

Migration Engine

GIS workflow migration that learns as it goes.

From legacy desktop to cloud pipeline, automated, verified, documented.

~30 min
Engine runtime

7
Stages

1
Human checkpoint

Parts.

01 Traditional way.

02 Migration engine

03 Agentic GIS.

Axis Spatial is the agent layer for geospatial intelligence.

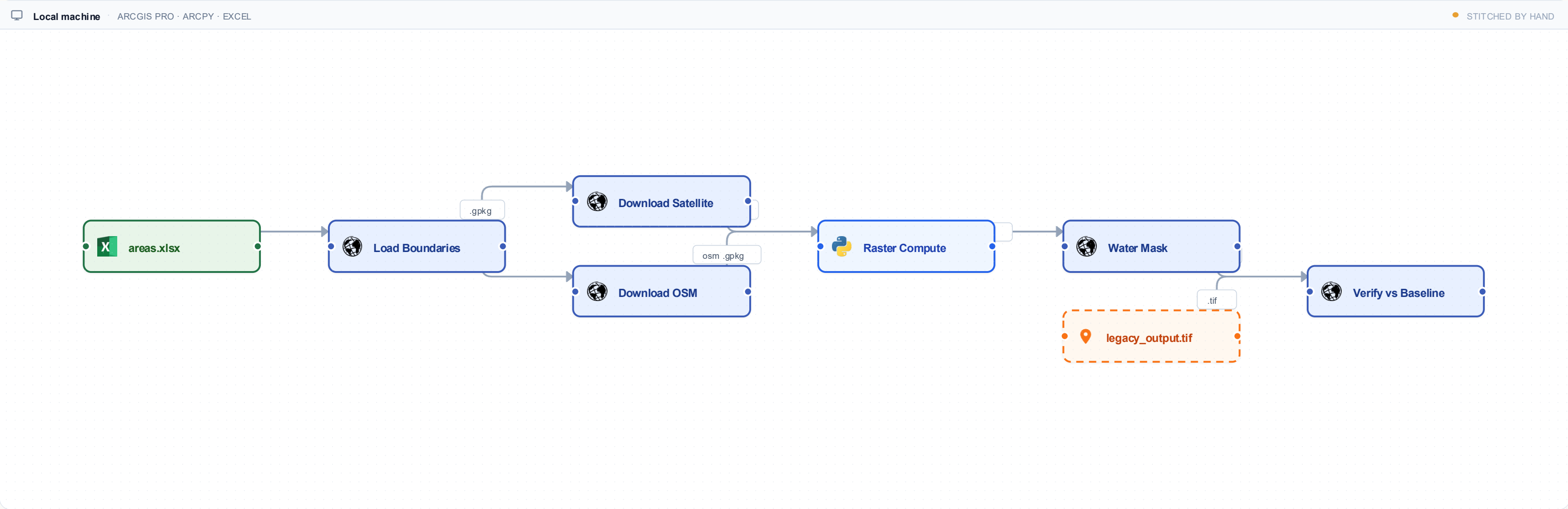
We sit underneath the products and teams that own the domain.
You keep the product. We run the agent layer.

Worked example · Today

A satellite data pipeline, the way it runs today.

Pull satellite tiles for a list of areas, combine them with open map data, do raster analysis, compare against a previous run. ArcPy + Python + Excel, stitched on a laptop.

TODAY · LOCAL + ARCPY

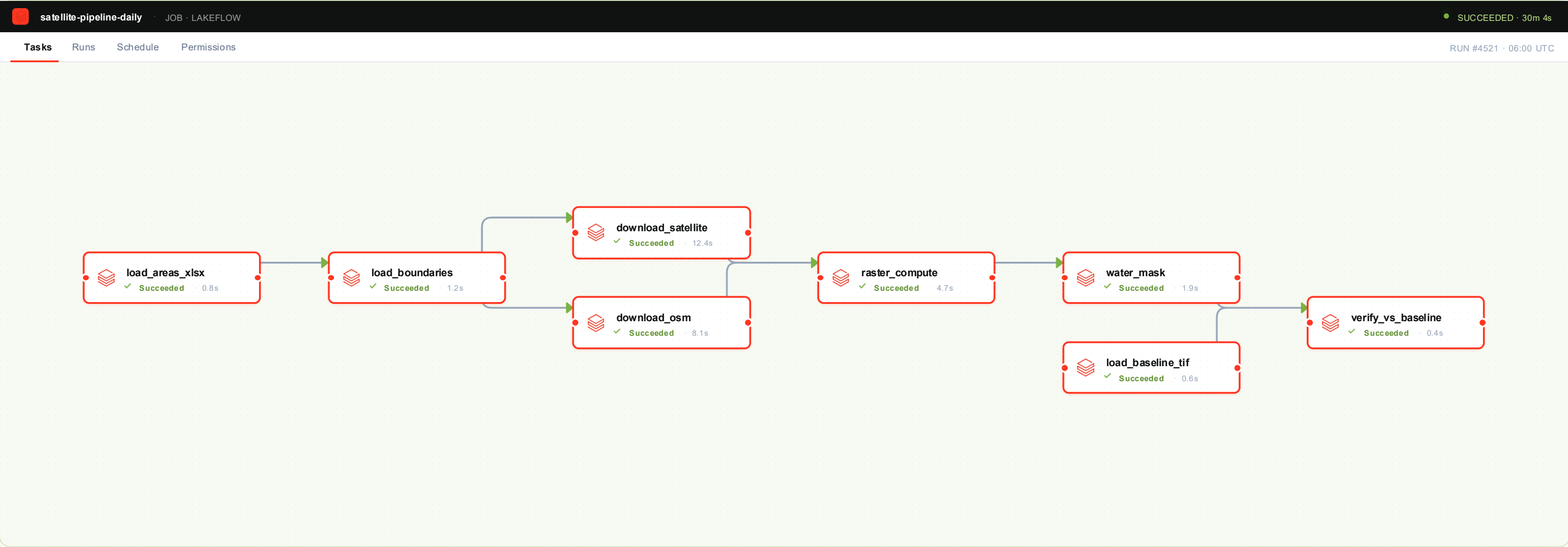


[Worked example](#) · [Migrated](#)

Same pipeline. Same logic. Native cloud job.

The Migration Engine rebuilds it as a single Lakeflow job on *Databricks* , also runs on Snowflake, AWS, GCP or Azure. One DAG. Tracked runs. Native scheduling.

CLOUD · NATIVE PIPELINE



First, the problem

Why do teams migrate from desktop to cloud?

01 · Vendor lock-in

Per-seat pricing. Maintenance fees going up every year. Your entire ecosystem depends on one vendor's roadmap.

02 · Scattered toolchain

ArcPy scripts. Excel processing. Manual checks. Tribal knowledge. It's inherently scattered, hard to reproduce, hard to scale, impossible to audit.

03 · No reproducibility, scalability, or auditability

Same workflow, different geographies, countries, years, each one requires different setup, different inputs. No autoscaling. No logs. No version control.

The old way. Six stages. Three months. Per workflow.

The old way

Six stages. Every one a dead end.

01FinderReverse-engineer~2 WK

02PlatformPick cloud + spatial~1 WK

03CodeTranslate by hand2-4 WK

04DeployOrchestration hell1-2 WK

05Verify"Looks right"~2 WK

06TestN × N casesWEEKS

Stage 01 · Finder

Reverse-engineer scripts & files

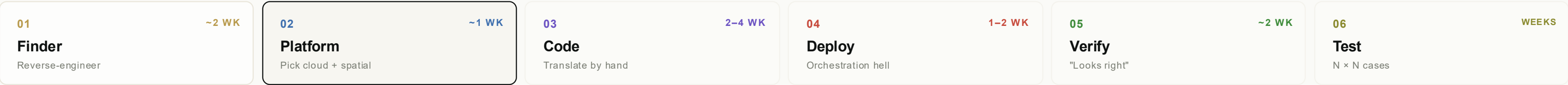
CONSULTANT · ~2 WK

C:\GIS\Projects\FloodAnalysis

NAME	TYPE
▶ data	3 ITEMS
project_data.gdb	GDB
wetlands.gdb	B GDB
flood_zones.shp	SHP
▶ rasters	2 ITEMS
elevation_dem.tif	TIF
landuse_raster.tif	TIF
▶ scripts	2 ITEMS
process_batch.py	PY
transform_data.py	PY
▶ tabular	3 ITEMS
FloodAnalysis.aprx	APRX
backup_analysis.qgz	QGZ
README.md	MD

The old way

Six stages. Every one a dead end.



03

Code

Translate by hand

2-4 WK

04

Deploy

Orchestration hell

1-2 WK

05

Verify

"Looks right"

~2 WK

06

Test

N × N cases

WEEKS

Stage 02 · Platform

Pick cloud + spatial library

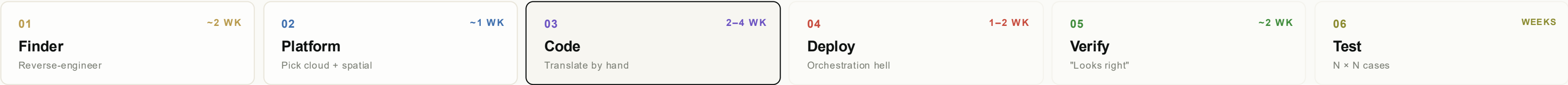
CONSULTANT · ~1 WK

PLATFORM COMPARISON

CAPABILITY	DATABRICKS	SNOWFLAKE	AWS
Vector ops	Mosaic	❌ LAO Mosaic	Sedona on EMR
Raster	Mosaic + GDAL	No native	GDAL on Lambda
CRS handling	Partial	❌ PROJ	Proj4 DIY
ArcPy compat	None	None	None
Cost model	DBU	FINANCE	Per-service

The old way

Six stages. Every one a dead end.



Stage 03 · Code

Translate ArcPy by hand

ENGINEER · 2–4 WK

TRANSLATE · ARCPY → PYSPARK

– ARCPY (BEFORE)

```
import arcpy
arcpy.env.workspace = "C:/GIS/Data"
arcpy.analysis.Buffer(
    "flood_zones.shp",
    "buffered.shp",
    "500 Meters",
    "FULL", "ROUND", "ALL"
)
arcpy.management.Dissolve(
    "buffered.shp",
    "dissolved.shp",
    ["region"]
)
```

+ PYSPARK (AFTER)

```
from mosaic import *
from pyspark.sql.functions import *

df = spark.read.format("ogr")\
    .load("s3://flood_zones.shp")

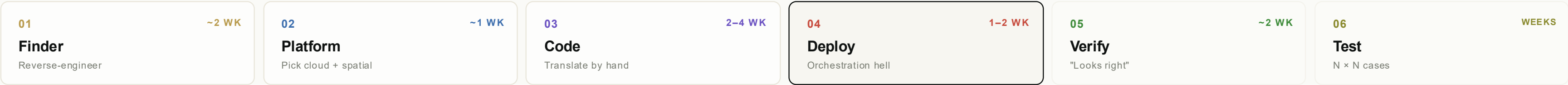
buf = df.withColumn(
    "geom_buf",
    st_buffer(col("geom"), lit(500))
).groupBy("region")\
    .agg(st_union_agg(col("geom_buf")))

buf.write.format("delta").save(...)
```

Every ESRI function → new library. Every quirk → manual fix.

The old way

Six stages. Every one a dead end.



Stage 04 · Deploy

Orchestration, IAM, env drift

ENGINEER · 1-2 WK

CLUSTER DEPLOY · SECOND ATTEMPT

```
$ databricks jobs run --job-id 4827
→ Cluster spinning up...
✓ Node acquired (i3.xlarge)
✓ Env attached

⚠ ImportError: No module named 'arcpy'
  at process_batch.py:3

⚠ GDALError: PROJ_LIB not set on cluster
  MP PROJ_LIB not set on cluster

⚠ CRSMismatch: EPSG:4326 vs EPSG:3857
  reprojection needed on ingest

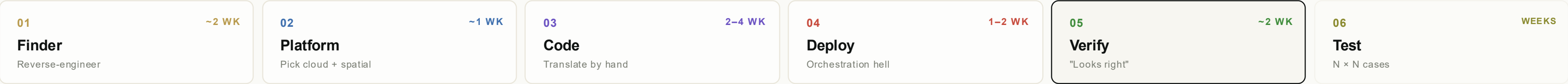
⚠ IAMError: s3://flood-data · access denied

$ # Runs on my laptop.
$ # Does not run on the cluster.

JOB FAILED · 3m 12s
```

The old way

Six stages. Every one a dead end.



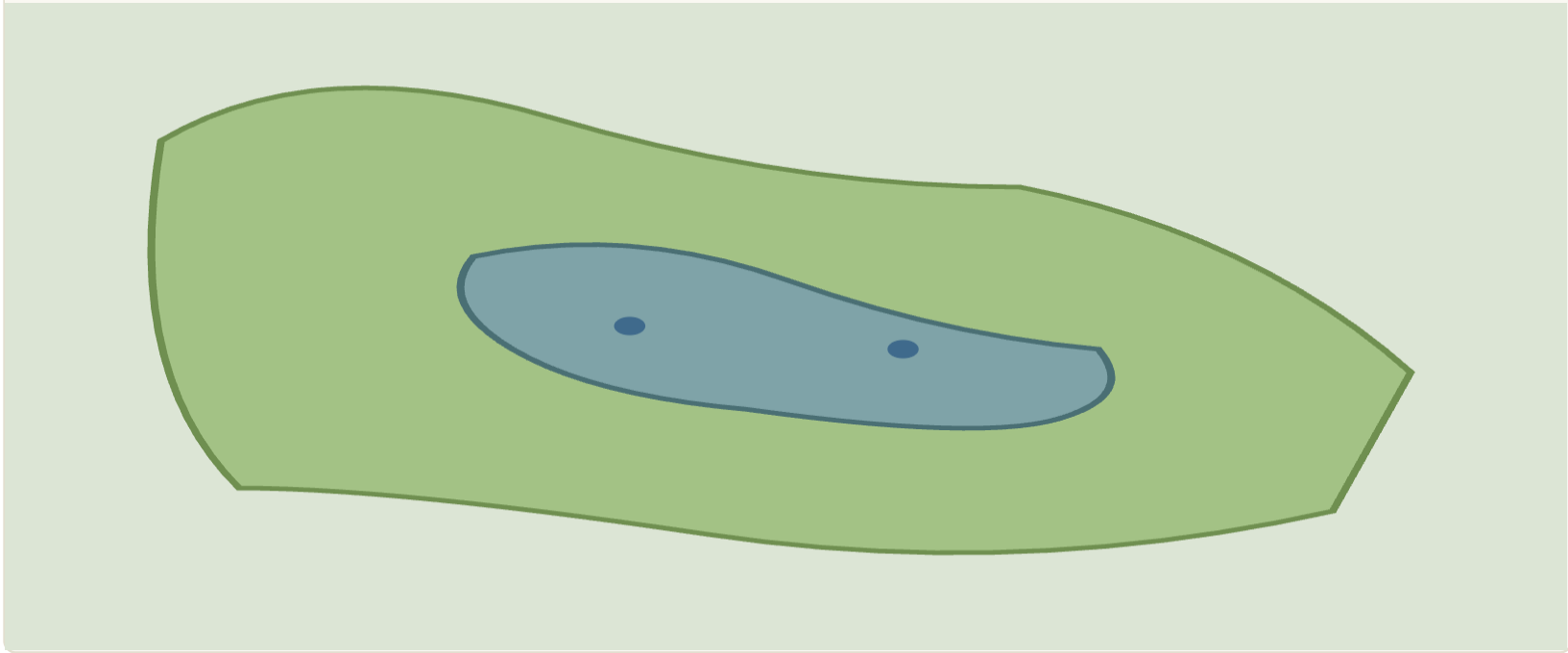
Stage 05 · Verify

"Looks right, ship it"

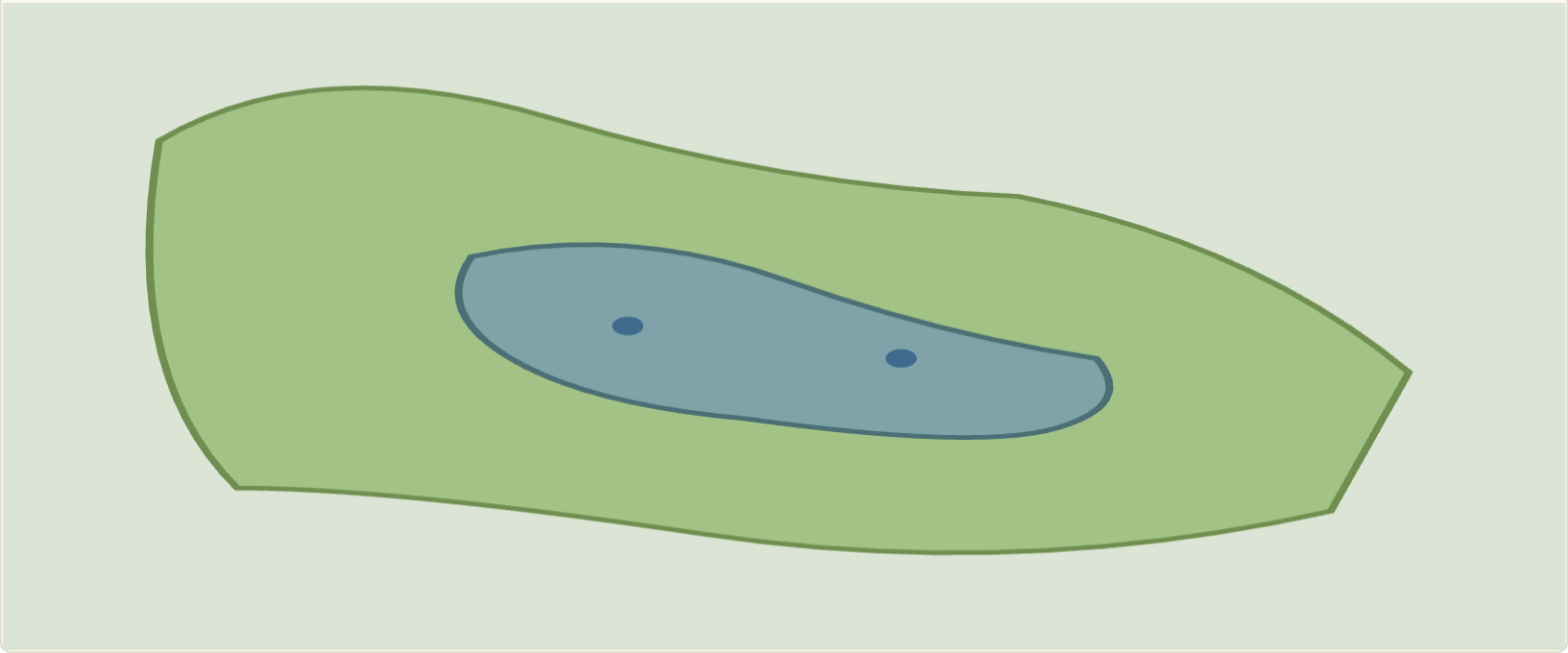
ANALYST · ~2 WK

VERIFY · EYEBALL

ARCPRO · OUTPUT



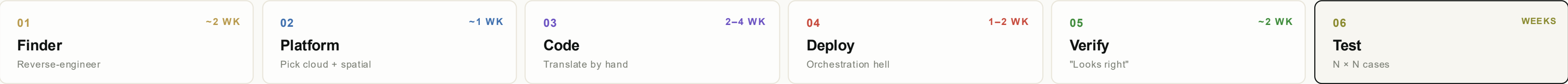
DATABRICKS · OUTPUT



LOOKS RIGHT — no tolerance check, no reference dataset, no sign-off log

The old way

Six stages. Every one a dead end.



Stage 06 · Test

N × N parametric cases

ANALYST · WEEKS

TEST · PARAMETRIC COVERAGE

COUNTRIES · 12

US

CA

ì ÿ

BR

AR

UK

FR

DE

IT

ES

Ĥ

AU

SCENARIOS · 3

BASELINE

+2°C

+4°C

RETURN PERIODS · 5

10-year flood

25-year flood

50-year flood

100-year flood

250-year flood

12 × 5 × 3

180

RUNS · MANUALLY

Add it up

≈ 3 months.

Per workflow. Then you start over for the next one.
Legacy teams have dozens.

The Migration Engine. Same stages. Run by agents. Plus Learn.

Positioning

We're not trying to be the platform. We're the bridge onto whichever one you pick.

Your legacy workflows

 ArcGIS Pro

 ArcPy

 QGIS

 Excel

 FME

 MATLAB

On desktop · tribal knowledge · stuck.



 Migration Engine

A connector / rebuilder.
Not a platform.

Your platform choice

 Databricks

 Snowflake

 AWS

 BigQuery

 Azure

 PostGIS

Your stack. Your contract. No lock-in.

We are not

A new warehouse, a new GIS, or another platform to buy into.

We are

The bridge that rebuilds your local workflows onto whichever platform you've already chosen.

Seven stages. One human checkpoint.

01

Finder
Auto-scan → DAG

02

Refine
The one human checkpoint

03

Platform
Databricks · Snowflake · AWS

04

Code
Builder · Validator · Reviewer

05

Deploy
Auto-heal on failure

06

Verify
Artifact-by-artifact diff

07

Learn
Compounds in your tenant

Stage 01 · Finder

Auto-scan scripts & files

AXIS SPATIAL

Discover

2 Refine

3 Platform

4 Generate

5 Deploy

6 Verify

Chat

Jobs

0

P

Discover > Refine > Platform > Generate > Deploy > Verify

Migration Workspace

Upload a workflow, create the canonical pipeline session, and move into refinement without touching builder chat.

Back to Discover

Migration Intake

Select a folder in Chrome or upload workflow files directly. The browser parses the workflow locally, then Axis Spatial creates the canonical migration session used by Refine, Platform, Generate, Deploy, and Verify.

Select a folder to scan

Your files stay local. Only metadata is analysed.

Upload folder contents

Upload workflow files

Use folder upload if the native folder picker is blocked, or upload workflow files directly if you only need a file-based scan.

Running AI analysis (1/32)...

Loading AI model (100%)

OSM/Documentation/OSM_key_value_pairs_v2.csv: Using cloud AI for OSM_key_value_pairs_v2.csv...

32 Found

32 Parsed

0 Skipped

0 Errors

AI analysed: 1 / 32 Python files

Cancel scan

Parsed Workflow Summary

FILES32

PYTHON4

QGIS/FME0

SPATIAL DATA19

EXCEL8

SCAN STATEai-analysis

Runtime Boundary

Builder chat remains separate. This migration surface only creates the wc context and canonical session. Cloud targets like Databricks, Snowflake, and GCP still get selected later on the Platform step.

Detected Files

OSM_IRL_key_value_statistics.xlsx

OSM_IRL_key_value_statistics.xlsx

OSM_key_value_pairs_v2.csv

OSM/Documentation/OSM_key_value_pairs_v2.csv

OSM_key_value_pairs_v1.csv

OSM/Documentation/OSM_key_value_pairs_v1.csv

OSM_key_value_pairs.csv

Part 2 · The Migration Engine

Seven stages. One human checkpoint.

- 01

Finder
Auto-scan → DAG
- 02

Refine
The one human checkpoint
- 03

Platform
Databricks · Snowflake · AWS
- 04

Code
Builder · Validator · Reviewer
- 05

Deploy
Auto-heal on failure
- 06

Verify
Artifact-by-artifact diff
- 07

Learn
Compounds in your tenant

Stage 02 · Refine

The one human checkpoint

AXIS SPATIAL

Discover

Refine

3 Platform

4 Generate

5 Deploy

6 Verify

Chat

Jobs

0

P

Discover > Refine > Platform > Generate > Deploy > Verify

7 nodes0 confirmed

Refine Workflow

Click nodes to configure. Review AI assumptions below.

+📄

+⚙️

↶↷

Context

Re-layout

Test Code

↗️

Load and Validate Inputs

exposure_fc, buildings_fc

Spatial Normalization

exposure_srsj, buildings_srsj

pop_raster

Building Footprint Join

exposure_buildings_join, occupancy_weights.csv

Residential Disaggregation

nonres_disagg_fc

Merge and Finalize Output

Generate Occupancy Weights

Non-Residential Disaggregation

React Flow

+ - ↺ ↻ 📄

AI Assumptions

The current script relies on local Esri File Geodatabases (.gdb) for input and scratch workspaces. How should we store and manage the vector data (exposure points, building footprints) in the cloud environment?

Migrating away from proprietary local file formats is required for cloud-native scaling. The choice impacts the compute engine (e.g., Spark vs. PostGIS) used for the spatial join.

Cloud Data Warehouse with native geospatial support (e.g., Snowflake, BigQuery) (Recommended)

Cloud Object Storage using cloud-native formats (e.g., S3/GCS with GeoParquet)

Managed Relational Database with Spatial Extension (e.g., PostgreSQL + PostGIS)

Other (specify)

Confirm

Skip All

Reviewed: 0 assumptions

Part 2 · The Migration Engine

The Migration Engine

Seven stages.One human checkpoint.

01

Finder

Auto-scan → DAG

02

Refine

The one human checkpoint

03

Platform

Databricks · Snowflake · AWS

04

Code

Builder · Validator · Reviewer

05

Deploy

Auto-heal on failure

06

Verify

Artifact-by-artifact diff

07

Learn

Compounds in your tenant

Stage 03 · Platform

Pick target platform

AXIS SPATIAL

Discover

Refine

Platform

4Generate

5Deploy

6Verify

Chat

Jobs

0

P

← Back to Refine

Discover >

Refine >

Platform >

Generate >

Deploy >

Verify

Choose Target Platform

Select where your pipeline should run. Customer-cloud targets generate platform-native code, while the standalone runtime keeps execution on the Axis-managed E2B backend.

Workflow: Based on the provided scripts and...

Databricks

PySpark ETL with Delta Lake and Unity Catalog

SPATIAL CAPABILITIES

- GeoPandas + Shapely (pure Python, serverless)
- Delta Lake writes via deltalake library
- Unity Catalog Volumes for persistent storage

CORE FEATURES

Unity Catalog governance

Multi-task Jobs API (auto-generated)

Volumes for file storage

Serverless compute

Large-scale geospatial ETL with governance

CUSTOMER CLOUD EXECUTION

AWS Glue

PySpark ETL with S3 storage and Step Functions

SPATIAL CAPABILITIES

- Apache Sedona JAR for spatial SQL
- GeoPandas in Glue Python Shell (small data)
- EMR with Sedona for large-scale processing

CORE FEATURES

S3 for all I/O

Step Functions orchestration (auto-generated)

Glue Data Catalog

Auto-scaling workers

AWS-native geospatial pipelines with S3

CUSTOMER CLOUD EXECUTION

Google Cloud

BigQuery Geography SQL with Dataproc

SPATIAL CAPABILITIES

- Native GEOGRAPHY type in BigQuery
- ST_GEOPOINT, ST_BUFFER, ST_DISTANCE functions
- H3 spatial indexing via BigQuery GIS

CORE FEATURES

BigQuery serverless SQL

Cloud Workflows orchestration (auto-generated)

Cloud Storage for files

Dataproc for Spark workloads

SQL-first spatial analytics with BigQuery

CUSTOMER CLOUD EXECUTION

Snowflake

Snowpark Python with native H3 and GEOGRAPHY

SPATIAL CAPABILITIES

- Native GEOGRAPHY and GEOMETRY types
- Native H3 functions (H3_LATLNG_TO_CELL)
- ST_BUFFER, ST_DISTANCE, ST_CONTAINS

CORE FEATURES

Snowpark Python (runs on Snowflake compute)

Iceberg Tables for open format

Tasks for scheduling

Streams for CDC

Snowflake-native geospatial with Snowpark

CUSTOMER CLOUD EXECUTION

Standalone Runtime

Axis-managed runtime for one-off runs on E2B

SPATIAL CAPABILITIES

- GeoPandas for vector operations
- Rasterio for raster processing
- Shapely for geometry manipulation

CORE FEATURES

Axis-managed runtime

Managed E2B execution backend

No customer cloud account required

Downloadable Python bundle

One-off verification and pilot runs

Fast managed verification, pilots, migrations, and no-infra onboarding

MANAGED EXECUTION BACKEND

AUTOMATED

Part 2 · The Migration Engine

The Migration Engine

Seven stages. One human checkpoint.

01

Finder
Auto-scan → DAG

02

Refine
The one human checkpoint

03

Platform
Databricks · Snowflake · AWS

04

Code
Builder · Validator · Reviewer

05

Deploy
Auto-heal on failure

06

Verify
Artifact-by-artifact diff

07

Learn
Compounds in your tenant

Stage 04 · Code

Builder · Validator · Reviewer

AXIS SPATIAL

Discover

Refine

3 Platform

Generate

5 Deploy

6 Verify

Chat

Jobs

P

Back to Platform

Discover >

Refine >

Platform >

Generate >

Deploy >

Verify

Generate

In Progress

Step 1/5: Queued

occupancy_weights.xlsx

Target: Standalone Runtime

Change

Infer Cloud Paths

Agents processing...

GENERATED FILES

docs

MIGRATION_NOTES.md

DATA_DICTIONARY.md

pipeline

sentinel_ndvi_pipeline.py

stac_utils.py

config

databricks_job.json

sentinel_ndvi_pipeline.py

```
1 """
2 Sentinel-2 NDVI Pipeline - Cloud-Native on Databricks
3 Source: Copernicus Data Space Ecosystem (CDSE) STAC API
4 Generated by Axis Spatial AI Agents
5 """
6
7 from pystac_client import Client
8 import rioarray
9 import xarray as xr
10 import numpy as np
11 from pathlib import Path
12
13 # --- Connect to Copernicus CDSE STAC ---
14 STAC_URL = "https://catalogue.dataspace.copernicus.eu/stac"
15
16 def search_sentinel2(bbox: list, date_range: str, max_cloud: int = 20):
17     """Search for Sentinel-2 L2A scenes via CDSE STAC API."""
18     client = Client.open(STAC_URL)
19     search = client.search(
20         collections=["sentinel-2-l2a"],
21         bbox=bbox,
22         datetime=date_range,
23         query={"eo:cloud_cover": {"lt": max_cloud}},
24     )
25     return list(search.items())
26
27 # --- Band Loading ---
28 def load_bands(item):
29     """Load Red (B04), NIR (B08), and SCL bands from a STAC item."""
30     red = rioarray.open_rasterio(item.assets["B04"].href).squeeze()
31     nir = rioarray.open_rasterio(item.assets["B08"].href).squeeze()
32     scl = rioarray.open_rasterio(item.assets["SCL"].href).squeeze()
33     return red, nir, scl
34
35 # --- Cloud Masking ---
36 def create_cloud_mask(scl, red):
37     """Create cloud mask from SCL band, resampled to 10m."""
38     # Resample 20m SCL to match 10m band resolution
39     scl_10m = scl.rio.reproject_match(red, resampling='nearest')
40     # Clear pixels: 4=vegetation, 5=bare soil, 6=water
41     clear_mask = scl_10m.isin([4, 5, 6])
42     return clear_mask
43
44 # --- NDVI Calculation ---
```

Part 2 · The Migration Engine

The Migration Engine

Seven stages. One human checkpoint.

01

Finder

Auto-scan → DAG

02

Refine

The one human checkpoint

03

Platform

Databricks · Snowflake · AWS

04

Code

Builder · Validator · Reviewer

05

Deploy

Auto-heal on failure

06

Verify

Artifact-by-artifact diff

07

Learn

Compounds in your tenant

Stage 05 · Deploy

Auto-heal on failure

AXIS SPATIAL

Discover

Refine

Platform

Generate

Deploy

Verify

Chat

Jobs

0

P

Back to Generate

Discover

Refine

Platform

Generate

Deploy

Verify

Deploy Pipeline

Configure and deploy to Standalone Runtime

Checking deployment...

Provisioning runtime and validating configuration. This may take a few moments.

Standalone Runtime

Run the standalone Python bundle on the Axis-managed runtime

Change Platform

Deployment Preview

Standalone Runtime Job

occupancy_weights.xlsx

RUN

Cluster

Managed Standard

Schedule

Manual (on-demand)

Pipeline Summary

4 files 15 operations

Code Snippet

```
# Databricks Notebook
# Generated by Axis Spatial

from pyspark.sql import SparkSession
import mosaic as mos

# Enable H3 functions
mos.enable_mosaic(spark)

# Read Delta Lake table
df = spark.read.format("delta").load(
    "dbfs:/mnt/geospatial/flood_zones"
)

# Apply H3 indexing at resolution 9
df_indexed = df.withColumn(
    "h3_index",
    mos.grid_pointascellid(f.col("geometry"), F.lit(9))
)


```

Managed Runtime

Standalone runtime is ready

This run uses the backend-managed E2B runtime. No customer cloud connection is required.

Configuration

Cluster Size

Managed Standard (4 cores, 8 GB)

Axis-managed runtime for MVP-sized jobs

Schedule

Manual (on-demand)

Enable Alerts

Show Advanced Options

Data Source Configuration

Configure source and output paths for your STANDALONE deployment

Primary Source

DBFS

Unity Catalog Volume

External Mount

Source Path

dbfs:/mnt/data/raw-data/shapefiles/

Output Path

dbfs:/mnt/data/processed/output/

Format

Shapefile (.shp)

Description

e.g. Parcel boundary

Deploying to Standalone Runtime

Checking...

```
$ axis-spatial deploy --platform standalone-runtime

Checking deployment...
- Provisioning managed E2B sandbox... [Checking]
- Executing canonical standalone bundle... [Checking]
- Collecting runtime artefacts... [Checking]

Elapsed: 00:12
```

Next: Verify

AUTOMATED

Seven stages. One human checkpoint.

01

Finder
Auto-scan → DAG

02

Refine
The one human checkpoint

03

Platform
Databricks · Snowflake · AWS

04

Code
Builder · Validator · Reviewer

05

Deploy
Auto-heal on failure

06

Verify
Artifact-by-artifact diff

07

Learn
Compounds in your tenant

Stage 06 · Verify

Automated

Artifact-by-artifact diff

AXIS SPATIAL

Discover

Refine

Platform

Generate

Deploy

6 Verify

ChatJobs0P

Back to Deploy > Discover > Refine > Platform > Generate > Deploy > Verify

Verify Outputs

4 PASS1 WARN0:20

Confirming legacy and automated results match for migration integrity

LEGACY VS CLOUD COMPARISON

Each legacy file is compared against its cloud-generated equivalent. Matching results prove migration integrity.

FILE PAIR	TYPE	MATCH
affected_parcelshp -> .parquet	Vector	PASS
flood_zones_bufferedshp -> .parquet	Vector	PASS
damage_report.csv -> .parquet	Tabular	WARN
dem_clipped.tif -> .tif	Raster	PASS
watershed_boundarystate -> .parquet	Vector	PASS

watershed_boundaryPASS

Single feature geometry match - area delta 0.00 sq m

LEGACY OUTPUT

FormatShapefile

CRSEPSG:4326

Count1 feature

Size0.3 MB

CLOUD OUTPUT

FormatGeoParquet

CRSEPSG:4326

Count1 feature

Size0.1 MB (67% smaller)

INTEGRITY CHECKS

Feature count matches (1 feature = 1 feature)

CRS identical (EPSG:4326)

Geometry hash match (SHA-256)

Attribute schema matches

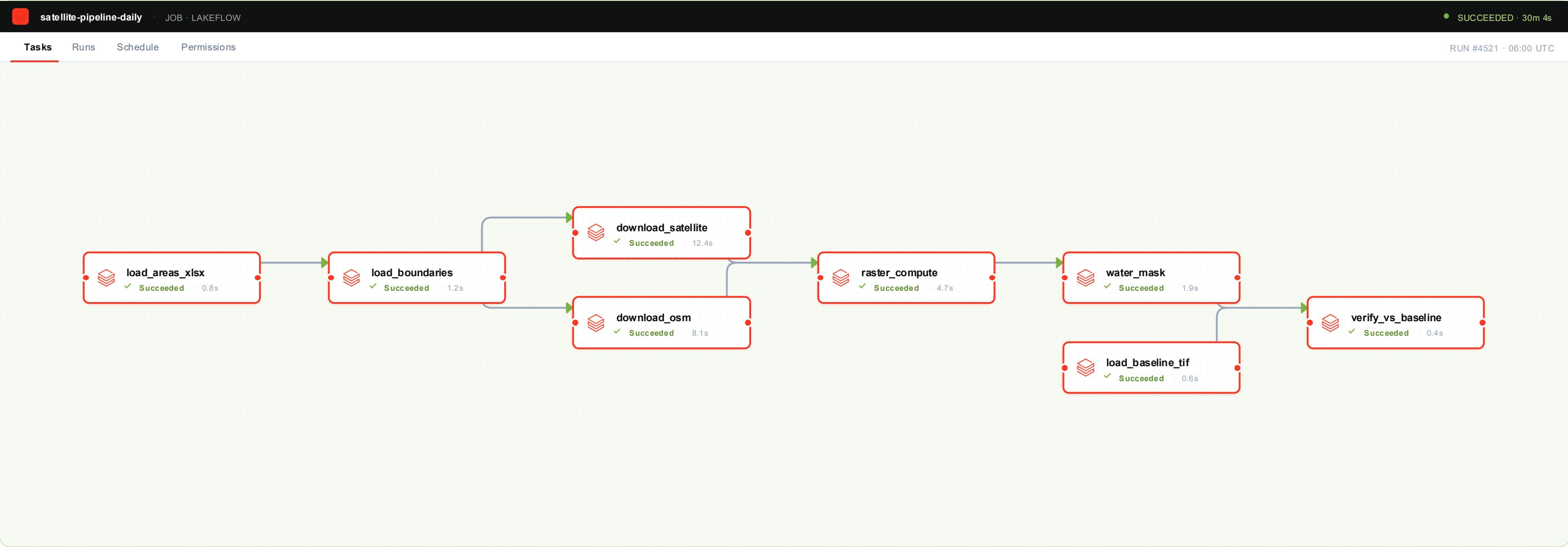
Part 2 · The Migration Engine

[Worked example](#) · [Migrated](#)

Same pipeline. Same logic. Native cloud job.

The Migration Engine rebuilds it as a single Lakeflow job on *Databricks* , also runs on Snowflake, AWS, GCP or Azure. One DAG. Tracked runs. Native scheduling.

CLOUD · NATIVE PIPELINE



The Migration Engine

Seven stages. One human checkpoint.

01

Finder
Auto-scan → DAG

02

Refine
The one human checkpoint

03

Platform
Databricks · Snowflake · AWS

04

Code
Builder · Validator · Reviewer

05

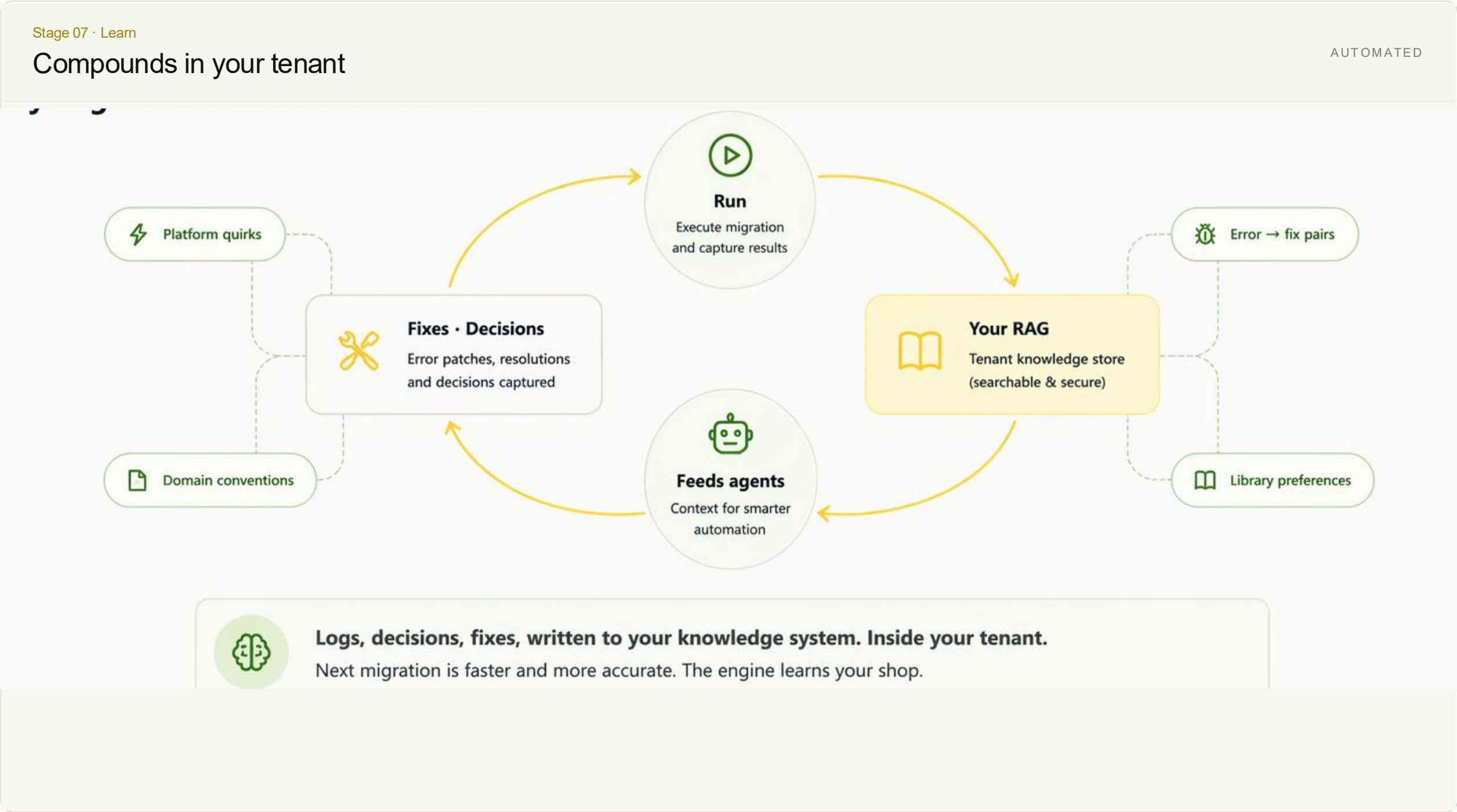
Deploy
Auto-heal on failure

06

Verify
Artifact-by-artifact diff

07

Learn
Compounds in your tenant



Add it up

≈ 30 min.

Per workflow. One checkpoint. 100% verified.
Then your tenant gets faster at the next one.

~30 min

ENGINE RUNTIME

1

HUMAN CHECKPOINT

100%

VERIFIED OUTPUT

What comes out

Our satellite pipeline. ArcPy + Excel + manual → Databricks Lakeflow job.

The old way
~~3-4~~
~~wk~~



The Migration Engine
~30 min

Consultant estimate

Engine runtime · same output · verified

Positioning

We're not trying to be the platform. We're the bridge onto whichever one you pick.

Your legacy workflows

 ArcGIS Pro

 ArcPy

 QGIS

 Excel

 FME

 MATLAB

On desktop · tribal knowledge · stuck.



 Migration Engine

A connector / rebuilder.
Not a platform.

Your platform choice

 Databricks

 Snowflake

 AWS

 BigQuery

 Azure

 PostGIS

Your stack. Your contract. No lock-in.

We are not

A new warehouse, a new GIS, or another platform to buy into.

We are

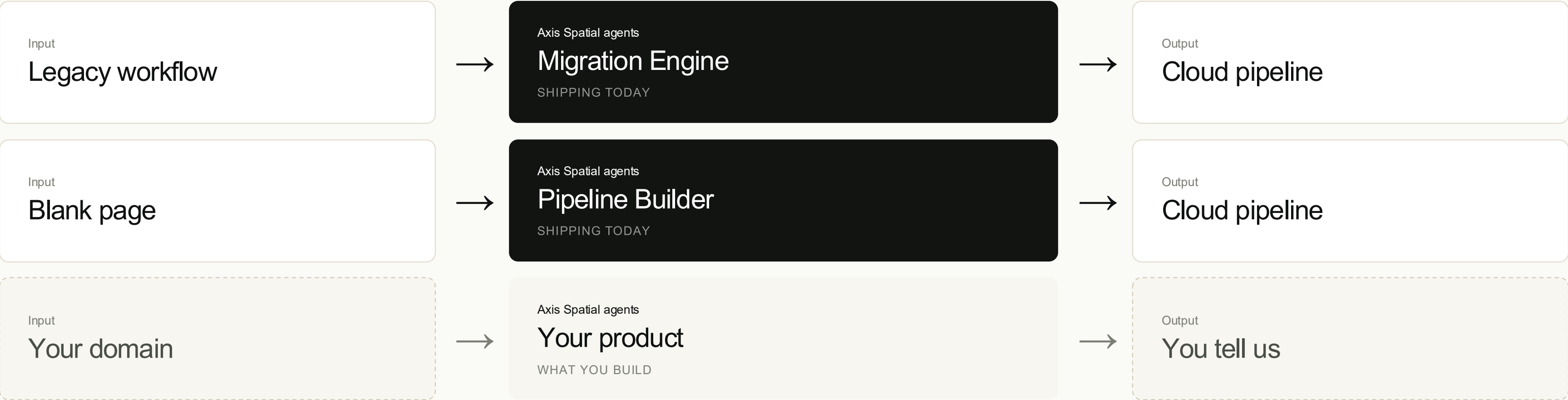
The bridge that rebuilds your local workflows onto whichever platform you've already chosen.

Beyond migration. The agent layer for geospatial teams.

Beyond migration

Migration is where Axis Spatial started.

The same agents power every geospatial workflow.



Same agents. Same validation. Same learning loop.

Build vs buy

Stay on the product. *We run the agents.*

Your team ships algorithms and domain logic. We ship the agent layer and the infrastructure it runs on.

Build it yourself

- ✗ 6–12 months of infra work
- ✗ Stand up agent framework, sandboxes, evals
- ✗ Best engineers on runtime glue
- ✗ Own the agent bugs forever
- ✗ Domain expertise diluted by plumbing

With Axis Spatial

- ✓ Plug in, week one
- ✓ Agent framework, sandboxes, evals — already built
- ✓ Best engineers on product & algorithms
- ✓ We own the agent bugs
- ✓ Domain expertise compounds into product

01 Geospatial-native.

Spatial libraries, CRS, raster and vector — our agents speak it. Generic coding agents don't.

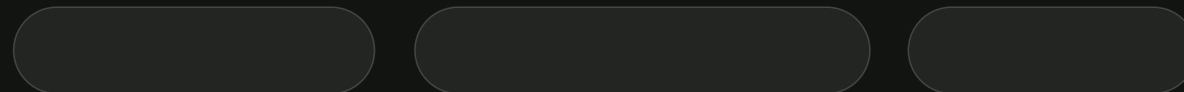
02 Runs in your tenant.

Your data never leaves. Your knowledge compounds to you, not to us.

03 Learning loop.

Every run makes the next one faster. For your team, not our other customers.

Bring your messiest workflow.
We'll map it live.



najah@axisspatial.com

axisspatial.com · Geomob Berlin · 2026