

Architecting Multi-Tenant ML Governance & Privacy in Airflow

Serjesh Sharma

Manager, Software Engineering, AI Platform

John Macmillan

DevOps, Software Engineering, Workforce Scheduling

Workday

Agenda

1 Overview & challenge

2 Multi-tenant ML privacy in SaaS

3 Dynamic Dag architecture

4 Customer Decides - Opt-Ins & tenant configurations

5 Lifecycle compliance - Opt-outs & automated deletion pipelines

6 Scaling & infrastructure

PART

1

of 6

Overview & challenge

One Airflow environment. Every customer owns its data.

Introduction

ENGINE CONTEXT

Data processing engine behind a feature of

Scheduling and Workforce Optimisation Software

BUSINESS PROBLEM

Business problem can be read through the blog:

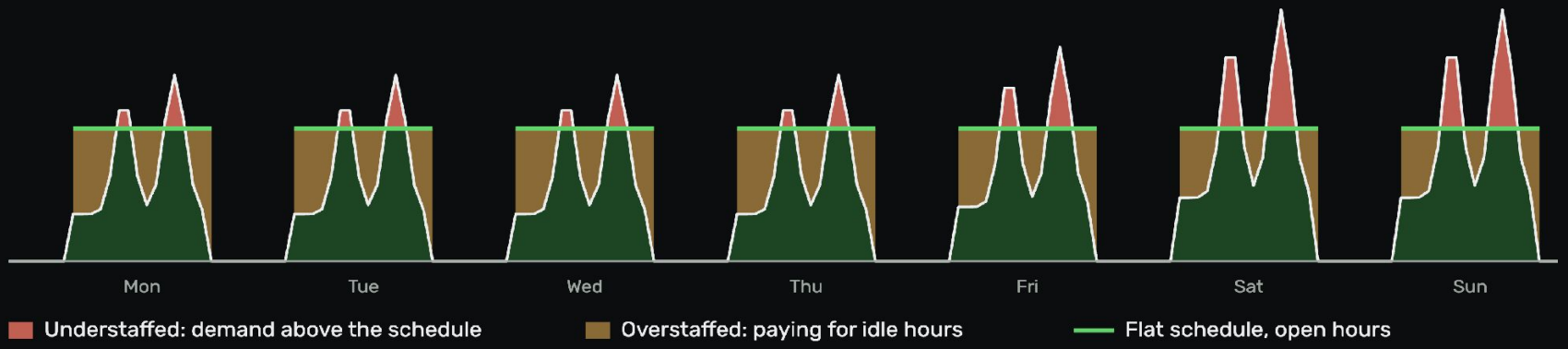
Deep Dive in to ML Powered Labor Forevasting

TECHNICAL SCOPE

Data processing consisting of classical model training and scaling to multiple tenants

How many people do we need, and when?

ILLUSTRATIVE · ONE STORE, ONE WEEK, HOURLY DEMAND



Too many on shift: labor cost goes up for idle hours.

Too few: service suffers and staff burn out.

Every business peaks differently: a grocery store, a hospital and a warehouse each have their own cycle.

Every customer gets its own forecasts



1 customer tenant

= one customer in one environment, e.g. `acme01` in `r1-sandbox`

1 dynamic pipeline

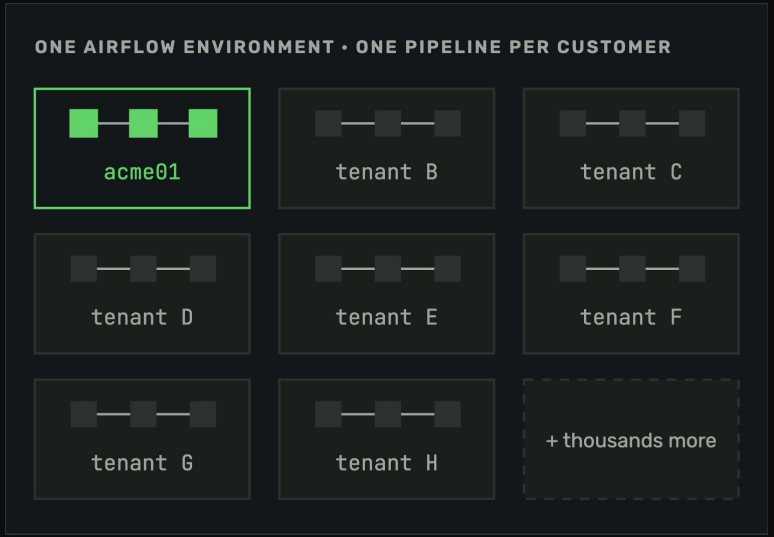
Engineers design the pipelines. Customer's opts in to start using them.

up to 10 DAGs

for each customer, with hardened clones for our engineers to experiment on

One model per customer, at SaaS scale

- **One model per customer.** Every customer's demand is different.
- **Trained only on their own data.** Customers own their data and model artifacts.
- **Customer-controlled.** Opt in or out per data category, at any time. Opting out turns forecasting off and starts deletion.
- **Customer-configured.** Run schedules and model choices vary per customer.
- **Thousands of customers** across regionally shared Airflow environments on Kubernetes.



> **So:** a pipeline per customer, generated dynamically, isolated by design.

Three problems, one Airflow environment

01

Strict data & model privacy

Train each tenant's models only on that tenant's own data, across thousands of tenants.

Hard in Airflow: every tenant in a stack shares the scheduler, namespace, AWS role and node pool.

02

Lifecycle compliance

Create pipelines on opt-in. On opt-out, stop using the data at once and purge it within 30 days.

Hard in Airflow: Dags are code, but tenants join and leave every day with no deploy in between.

03

Operational efficiency

Every tenant adds 10 DAGs. Resource requirements are hard to predict.

Hard in Airflow: every Dag file is re-parsed continuously, so each tenant adds parse work.

> **Airflow has no built-in concept of a tenant.** The rest of this talk is how we added one.

PART

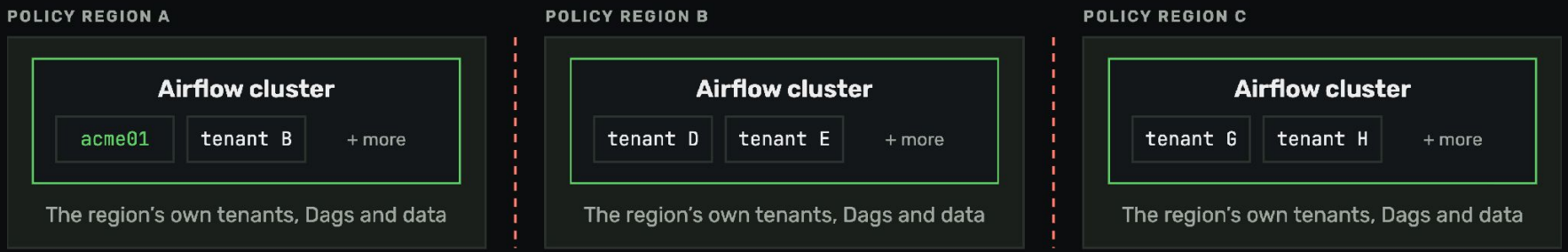
2

of 6

Multi-tenant ML privacy in SaaS

What privacy requires of the pipeline, written down before the code.

One Airflow cluster per policy region



01 · Between regions

Strict separation at the cluster level

Each policy region runs its own Airflow cluster. No resources or components are shared across regions, so regional data policies hold by design.

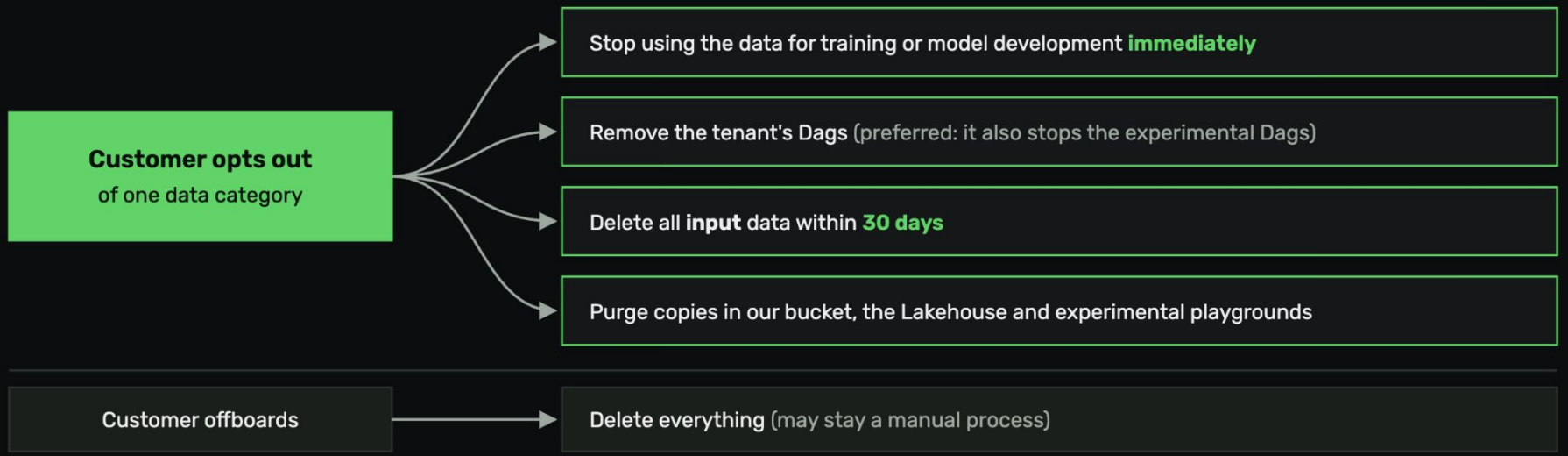
02 · Within a region

Airflow cluster handles its own tenancy

A single cluster serves every tenant in its region. Tenancy is handled inside Airflow: one Dag per tenant, with the tenant bound at build time.

The cluster is the regional boundary. Inside it, the tenant is the boundary, enforced by design

Enforcing data privacy & offboarding



PART

3

of 6

Dag architecture

One definition, generated for every customer tenant.

AIRFLOW SUMMIT | ONLINE RECONNECT

Static vs. Dynamic DAGs

	STATIC DAG, INGEST CONFIG PER TENANTS	DYNAMIC DAG PER TENANT • OUR CHOICE	ONE AIRFLOW PER TENANT
Schedule	One schedule for everyone	Each tenant's own schedule	Each tenant's own schedule
Failures & retries	Shared run: one tenant's failure or backfill affects the rest	Contained to that tenant's Dag	Contained to the deployment
Removing a tenant	Dropping it from registry doesn't reflect in UI	Drop it from the registry; its Dags disappear at the next parse	Tear down a deployment
Visibility	One Dag to read for all tenants	Filter, pause and alert on one tenant	A separate UI per tenant
Latency	N/A	Dag count grows with tenants, so parse time matters	A full Airflow per customer

What led us here

2022

Hands on approach

Parameterized DAGs shared by all tenants, no schedules, difficult to handle concurrent runs

No schedules. No concurrency

Next, a manual DAG triggered from the UI, creates training Dags. Those are triggered by hand too.

2023-2025

Dynamic DAGs, still requires intervention

Hourly generator Dag: one Dag per customer grouping, tenants forked as tasks

One Dag per tenant, tagged customer / env / tenant. Each tenant's schedule read from its config

Generator precomputes parse inputs hourly to cut parse time (part 6)

2026

Driven by opt-in/out-out, Customer defined Schedules

DAG generation is driven by customers opt-in status and existence of data

Opted-out tenants are dropped from the registry, so their Dags are no longer generated, 7-day confirmation window before a tenant is removed

Controller DAG drives schedules and orchestration with no intervention

One Dag per tenant was the turning point. Every step after it builds on that: per-tenant schedules, experimental twins, and removing a tenant's Dags when it opts out.

Static vs. Dynamic DAGs

	STATIC DAG, INGEST CONFIG PER TENANTS	DYNAMIC DAG PER TENANT • OUR CHOICE	ONE AIRFLOW PER TENANT
Schedule	One schedule for everyone	Each tenant's own schedule	Each tenant's own schedule
Failures & retries	Shared run: one tenant's failure or backfill affects the rest	Contained to that tenant's Dag	Contained to the deployment
Removing a tenant	Dropping it from registry doesn't reflect in UI	Drop it from the registry; its Dags disappear at the next parse	Tear down a deployment
Visibility	One Dag to read for all tenants	Filter, pause and alert on one tenant	A separate UI per tenant
Parsing Latency	N/A	Dag count grows with tenants, so parse time matters	A full Airflow per customer

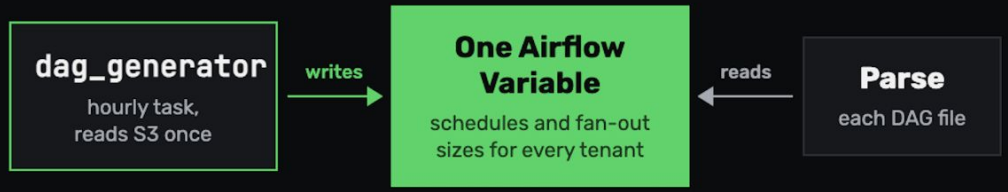
Dynamic DAGs - DAG parsing

BEFORE · UNTIL DEC 2025



- Every parse rebuilt every tenant's config from S3
- An in-memory cache didn't help: each DAG file is parsed in a fresh process

NOW · SINCE DEC 2025



- The S3 work runs once an hour, off the parse path
- Still one Variable read per tenant × mode: cheaper, not flat

PART

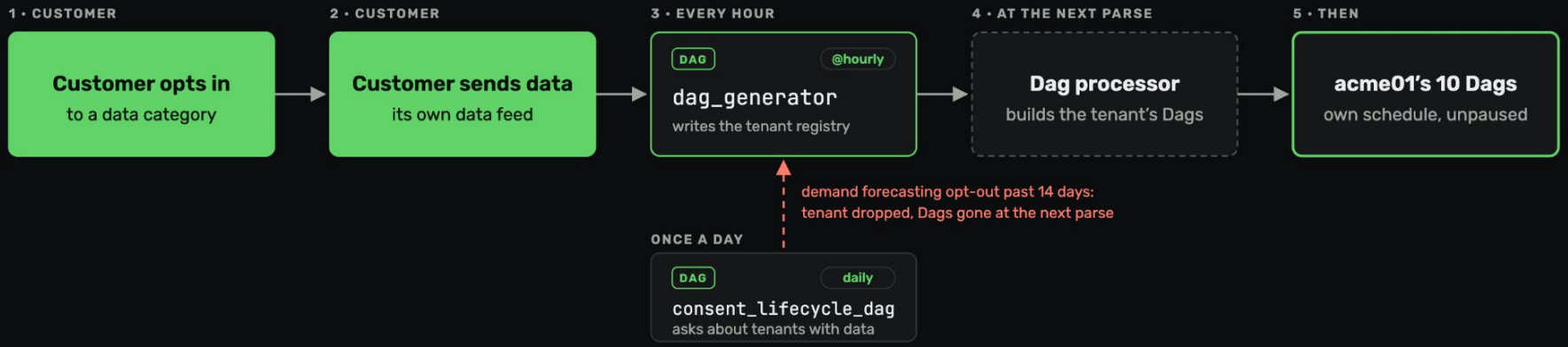
4

of 6

Customer Decides - Opt-Ins & Tenant Configuration

How a tenant appears in Airflow, and how it's carried into every task.

Customer opts-in to start using Airflow



> Opting in alone builds nothing. Dags appear when the tenant's **data** arrives.

Tenant Key drives config

TENANT KEY

family	acme
env	sandbox
tenant	acme01

From the tenant registry. Bound when the DAG file is parsed, never passed at run time.

Creates the tenant's DAGs

dag_id	`pre_processing_acme01_sandbox` + `exp_` twin
tags	`customer=acme` · `env=sandbox` · `tenant=acme01`
schedule, fan-out	that tenant's entry in the parse-time snapshot
every task	gets the tenant in its kwargs; pod config and callbacks built from it

Partitions the tenant's data

data paths	`s3://<bucket>/[exp/]<dataset>/acme/sandbox/acme01/…`
execution spec	`…/acme/sandbox/acme01/<dag_id>/<run_id>/`
deletion guard	refuses any path without the tenant *and* env segments

SHARED BY EVERY TENANT Kubernetes namespace · One AWS role + IAM trust · Node pool — Separation is in the DAGs and paths, not the infra

PART

5

of 6

Lifecycle compliance - Opt-Outs & automated deletions

What happens when a customer opts in? What happens when they opt out?

AIRFLOW SUMMIT | ONLINE RECONNECT

Lifecycle Compliance DAG

1 • Detect & alert

- Record new opt-outs; after 14 days, alert once
- List tenant / env / category in our bucket; ask the consent API about each
- Drop records for tenants who opted back in

→

2 • Delete from Airflow bucket

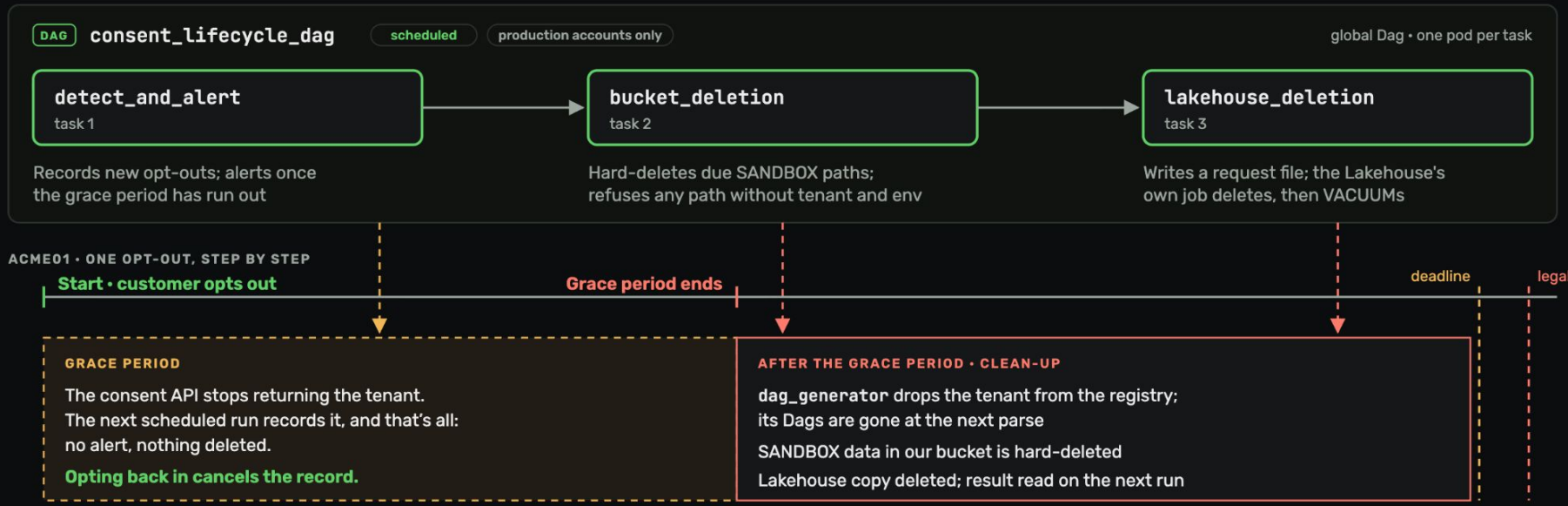
- Determine action based on deadline
- Either alert engineers for immediate action, or automated cleanup, depending on environment type

→

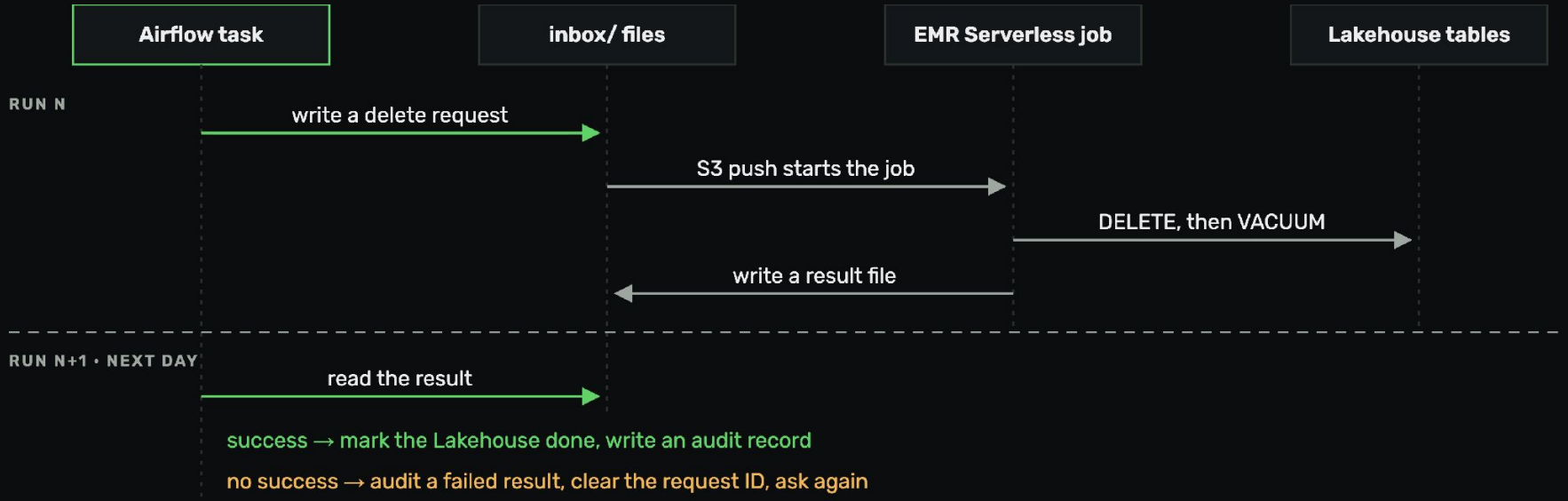
3 • Delete from the Lakehouse

- Read results of earlier requests
- Write new requests for due records
- The Lakehouse's own job deletes

Lifecycle Compliance DAG (detail)



Deletion is driven tenant key



Why files? Airflow couldn't trigger or list the Lakehouse's EMR jobs, and could only read and write under `inbox/`.

Why VACUUM? A Delta DELETE is logical; tombstoned files stay until vacuumed, so the job vacuums with zero retention.

Designed vs. shipped: the design polled inside one run for up to an hour. What shipped reads the result on the next daily run.

PART

6

of 6

Scaling & infrastructure

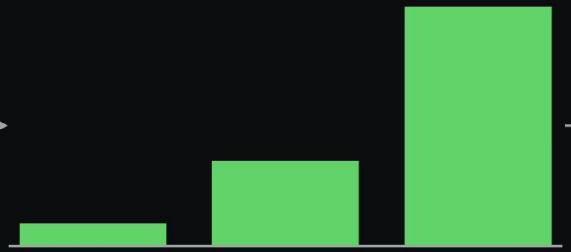
The cost of one Dag per tenant, and how tasks get compute.

One pod per task, protected from eviction

Scheduler
KubernetesExecutor
parallelism 200 (US prod)

One pod
per task, config
built from the tenant

POD TIERS · BAR HEIGHT = CPU REQUEST, TO SCALE



Small
4 / 8 CPU
10 GiB

Medium
16 / 16 CPU
90 GiB

Large
45 / 46 CPU
250 GiB

350 GiB for one pod group

Dedicated node pool

tainted; task pods require its label and tolerate its taint

migration and sync jobs run on static nodes instead

Dedicated node pool prevents noisy neighbor issues. Pods carry `do-not-disrupt: "true"` (Karpenter) and `safe-to-evict: "false"` to block eviction.

Linear backoff retries when node limit is reached (**3 retries**, 5 mins apart) give Karpenter time to scale nodes and pods.

Data variability makes pod size variable for the same task; one Airflow stack per prod region deployed by Argo CD.

Autoscaling, attempts and outcomes

VPA & ScaleOps

Evaluated VPA and commercial tools (e.g. ScaleOps) to learn workload patterns and right-size pod resource requests based on tenant-specific usage.

Open risk

If a right-sized pod encounters OOM, retries keep receiving the same trimmed request. Pattern-learning algorithms risk under-provisioning during peak tenant runs.

What breaks?

Tasks sharing a pod name across tenants that require distinct resource profiles

Tasks with spiky resource usage that pattern learning sizes too small

FROM THE ABSTRACT

Takeaways

01

Dynamic per-tenant DAG generation

- Registry in S3, outside the metadata DB
- Dags follow data, via an hourly generator
- Parse-time values computed off the parse path

02

Tenant isolation, security & governance

- Bind the tenant when the Dag is built
- Fail closed on every destructive path
- Write down what's still shared

03

Opt-in / opt-out compliance workflows

- Policy by environment, scope by category
- Each store deleted by its owner
- Flags as retries, an audit record per attempt



Questions?

Bind the tenant when the Dag is built.

Record now, confirm later.

Write down what's still shared.

Serjesh Sharma • John Macmillan
Workday