Airflow Summit
September 19 - 21, 2023
Toronto, Canada
airflowsummit.org

Let's flow together

# Airflow at Reddit - How we migrated from Airflow 1 to Airflow 2

Dave Milmont
Senior Software
Engineer at Reddit

Branden West
Software Engineer III
at Reddit

01.

# Infrastructure Overview

reddit

# Scale

***1500+ DAGs***

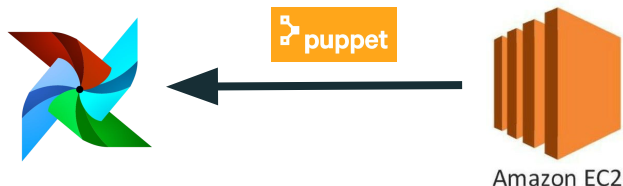
***30000+ Daily Tasks***

***Over a Petabyte of Data Processed Daily***



reddit

# Old Setup and its Challenges

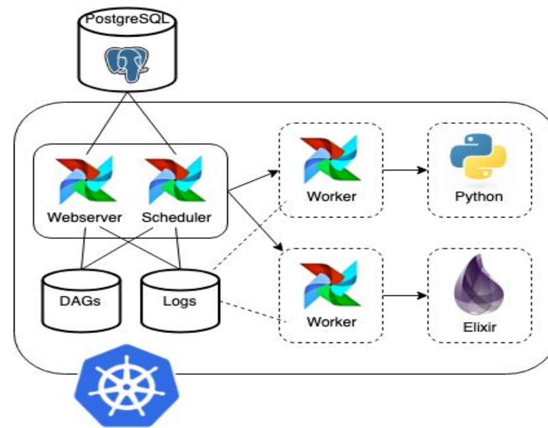



- Airflow deployed on single ec2 instance managed with puppet, with unknown amount of manual modifications.
- Awkward DAG deployment tooling - manual process that took ~4 minutes.
- Python dependency hell
- LocalExecutor
- DAG backilling was very difficult/impossible to perform.
- Reddit was experiencing large growth. Scaling this old infrastructure was challenging.
- No staging environment



reddit

# Moving to Kubernetes




- Reddit manages our own kubernetes clusters
- Airflow Scales easily on kubernetes
- KubernetesExecutor:
  - Ability to handle processing tasks on specific nodes.
  - Containers for different environments and languages e.g. Scala job
  - We can scale up when necessary and then release resources when finished.
- Take advantage of the official airflow helm chart - a more standard deployment.



reddit

02.

# Migration Philosophies

reddit

# Infra Migration Strategy

## Option 1: In Place

### Steps

- Upgrade ec2 based airflow to the bridge release
- Update DAG code to be compatible with airflow 2.0
- Point DAGs to airflow 2 running on k8s.

### Pros:

- Simplest approach - focus only on DAG compatibility.
- DAG dependencies remain in place.

### Cons:

- Resolving delta between puppet and ec2 - major problem.
- Untangling python dependency hell.
- Unknown amounts of downtime that might occur from the previous two actions.



reddit

# Infra Migration Strategy

## Option 2: Shift to Separate Airflow 2 Instance

### Steps

1. Stand up new airflow 2 instance on k8s.
2. Shift DAGs to new repository, update code to work on airflow 2 from there.

### Pros:

- Allows airflow instance and repo to start from a clean slate using best practices.
- Precarious airflow 1 instance does not need to be modified, less risk of downtime.

### Cons:

- More time consuming than in place, and less convenient.
    - Moving code between repositories.
    - DAG dependencies can be tricky to handle.

# DAG Migration Strategy: Big Bang vs Piecemeal

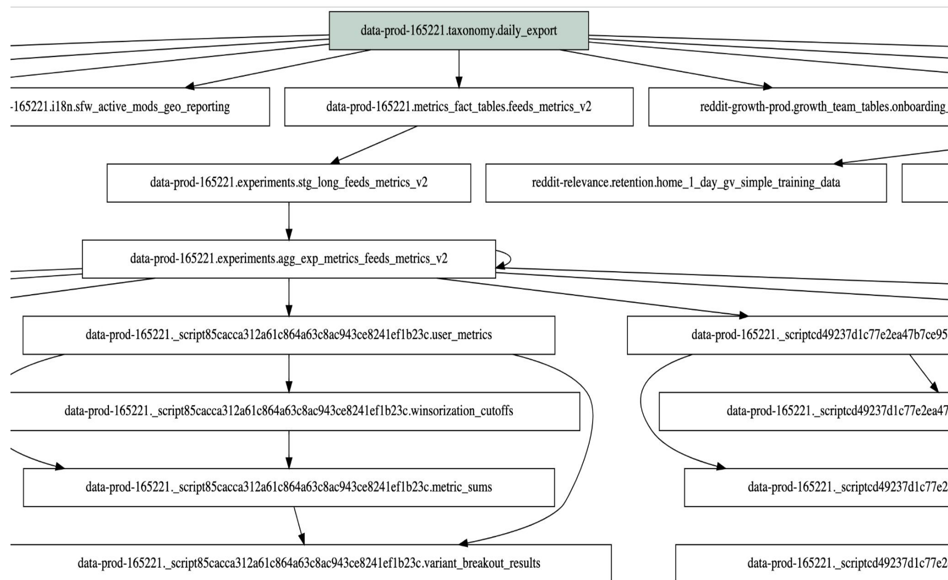
- **Big Bang**: Migrate all DAG’s over all at once. Turn on all at the same time.
  - Pros: DAG dependencies no longer a problem
  - Cons: Requires total code freeze
- **Piecemeal:** Separately migrate groups of dependent DAG’s
  - Pros: Can tackle in smaller batches
  - Cons:
    - Difficult for DAG owners to coordinate dependencies.
    - Most of our DAGs depend on each other.



reddit

# Why we chose the Big Bang approach

- **Complex Dag Dependencies prohibited a piecemeal approach**
  - The large majority of our DAGs were dependent on each other.
- **Big Bang allows for:**
  - A large but brief code freeze
    - No Code Drift






reddit

reddit

02.

# Migration Steps

reddit

# How we determined which DAG's were actually being used

- Vast majority of our DAGs write to tables in BigQuery.
- Find DAG to table mapping, and find usage of table.
- Rank DAG's by usage
- Cut all the DAG's whose tables were unused
  - This meant tables that were only accessed by Airflow, no other users

```
1  import copy
2  from pathlib import Path
3
4  from airflow.models import DagBag
5
6  dag_path = Path("dags")
7
8  dagbag = DagBag(dag_folder=str(dag_path.absolute()), include_examples=False)
9  data = {}
10
11 for dag_id in dagbag.dag_ids:
12     dag = dagbag.get_dag(dag_id)
13     owner = dag.owner
14     tables_list = []
15     for ti in dag.get_task_instances():
16         task_id = ti.task_id
17         task = copy.copy(dag.get_task(task_id))
18         if task.task_type == "BatchBigQueryOperator":
19             tables_list.append(task.destination_dataset_table)
20     data[dag_id] = {"tables": set(tables_list), "owner": owner}
21
22 for dag_id, values in data.items():
23     for table in values["tables"]:
24         print(f"{dag_id},{table},{values['owner']}")
```

This gets us a map of DAG's to the table(s) they write to



reddit

# Migration Timeline/Comms/Instructions

- Code Freeze
- Communication is key
- Team Effort - dividing up work among team
- Swapping out airflow 1.0 operators to airflow 2.0 operators using provider packages
- Runbooks are useful

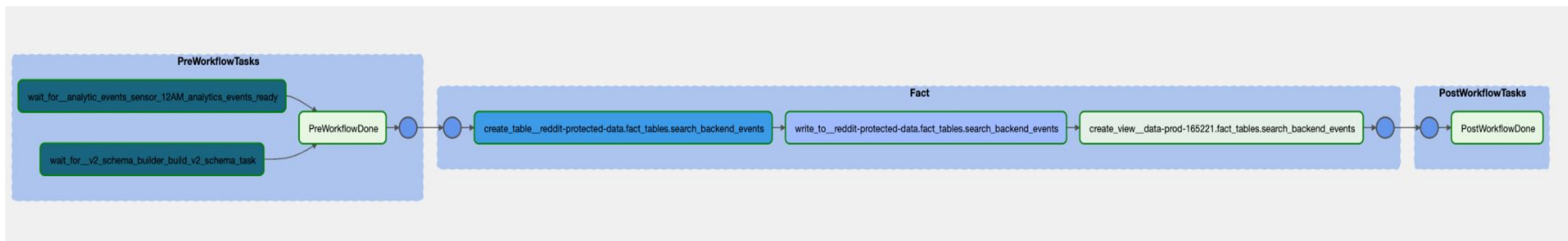
03.

# DAG Factory

reddit

# What is a DAG Factory

- Creates DAG’s dynamically from a config file
    - Reduces boilerplate

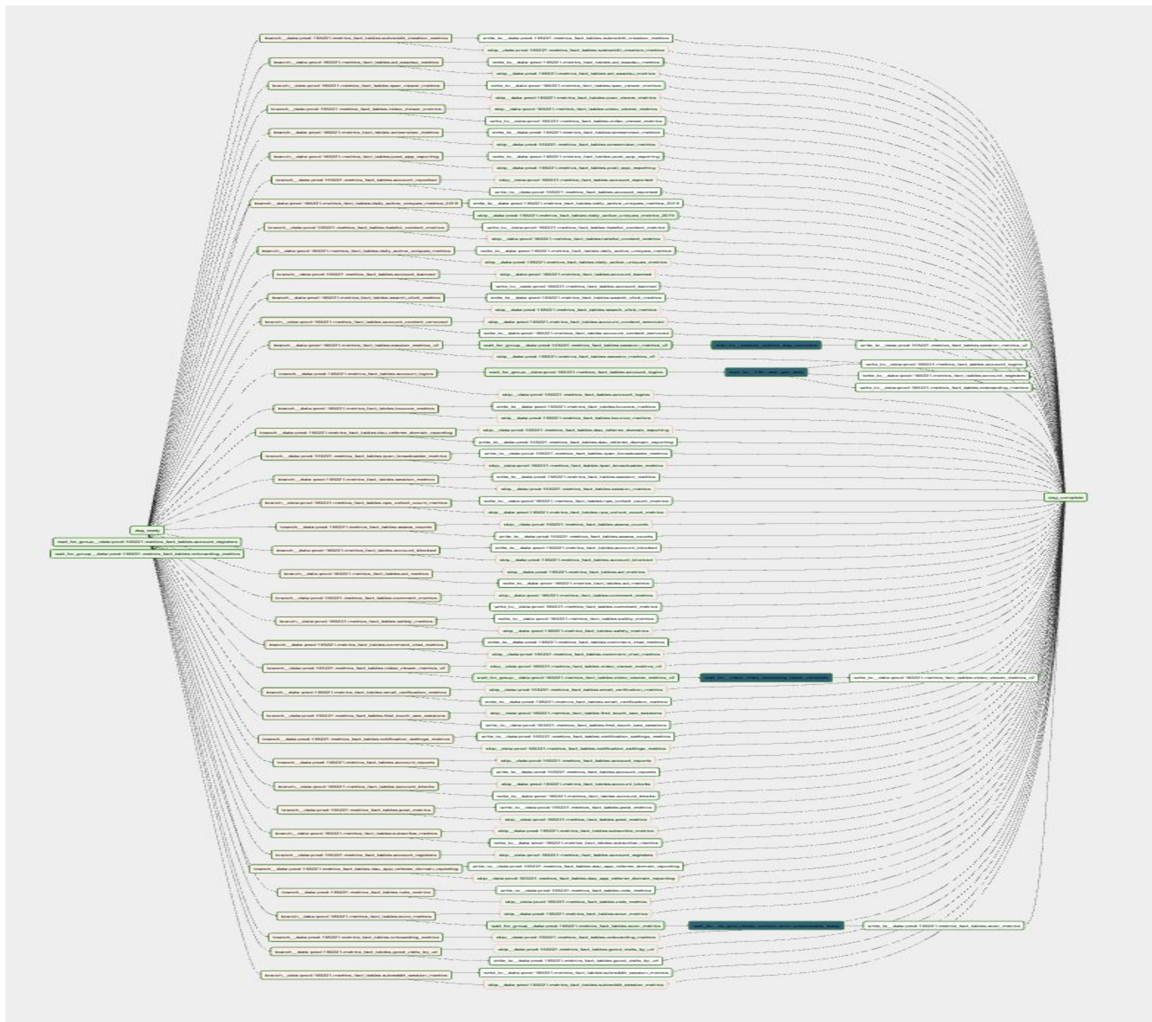





reddit

# Issues we Faced

- No consistent pattern for writing DAG's
  - Disorganized
  - Prone to errors
- Some DAG's were grouped into "mega dags"
  - Backfills were time consuming and computationally expensive
  - Brittle – a failure in one task would delay all others



# How a DAG Factory Helps

- DAG’s are all written in a consistent way.
- Easy, self-serve way to create a new DAG. Just need to create a config file.
- Saved us a lot of effort during migration
- Easy to add new capabilities that automatically apply to all DAG’s at once

# Finding our most used Operators

- Finding list of most used operators to see how we can create workflows
- 80/20 principle
- Most DAG's involved:
  - sensing on upstream tables
  - creating a new table, writing to that table
  - creating a view on that table
  - triggering a dashboard
- Slack and Email were our main alerting operators

```python
from airflow.models import DagBag
from collections import Counter

subdir = "dags"
dagbag = DagBag(dag_folder=str(subdir), include_examples=False)

def get_dag_task_types(dagbag: DagBag):
    tasks = []

    for dag_id in dagbag.dag_ids:
        dag = dagbag.get_dag(dag_id)
        for _, values in dag.task_dict.items():
            tasks.append(values.operator_name)
    counts = Counter(tasks)
    percents = [(i, counts[i] / len(tasks) * 100.0) for i in counts]
    return counts, percents, tasks


counts, percents, tasks = get_dag_task_types(dagbag)

```



reddit

03.

# Challenges and Learnings

reddit

# Challenges

- Migrations are hard and take a long time.
- Requires buy-in, communicate the benefits!
- Documentation on the new system - how to use it.
- No interruptions during the migration process.



reddit

# Learnings

- Team effort and great communication is essential.
- Plan for contingencies (failures, runbooks)
- Reduce the migration workload as much as possible. Don’t blindly migrate everything.
- Tooling to make migration easier. DAG factories are great!



reddit

# Thanks for listening!!