NOV 4

Airflow across four regions and a fleet of agents

How Notion runs its data platform, and keeps every pipeline where its customers' data lives.



Yunhao Qing · Senior SWE, Notion Data Platform

ABOUT ME

Hi, I’m Yunhao

What I do

Senior SWE on Notion’s Data Platform team.

Airflow, streaming, and the compute behind Notion AI, in every region we serve.

I have two lovely cats.

What you’ll leave with

A way to model regions and cells as data.

DAG factories that fan out without copying business logic.

Compute isolated by environment and geography, and how to migrate it.

AGENDA

Four parts, twenty-five minutes

01	The platform: regions, cells, one control plane	5 min
02	Patterns: cell model, DAG factories, compute isolation	9 min
03	Case study: indexing connectors for Notion AI	6 min
04	Takeaways and questions	2 min

PART 1 · THE PLATFORM

Four regions, and cells inside them

us-west-2

Oregon

Home region. Dev, staging and prod cells.

eu-central-1

Frankfurt

EU data residency. Dev and prod cells.

ap-northeast-1

Tokyo

Japan residency. Prod cells only.

ap-northeast-2

Seoul

Korea residency. Prod cells only.

```
cell = env - kind - region - index e.g. prod-space-euc1-0001
```

Airflow across four regions · Notion · more regions on the way

PART 1 · THE PLATFORM

The platform at a glance

2
Astro deployments: core and ai

500+
DAGs

4
AWS regions

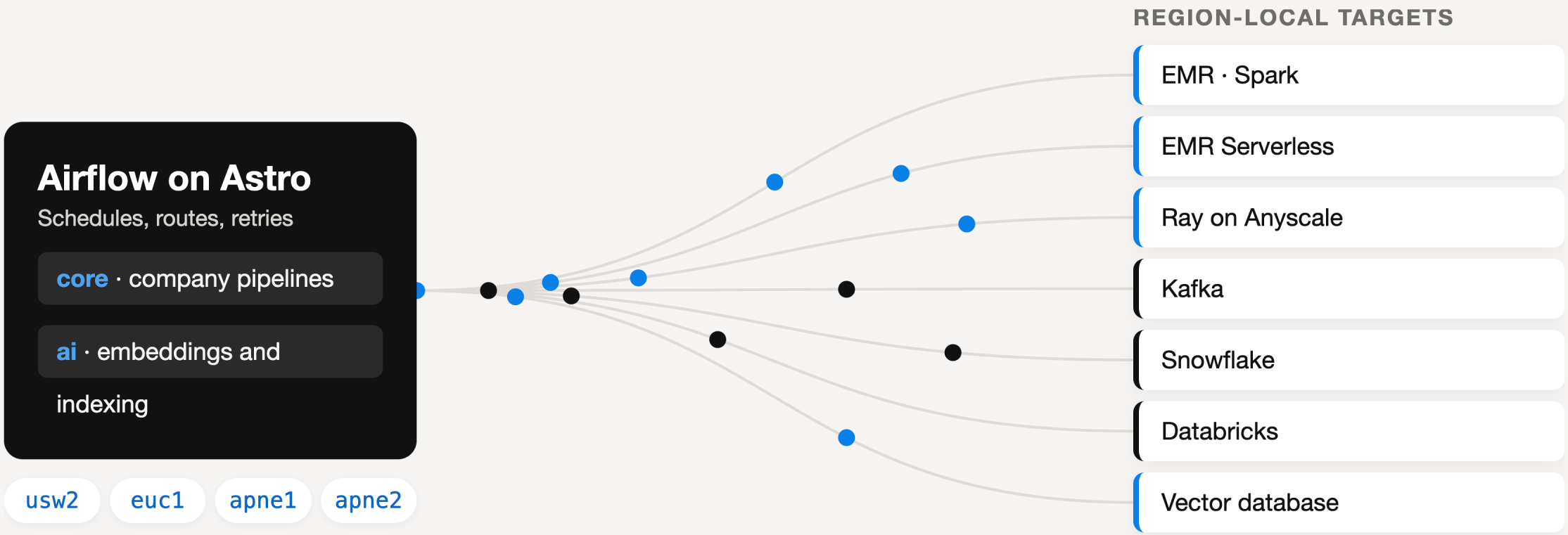
21
active cells

7
compute and data targets

A small team runs all of it. That only works if the platform does the multiplication.

PART 1 · THE PLATFORM

Airflow is the control plane



Blue: compute Airflow submits jobs to · Black: data systems it reads and writes

THE RULE EVERYTHING FOLLOWS

Every pipeline must know where its data lives.

Customer data is read, processed and written in its own region.

Regions and cells change without pipeline code changing.

Dev jobs never run on prod compute.

PART 2 · PATTERN 1

One generated cell and region model

```
# AUTO-GENERATED. Do not edit.
RegionShort = Literal[
  "apne1", "apne2", "euc1", "usw2"]

CELL_ID_TO_STATUS = {
  "prod-space-apne1-0001": "active",
  "prod-space-euc1-0001":  "active",
  "prod-space-usw2-0004":  "active",
  ...
}
```

Generated, not written

The app repo's cell registry is the source of truth.

Typed

A typo in a region name fails at parse time.

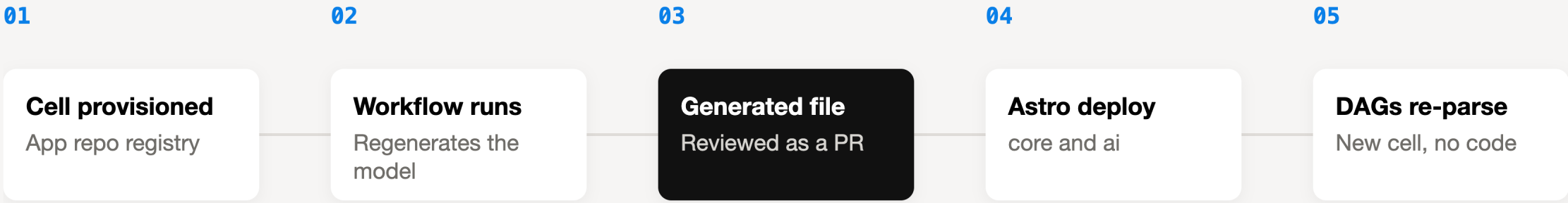
Shared by every DAG

Both deployments import the same module.

Simplified from our shared Airflow modules

PART 2 · PATTERN 1

A new cell reaches every DAG



Retiring a cell is the same path in reverse: its status changes, and its tasks stop.

PART 2 · PATTERN 2

One config, one DAG per region

Region config

```
region = "euc1"
suffix = "_euc1"
compute = v3
tags = ("eu",)
```

factory.py

Business logic, written once

GENERATED DAGS

- indexer_prod_usw2
- indexer_prod_euc1
- indexer_prod_apne1
- indexer_prod_apne2
- indexer_dev_usw2
- indexer_dev_euc1
- indexer_stg_usw2
- …

Blue: prod DAGs · Gray: dev and staging DAGs · Names simplified

PART 2 · PATTERN 2

The factory, in a dozen lines

```
for env in ALL_ENVS:
    for region in ALL_REGIONS:
        dag_id = f"indexer_{env.short}{region.dag_id_suffix}"
        with DAG(dag_id, tags=region.tags, is_paused_upon_creation=True) as dag:
            SubmitCustomAnyscaleJob(
                cloud=get_anyscale_cloud_name(region.region_long, env.env_long),
                compute_config=build_compute_config(env, region),
                entrypoint=f"... --region {region.region_long}",
            )
        globals()[dag_id] = dag
```

Region picks the compute and the data. Env picks dev or prod cloud. Paused until someone validates it.

Illustrative, simplified from our vector indexing factory

PART 2 · PATTERN 3

Compute isolated by env and region

Env	Region	Anyscale cloud
dev	usw2	notion_ml_usw2_dev
staging	usw2	notion_ml_usw2_prod
prod	usw2	notion_ml_usw2_prod
prod	euc1	notion_ml_euc1_prod
prod	apne1	notion_ml_apne1_prod
prod	apne2	notion_ml_apne2_prod

The name is the boundary

One helper turns env and region into a cloud name.

Each cloud has its own account, network and IAM.

EMR and EMR Serverless follow the same split.

PART 2 · PATTERN 3

Our custom Anyscale operator

Deferrable waits

Hours-long Ray jobs hold no worker slot.

One image per Containerfile

Jobs that share a file share a build.

Guards on cloud names

Unsuffixed legacy clouds fail before the API call.

Fingerprinted compute

Config drift between regions shows up in review.

Built on the open-source Anyscale Airflow provider

PART 2 · PATTERN 3

A cloud migration from config alone

One shared cloud per region

Dev and prod jobs side by side.

New dev and prod clouds

Created per region, with their own access.

Clone compute configs

Versions restart at 1 in each cloud.

Flip one helper

Factories pick the new names on next parse.

Block the old names

The operator rejects them at task start.

PART 3 · CASE STUDY

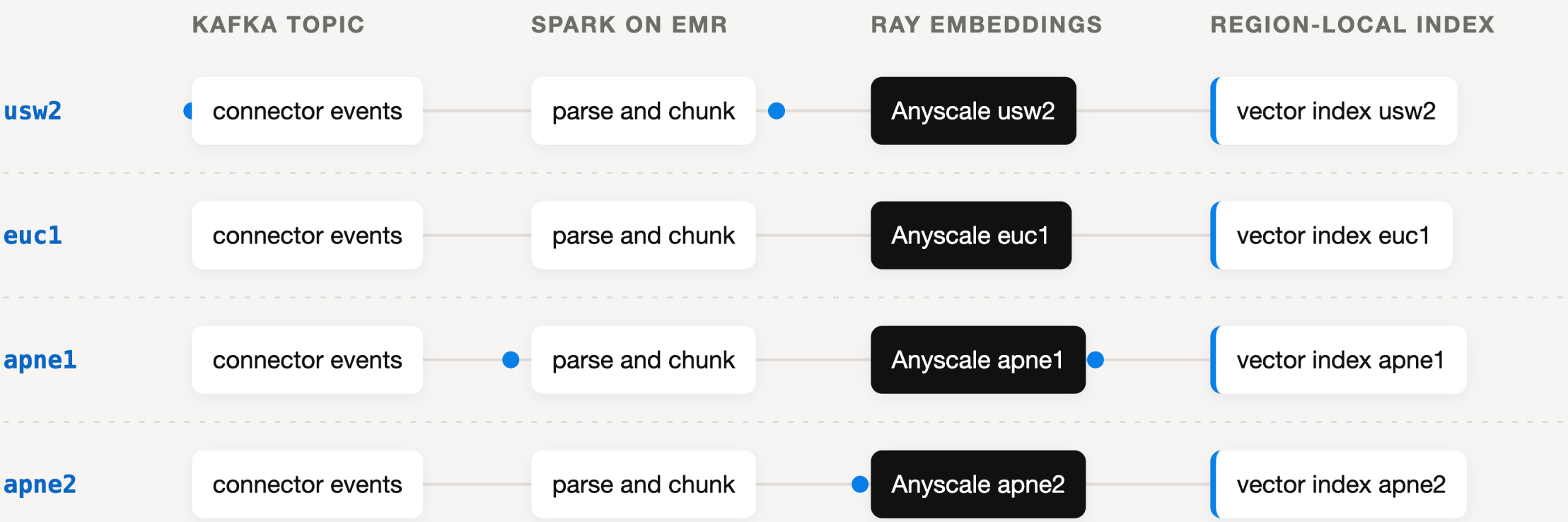
Notion AI searches your other tools

Jira	Slack	Google Drive	GitHub
Confluence	Microsoft Teams	SharePoint	Salesforce
Linear	Asana	Box	and more

Each connector is indexed in every region where customers use it

PART 3 · CASE STUDY

The same pipeline, four times



Dashed lines: region boundaries no customer content crosses

PART 3 · WHAT HELD UP

Where the abstractions held

Seoul was data entry

A new region meant new config entries, not new pipeline code.

Residency by default

Region in, region out. No DAG names a bucket or cloud by hand.

Clean cloud moves

Every Ray workload moved clouds without touching its DAG.

Where a workload went through a factory, it scaled. Where it didn’t...

PART 3 · WHERE IT BROKE DOWN

The per-region DAG file explosion

90

near-identical indexer DAG files, one per connector, env and region

```
ai_kafka_indexer_jira_prod.py
ai_kafka_indexer_jira_prod_eu.py
ai_kafka_indexer_jira_prod_apne1.py
ai_kafka_indexer_jira_prod_apne2.py
ai_kafka_indexer_jira_dev_eu.py
ai_kafka_indexer_box_prod_apne1.py
ai_kafka_indexer_slack_prod_turbopuffer_eu.py
ai_kafka_indexer_slack_ray_dev.py
… 82 more
```

Each one a copy. Each copy a chance to drift.

PART 3 · PAYING IT DOWN

From copies to one factory

BEFORE

A file per connector, env and region

A new region means copying every connector's file. A fix means editing all of them.

AFTER

One factory, connectors as config

A new region is one config entry. A fix lands everywhere at once.

TAKEAWAYS

Four things to steal

01	**Model regions and cells as generated data**	One typed source, owned by whoever provisions them.
02	**Factories own the fan-out**	Logic written once; regions and envs are config.
03	**Put the boundary in the name**	One helper maps env and region to compute.
04	**Make the right way the fast way**	Or people will copy the file.

IF YOU REMEMBER ONE THING

Model regions as data. Let the platform do the multiplication.

Then a new region, a retired cell, or a cloud migration is a config change, not a project.

Thank you

Questions?



FIND ME

Yunhao Qing · https://x.com/yunhao_q

WE'RE HIRING

notion.com/careers