Operation Airlift

Uber's ongoing journey to migrate to Airflow 3

3.0

Sumit Maheshwari
Tech Lead @Uber, PMC Apache Airflow


Uber

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**

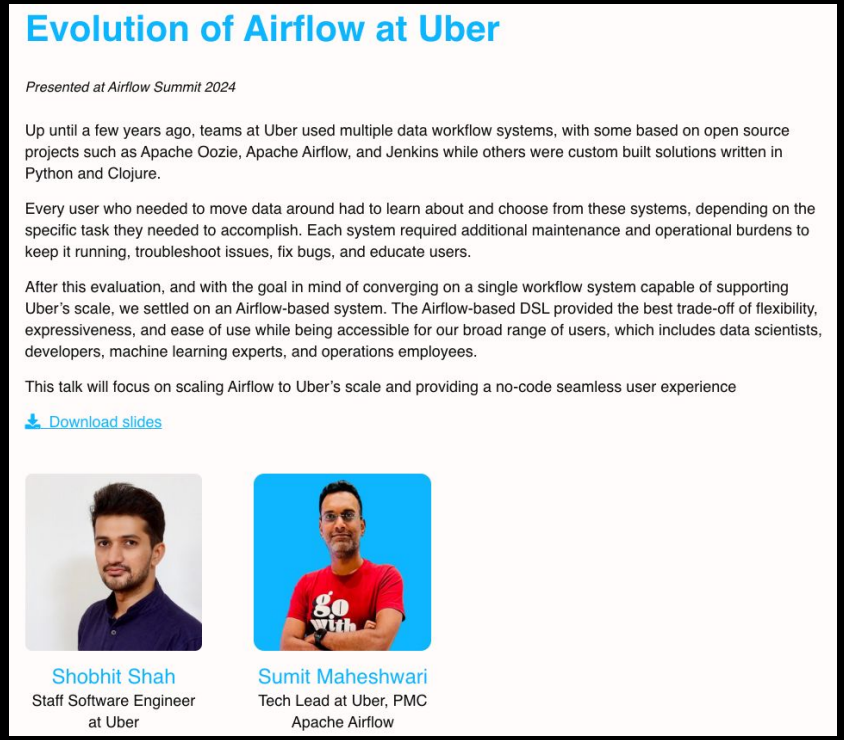
The Story of Piper

A Look Back

Recap

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

https://youtu.be/EhrtShXSrsw



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

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Outdated SDK authoring exp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

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
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

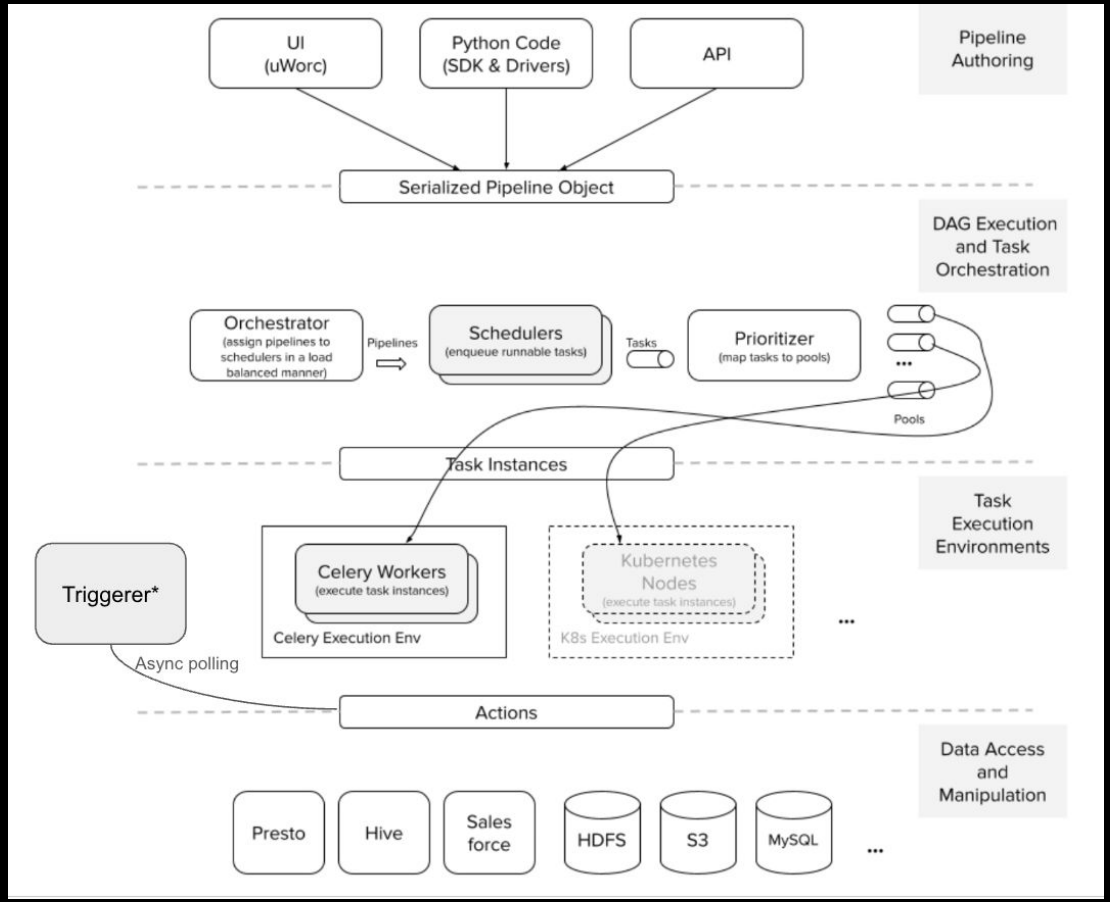
Challenges

- 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

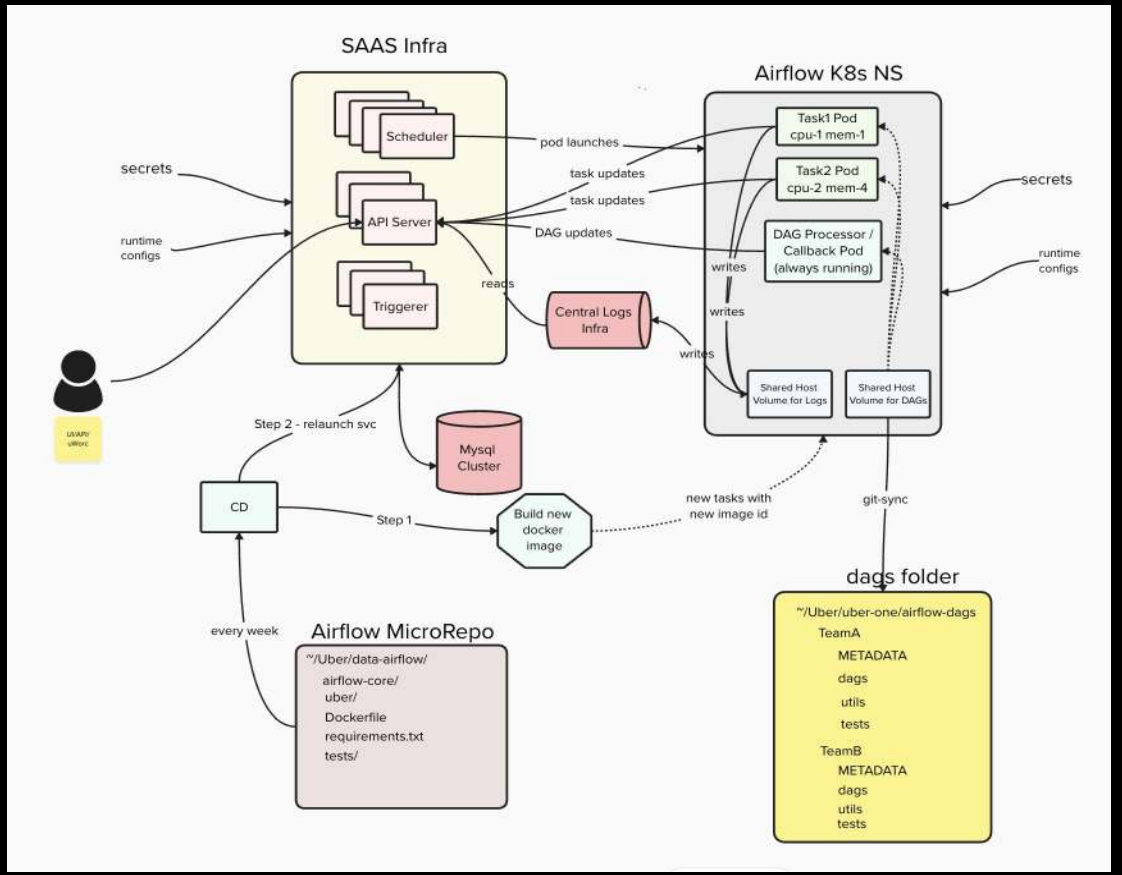
Under The Hood

Implementation Details

Piper HLD

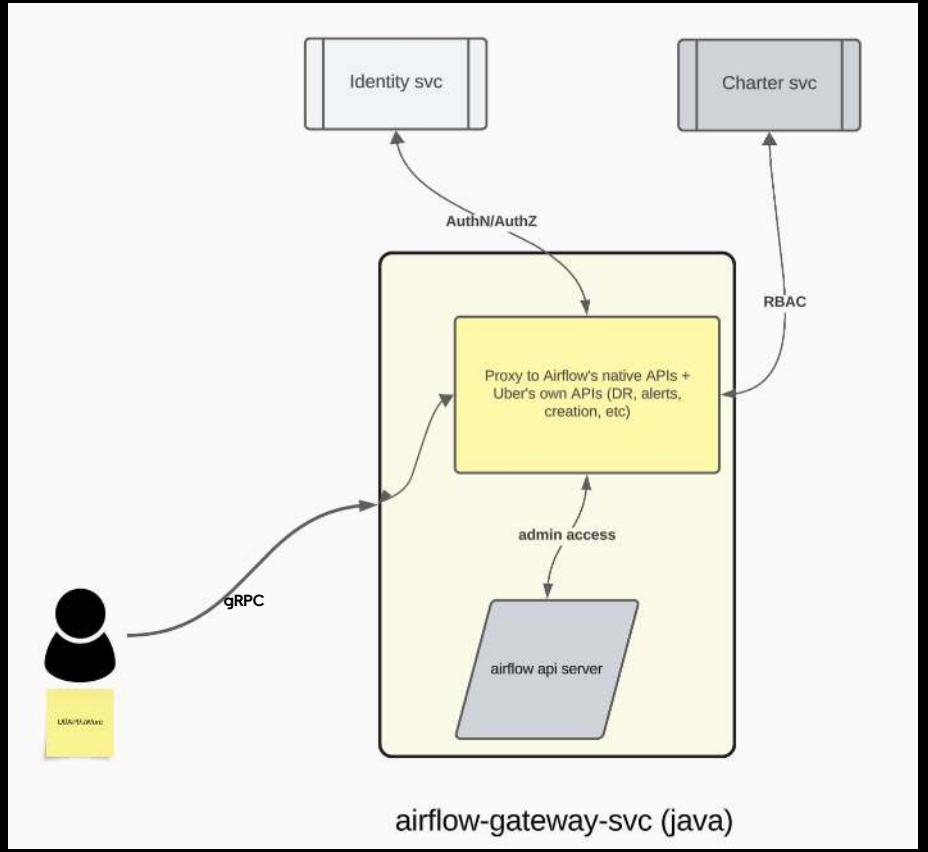

UI
(uWorc)
Python Code
(SDK & Drivers)
API
Pipeline Authoring
Serialized Pipeline Object
DAG Execution and Task Orchestration
Orchestrator
(assign pipelines to schedulers in a load balanced manner)
Pipelines
Schedulers
(enqueue runnable tasks)
Tasks
Prioritizer
(map tasks to pools)
Pools
Task Instances
Task Execution Environments
Triggerer*
Celery Workers
(execute task instances)
Celery Execution Env
Kubernetes Nodes
(execute task instances)
K8s Execution Env
Async polling
Actions
Data Access and Manipulation
Presto
Hive
Sales force
HDFS
S3
MySQL

Airflow HLD

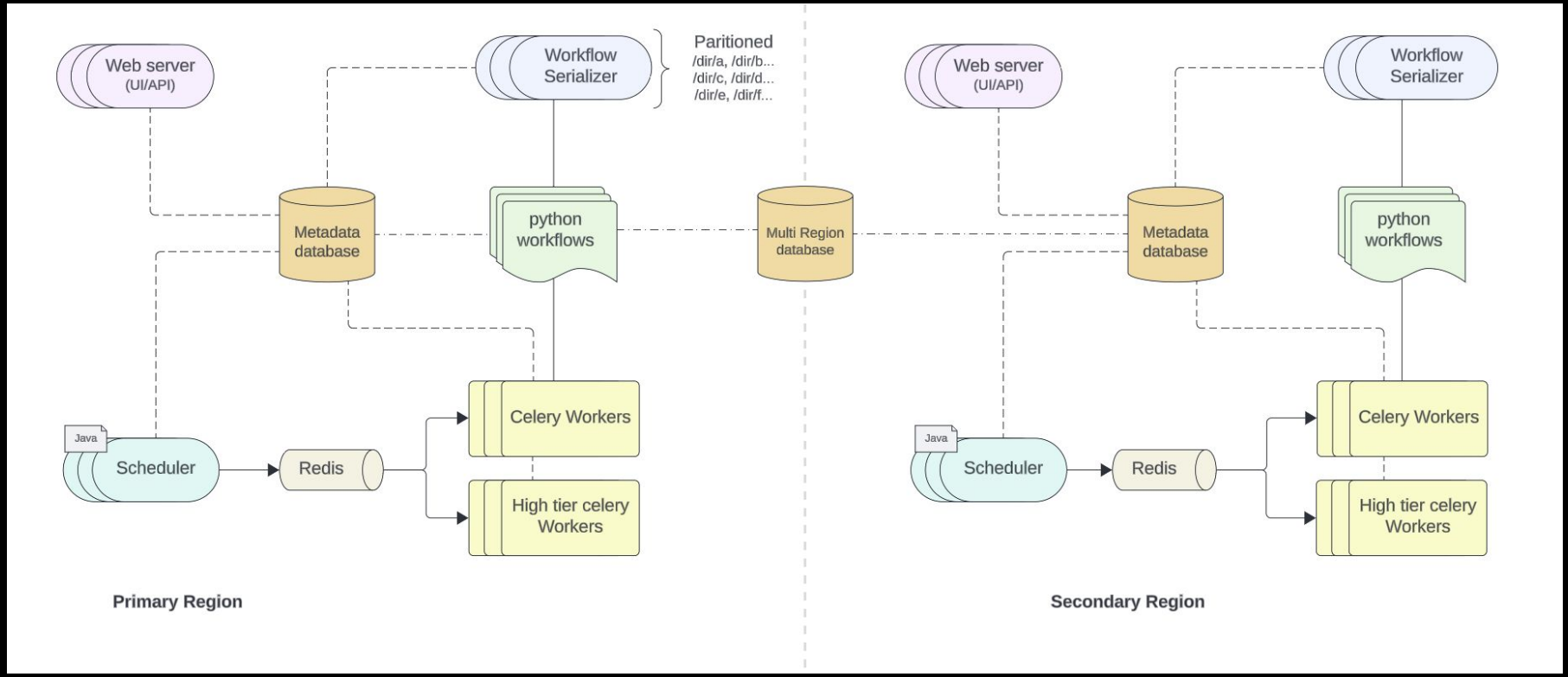

SAAS Infra
Airflow K8s NS
Scheduler
API Server
Triggerer
Central Logs Infra
Mysql Cluster
Task1 Pod cpu-1 mem-1
Task2 Pod cpu-2 mem-4
DAG Processor / Callback Pod (always running)
Shared Host Volume for Logs
Shared Host Volume for DAGs
secrets
runtime configs
pod launches
task updates
task updates
DAG updates
reads
writes
writes
writes
secrets
runtime configs
Step 2 - relaunch svc
CD
Step 1
Build new docker image
new tasks with new image id
git-sync
every week
Airflow MicroRepo
~/Uber/data-airflow/
airflow-core/
uber/
Dockerfile
requirements.txt
tests/
dags folder
~/Uber/uber-one/airflow-dags
TeamA
METADATA
dags
utils
tests
TeamB
METADATA
dags
utils
tests

AuthN & AuthZ

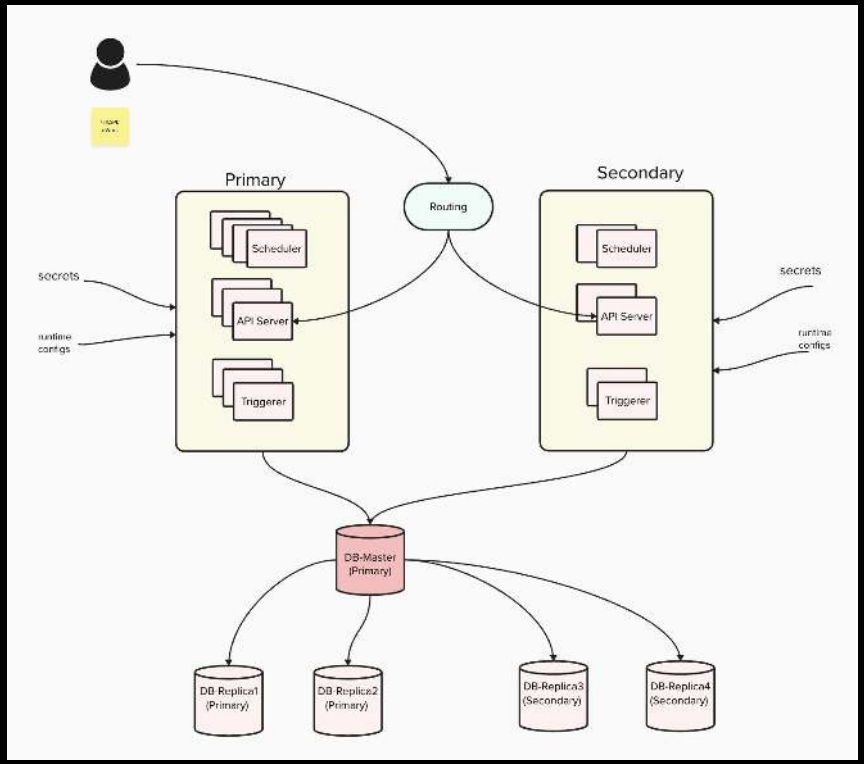
Airflow-web-gateway service written in Java to do all Uber specific authentication & authorization out of Airflow core.

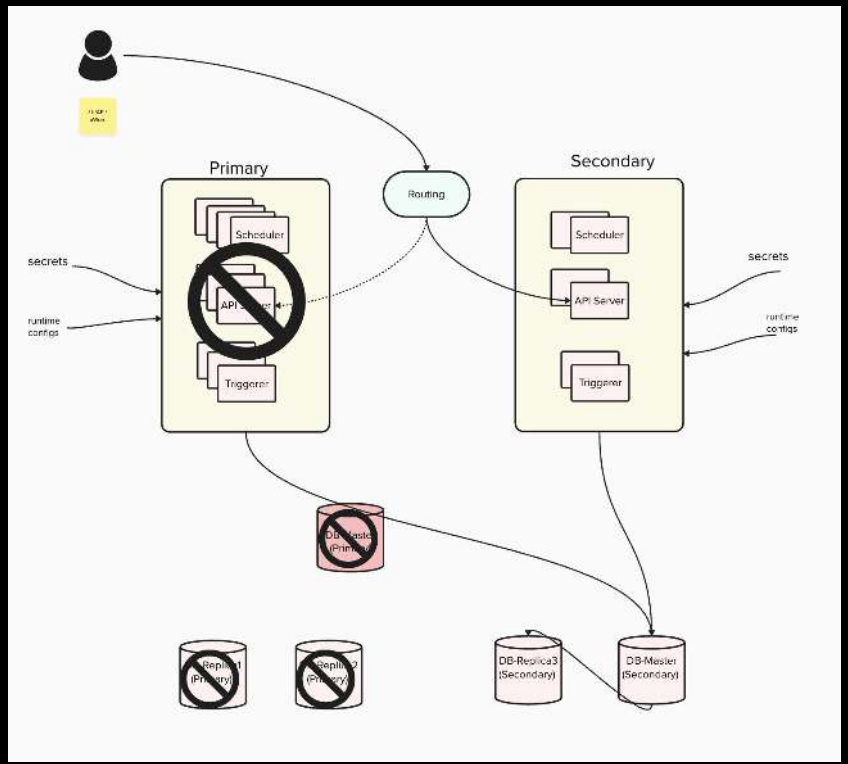


Piper - DR


Web server
(UI/API)
Workflow
Serializer
Paritioned
/dir/a, /dir/b...
/dir/c, /dir/d...
/dir/e, /dir/f...
Metadata
database
python
workflows
Multi Region
database
Java
Scheduler
Redis
Celery Workers
High tier celery
Workers
Primary Region
Web server
(UI/API)
Workflow
Serializer
Metadata
database
python
workflows
Java
Scheduler
Redis
Celery Workers
High tier celery
Workers
Secondary Region

Airflow - DR


Primary
Routing
Secondary
Scheduler
API Server
Triggerer
secrets
runtime configs
DB-Master (Primary)
DB-Replica1 (Primary)
DB-Replica2 (Primary)
DB-Replica3 (Secondary)
DB-Replica4 (Secondary)


Primary
Routing
Secondary
Scheduler
API Server
Triggerer
secrets
runtime configs
DB-Replica3 (Secondary)
DB-Master (Secondary)

Making The Move

Adoption & Migration

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

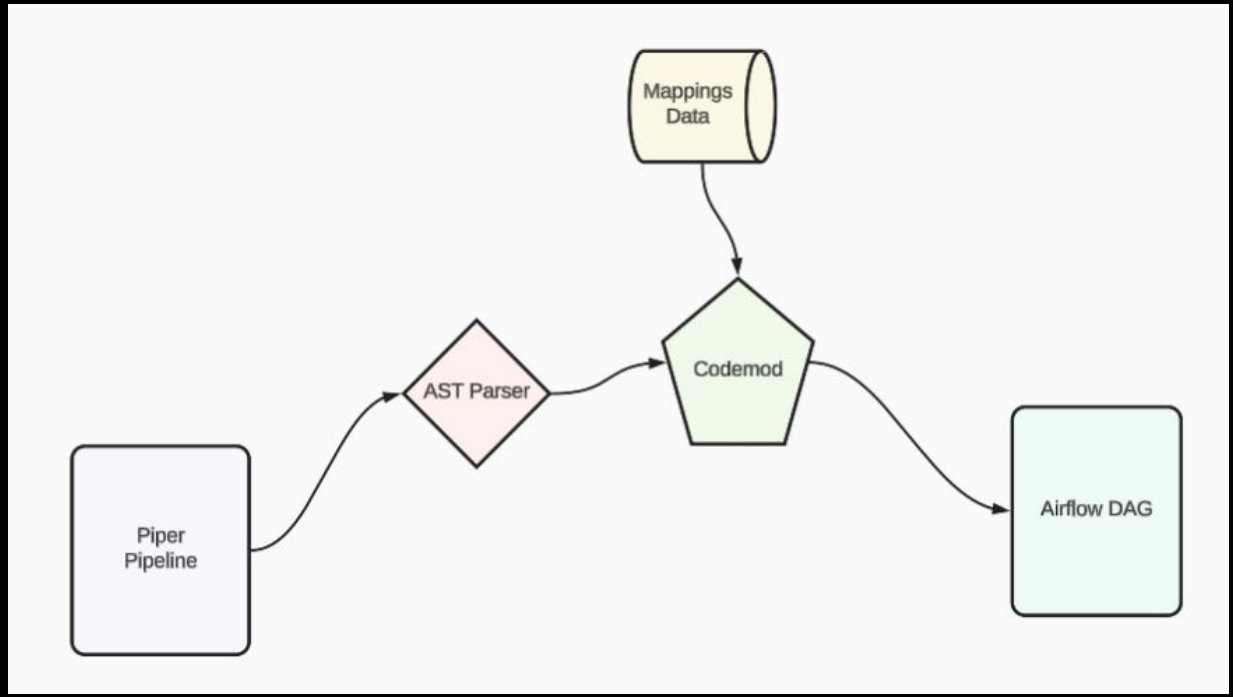
Framework

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/
```



27 removals — 32 lines — Copy

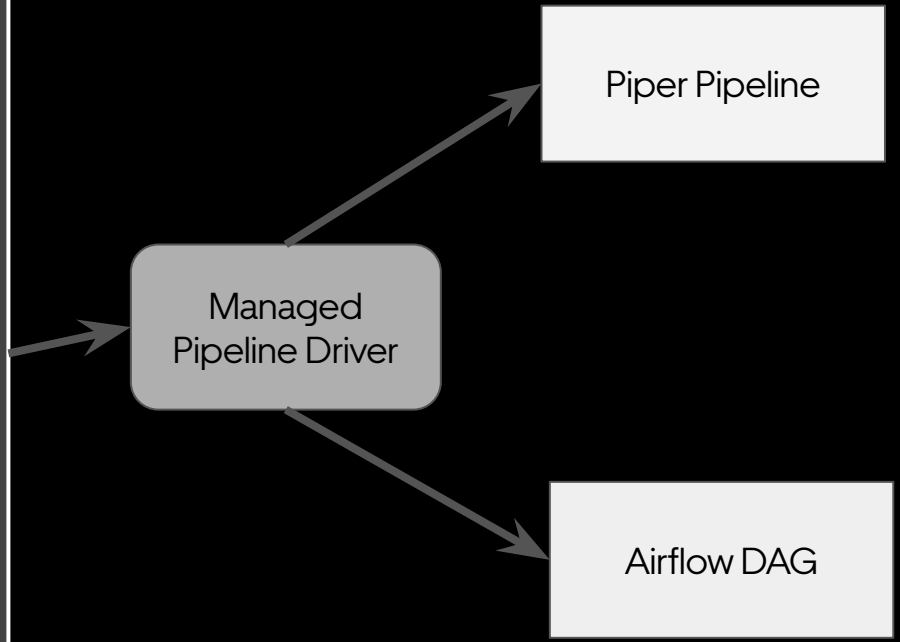
```
# -*- coding: utf-8 -*-
""" Pipeline to watch pipeline failed status and publish stats """
from piper.models import Pipeline
from clay import config
from datetime import datetime, timedelta
from piper.tasks.dummy_task import DummyTask
from pipelines.core.dwm_admin.pipeline_watcher.tasks.watch_pipeline_failed_task import WatchPipelineFailTask
from piper.models.pipeline import RUN_IN_ALL_DATACENTERS
logger = config.get_logger('piper')
POOL_NAME = 'piper_admin'
PID = 'metric_pipeline_fail_watcher'
args = {
'owner': 'data_workflow_management',
'owner_ldap_groups': ['piper', 'data_workflow_management'],
'depends_on_past': False,
'auto_backfill': False,
'pool': POOL_NAME,
'email': ['redacted@abc.com'],
'retries': 3,
}
pipeline = Pipeline(
pipeline_id=PID,
schedule_interval=timedelta(hours=4),
secure=True,
start_date=datetime(2020, 5, 6, 0, 0, 0),
default_args=args,
datacenter_choice_mode=RUN_IN_ALL_DATACENTERS
)
# last task
complete_task = DummyTask(task_id='complete', pipeline=pipeline, default_args=args)
watch_pipeline_failed_task = WatchPipelineFailTask(task_id='watch_pipeline_fail', pipeline=pipeline, default_args=args)
watch_pipeline_failed_task.set_downstream(complete_task)
```

20 additions — 27 lines — Copy

```
""" Pipeline to watch pipeline failed status and publish stats """
from airflow.models import DAG
from airflow.configuration import conf as config
from datetime import datetime, timedelta
from airflow.providers.standard.operators.empty import EmptyOperator
from pipelines.core.dwm_admin.pipeline_watcher.tasks.watch_pipeline_failed_task import (
WatchPipelineFailTask,
)
POOL_NAME = "piper_admin"
PID = "metric_pipeline_fail_watcher"
args = {
"owner": "data_workflow_management",
"depends_on_past": False,
"pool": POOL_NAME,
"email": ["redacted@abc.com"],
"retries": 3,
}
pipeline = DAG(
dag_id=PID,
schedule=timedelta(hours=4),
start_date=datetime(2020, 5, 6, 0, 0, 0),
default_args=args,
)
complete_task = EmptyOperator(task_id="complete", dag=pipeline)
watch_pipeline_failed_task = WatchPipelineFailTask(
task_id="watch_pipeline_fail", dag=pipeline)
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"
}
```

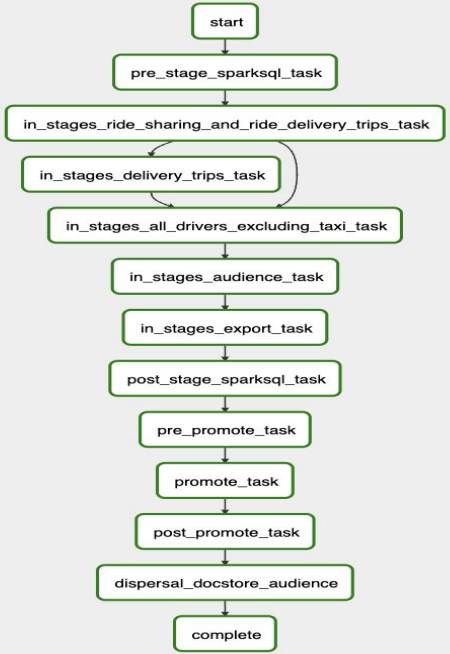
Conversion of Framework Pipelines

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
```

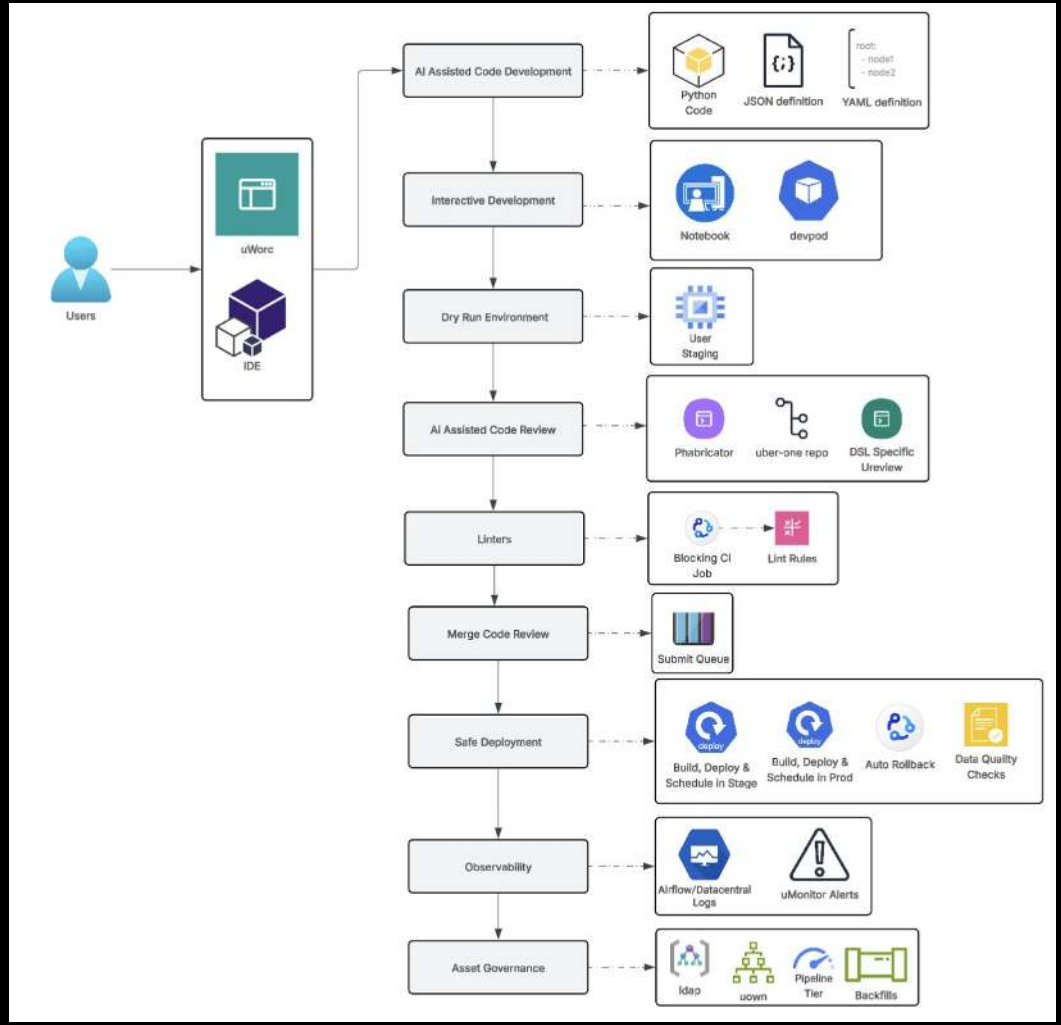
Example pipeline yaml

```
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
```

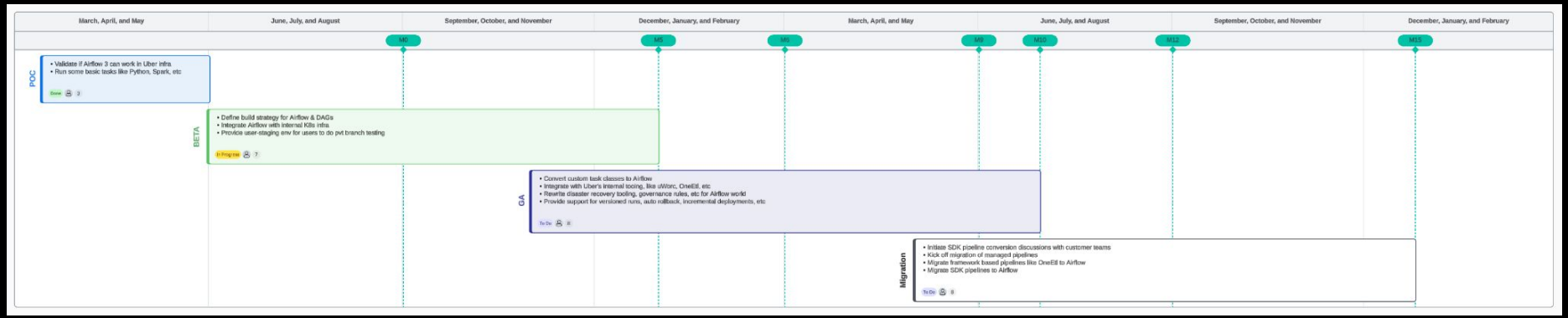


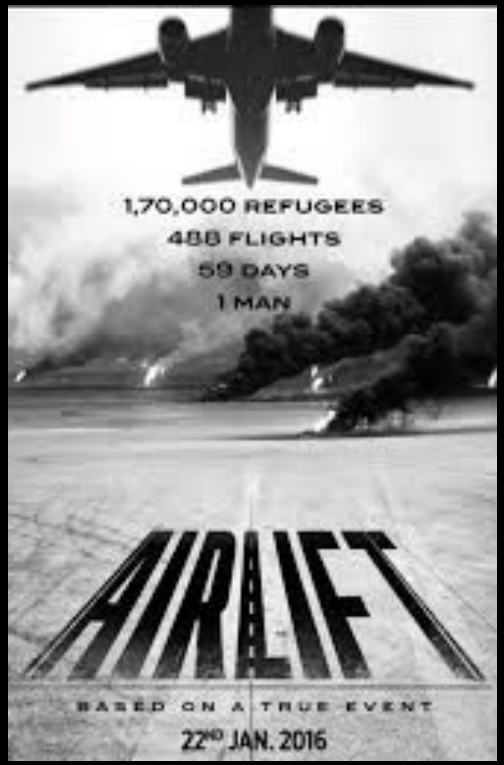
Conclusion & Next Steps

End Goal


AI Assisted Code Development
Python Code
JSON definition
YAML definition
Interactive Development
Notebook
devpod
Dry Run Environment
User Staging
AI Assisted Code Review
Phabricator
uber-one repo
DSL Specific Ureview
Linters
Blocking CI Job
Lint Rules
Merge Code Review
Submit Queue
Safe Deployment
Build, Deploy & Schedule in Stage
Build, Deploy & Schedule in Prod
Auto Rollback
Data Quality Checks
Observability
Airflow/Datacentral Logs
uMonitor Alerts
Asset Governance
Idap
uown
Pipeline Tier
Backfills
Users
uWorc
IDE

Timelines


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
M0
M5
M6
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
7
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