AXIS SPATIAL

Migration Engine

GIS workflow migration that learns as it goes.

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

~30 min	7	1
Engine runtime	Stages	Human checkpoint

Parts.

01 Traditional way.

02 Migration engine

03 Agentic GIS.

Who we are

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.

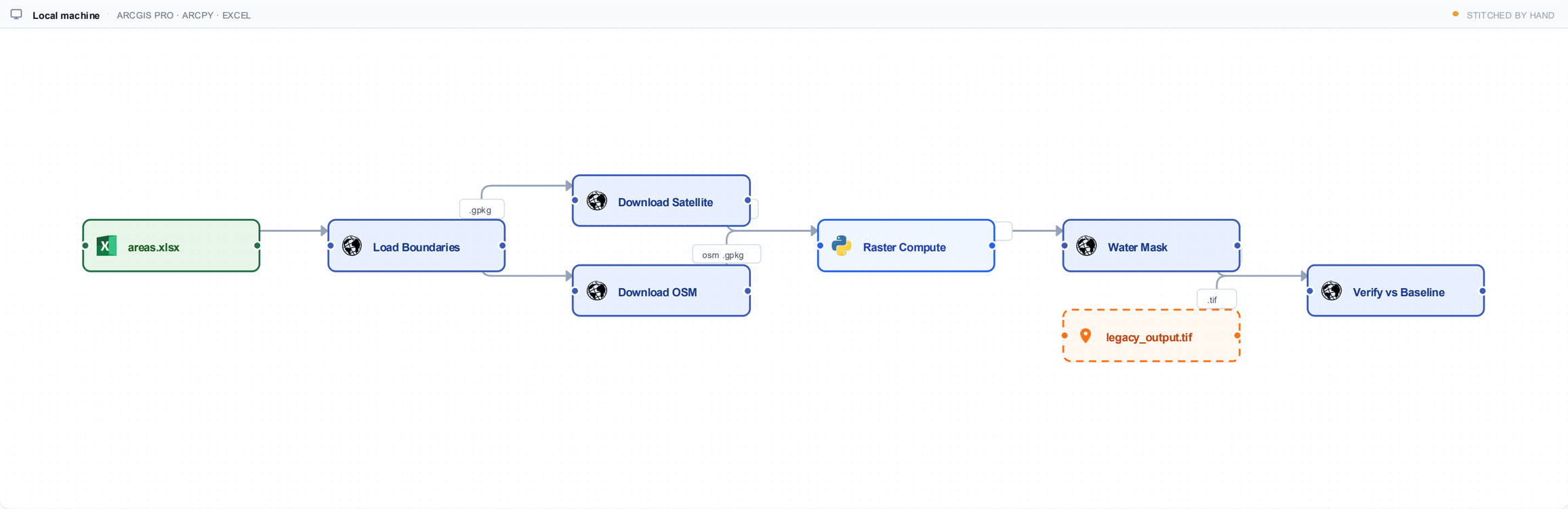
Najah Ghalyoun · najah@axisspatial.com

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

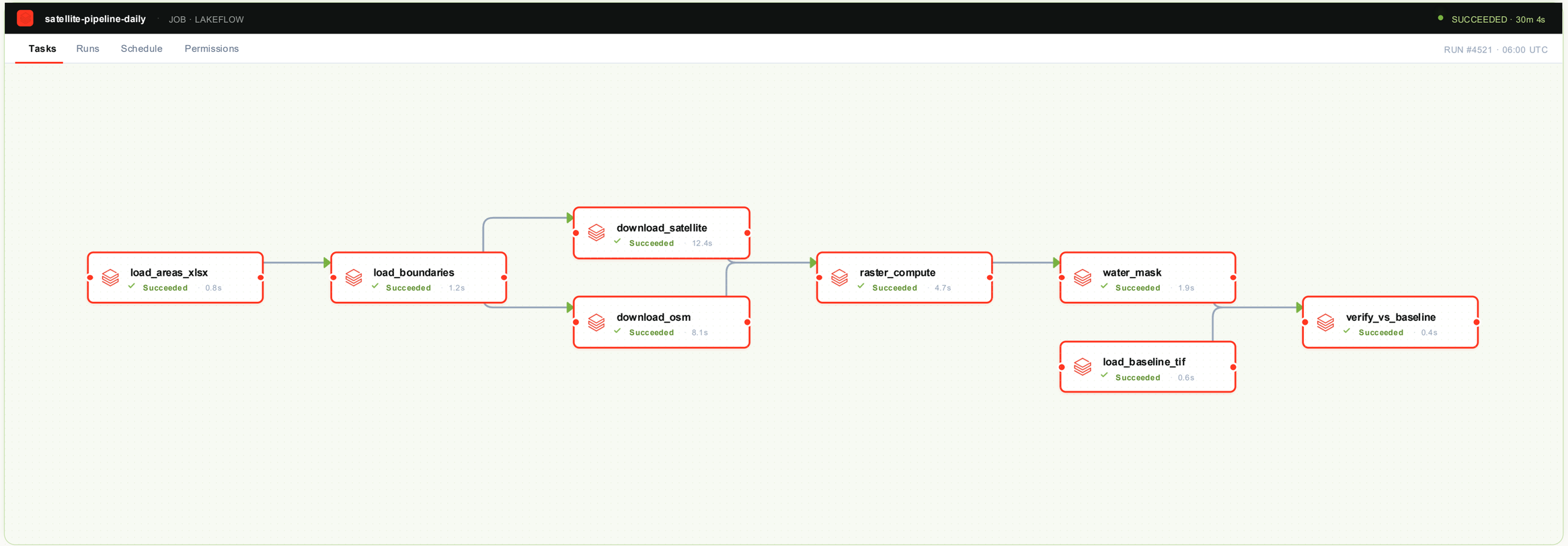


Worked example · Migrated

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.

Part 01

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

The old way

Six stages.Every one a dead end.

01	02	03	04	05	06
~2 WK	~1 WK	2–4 WK	1–2 WK	~2 WK	WEEKS
Finder	**Platform**	**Code**	**Deploy**	**Verify**	**Test**
Reverse-engineer	Pick cloud + spatial	Translate by hand	Orchestration hell	"Looks right"	N × N cases

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	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.

01	02	03	04	05	06
~2 WK	~1 WK	2–4 WK	1–2 WK	~2 WK	WEEKS
Finder	**Platform**	**Code**	**Deploy**	**Verify**	**Test**
Reverse-engineer	Pick cloud + spatial	Translate by hand	Orchestration hell	"Looks right"	N × N cases

Stage 02 · Platform

Pick cloud + spatial library

CONSULTANT · ~1 WK

PLATFORM COMPARISON

CAPABILITY	DATABRICKS	SNOWFLAKE	AWS
Vector ops	Mosaic	Ï Ì ŁÅÕMŘŒÑ	Sedona on EMR
Raster	Mosaic + GDAL	No native	GDAL on Lambda
CRS handling	Partial	Ï MØŘŒVÕ	Proj4 DIY
ArcPy compat	None	None	None
Cost model	DBU	F ØÑÑŒCE	Per-service

The old way

Six stages.Every one a dead end.

01	02	03	04	05	06
~2 WK	~1 WK	2–4 WK	1–2 WK	~2 WK	WEEKS
Finder	**Platform**	**Code**	**Deploy**	**Verify**	**Test**
Reverse-engineer	Pick cloud + spatial	Translate by hand	Orchestration hell	"Looks right"	N × N cases

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.

01 · ~2 WK	02 · ~1 WK	03 · 2–4 WK	04 · 1–2 WK	05 · ~2 WK	06 · WEEKS
Finder	**Platform**	**Code**	**Deploy**	**Verify**	**Test**
Reverse-engineer	Pick cloud + spatial	Translate by hand	Orchestration hell	"Looks right"	N × N cases

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 PǾMŌCĘNŎǾÖ ŁŇMPMBŐŘĘÇĐ
⚠ 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.

01	02	03	04	05	06
~2 WK	~1 WK	2–4 WK	1–2 WK	~2 WK	WEEKS
Finder	**Platform**	**Code**	**Deploy**	**Verify**	**Test**
Reverse-engineer	Pick cloud + spatial	Translate by hand	Orchestration hell	"Looks right"	N × N cases

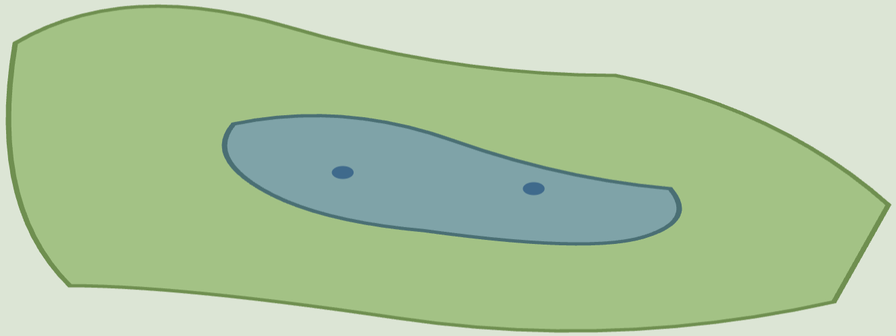
Stage 05 · Verify

"Looks right, ship it"

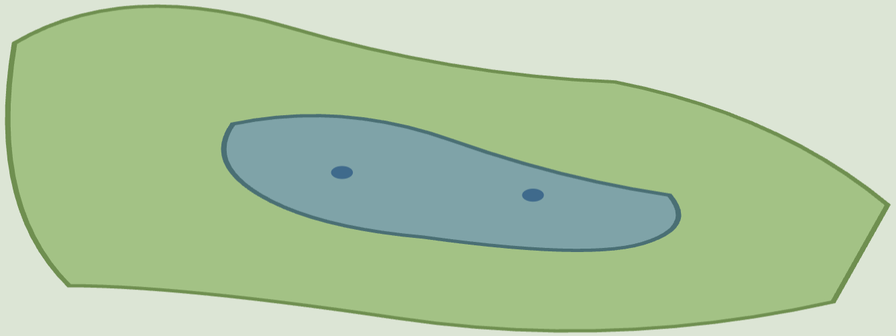
ANALYST · ~2 WK

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.

01	02	03	04	05	06
~2 WK	~1 WK	2–4 WK	1–2 WK	~2 WK	WEEKS
Finder	**Platform**	**Code**	**Deploy**	**Verify**	**Test**
Reverse-engineer	Pick cloud + spatial	Translate by hand	Orchestration hell	"Looks right"	N × N cases

Stage 06 · Test

N × N parametric cases

ANALYST · WEEKS

TEST · PARAMETRIC COVERAGE

COUNTRIES · 12

US	CA	[illegible]	BR
AR	UK	FR	DE
IT	ES	[illegible]	AU

SCENARIOS · 3

BASELINE	+2°C	+4°C

RETURN PERIODS · 5

10-year flood	[illegible]
25-year flood	RUN
50-year flood	RUN
100-year flood	RUN
250-year flood	[illegible]

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.

Part 02

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.

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

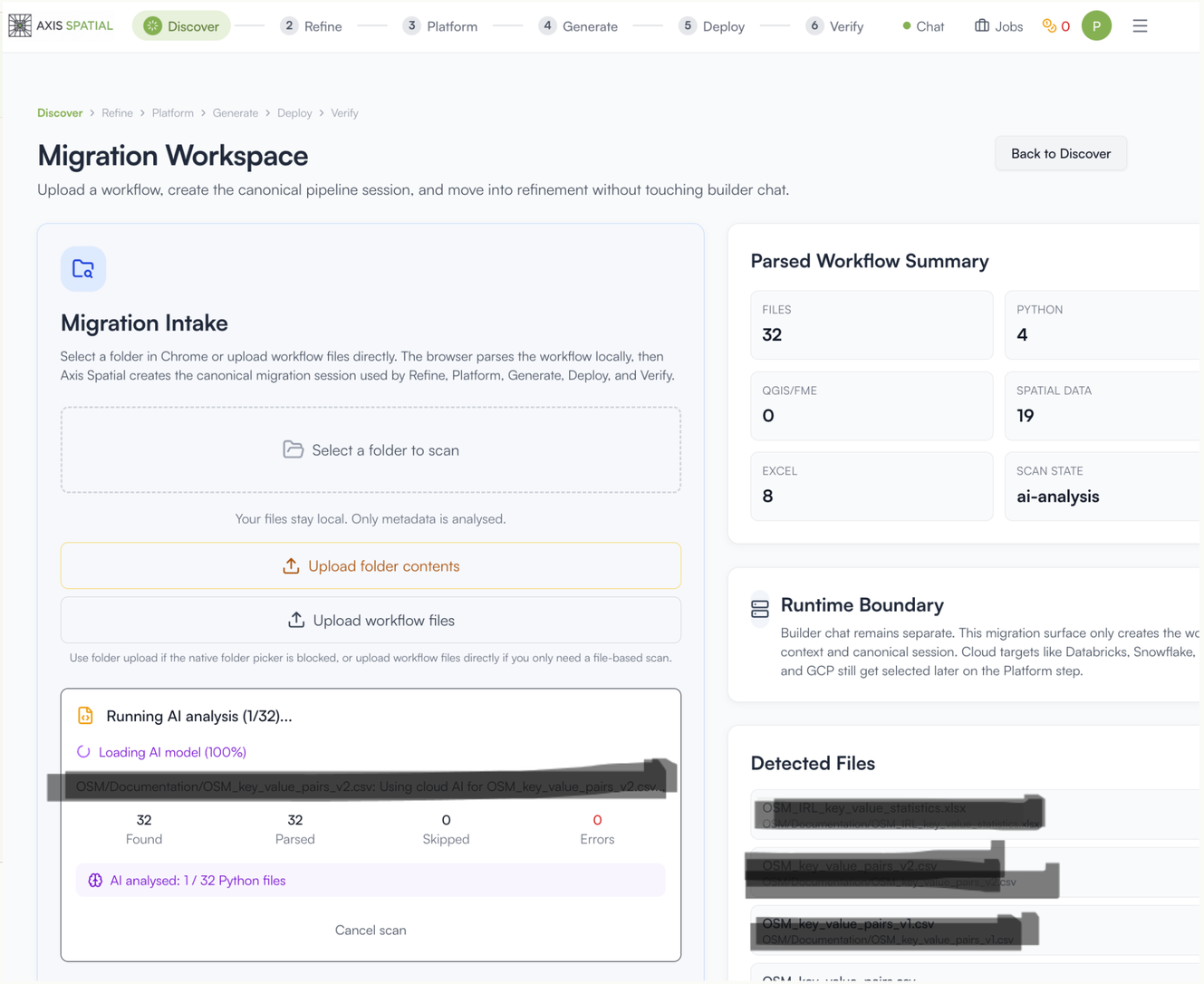
05 **Deploy**
Auto-heal on failure

06 **Verify**
Artifact-by-artifact diff

07 **Learn**
Compounds in your tenant

Stage 01 · Finder

Auto-scan scripts & files



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

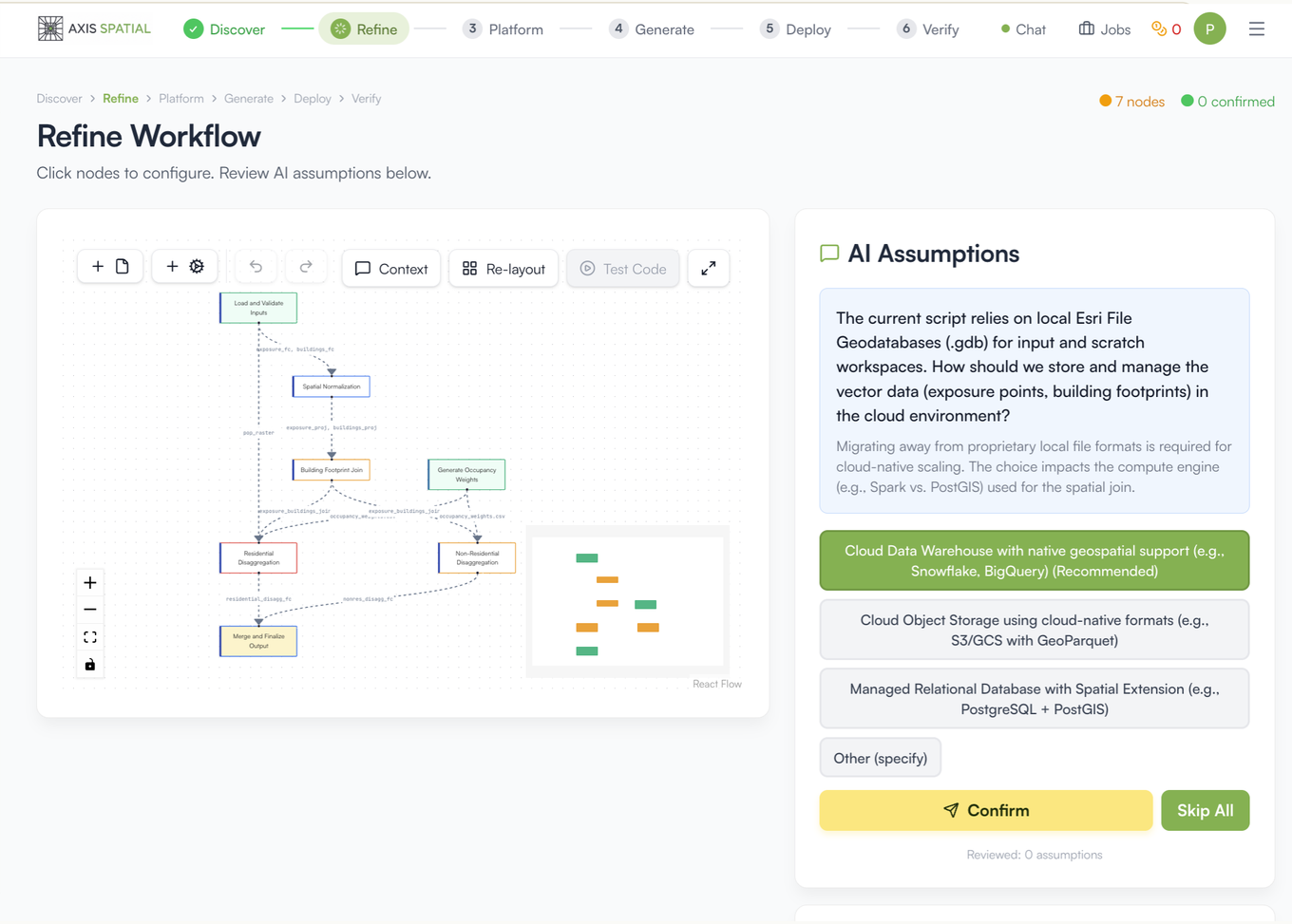
05 Deploy
Auto-heal on failure

06 Verify
Artifact-by-artifact diff

07 Learn
Compounds in your tenant

Stage 02 · Refine

The one human checkpoint



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

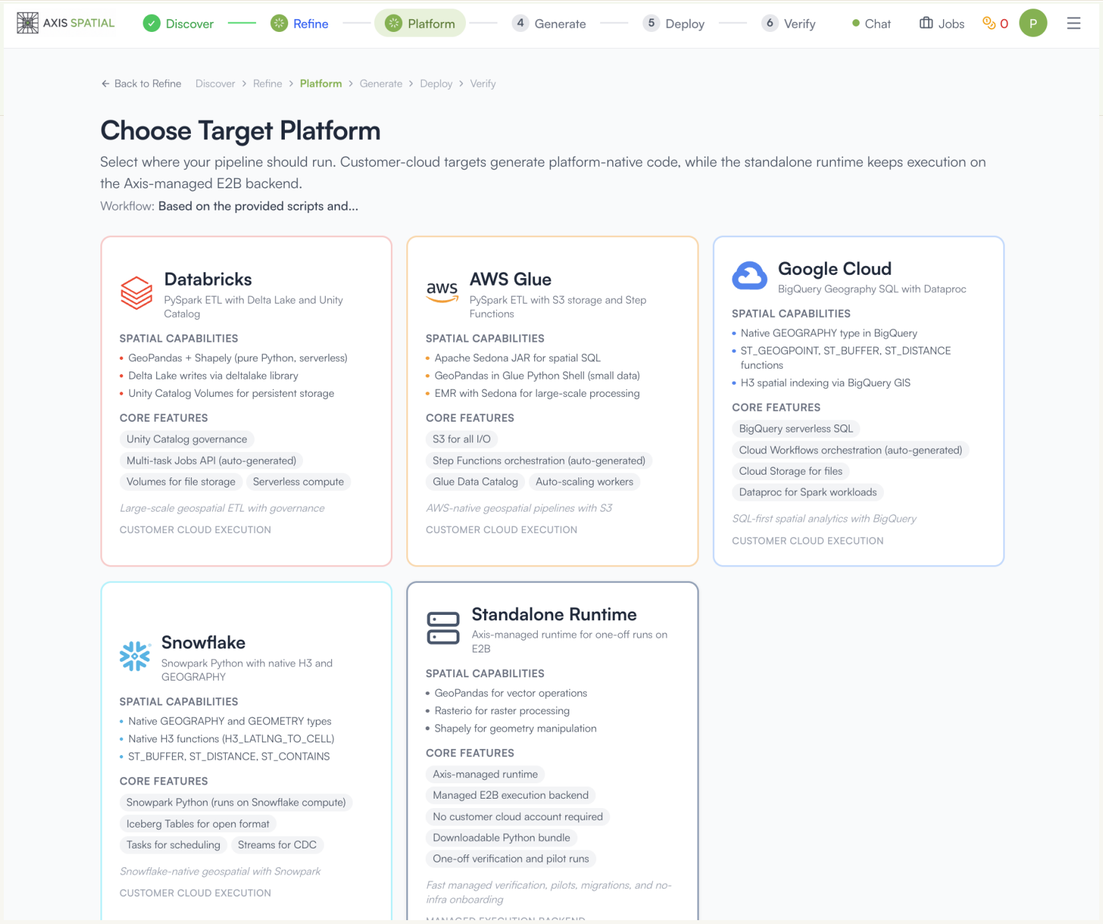
06 **Verify**
Artifact-by-artifact diff

07 **Learn**
Compounds in your tenant

Stage 03 · Platform

Pick target platform

AUTOMATED



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

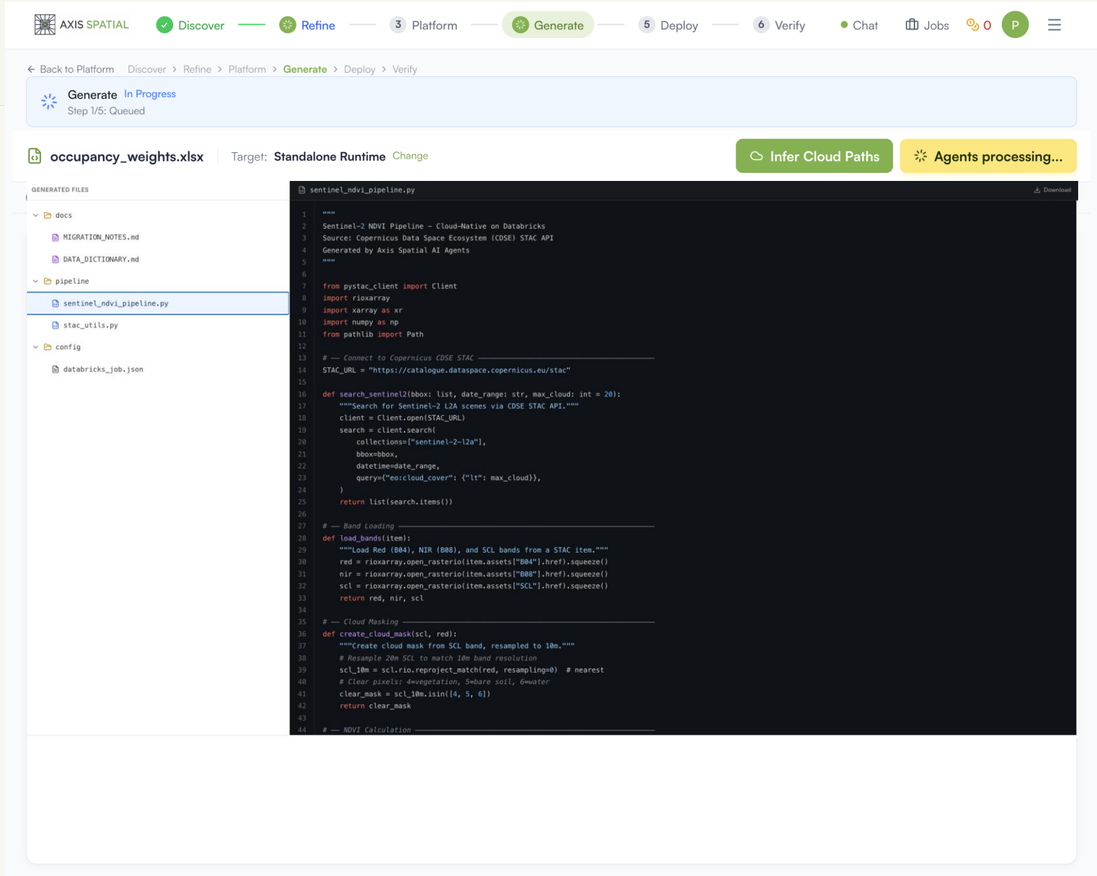
05 **Deploy**
Auto-heal on failure

06 **Verify**
Artifact-by-artifact diff

07 **Learn**
Compounds in your tenant

Stage 04 · Code

Builder · Validator · Reviewer



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

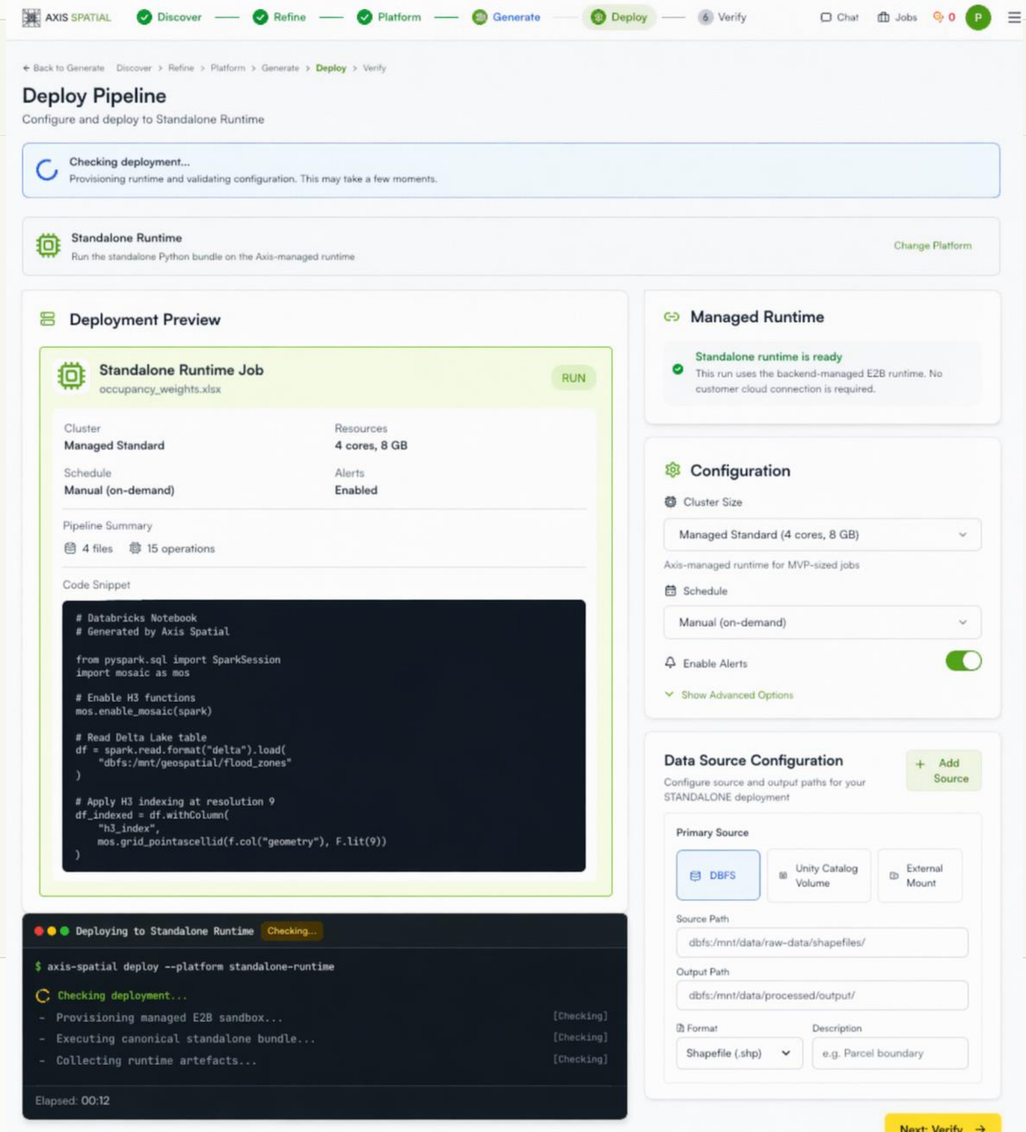
06 **Verify**
Artifact-by-artifact diff

07 **Learn**
Compounds in your tenant

Stage 05 · Deploy

Auto-heal on failure

AUTOMATED



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

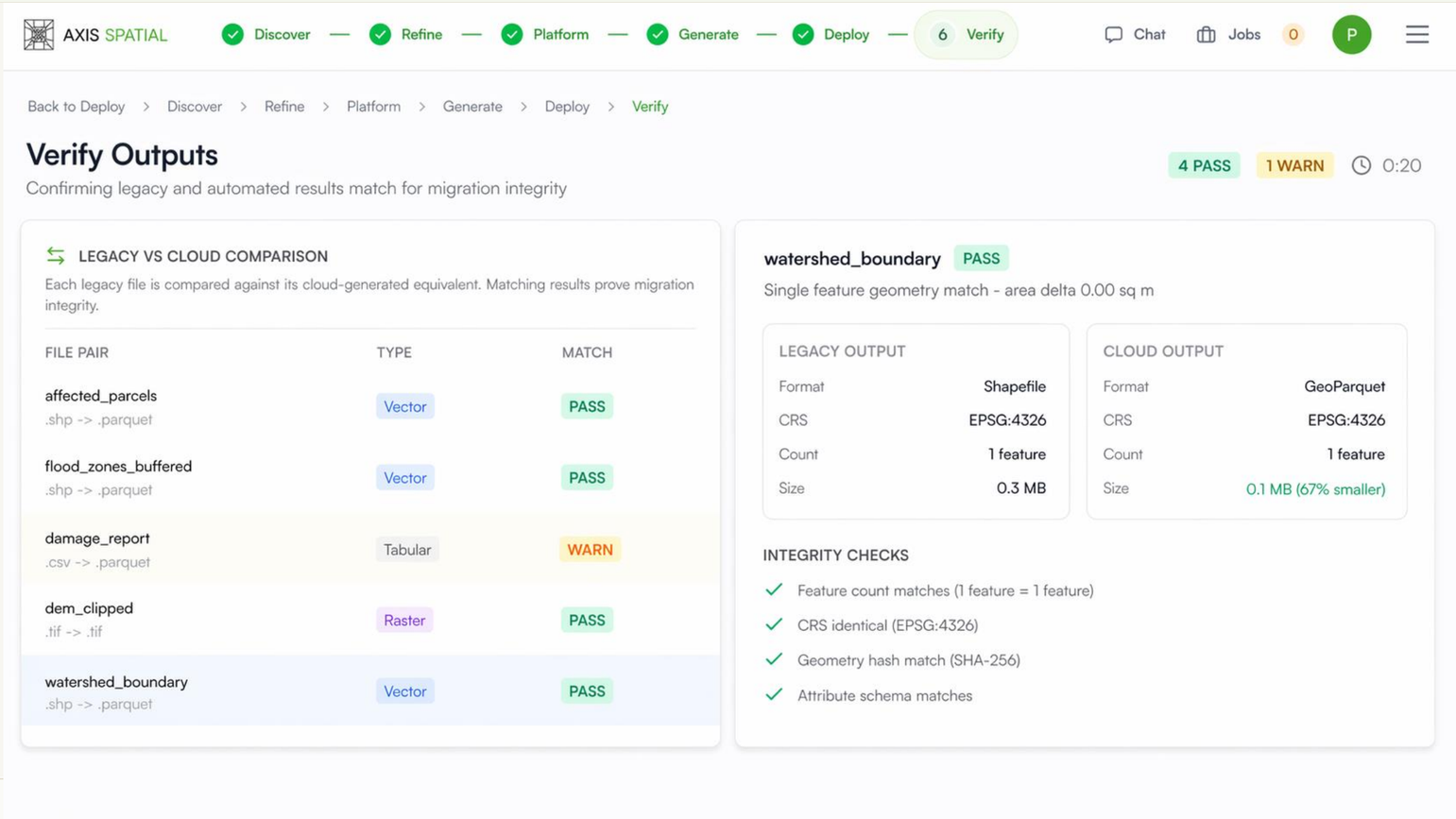
06 **Verify**
Artifact-by-artifact diff

07 **Learn**
Compounds in your tenant

Stage 06 · Verify

Artifact-by-artifact diff

AUTOMATED

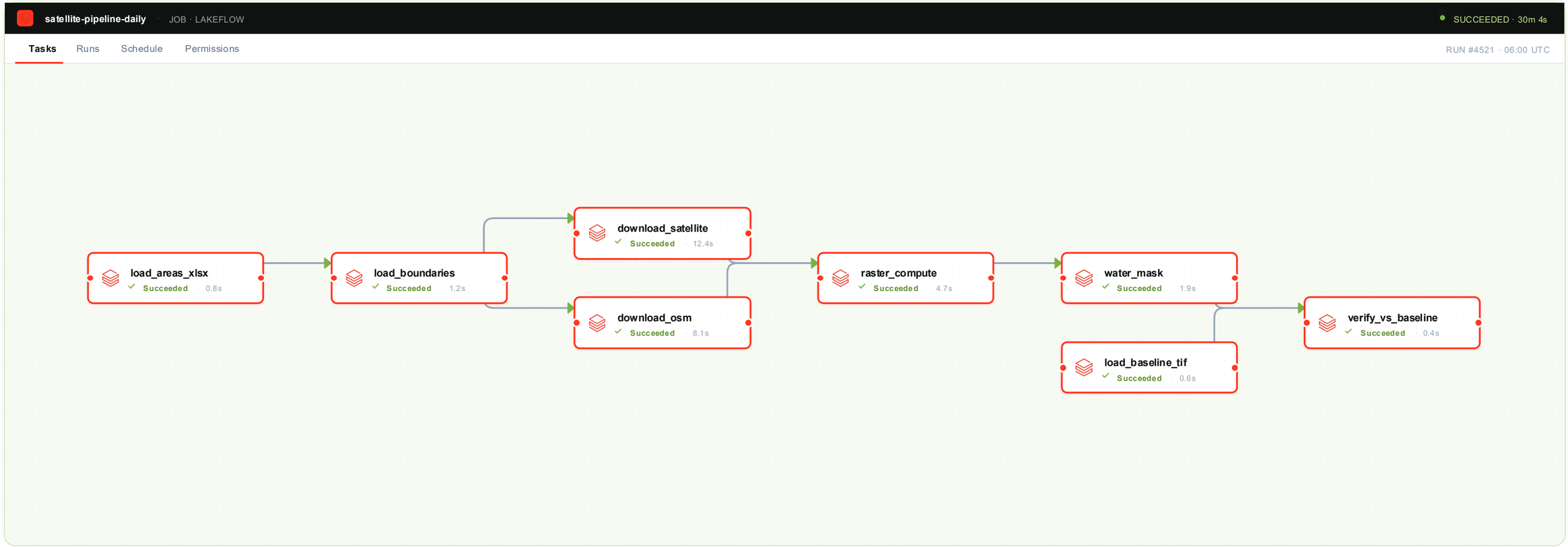


Worked example · Migrated

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

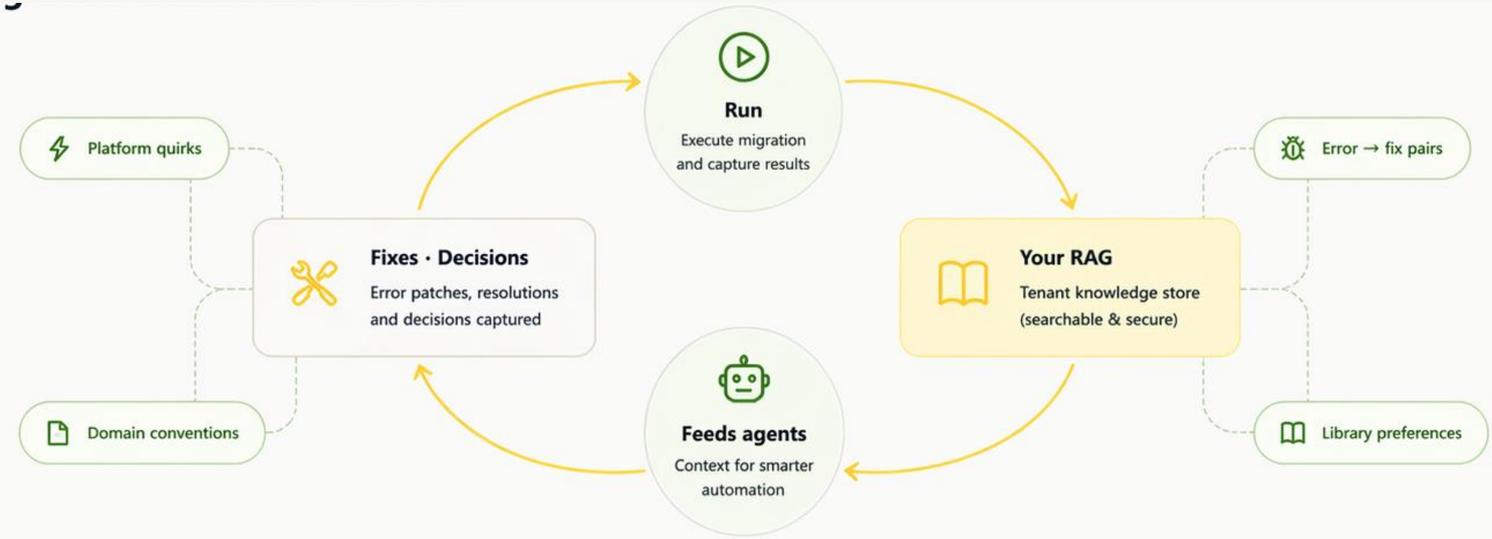
06 **Verify**
Artifact-by-artifact diff

07 **Learn**
Compounds in your tenant

Stage 07 · Learn

Compounds in your tenant

AUTOMATED



Logs, decisions, fixes, written to your knowledge system. Inside your tenant.
Next migration is faster and more accurate. The engine learns your shop.

Add it up

≈ 30 min.

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

~30 min	1	100%
ENGINE RUNTIME	HUMAN CHECKPOINT	VERIFIED OUTPUT

What comes out

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

The old way

~~3–4 wk~~

Consultant estimate

→

The Migration Engine

~30 min

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.

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.

Part 03

Beyond migration. The agent layer for geospatial teams.

Beyond migration

Migration is where Axis Spatial started.

The same agents power every geospatial workflow.

Input		Axis Spatial agents		Output
Legacy workflow	→	Migration Engine SHIPPING TODAY	→	Cloud pipeline
Blank page	→	Pipeline Builder SHIPPING TODAY	→	Cloud pipeline
Your domain	→	Your product WHAT YOU BUILD	→	You tell us

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

Why Axis Spatial

Why us.

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