Airflow at DigitalOcean

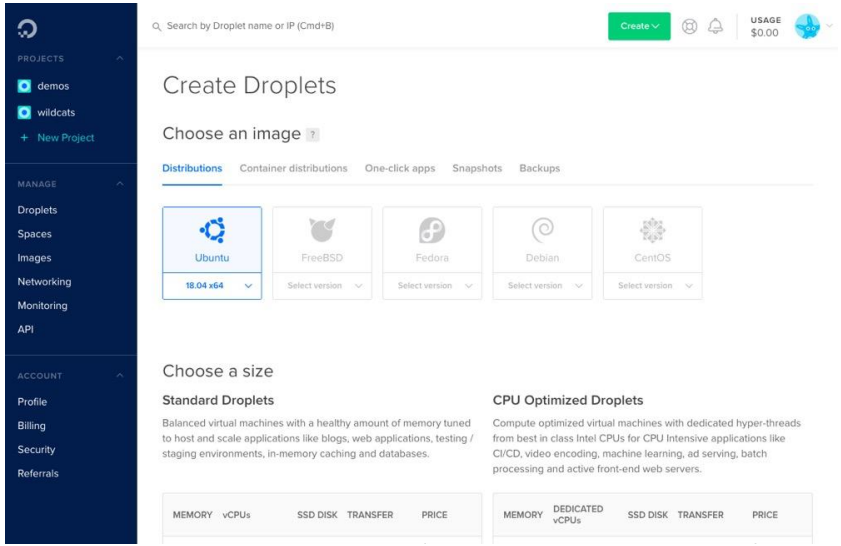
Adam Boscarino, Data Engineer
NYC Data Engineering Meetup, 2018-10-24

- Data Engineer at DO for ~1 year
- Previously at Carbonite and Fitbit
- Worked with Airflow for ~2 years
- Github & Twitter: ajbosco


DigitalOcean

Cloud Platform for Developers & Teams

- Focused on simplicity
- Offering compute, block storage, object storage, networking, and monitoring
- Managed Kubernetes!



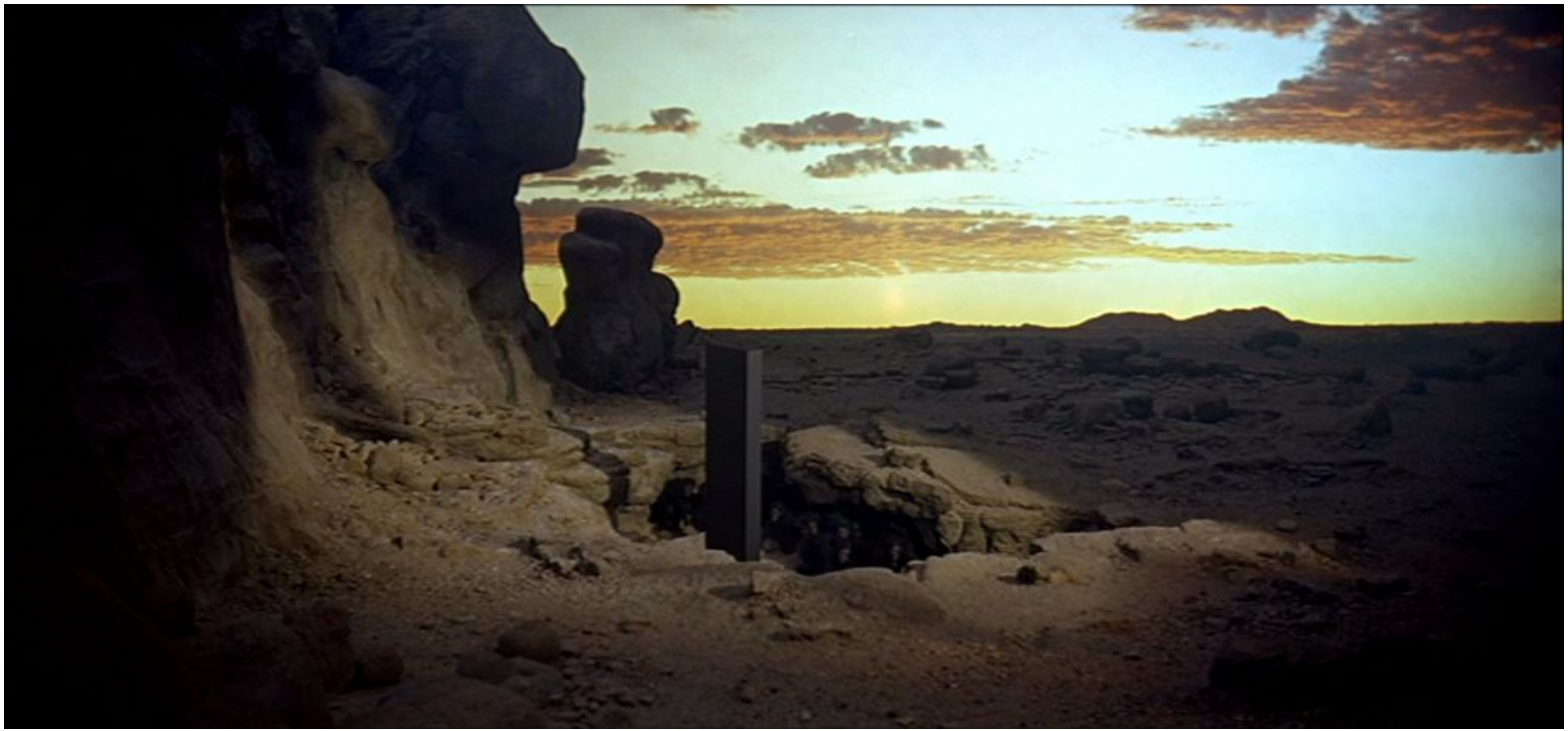
Data Engineering at DO

Mission: Develop and maintain tools and platforms to ingest, transform, and preserve data from a wide variety of sources, internal and external, in order to provide access for analysis, reporting, and data science modeling.

What we use:

- Looker
- Hadoop
- Presto
- Kafka
- **Airflow!**

IN THE BEGINNING...



THERE WAS **CRON**

Cron Scheduled Pipelines

- Over 60 regularly scheduled routines (usually once per day)
- Deployed on several virtual machines
- Logging via stdout/stderr
- Unable to manage dependencies
- Notifications enabled through *additional* cron jobs
- Duplicate code, especially for common data source access

AND THEN THERE WAS AIRFLOW

Why Airflow?

Research:

- Previous experiences
- Contribution activity
- Lots of testimonies from happy implementers

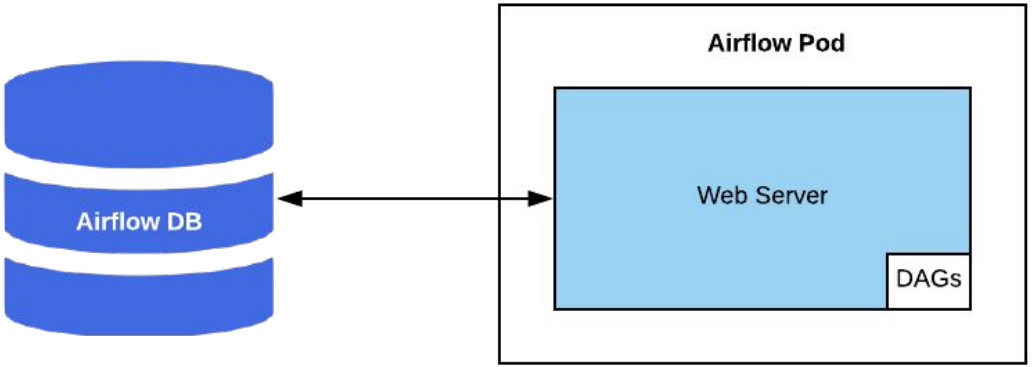
Leverage existing DO tools:

- Deployment with Kubernetes
- Configuration as Code makes CI/CD easier
- Python - DO Data Eng's language of choice

Solved Pain Points:

- Automatic retries
- Enforce dependencies across DAGs
- Alerting on failures (Slack/PagerDuty)
- Alerting on SLAs (data timeliness)

Airflow Architecture - Initial Setup



Problem Solved!

Issues with Initial Setup

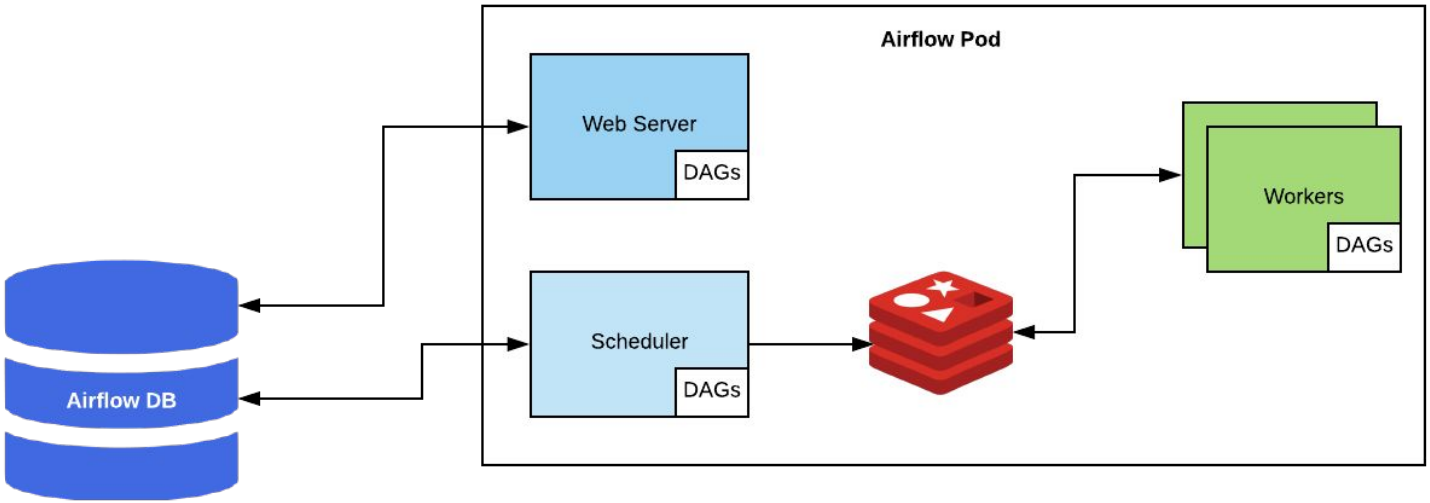
- Local Executor - Limiting DAG Concurrency
- Interest from other teams!
- Legacy jobs not designed with Airflow in mind
- Not part of Data Science team’s workflow

Scaling with Celery

Scaling with Celery Executor

- Split Web Server and Scheduler to separate containers
- Requires Celery broker (Redis, RabbitMQ, etc.)
- Easy to add additional workers
- Support for Pools and Queues
- Flower UI for monitoring

Airflow Architecture - Celery



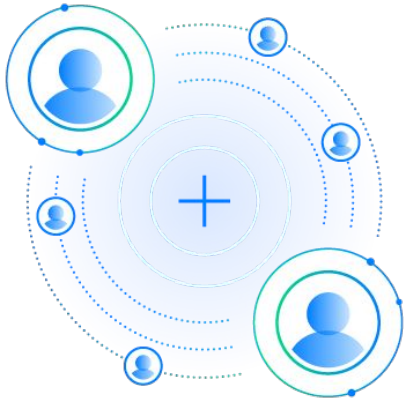
Not everything is perfect

- Additional complexity
- On k8s, more containers = more YAML
- Workers require all dependencies for DAGs
- Workers are not elastic - wasted resources
- Workers need to be sized properly - silent failures!

Working with Multiple Teams

Multiple Teams & Airflow

- Onboarding
- Deployment
- Resource Management



Onboarding to Airflow

Problem:

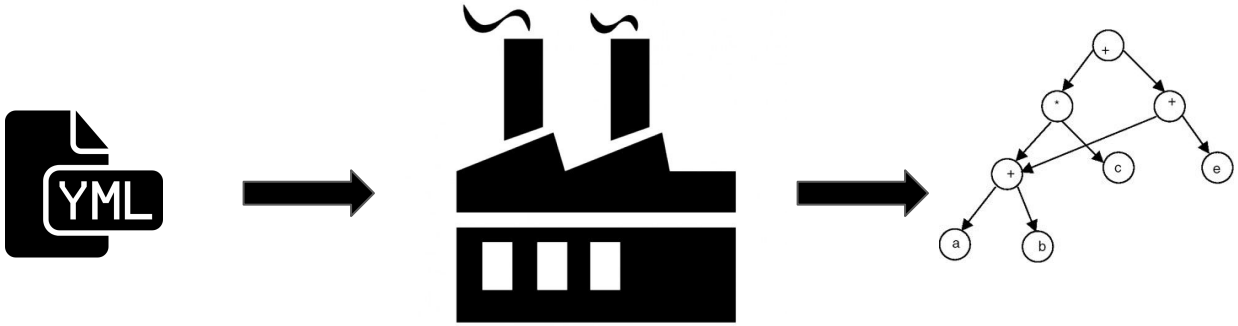
- Teams want to schedule jobs, not learn Airflow

Solution:

- Start with “simple tasks” that are developed outside of Airflow (Spark Submit or Docker)
- Abstract Airflow primitives away with DAG factories for common workflows

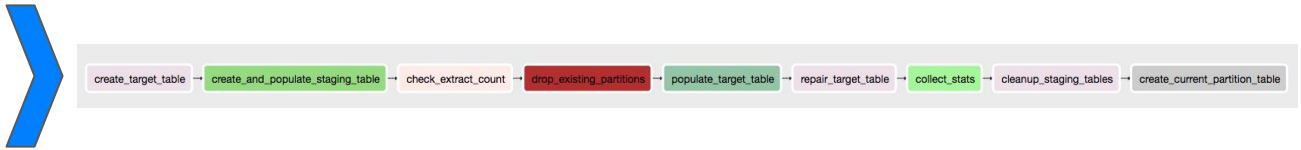
DAG Factories

- YAML configuration files for each common pattern
  - MySQL to HDFS
  - Data Warehouse Tables
  - Spark Jobs
- Files are parsed dynamically via Python scripts into Airflow DAGs
- Each entry generates a DAG
- Extensible - Easy to add additional tasks to multiple DAGs



DAG Factories - MySQL to HDFS

```yaml
demo_dag:
  dag_start_date: 2018-10-24
  retries: 4
  retry_exponential_backoff: True
  source_database: source_db
  source_schema: source_schema
  source_system: mysql
  create:
    table_type: external
    columns:
      id: INT
      name: STRING
      created_at: TIMESTAMP
      updated_at: TIMESTAMP
    partition:
      load_date: DATE
    file_format: parquet
    tblproperties:
      PARQUET.COMPRESS: SNAPPY
  extract_select:
    - id
    - name
    - created_at
    - updated_at
    - CAST('{{ ds }}' AS DATE) as load_date
  load_type: overwrite
  current_partition_table: True
```



DAG Factories - Dimension Tables

```yaml
scd_example:
  dag_start_date: 2018-10-24
  dag_depends_on_past: True
  include_metadata: True
  versioned: True
  create:
    columns:
      id: INT
      user_id: INT
      is_active: BOOLEAN
      create_time: TIMESTAMP
      distribution_name: STRING
    file_format: parquet
    tblproperties:
      PARQUET.COMPRESS: SNAPPY
  dag_dependencies:
    source_dag_1: populate_target_table
    source_dag_2: populate_target_table
    source_dag_3: populate_target_table
  source_query: files/sql/dimensions/scd_example/extract.sql
  comparison_key:
    user_id: INT
    is_active: BOOLEAN
  unique_key:
    id: INT
  load_type: overwrite
  views:
    scd_example_current: files/sql/dimensions/view.current.sql
```

Resource Management with Airflow

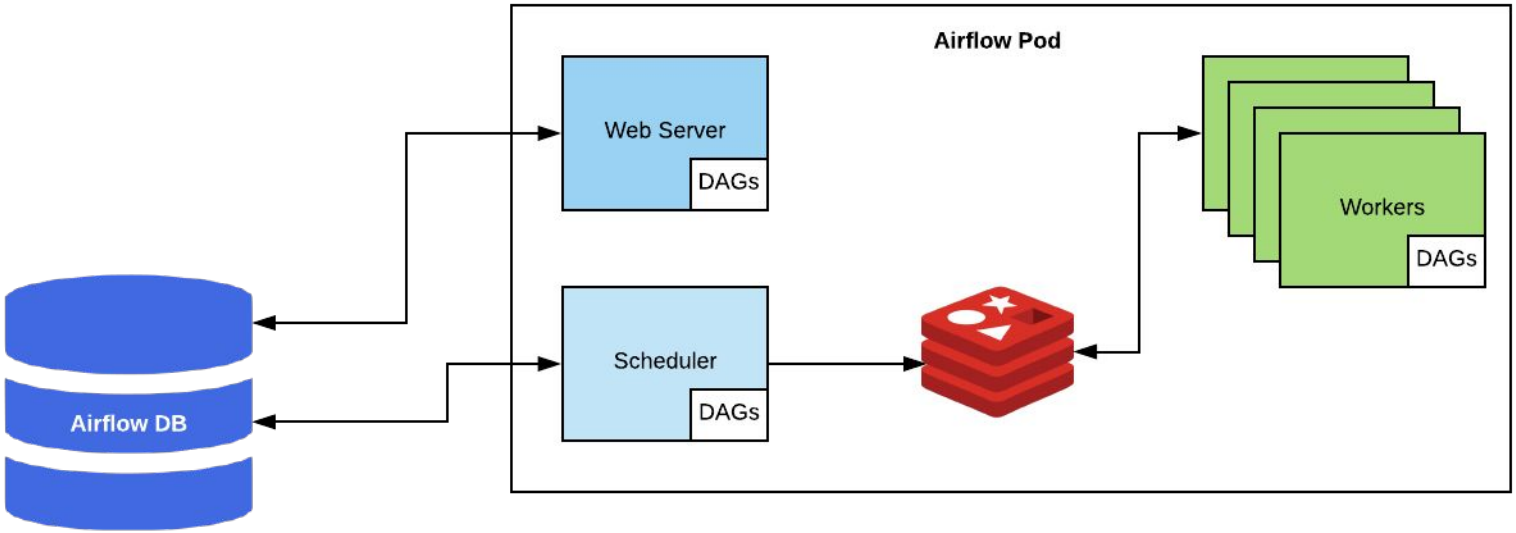
Problem:

- A “bad” DAG can use all task instance slots and block other DAGs

Solution:

- Use Airflow Pools to limit resources for each team
- Scale with Celery :)

Airflow Architecture - Celery & Multiple Teams



Deploying to Airflow

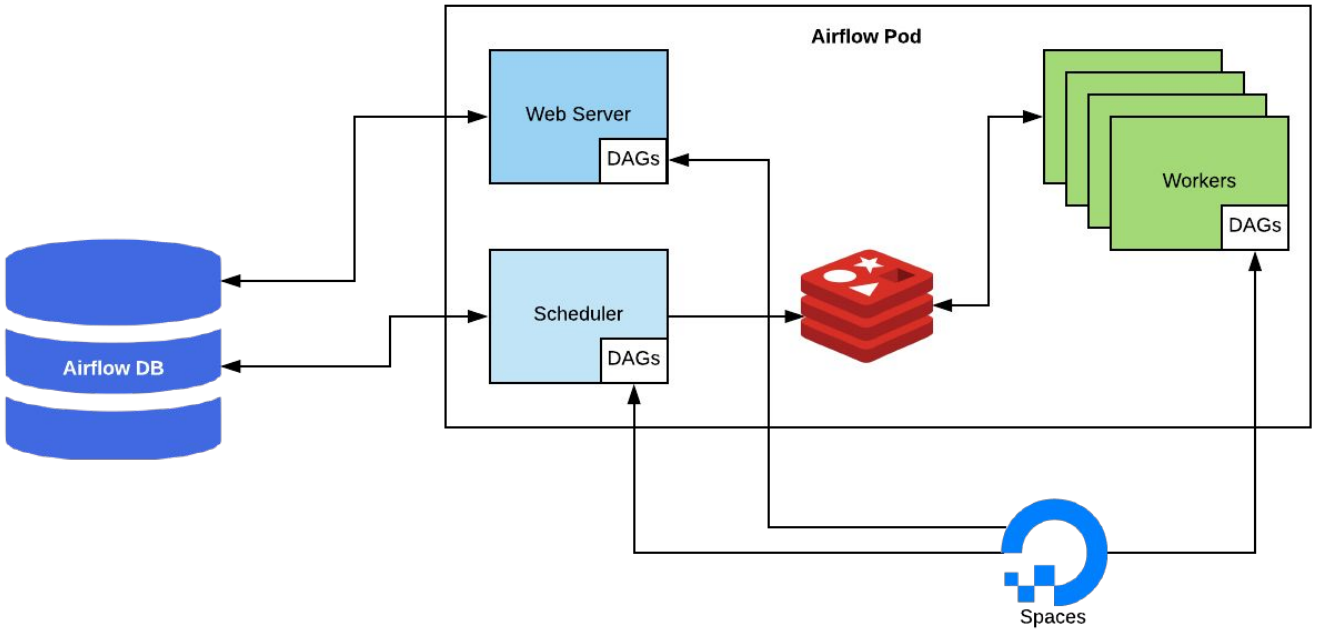
Problem:

- More teams, mean more frequent DAG changes and more deployments

Solution:

- Use Smoke Tests to verify DAGs can import
- Separate out DAG deployment from Airflow deployment
    - Use sidecar containers to load DAGs from GitHub or S3/Spaces

Airflow Architecture - Current



Kubernetes Pod Operator

Kubernetes Pod Operator

- Added in Airflow 1.9
- Developed by Bloomberg, Google, and others
- Creates Kubernetes pod for each task
- Run any task in a Docker container
- Simplifies dependency management
- Can use Kubernetes Secrets

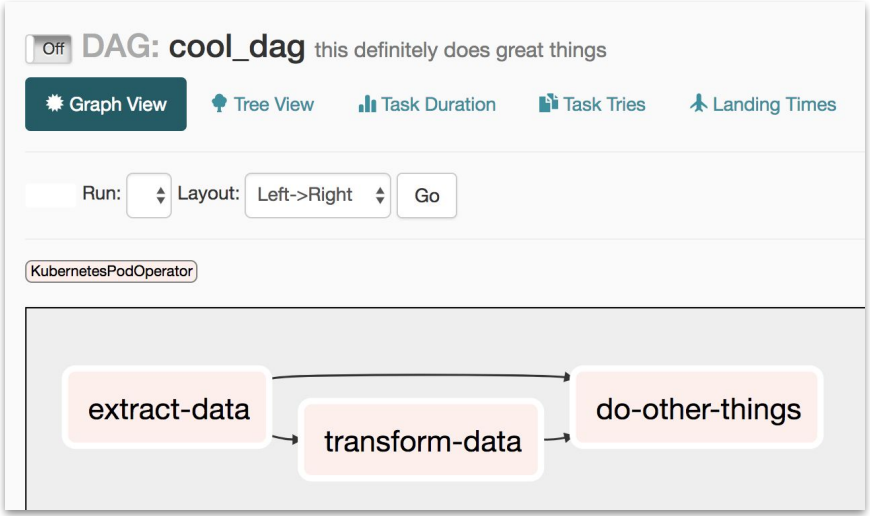
```
KubernetesPodOperator(
    namespace="default",
    image="ubuntu:16.04",
    cmds=["bash", "-cx"],
    arguments=["echo", "10"],
    labels={"foo": "bar"},
    name="airflow-test-pod",
    in_cluster=False,
    task_id="task",
    get_logs=True,
    dag=dag,
    is_delete_operator_pod=False,
    tolerations=tolerations,
)
```

Use Cases for Kubernetes Pod Operator

- Simple migration for existing cron jobs
- Easy to integrate into existing Data Science workflows
- Dynamic DAGs from config to reduce learning curve

Dynamic DAGs with Kubernetes Pod Operator

```yaml
cool_dag:
  dag_start_date: 2018-10-24
  dag_owner: 'dse'
  dag_description: "this definitely does great things"
  schedule_interval: '0 12 * * *'
  tasks:
    extract_data:
      task_type: kubernetes_pod
      image: docker-image
      cmds: ["python",
        "extract-data.py"]
    transform_data:
      task_type: spark_submit
      conf: DEFAULT_SPARK_CONF
      application: transform.py
      application_args:
        - -destination target_table
      image: spark_image
      dependencies: [extract_data]
    do_other_things:
      task_type: kubernetes_pod
      image: docker-image
      cmds: ["python",
        "do-other-things.py"]
      dependencies: [extract_data, transform_data]
  alert_type: slack
```



Future Plans

- More teams working with Airflow
- Kubernetes Executor
- Git Sync for DAG Deployment
- More robust Data Quality framework on top of Airflow

Thank you!