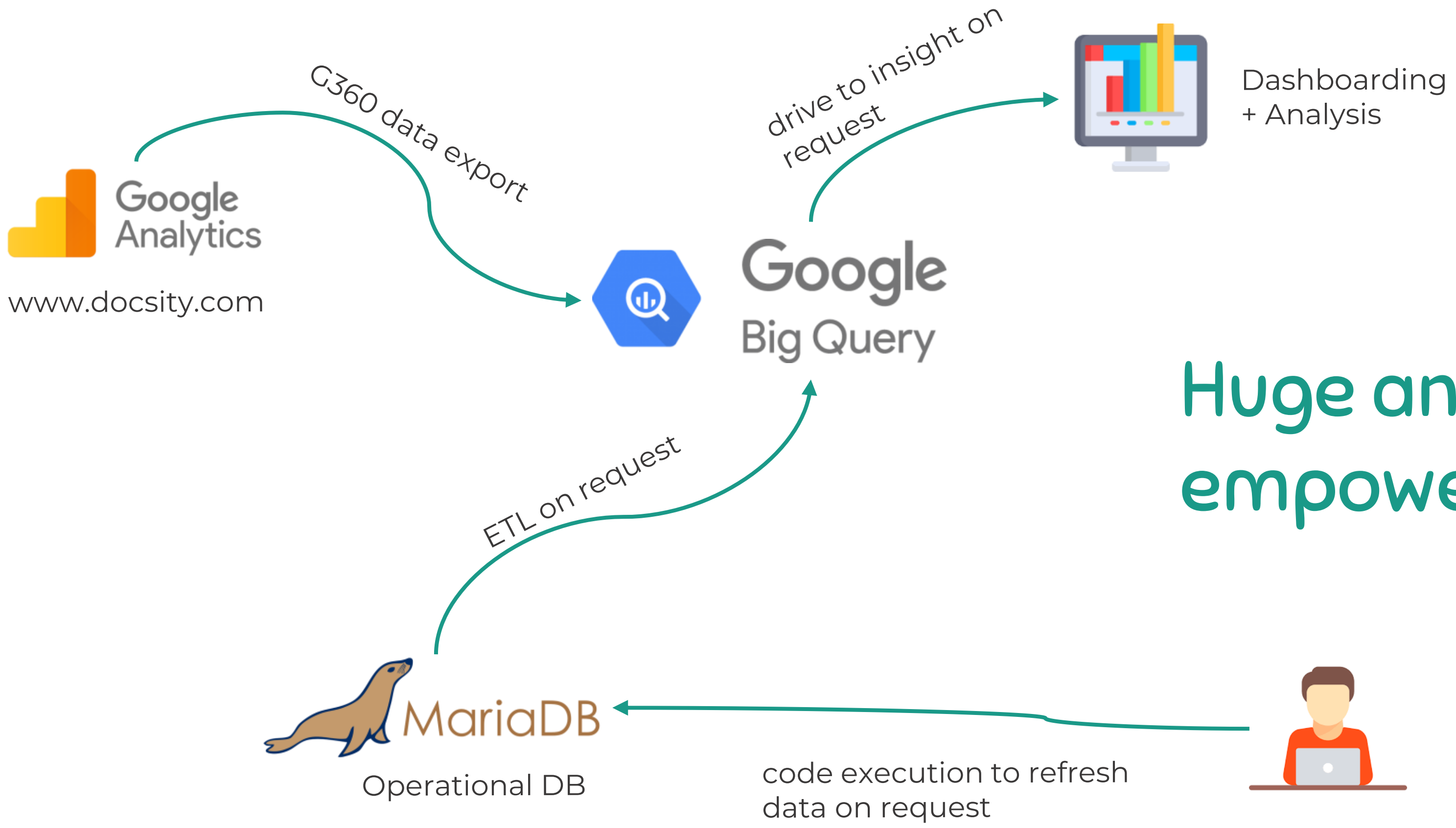
Google Cloud Storage

Google Search Console

Google Drive

Google Big Query

2020 Q2 data lake enhancement



Huge analytics empowerment

Terraform infrastructure as code

HashiCorp
Terraform

AUTOMATE DEPLOY/RECOVERY

VERSIONING

EASY TO INTEGRATE WITH CI

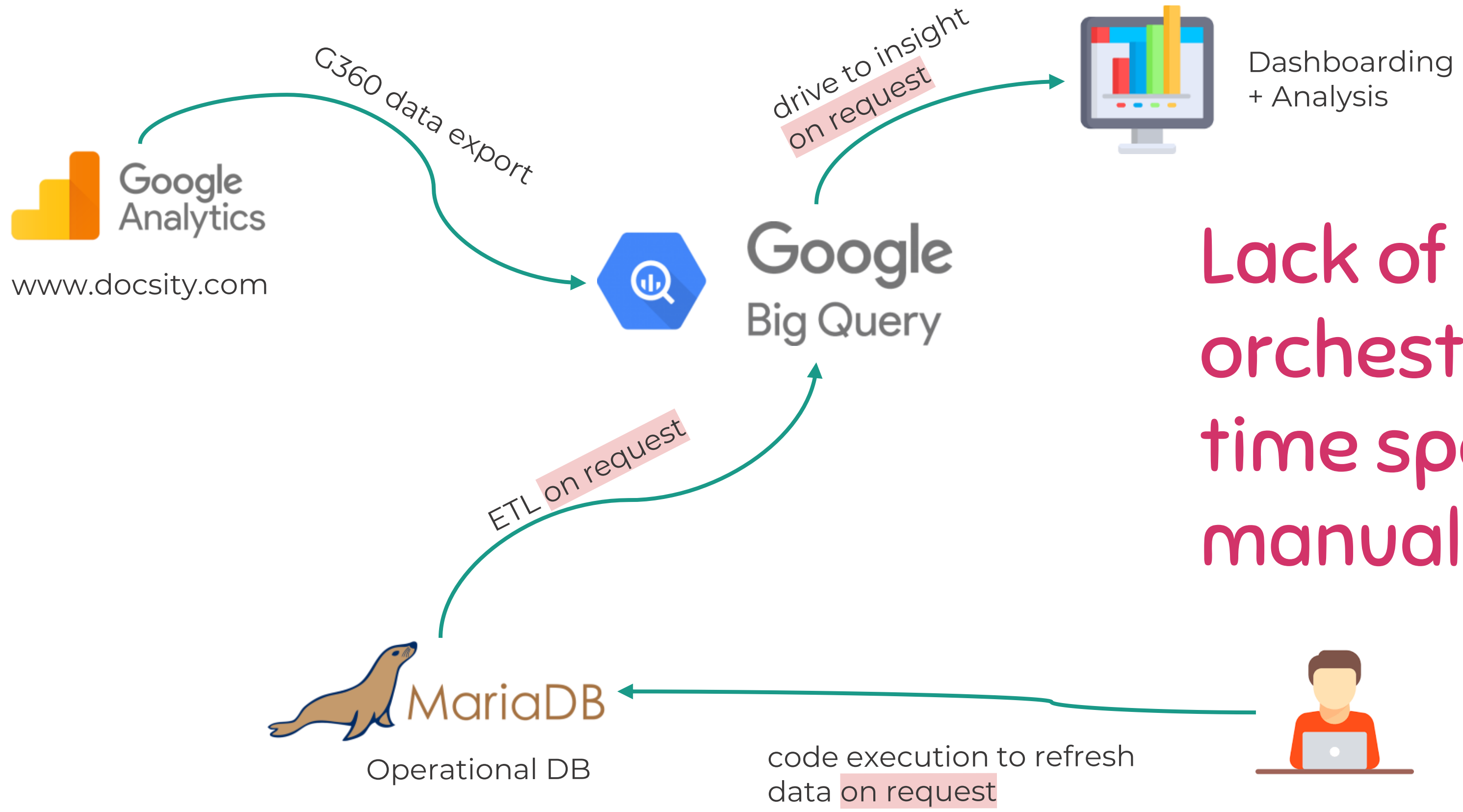
ONE CLICK TO DESTROY

NO NEED TO REPAIR, JUST REDEPLOY

```
resource "google_bigquery_table" "db_docsity_table" {
   dataset_id = google_bigquery_dataset.db_dataset.friendly_name
   table_id = “db_docsity”
   schema = <<EOF
[ { "name": "id", "type": "FLOAT", "mode": "NULLABLE" } ...]
}
```

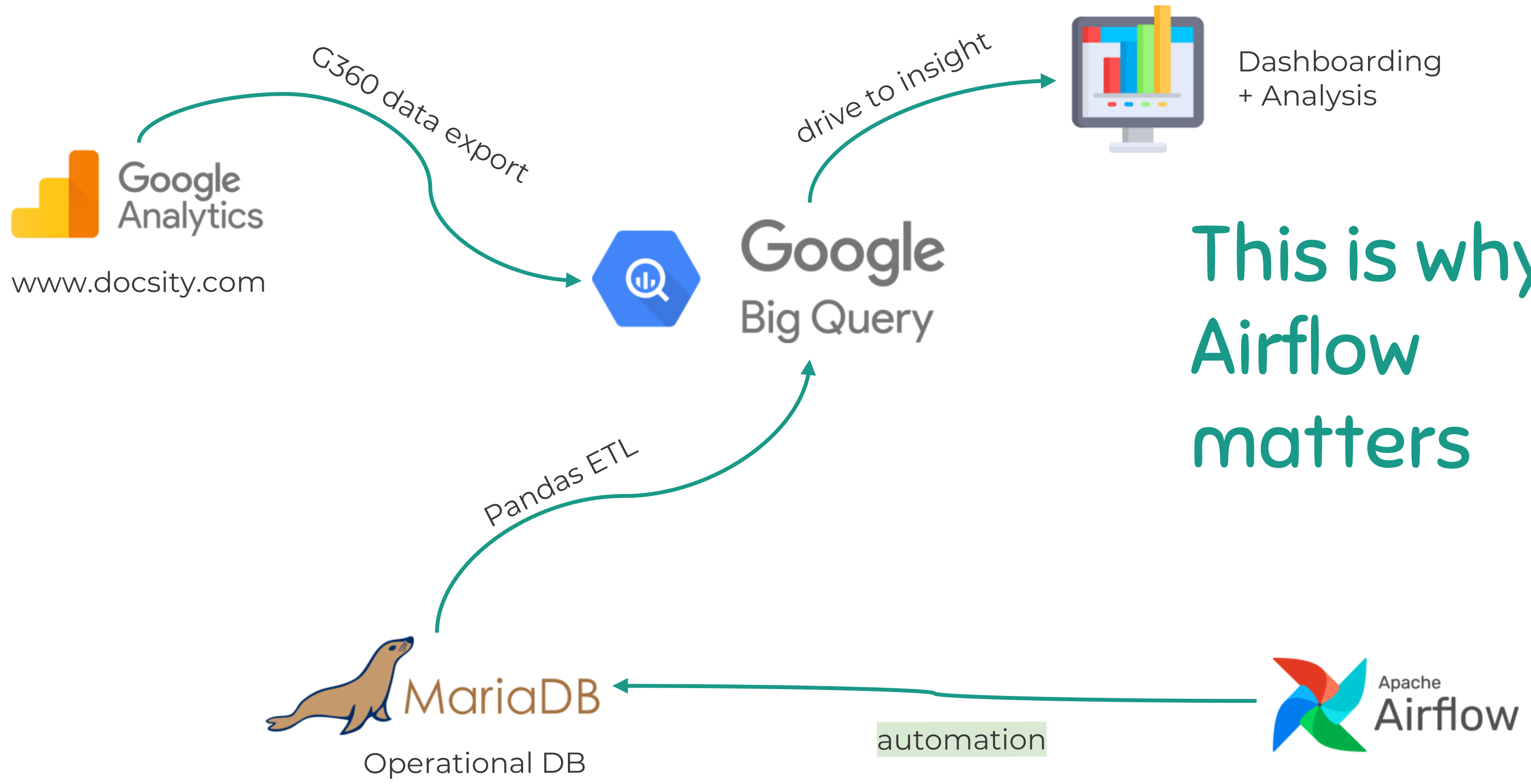
BigQuery table

2020 Q2 data lake enhancement



Lack of orchestration, time spent for manual refresh

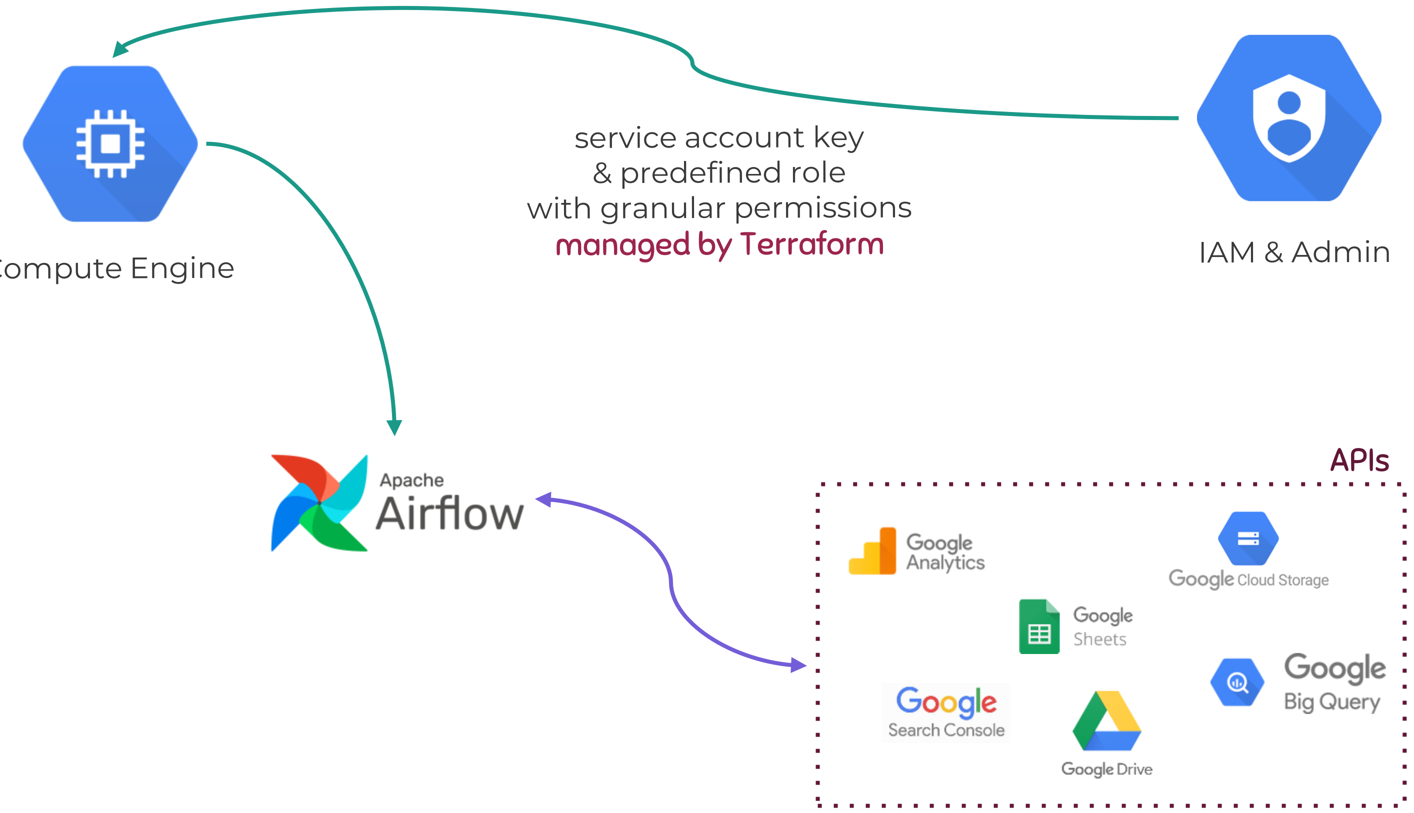
2020 Q2 data lake orchestration



This is why Airflow matters

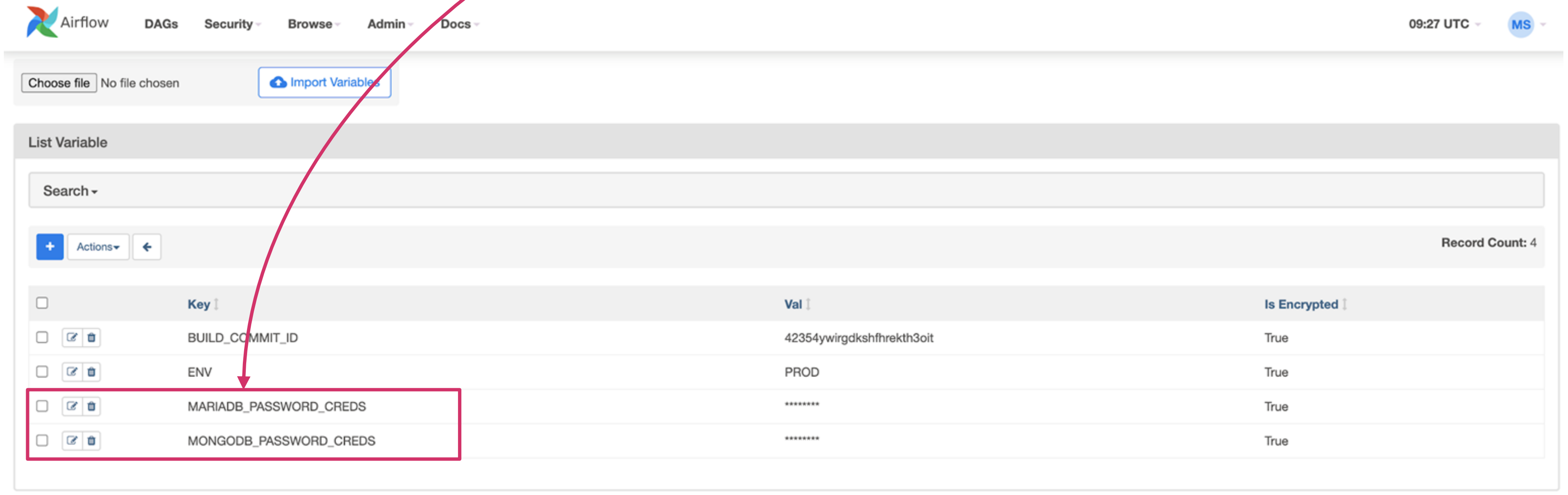
AIRFLOW SECURITY

Security for Google Compute Engine



Airflow encrypted variables & connections

Encrypted credentials pulled from Compute Engine metadata

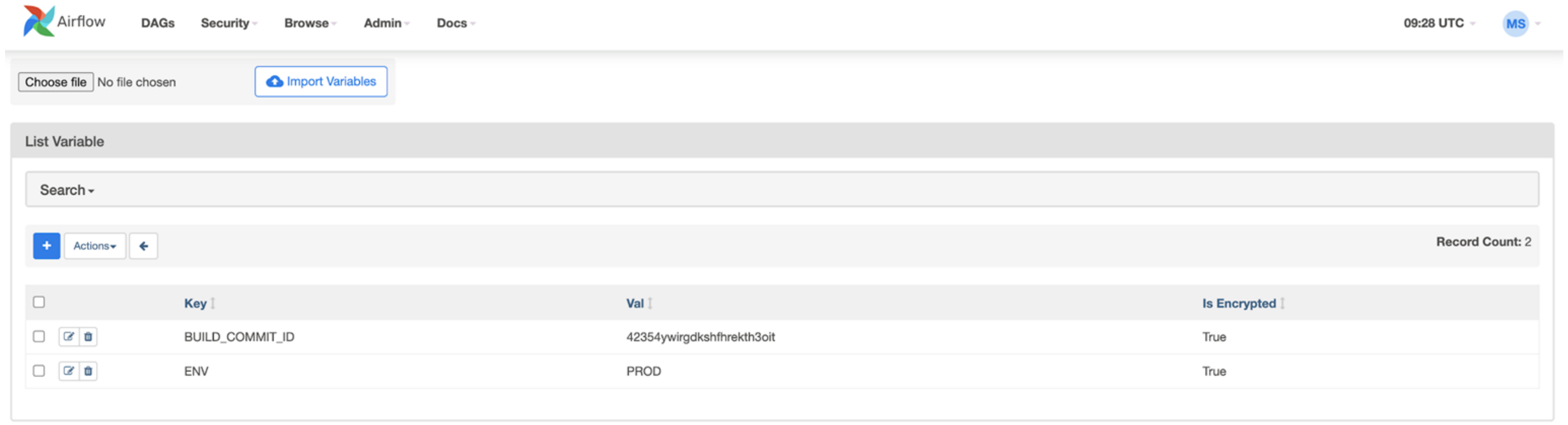


Key	Val	Is Encrypted
BUILD_COMMIT_ID	42354ywirgdkshfhrekth3oit	True
ENV	PROD	True
MARIADB_PASSWORD_CREDS	********	True
MONGODB_PASSWORD_CREDS	********	True

Airflow + Google Secret Manager

Google Secret Manager

credentials pulled from Google Secret Manager

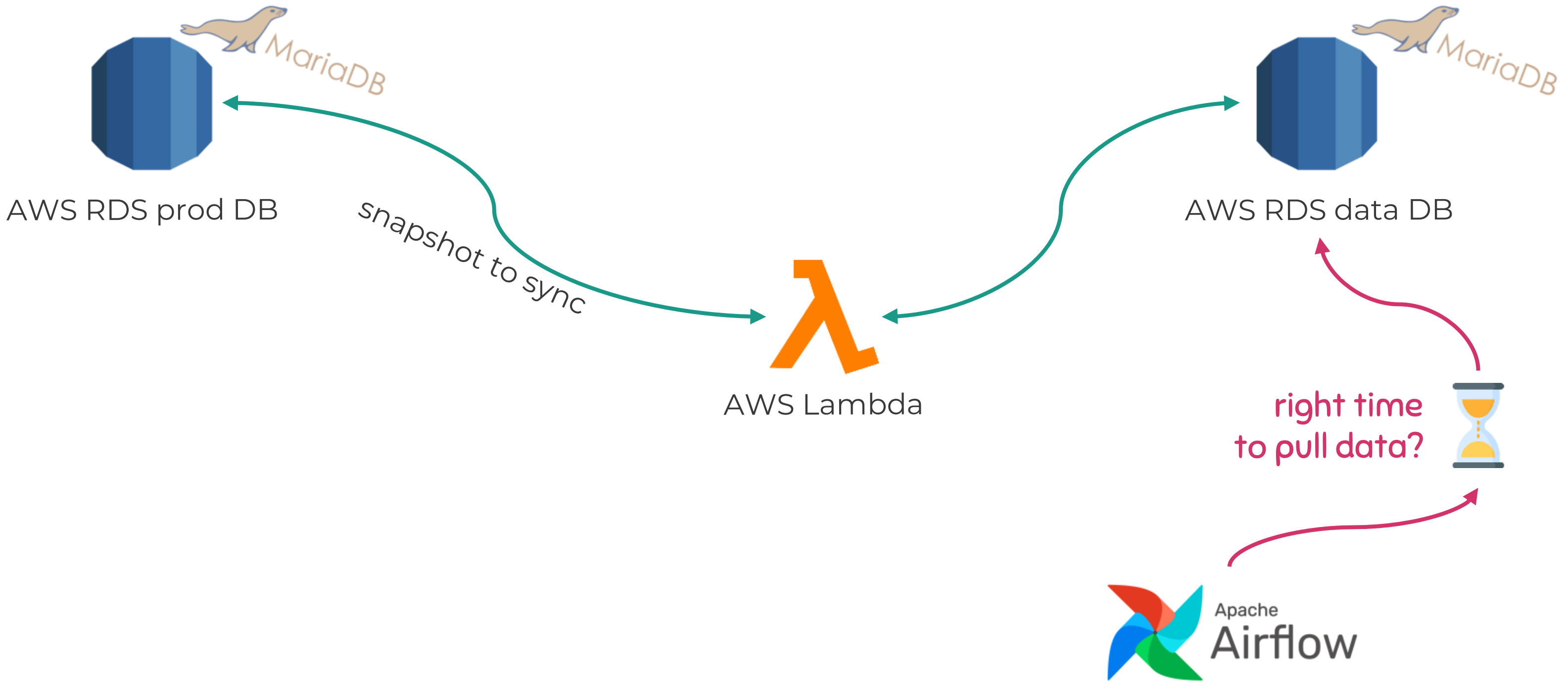


Record Count: 2

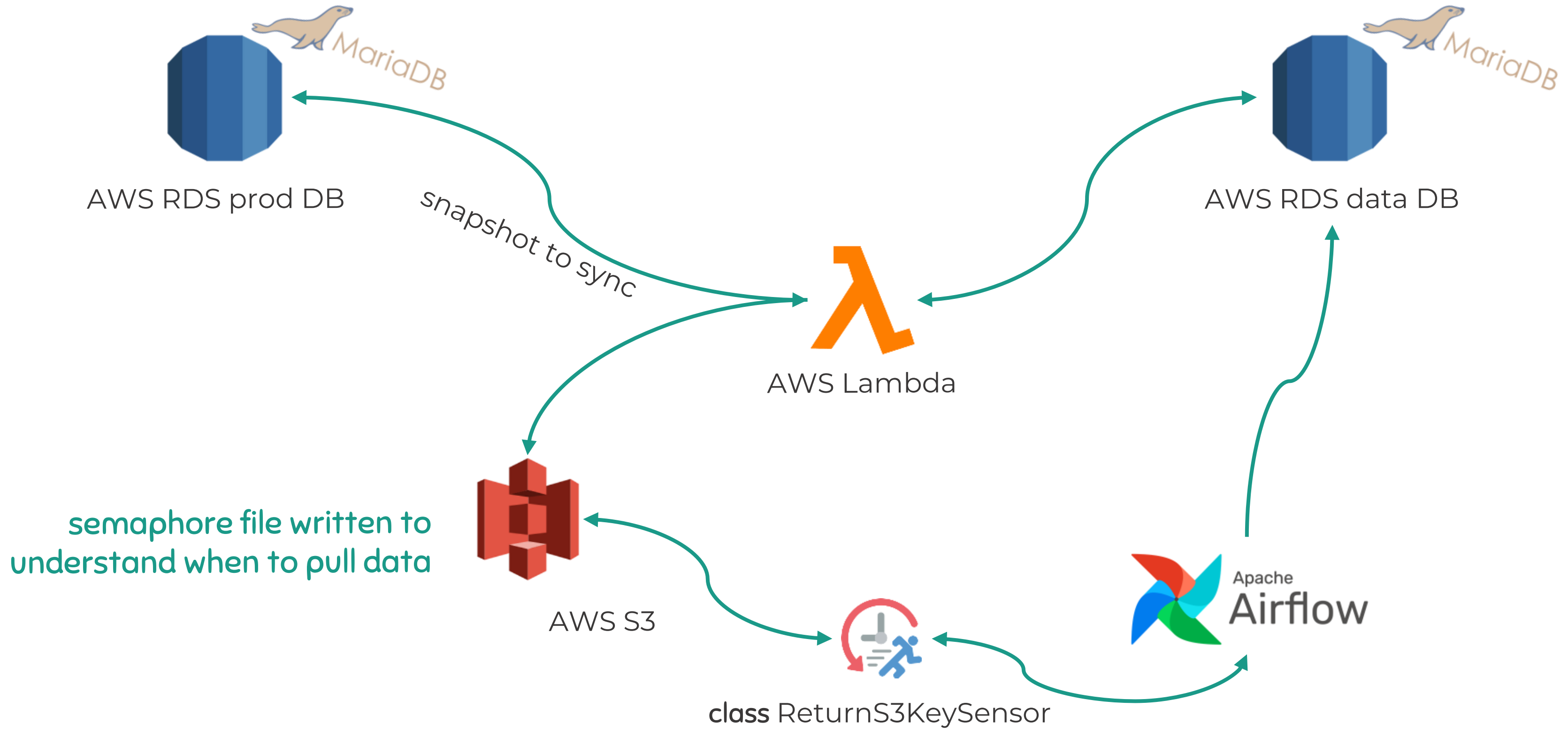
	Key	Val	Is Encrypted
	BUILD_COMMIT_ID	42354ywirgdkshfhrekth3oit	True
	ENV	PROD	True

WHAT TIME TO UPDATE?

Schedule time?



airflow-add-ons



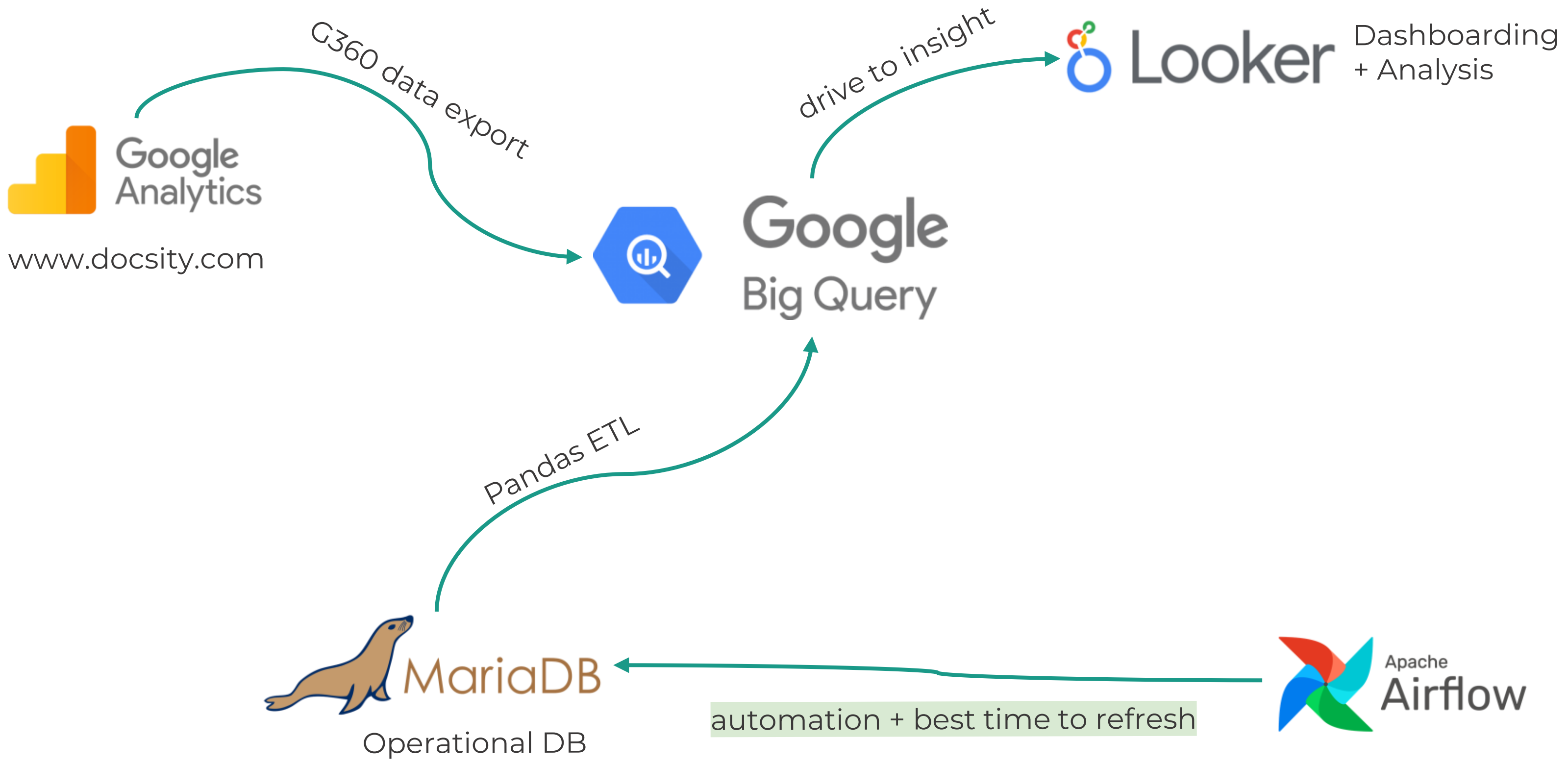
airflow-add-ons

```
pip install airflow-add-ons
```

Apache
Airflow

via custom Airflow Operators & Sensors

2020 Q3 data lake orchestration

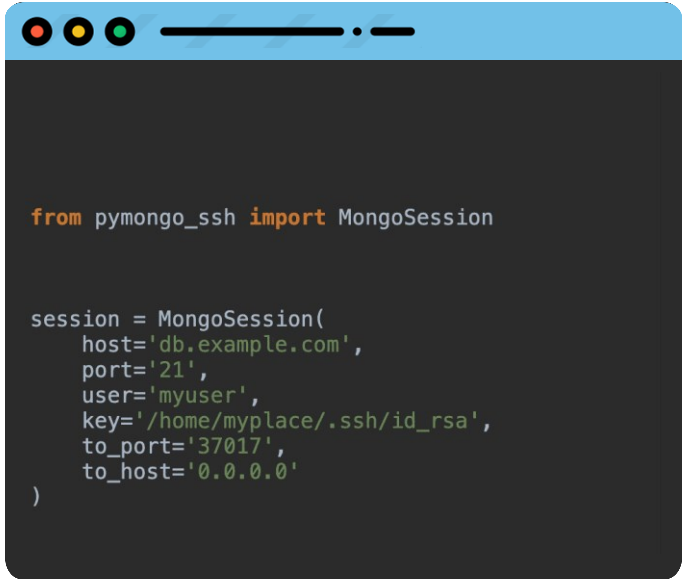


pymongo-ssh

```
pip install pymongo-ssh
```

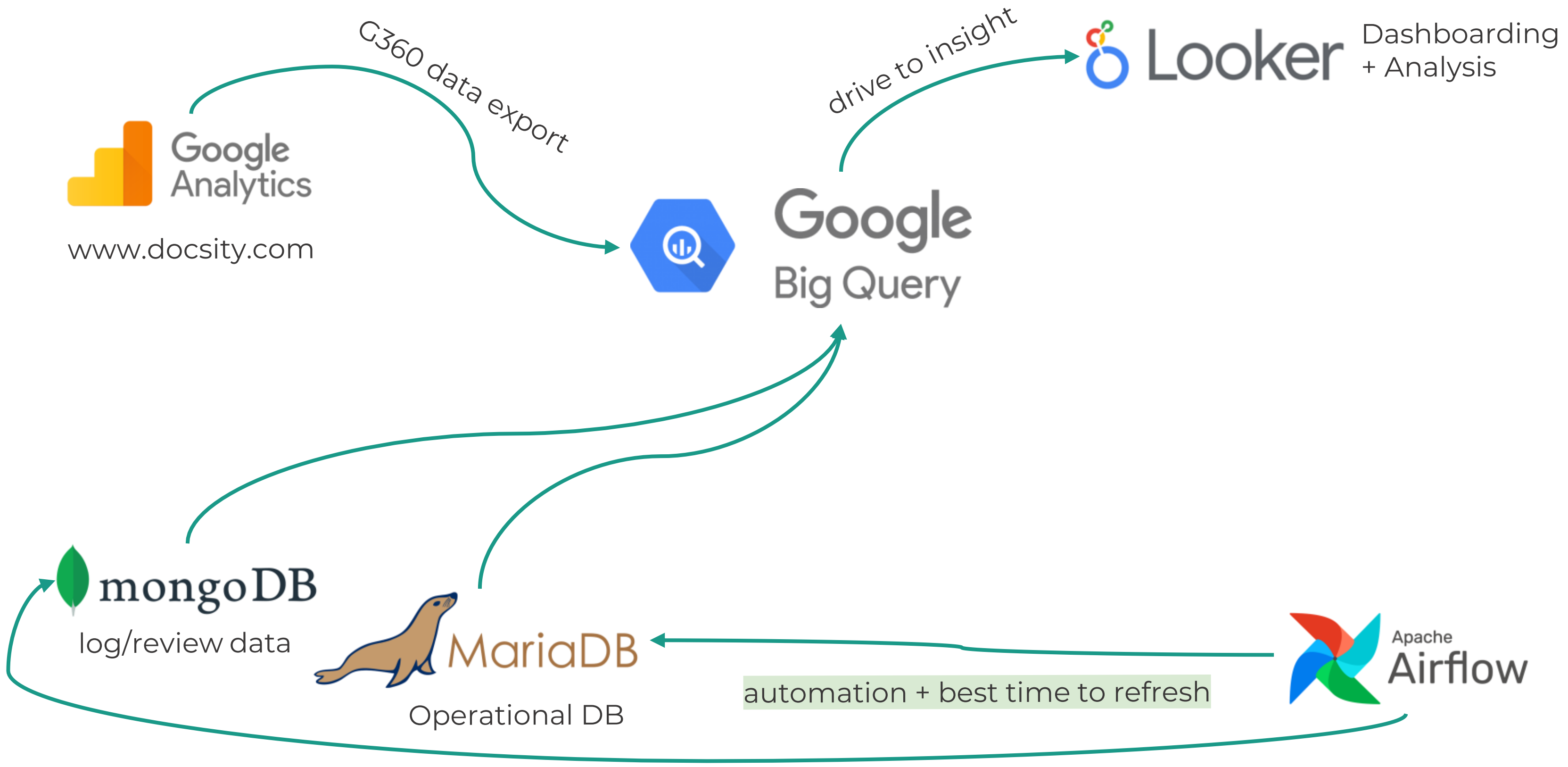
INSTALL ME
https://pypi.org/project/pymongo-ssh/

DRY

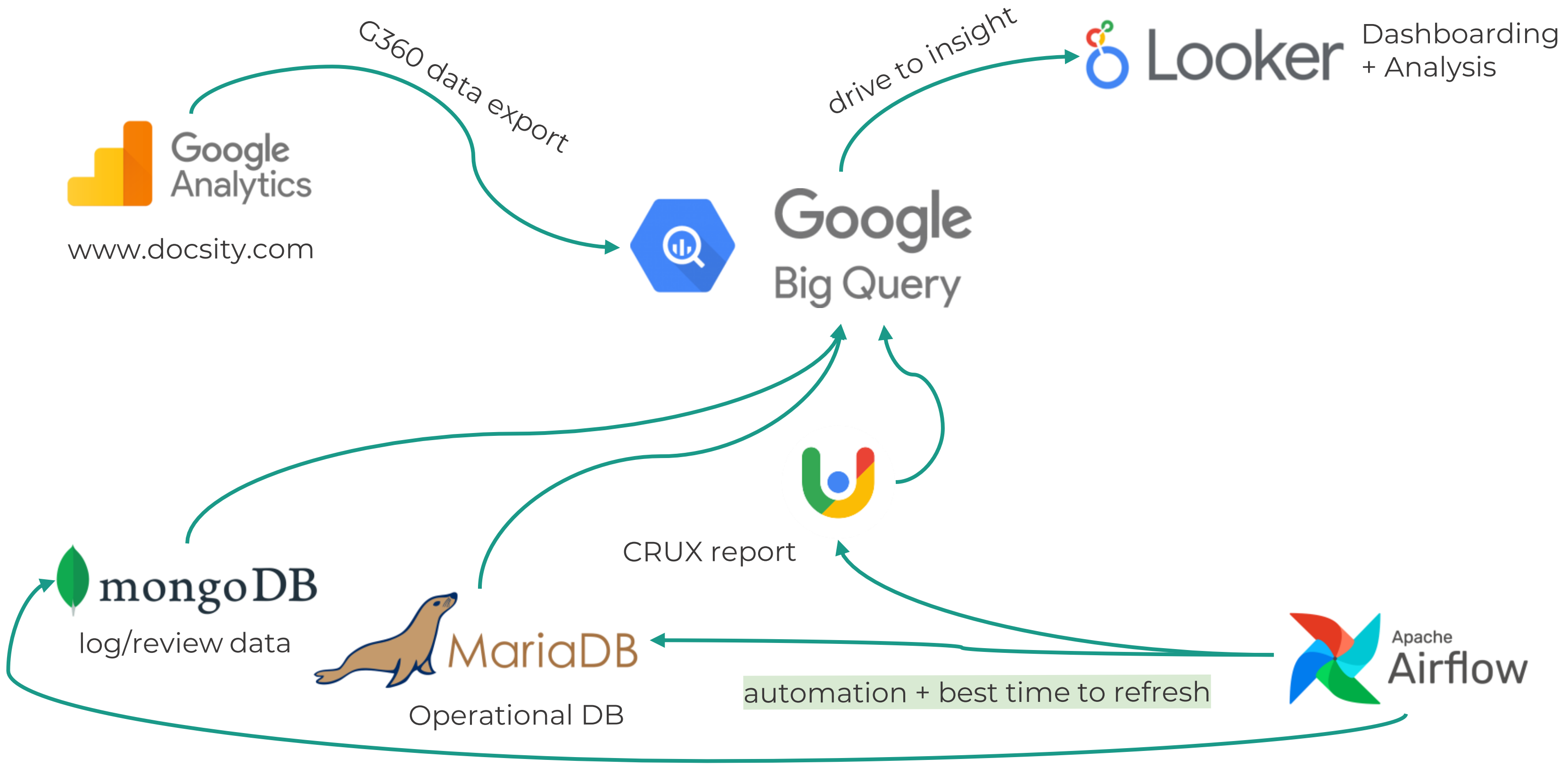


```python
from pymongo_ssh import MongoSession


session = MongoSession(
    host='db.example.com',
    port='21',
    user='myuser',
    key='/home/myplace/.ssh/id_rsa',
    to_port='37017',
    to_host='0.0.0.0'
)
```

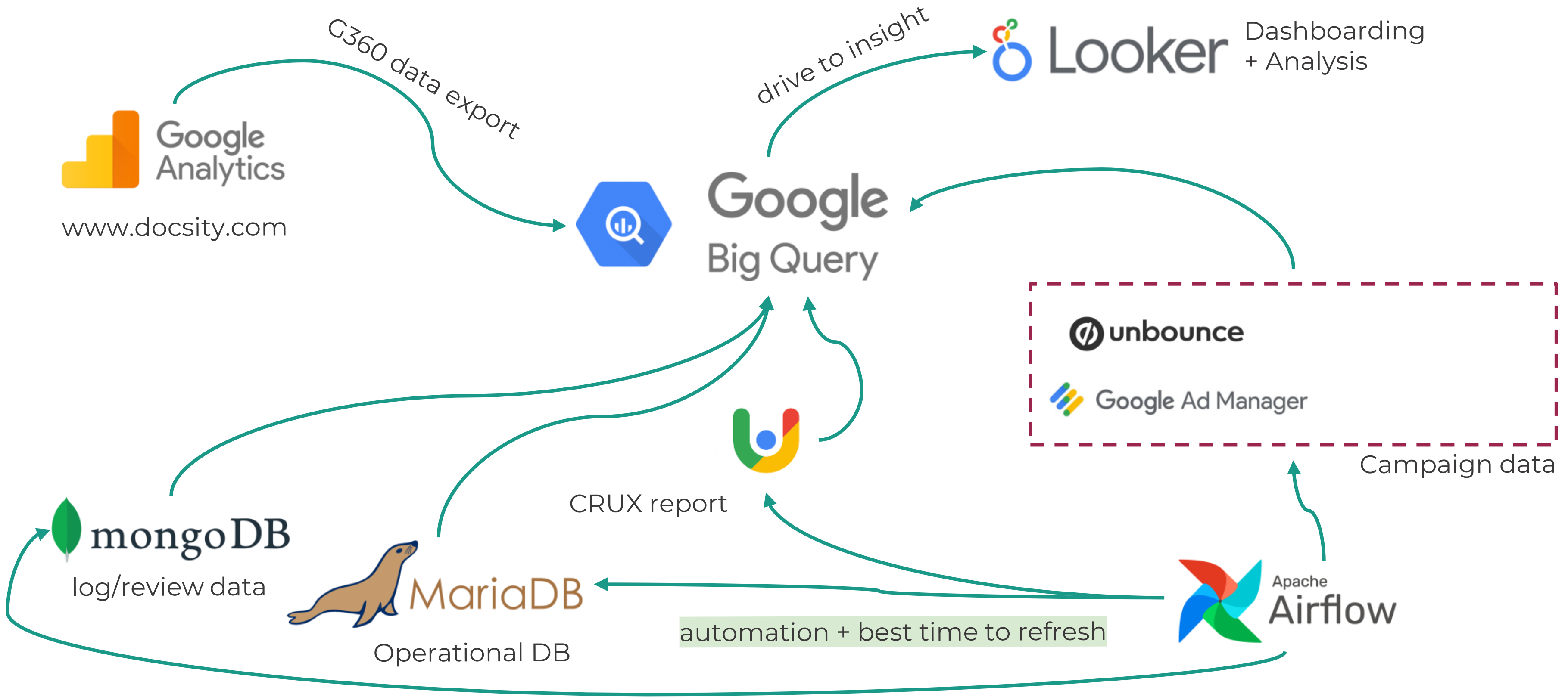
2020 Q4 data lake orchestration



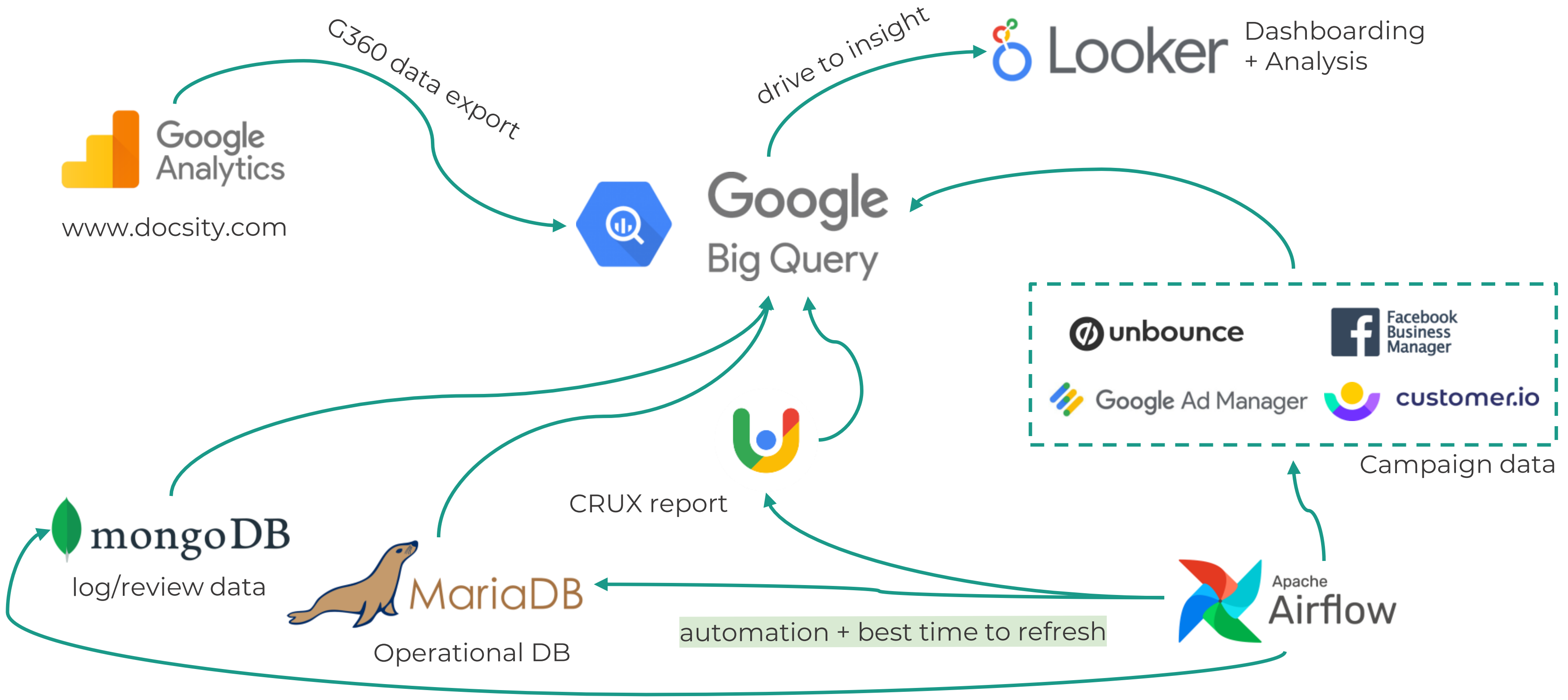
2021 Q1 data lake Orchestration



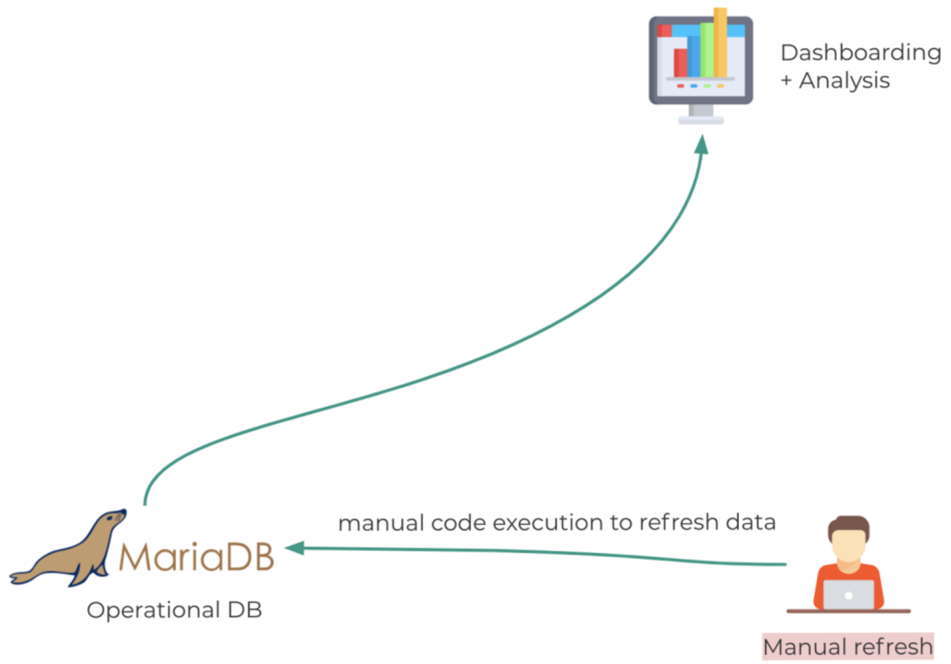
2021 Q2 data lake Orchestration



2021 Q3 data lake Orchestration



PRE Airflow



SIMPLE

ERRORS

TIME SPENT

LACK OF CONTROL

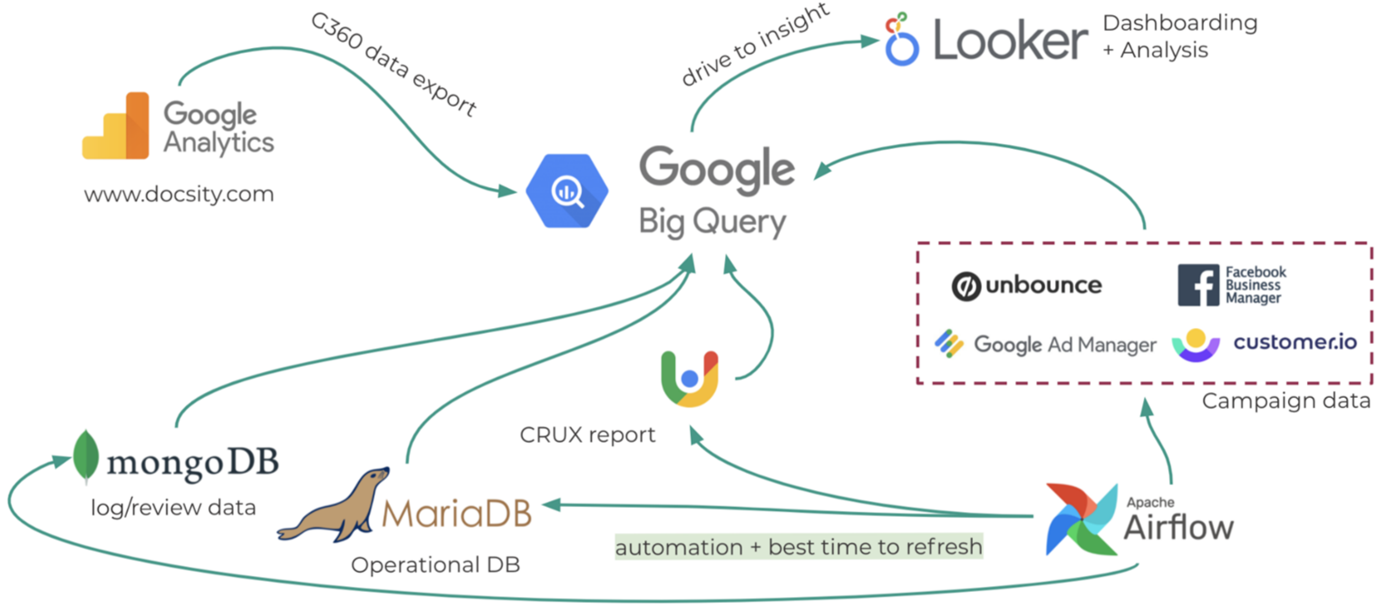
Airflow

MANAGEMENT

MONITORING

SECURITY

VERSIONING



Thanks!

Does anyone have questions?

matteo.s@docsity.com
https://github.com/pualien
irene.s@docsity.com
docsity.com