AIRFLOW SUMMIT

Orchestrating and Testing RAG pipelines with Airflow

Shrividya Hegde

THE PROBLEM

RAG pipelines fail silently

Chunk counts match, the DAG turns green. Your AI product quietly gets worse.

Bad retrievals

The right chunk exists but it just never gets surfaced to the model.

Hallucinated answers

The model answers confidently from context it was never given.

Stale vectors

New documents land, but old embeddings keep winning the similarity search.

Gradual quality decay

No single run breaks. Answer quality erodes over weeks, not minutes.

TODAY'S REFERENCE SYSTEM

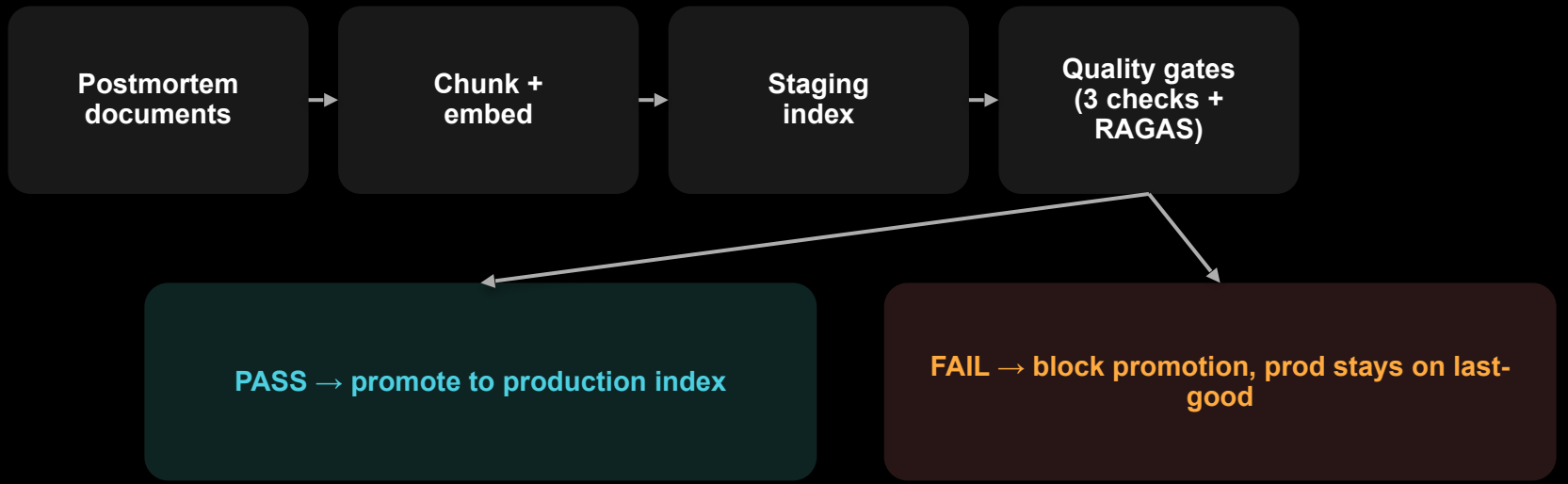
The corpus: our own incident postmortems

Every SEV-1 and SEV-2 writes a postmortem. Nobody re-reads them until the next similar incident , usually mid-outage, under pressure, searching a wiki that doesn't rank well.

Why this corpus

- Ask “what caused an outage like this before?” and get grounded answers, cited to the incident.
- New postmortems should show up in the index within a run but never at the cost of a wrong answer.
- This is the exact shape of most internal RAG systems: a document set that changes, and answers people will act on.

Staging first. Gates decide. Prod is earned.



Production is only ever written by one task, and only after every gate passes.

EVALUATING BEFORE PROMOTION

Inline quality gates, scored with RAGAS

The golden question set runs against the staging retriever before anything is promoted. Three metrics, three floors:

Faithfulness

Floor 0.85

Is every claim in the answer actually supported by the retrieved postmortem text?

Context precision

Floor 0.75

Are the relevant chunks ranked near the top of what was retrieved?

Context recall

Floor 0.75

Did retrieval surface everything needed to answer, or did it miss key context?

Floors are a starting point, not a universal constant — baseline against your own known-good index first.

TaskFlow API: business logic without boilerplate

@task for every step

A plain Python function becomes a task — no PythonOperator, no manual XComs. Return values pass straight into the next task's arguments.

@task.branch for gate routing

evaluate_quality_gates is one function that returns a task id. No BranchPythonOperator, no manual XCom push to decide the path.

@task.llm for generation & guardrails

apache-airflow-providers-common-ai turns an LLM call into a native task: system prompt in, structured output out, retried and logged like any other.

Dynamic task mapping

generate_eval_answer and check_groundedness run once per golden question via .expand() — the question set can grow without touching the DAG.

From tasks to data products: Assets

Airflow 3's @asset decorator and Asset outlets let this DAG declare what it produces, not just what it runs — so a chatbot's cache-warm DAG can schedule off postmortem_index_prod directly, instead of polling this one.

Two assets, one DAG

- **postmortem_index_staging** — written by every run, visible to nothing user-facing
- **postmortem_index_prod** — an Asset event only fires here after every gate passes
- **Downstream consumers** schedule on the asset, decoupled from this DAG's schedule

DAG versioning: reproducible when it matters most

Airflow 2 always reran a DAG with today's code, even if you were rerunning a run from six weeks ago. Airflow 3 tracks a version per structural change, automatically, and lets a rerun replay the exact code that produced it.

Why it matters here

When a quality gate rejects a run, you need to know exactly what chunking, prompt, and threshold code produced that verdict even six weeks later, on-call, under pressure.

`rerun_with_latest_version`

Set false for this DAG. Clearing or rerunning a past ingestion replays its original bundle version, not whatever is on main today.

What still runs latest

Backfills default to the latest version unless overridden — appropriate when you deliberately want new logic applied to historical data.

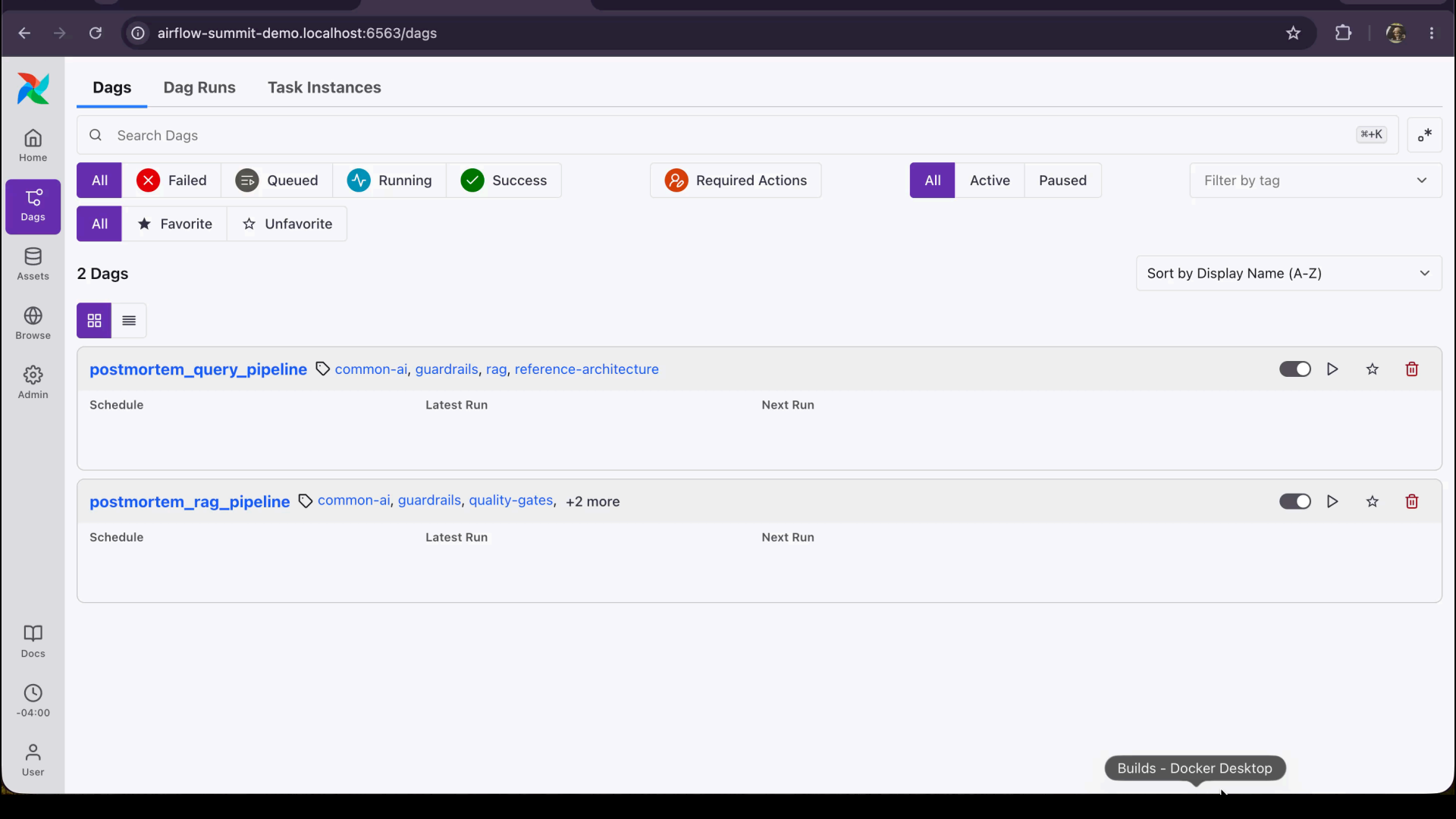
LIVE DEMO

Watching a bad run get blocked

postmortem_rag_pipeline — four postmortems, one golden question set, one deliberately broken run

1. **`build_staging_index`**
   chunk + embed all four postmortems
2. **`publish_staging_asset`**
   emits the staging Asset event
3. **`generate_eval_answer + check_groundedness`**
   @task.llm, mapped one task per golden question
4. **`evaluate_quality_gates`**
   checks all 4 failure modes + RAGAS, branches to promote or block

The graph also shows PII and groundedness guardrails with human-review branches ,a natural extension beyond today's four gates.


airflow-summit-demo.localhost:6563/dags
Dags
Dag Runs
Task Instances
Search Dags
⌘+K
All
Failed
Queued
Running
Success
Required Actions
All
Active
Paused
Filter by tag
All
Favorite
Unfavorite
2 Dags
Sort by Display Name (A-Z)
postmortem_query_pipeline
common-ai, guardrails, rag, reference-architecture
Schedule
Latest Run
Next Run
postmortem_rag_pipeline
common-ai, guardrails, quality-gates, +2 more
Schedule
Latest Run
Next Run
Home
Dags
Assets
Browse
Admin
Docs
-04:00
User
Builds - Docker Desktop

TRY IT YOURSELF

The reference DAG, ingestion code, and sample postmortems are open

Repo link:

https://github.com/Shrividya/Airflow-Summit-Demo

Sample corpus

4 synthetic postmortems + a golden question set for RAGAS

Stack

Apache Airflow 3 · Chroma · OpenAI embeddings · RAGAS

Thank you

Questions?

Would love to hear some feedback

Orchestrating and Testing RAG Pipelines with Airflow

Shrividya Hegde's website QR

https://shrividya.github.io