AIRFLOW SUMMIT | ONLINE RECONNECT

# Scale Without Scaling the Team:

## Metadata-Driven Airflow the Business Can Own

Ramzi Alashabi
Chapter Lead Data and Streaming @ ABNAMRO BANK

ABN·AMRO



ABN·AMRO




# Ramzi Alashabi

- 13+ Years Data Specialist
- Micro-services, ETLs, and Cloud Engineering
- Transform ideas to Production
- Focus on Simplification, Reusability, Scalability and Innovation.
- Yes, I'm a Dog Person

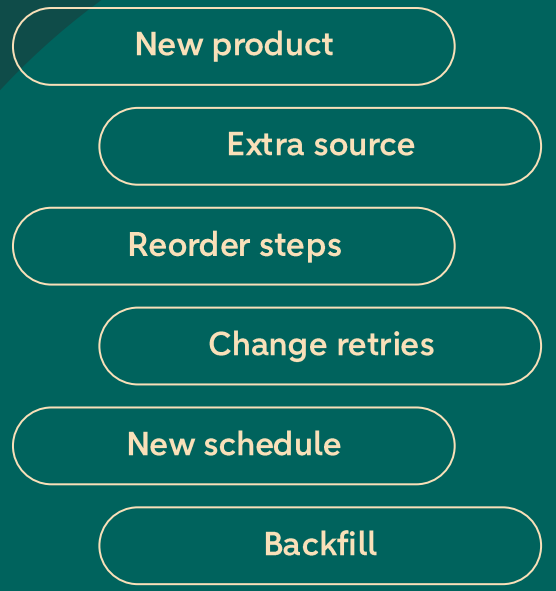


https://www.linkedin.com/in/ramzialashabi/

THE PROBLEM

# Every data team hits the same wall.

A handful of engineers write and maintain every Airflow DAG, while the business waits in the queue.




**3** DAG AUTHORS for every request

**Delivery scales with headcount, and the people who understand the data are not the people who can ship it.**

OUR APPROACH

# Don't write DAGs. Compile them.

01

## Describe

The business describes the data product in a declarative manifest: mappings, trigger, defaults. Plain JSON, validated against a versioned schema.

02

## Compile

A generator turns that manifest into an asset-aware, partitioned Airflow 3 DAG. Deterministic, traceable, never hand-edited.

03

## Compose

A visual canvas lets analysts draw the execution order on top of the generated graph, and it writes back to the same manifest.

THE TOOLING

# Describe in DIET. Compile with BRE.

**Why it exists:** the people who know the data author the pipeline; engineers own one compiler, not every DAG.

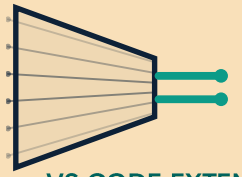
1 METADATA IN — 2 DESCRIBE — Generate — 3 COMPILE — 4 RUN →

ALREADY IN PLACE

**Unity Catalog**
source tables and views

**Enterprise Architect**
the target data model

**The analyst**
business rules and intent



DIET

DATA INTEGRATION ENABLEMENT TOOL

VS CODE EXTENSION

- Map sources, joins and rules to targets
- Group mappings into a product; draw the order on a canvas
- Validated as you type, versioned in Git

BRE

BUSINESS RULE ENGINE

- A pinned Python package, run on Generate
- Writes real code: PySpark per mapping, an Airflow 3.3 DAG per product
- Checks cycles and references; stamps every file with its versions

ALREADY IN PLACE

**Azure DevOps**
deploys the generated files

**Airflow 3.3 on AKS**
schedules every product DAG

**Databricks**
runs each mapping as a job

**DIET and BRE never run Spark. They generate the code; Airflow schedules it and Databricks runs it.**

# 01

FROM MANIFEST TO DAG

## One manifest in, one DAG out.

Same input, same DAG. Every file records BRE, contract and DIET versions plus the input SHA-256.

CREDITFACILITY.PRODUCT.JSON

```
"scheduling": {
  "trigger": "time",
  "schedule": "@daily",
  "snapshot_offset": -1
},
"mappings": [
  "kem_creditfacility.json", ...
],
"execution_dependencies": [{
  "upstream": "kem_creditfacility.json",
  "downstream": "kem_creditfacilityMDDR.json"
}, ...]
```

BRE

DAG_CREDITFACILITY.PY · GENERATED

```
schedule=sdk.CronPartitionTimetable(
    "@daily", timezone="UTC",
    run_offset=-1, key_format="%Y-%m-%d"),

def product_dag():
    business_date = snapshot_date()
    pipelines = {m: mapping_pipeline
        .override(group_id=m)(m, business_date)
        for m in MAPPINGS}
    for up, down in EXECUTION_DEPENDENCIES:
        pipelines[up] >> pipelines[down]
```

A real product from our platform: nine mappings, one manifest, one generated DAG.

WHERE TO DRAW THE LINE

# Derive what you can. Default the rest. Ask people only for intent.

THE MACHINE

## Derived by the compiler

- One task group per mapping
- Databricks job parameters: product, mapping, environment, snapshot date
- Timetable from the trigger type
- Partition key validated as the snapshot date
- Task IDs, imports, provenance header

CONFIGURABLE

## Defaulted, with an override

- Retries, retry delay, timeouts
- Pool, priority weight, trigger rule
- Max active runs and tasks
- Catchup, paused on creation
- Any operator argument via extra_kwargs

A PERSON

## Decided by a person

- Trigger: time, assets or manual
- Execution order between mappings
- Which mappings are enabled
- Per-mapping overrides
- Which assets the product publishes

**Rule of thumb:** if two reasonable people would pick the same value, derive it. If they would argue, make it a field.

# 02

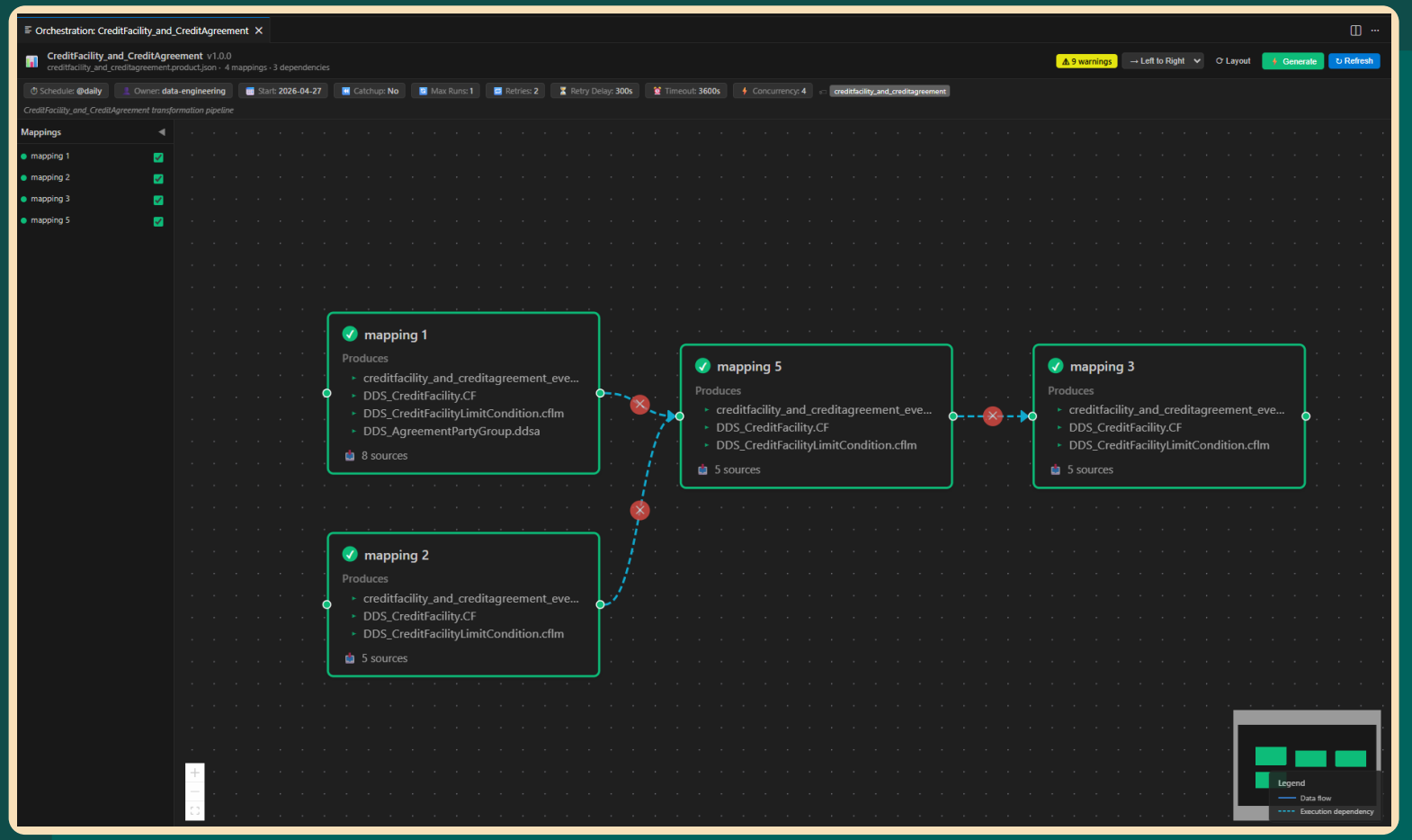
THE ORCHESTRATION CANVAS

## Drag a line, get a dependency.

**1 Open the product**
Every mapping in the product folder is a node; data lineage between them is drawn for you.

**2 Draw the order**
Drag from one mapping to the next. Remove a line, or disable a mapping, just as easily.

**3 Generate**
The canvas saves the manifest and runs the compiler. Errors, such as a cycle, come back in place.

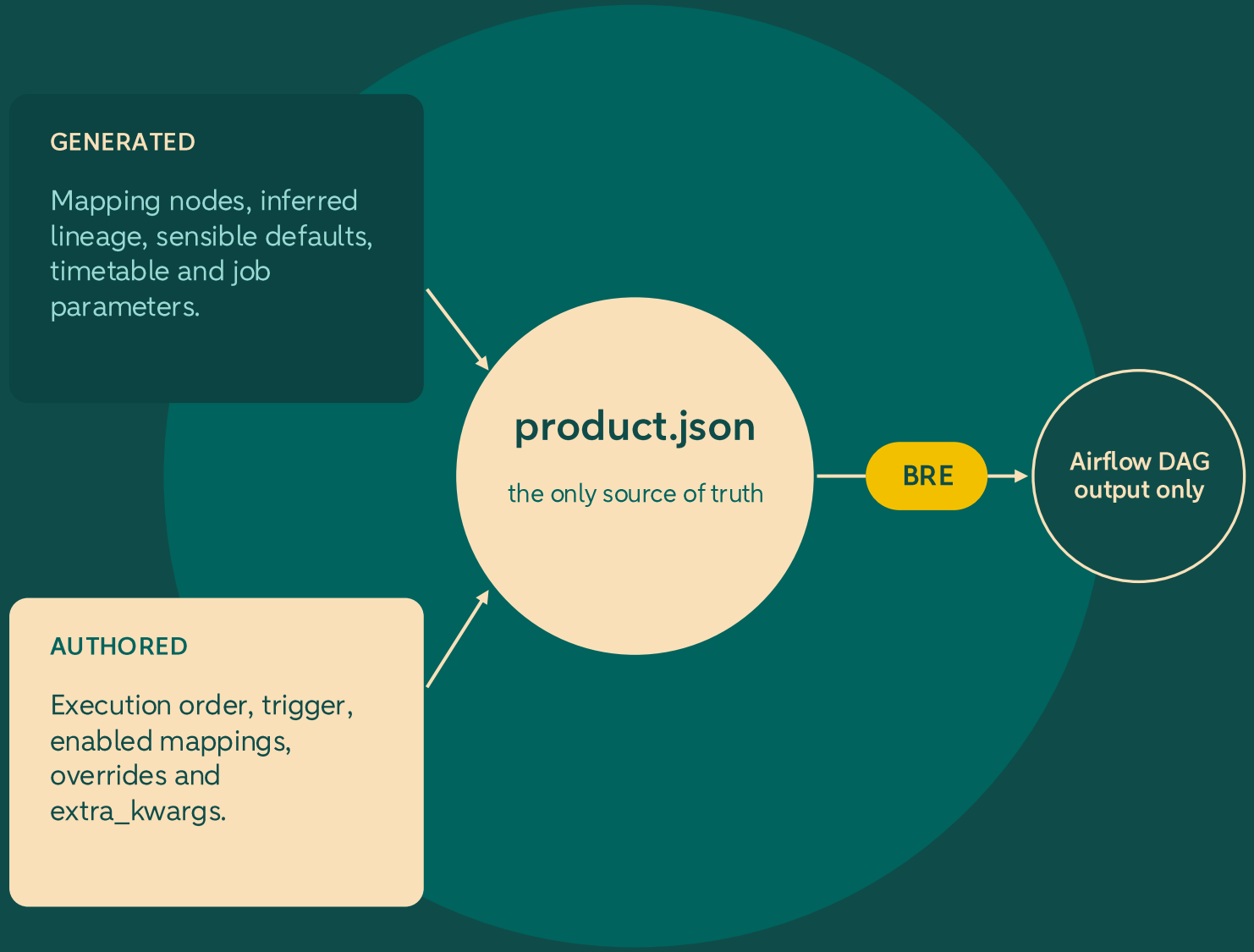


# 03

ONE SOURCE OF TRUTH

## The machine proposes. People decide. One file records.

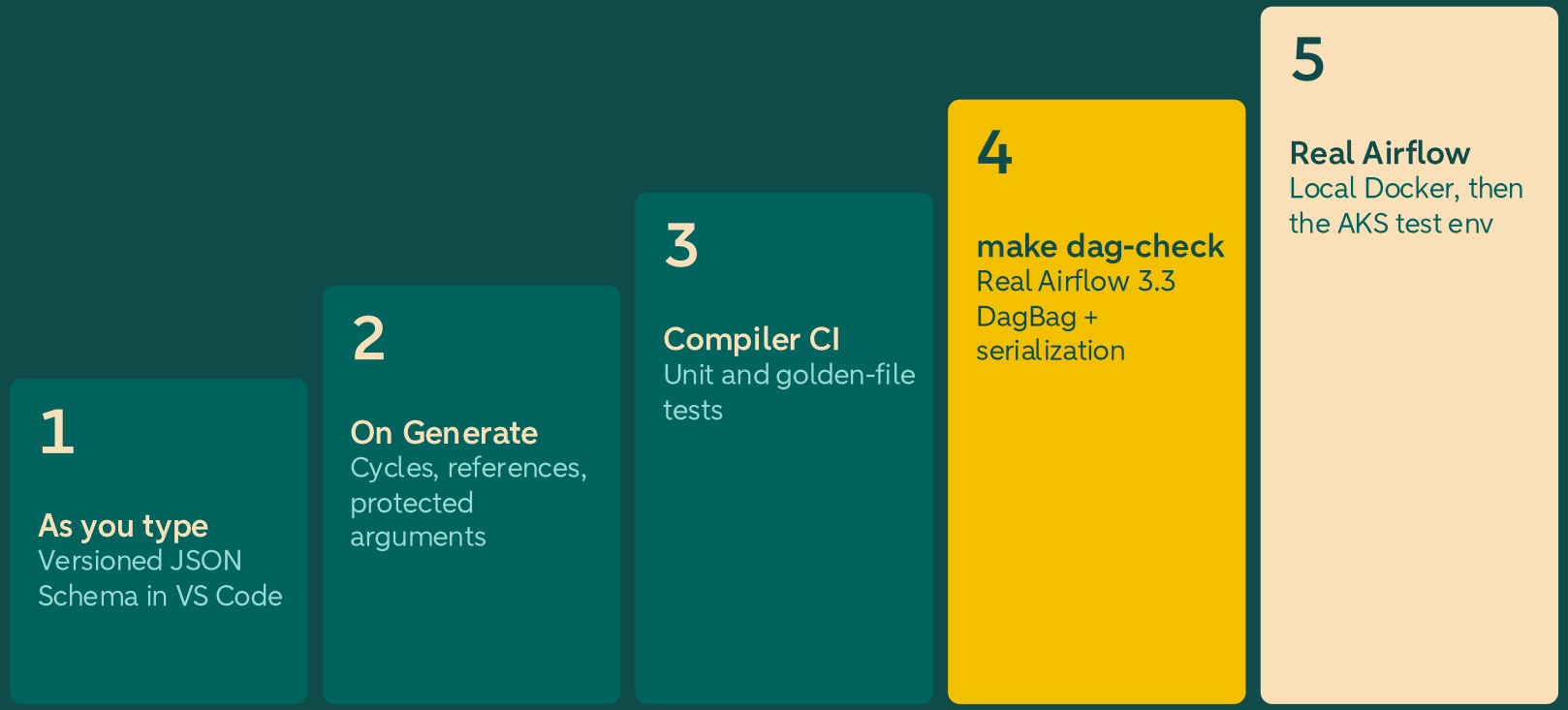
Generated and authored metadata share one file, one schema and one review flow, so they cannot drift apart.



GUARDRAILS & TESTING

# Business-owned, not unguarded

Each rung is cheaper than the next: nothing reaches the scheduler that Airflow has not already parsed.




## Cycle detection

A circular execution order is refused, and the error names every mapping in the loop.

## References & arguments

Edges and overrides must name real mappings; extra_kwargs cannot overwrite what the compiler owns.

## Pinned & traceable

DIET pins the exact BRE release; every DAG records the versions and an input hash.

LIVE DEMO

# From a dragged line to a running DAG.

1

## NYC_Taxi Demo

The canvas shows three mappings and their lineage.

2

## Draw one edge

Make one mapping wait for another.

3

## Show the diff

One new entry in product.json, nothing else.

4

## Generate

Regenerated DAG, then the new edge in the Airflow graph view.

# 04

DELIVERY

## Who is on the critical path changes.

[ ~5000 ] mappings in production

[ 14 ] days → [ ~2 ] hours lead time

[ 6 ] engineers own the compiler

| CHANGE | HAND-WRITTEN DAGS | METADATA-DRIVEN |
|---|---|---|
| **New data product** | Ticket, engineer writes Python, review, deploy | **Analyst authors a manifest, PR, generated** |
| **Change execution order** | Edit Python, re-review, redeploy | **Drag a line, PR, regenerate** |
| **Upgrade a pattern everywhere** | Touch every DAG file | **Change one template, regenerate all** |
| **Adopt a new operator feature** | Edit DAGs one by one | **One extra_kwargs line** |
| **Critical path** | The DAG engineers | **The people who know the data** |

HONEST BOUNDARIES

# Engineers own the platform, not every pipeline.

WHAT ENGINEERS OWN

- The contract and its versions
- The compiler and its templates
- The Databricks runner job
- Guardrails and CI, including dag-check
- The Airflow platform on Kubernetes

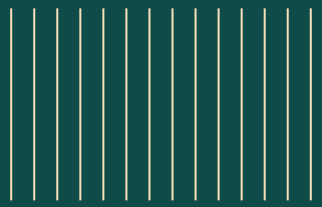
WHERE IT STOPS

# Deliberate limits, not missing features.

- The canvas orders mappings inside one product; products chain through assets
- One execution backend today: Databricks run-now
- No arbitrary Python in a manifest, by design
- Escape hatches are explicit: extra_kwargs, never hand edits
- New capabilities still start as an engineering change

TAKE IT HOME

# A blueprint for metadata-driven Airflow.



**01** **Version a contract**
A JSON Schema for the manifest, validated in the editor and in the compiler.

**02** **Derive everything deterministic**
Tasks, parameters, timetables and partitions are computed, never typed.

**03** **Separate the two dependencies**
Assets between products, task edges inside a product.

**04** **Give people a canvas over the same file**
Every gesture is metadata, reviewed like code.

**05** **Guard every layer**
Schema, references, cycles, protected arguments, and a real DagBag in CI.



# Don't scale the team that writes DAGs.

## Scale the people who can ship them.

Compile, don't hand-write

One source of truth

Guardrails at every layer

# Q&A

**Thank you for your attention and participation**

