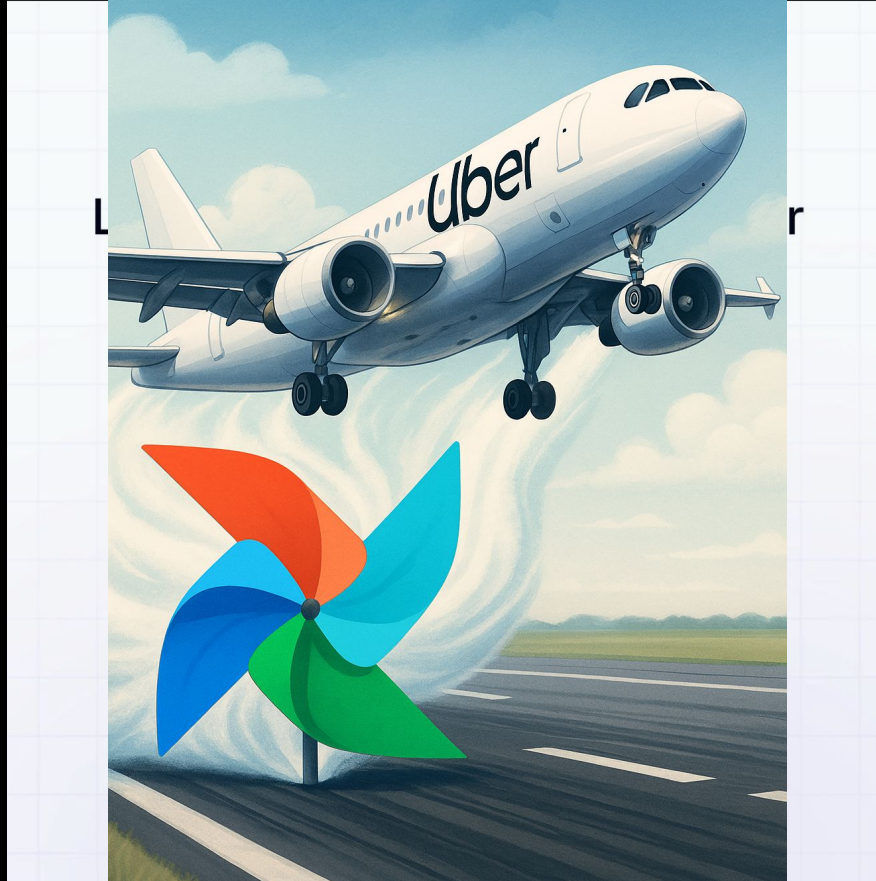# AIRFLOW SUMMIT

Operation Airlift

Uber's ongoing journey to migrate to Airflow 3

Sumit Maheshwari
Tech Lead @Uber, PMC Apache Airflow

3.0

Agenda

- **The Story of Piper:** A Look Back
- **The Need for Change:** Challenges with Piper
- **The Path Forward:** Why Apache Airflow?
- **Under the Hood:** Implementation Details
- **Making the Move:** Adoption & Migration
- **Conclusion & Next Steps**

Recap

- Uber's internal fork of Airflow
- Called Piper, born out of Airflow 1.x in 2017
- 200K+ Pipelines
- 800K+ TIs per day
- 1000+ Celery machines
- 50K+ python files
- 1K+ teams

https://youtu.be/EhrtShXSrsw

Evolution of Airflow at Uber

Presented at Airflow Summit 2024

Up until a few years ago, teams at Uber used multiple data workflow systems, with some based on open source projects such as Apache Oozie, Apache Airflow, and Jenkins while others were custom built solutions written in Python and Clojure.

Every user who needed to move data around had to learn about and choose from these systems, depending on the specific task they needed to accomplish. Each system required additional maintenance and operational burdens to keep it running, troubleshoot issues, fix bugs, and educate users.

After this evaluation, and with the goal in mind of converging on a single workflow system capable of supporting Uber's scale, we settled on an Airflow-based system. The Airflow-based DSL provided the best trade-off of flexibility, expressiveness, and ease of use while being accessible for our broad range of users, which includes data scientists, developers, machine learning experts, and operations employees.

This talk will focus on scaling Airflow to Uber's scale and providing a no-code seamless user experience

⬇ Download slides

Shobhit Shah
Staff Software Engineer at Uber

Sumit Maheshwari
Tech Lead at Uber, PMC Apache Airflow

Divergence

- DAG serialization format defined in JSON (17-18)
- Scheduler was divided and converted into 4 different services (17-18)
    - Organizer (Python) - Similar to DAG processor to serialize pipelines into JSON
    - Orchestrator (Java) - To assign pipelines to different scheduler nodes using Zookeeper
    - Scheduler (Java) - The brain of the system to do the real scheduling of tasks
    - Prioritizer (Java) - Service to enqueue tasks into Redis
- UI driven backfills (19)
- Connections backed by internal secrets platform (19)

Divergence - cont

- In house Disaster Recovery architecture (21)

- Drag & Drop interface to create pipelines (uWorc) (20-21)

- Utilizing read-replicas for load distribution from primary Mysql node (22)

- Jumpstart - Data aware scheduling (22)

- Various framework based pipeline creation frameworks (OneETL etc) (22)

- Full fledged workflow governance (23)

- Hybrid strategy - on-prem and cloud infra running in parallel (24)

- GenAI analysis for task failure summary & mitigation steps (25)

The Need For Change

Challenges With Piper

3.0

Piper – Pain Points

Modernization

No Task Level Identities

Slow Version Upgrades

Efficiency

Duplicate Pipelines

No Dynamic Pipelines

Missing Event Driven Scheduling

Reliability

Celery + Redis

Noisy Neighbours

Safe Deployments

Dev Velocity

Lack of local dev setup

No proper REST APIs

Outdated SDK authoring exp

The Path Forward

Why Apache Airflow

3.0

Why Airflow

Popularity

Most Popular

Battle Tested

Active Community

EcoSystem

Plenty of Connectors

Support of all major

cloud vendors

Familiarity

Similar DSL to Piper

Known-unknowns

Airflow 3

Isolated workers

Modern UI & APIs

DAG versioning

Tasks in any lang*

Piper <> Airflow

	Feature	Piper	Airflow 3	Feature	Piper	Airflow 3
1						
2	Serialised DAGs	Y	Y	Isolated Workers	N	Y
3	Scheduler HA	Y	Y	REST APIs	Y*	Y
4	UI Backfills	Y	Y	Custom Timetables	N	Y
5	Dynamic DAGs	N	Y	One pipeline for Trigger & Schedule	N	Y
6	Decorators	N	Y	Disaster Recovery	Y	N
7	Async Tasks	Y	Y	Drag & Drop UI	Y	N
8	K8s Executors/Operators	N	Y	YAML Frameworks	Y	Y*
9	Secret Manager	Y	Y	Governance	Y	N
10	Custom XCom Backends	N	Y	DB Read Replicas	Y	N
11	Data Aware Scheduling	Y*	Y	Non-Pythonic Tasks	N	Y*
12	Event Driven Scheduling	N	Y	GenAI Summary & Mitigation	Y	N
13	DAG Versioning	N	Y	Open Source Community	N	Y

Convergence

- Piper's backfill replaced by Airflow's native backfill functionality

- YAML based frameworks like OneETL rewritten using Dag-Factory

- Data aware scheduling (Jumpstart) moves to Airflow's event driven scheduling

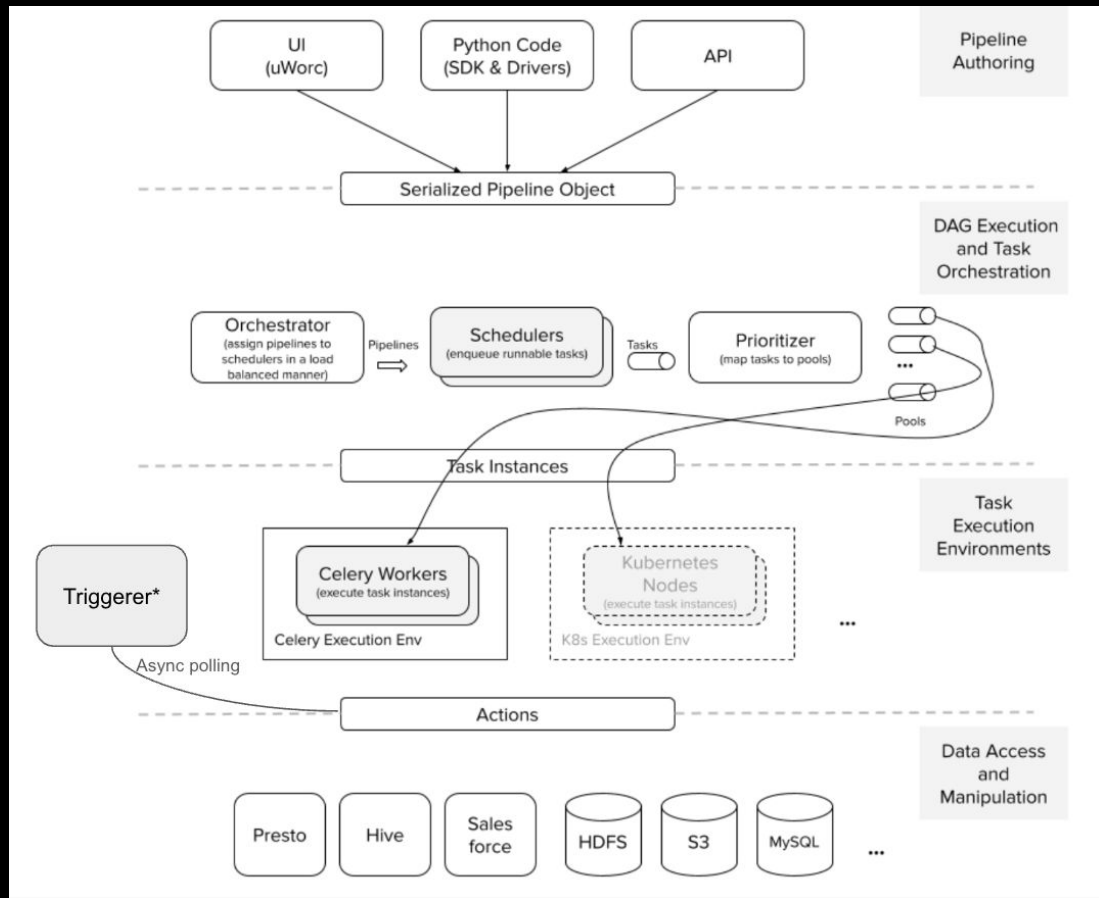- uWorc (drag & drop UI) gets upgraded to work with Airflow as well

Challenges

- DAG-Processor is still running in control-plane, exposing metadata DB (AIP-92)

- All Airflow services connects to the same database in read-write mode (AIP-94)

- Airflow's inbuilt Scheduler HA may not be sufficient for Uber's scale (AIP-XX)

- Disaster Recovery framework needs to be rewritten for Airflow

- Missing safe-deployment constructs like auto rollbacks, incremental rollouts, etc
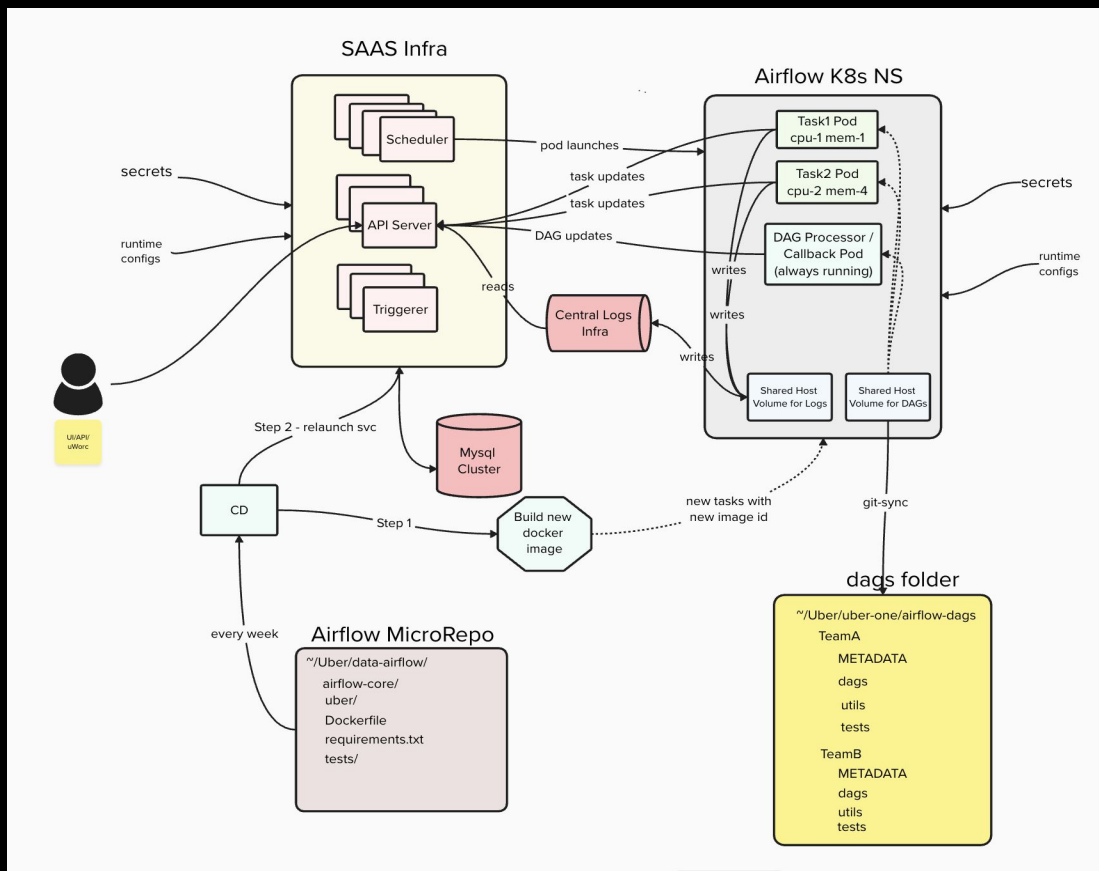
Under The Hood
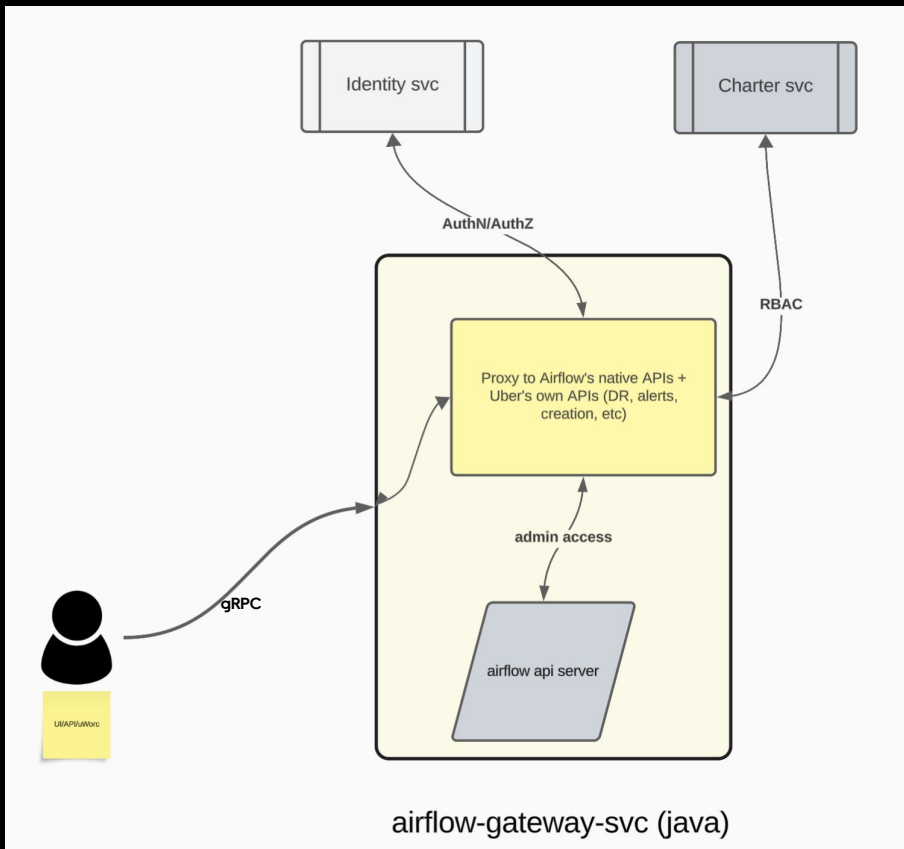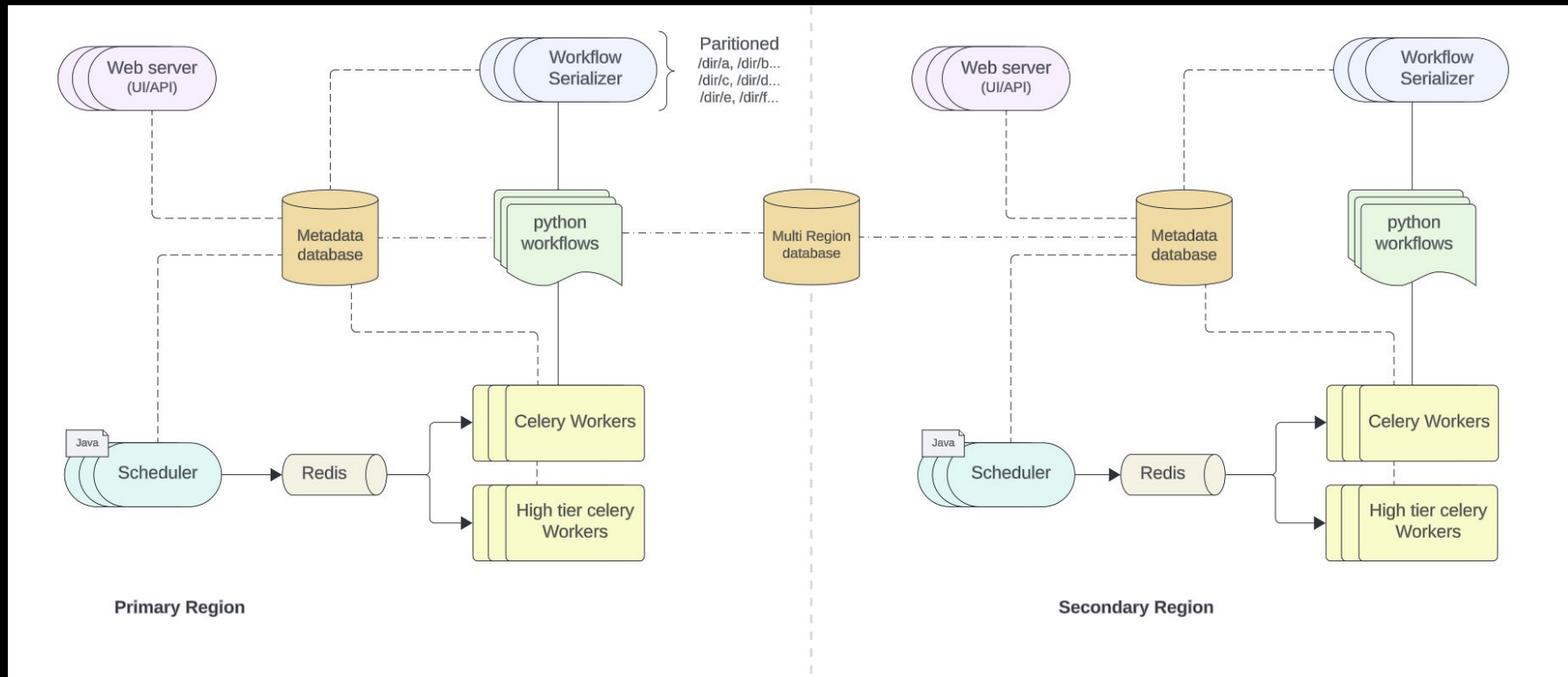
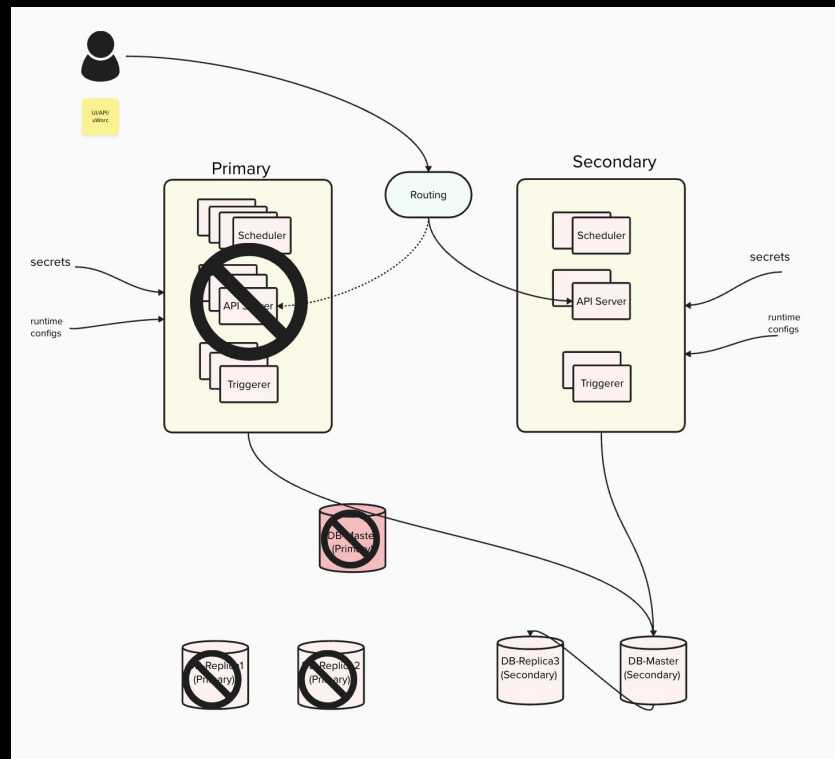**Implementation Details**

3.0

Piper HLD

Airflow HLD

AuthN & AuthZ

Airflow-web-gateway service written in Java to do all Uber specific authentication & authorization out of Airflow core.

airflow-gateway-svc (java)

Piper - DR

Airflow – DR

Making The Move

Adoption & Migration

3.0

Types of Pipelines

SDK

Pythonic in nature

Syntax similar to Airflow 1.x

Managed

Drag & Drop pipelines

Stored in DB as JSON

Backfill

Associated with Parent Pipelines

UI based creation

Stored in DB as JSON

Framework

YAML based pipelines

Dozens of diff frameworks

Conversion of SDK Pipelines

```
# Convert a single pipelines
piperdev convert
pipelines/pipeline_watcher/pipeline_
tier_check_watcher.py

# Convert a whole folder
piperdev convert
pipelines/pipeline_watcher/
```
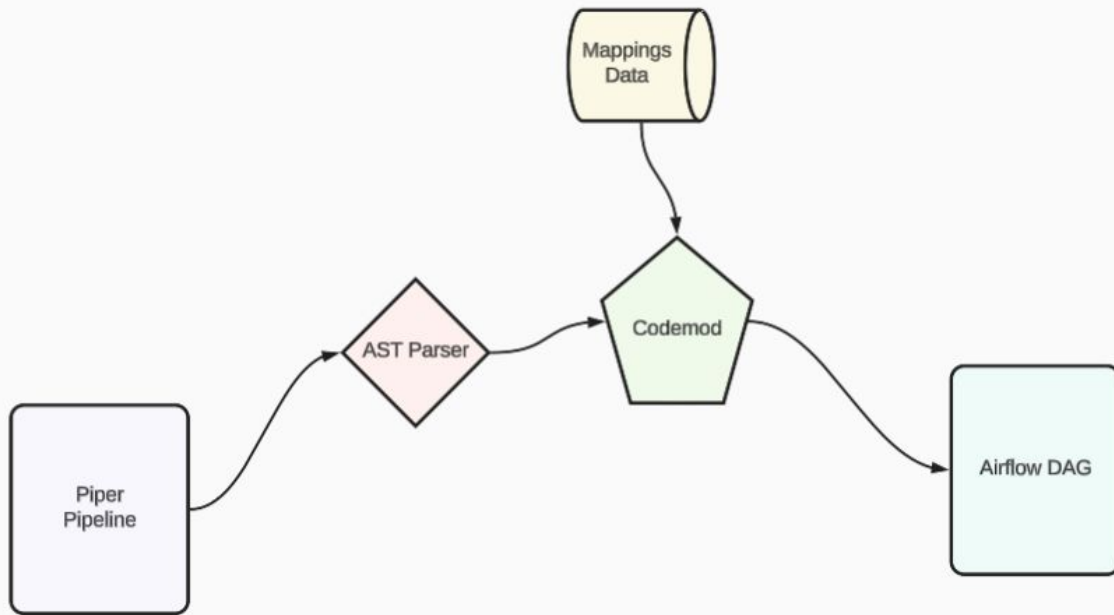
Left side (removals):

```python
1  # -*- coding: utf-8 -*-
2  """ Pipeline to watch pipeline failed status and publish stats """
3  from piper.models import Pipeline
4  from clay import config
5  from datetime import datetime, timedelta
6  from piper.tasks.dummy_task import DummyTask
7  from pipelines.core.dwm_admin.pipeline_watcher.tasks.watch_pipeline_failed_task
   import WatchPipelineFailTask
8  from piper.models.pipeline import RUN_IN_ALL_DATACENTERS
9  logger = config.get_logger('piper')
10 POOL_NAME = 'piper_admin'
11 PID = 'metric_pipeline_fail_watcher'
12 args = {
13 'owner': 'data_workflow_management',
14 'owner_ldap_groups': ['piper', 'data_workflow_management'],
15 'depends_on_past': False,
16 'auto_backfill': False,
17 'pool': POOL_NAME,
18 'email': ['redacted@abc.com'],
19 'retries': 3,
20 }
21 pipeline = Pipeline(
22 pipeline_id=PID,
23 schedule_interval=timedelta(hours=4),
24 secure=True,
25 start_date=datetime(2020, 5, 6, 0, 0, 0),
26 default_args=args,
27 datacenter_choice_mode=RUN_IN_ALL_DATACENTERS
28 )
29 # last task
30 complete_task = DummyTask(task_id='complete', pipeline=pipeline, default_args=args)
31 watch_pipeline_failed_task = WatchPipelineFailTask(task_id='watch_pipeline_fail',
   pipeline=pipeline, default_args=args)
32 watch_pipeline_failed_task.set_downstream(complete_task)
```
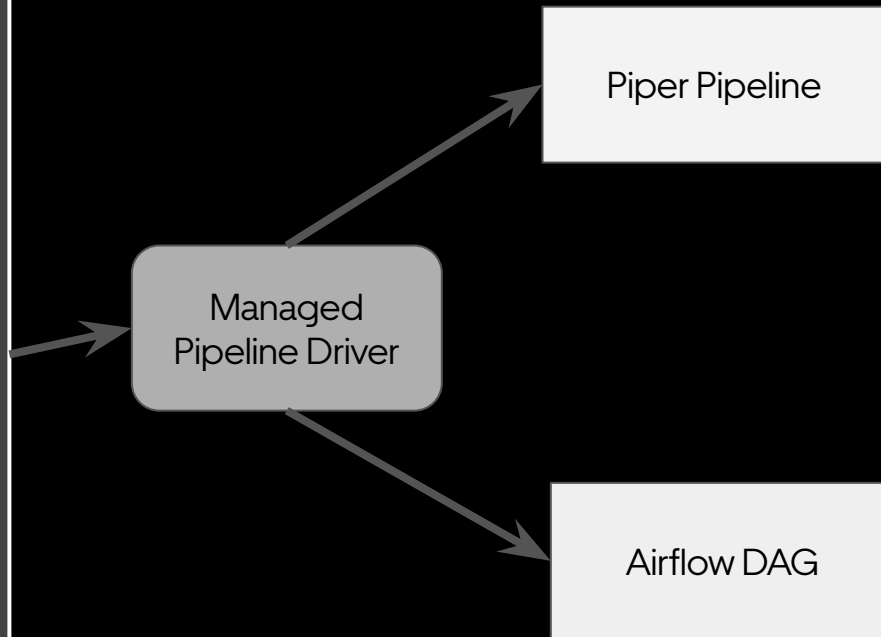
Right side (additions):

```python
1  """ Pipeline to watch pipeline failed status and publish stats """
2  from airflow.models import DAG
3  from airflow.configuration import conf as config
4  from datetime import datetime, timedelta
5  from airflow.providers.standard.operators.empty import EmptyOperator
6  from pipelines.core.dwm_admin.pipeline_watcher.tasks.watch_pipeline_failed_task
   import (
7  WatchPipelineFailTask,
8  )
9  POOL_NAME = "piper_admin"
10 PID = "metric_pipeline_fail_watcher"
11 args = {
12 "owner": "data_workflow_management",

13 "depends_on_past": False,

14 "pool": POOL_NAME,
15 "email": ["redacted@abc.com"],
16 "retries": 3,
17 }
18 pipeline = DAG(
19 dag_id=PID,
20 schedule=timedelta(hours=4),

21 start_date=datetime(2020, 5, 6, 0, 0, 0),
22 default_args=args,

23 )
24 complete_task = EmptyOperator(task_id="complete", dag=pipeline)
25 watch_pipeline_failed_task = WatchPipelineFailTask(
26 task_id="watch_pipeline_fail", dag=pipeline)
27 watch_pipeline_failed_task << complete_task
```

Conversion of Managed Pipelines

```
{
  "owner": "abc",
  "tasks":
  [
    {
      "pool": "adhoc",
      "retries": 1,
      "task_id": "emit_metric_to_m3",
      "task_class": "pipelines.core.dwm..emit_metric_to_m3_task.EmitMetricToM3Task",
      "task_params": {},
      "dependencies": []
    }
  ],
  "end_date": "2025-04-13T16:00:48Z",
  "execution_engine": "airflow",
  "start_date": "2024-04-13T16:00:00Z",
  "pipeline_id": "000046de-f9af-11ee-af7e-1070fd426416",
  "json_version": "1.0",
  "pipeline_name": "managed_pipeline_test_sample",
  "owner_ldap_groups": ["piper"],
  "schedule_interval": 600,
  "selected_datacenters": [],
  "datacenter_choice_mode": "run_in_all_datacenters"
}
```

Piper Pipeline

Managed Pipeline Driver

Airflow DAG

Conversion of Framework Pipelines

Config directory format for pipelines

```
one_etl/
    config/
        ddl/
            //CREATE TABLE statement
            <table_1_name>.ddl
            <table_2_name>.ddl
        jobs/
            //Pipeline configuration
            <table_1_name>.yaml
            <table_2_name>.yaml
    sql/
        //Transformation logic
        <table_1_name>.sql
        <table_2_name>.sql
```
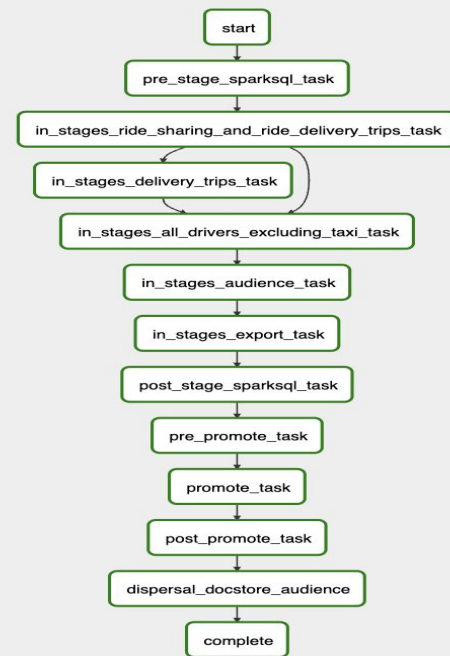
Example pipeline yaml

```
owner: alice
hive_table: schema.table
spark_opts:
  queue: spark-adhoc
ddl_file: <TEAM>/ddl/example_ddl.ddl
sql_file: <TEAM>/sql/example_sql.sql
execution_engine: spark
start_date: '2025-09-15'
hive_staging_schema: stg_schema
hive_test_schema: test_schema
hive_partition_key: datestr
execution_engine: airflow
```



```
start
pre_stage_sparksql_task
in_stages_ride_sharing_and_ride_delivery_trips_task
in_stages_delivery_trips_task
in_stages_all_drivers_excluding_taxi_task
in_stages_audience_task
in_stages_export_task
post_stage_sparksql_task
pre_promote_task
promote_task
post_promote_task
dispersal_docstore_audience
complete
```

Conclusion & Next Steps

3.0

End Goal

Timelines

Timeline chart with columns: March, April, and May | June, July, and August | September, October, and November | December, January, and February | March, April, and May | June, July, and August | September, October, and November | December, January, and February

Milestones: M0, M5, M1, M9, M10, M12, M15

POC
- Validate if Airflow 3 can work in Uber infra
- Run some basic tasks like Python, Spark, etc
  - Done · 3

BETA
- Define build strategy for Airflow & DAGs
- Integrate Airflow with internal K8s infra
- Provide user-staging env for users to do pvt branch testing
  - In Progress · ?

GA
- Convert custom task classes to Airflow
- Integrate with Uber's internal tooling, like uWorc, OneEtl, etc
- Rewrite disaster recovery tooling, governance rules, etc for Airflow world
- Provide support for versioned runs, auto rollback, incremental deployments, etc
  - To Do · 8

Migration
- Initiate SDK pipeline conversion discussions with customer teams
- Kick off migration of managed pipelines
- Migrate framework based pipelines like OneEtl to Airflow
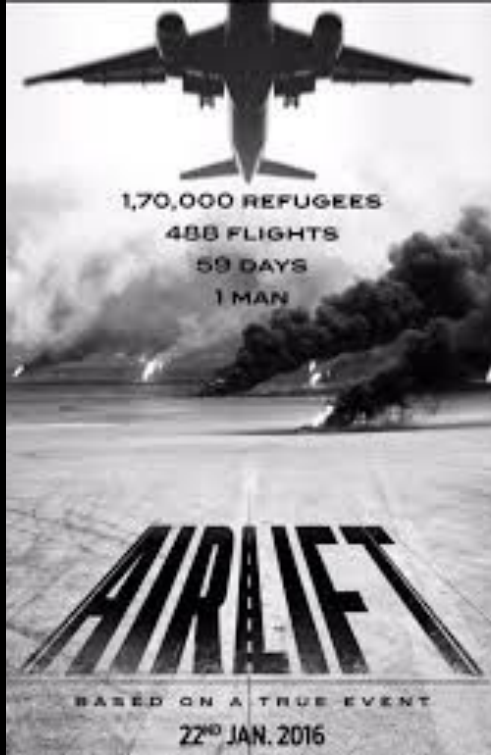- Migrate SDK pipelines to Airflow
  - To Do · 8

PC: bollywoodirect.com

PC: chat.openai.com

Questions?

Sumit Maheshwari