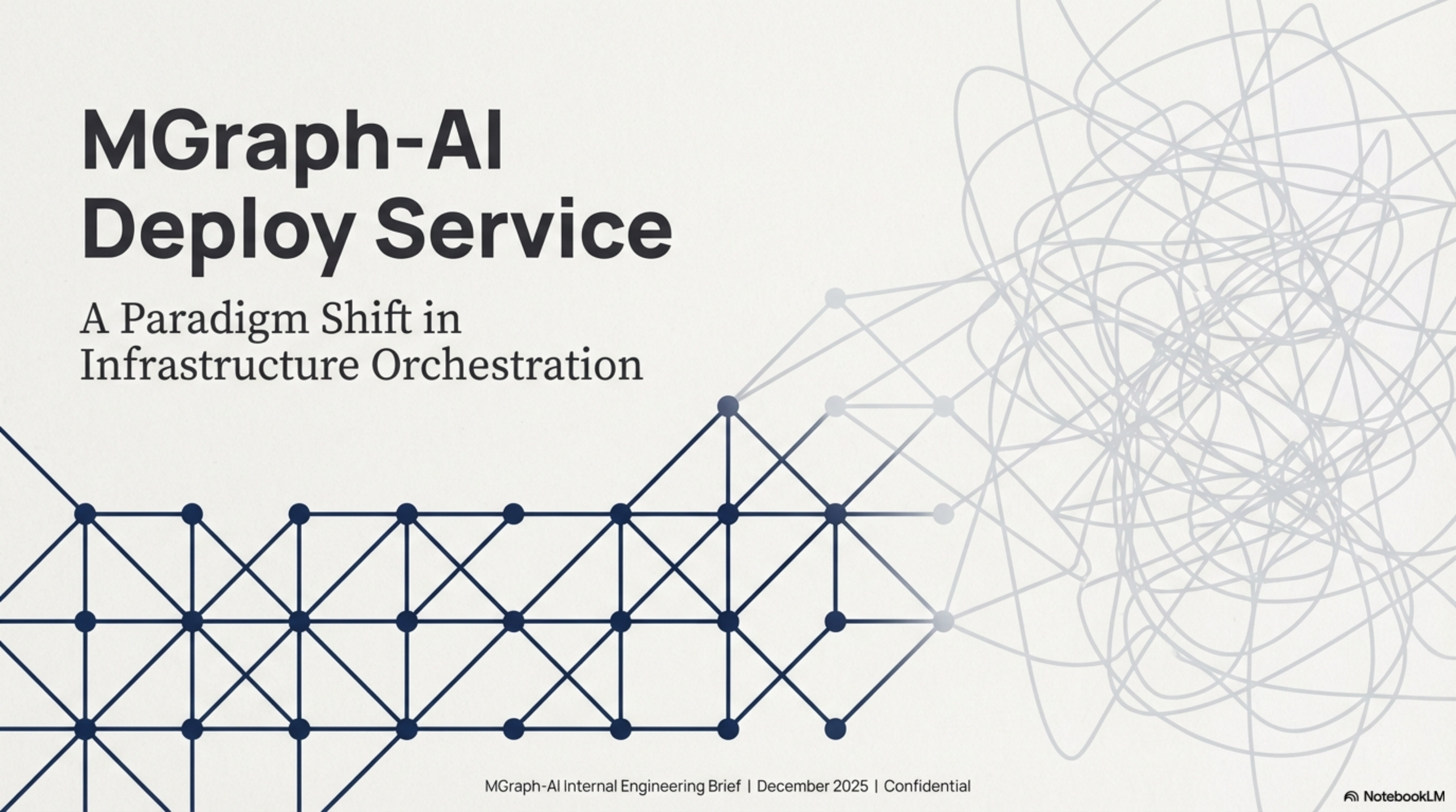
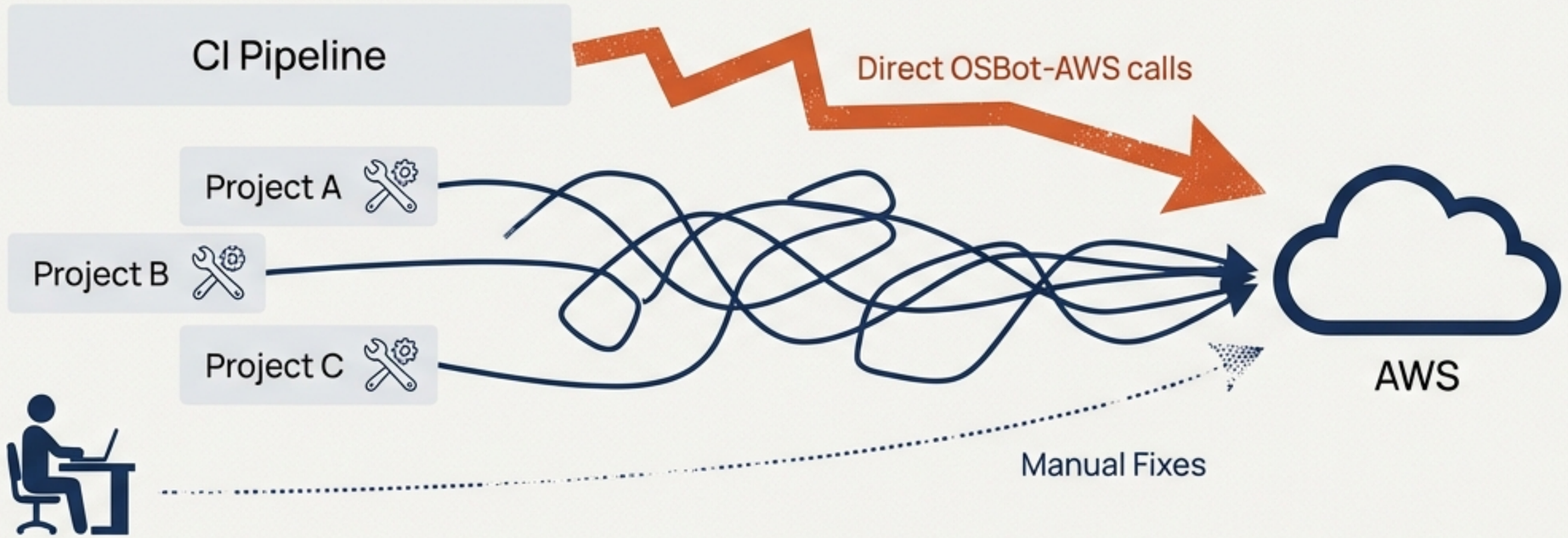
MGraph-AI Deploy Service

A Paradigm Shift in Infrastructure Orchestration



Our current infrastructure deployment is fragmented and fragile.

We rely on a mix of CI pipeline calls, scattered helper methods, and manual interventions. This creates significant challenges.



No Unified State: Changes are untracked and often conflict.

No Verification Layer: We deploy blind, hoping for the best.

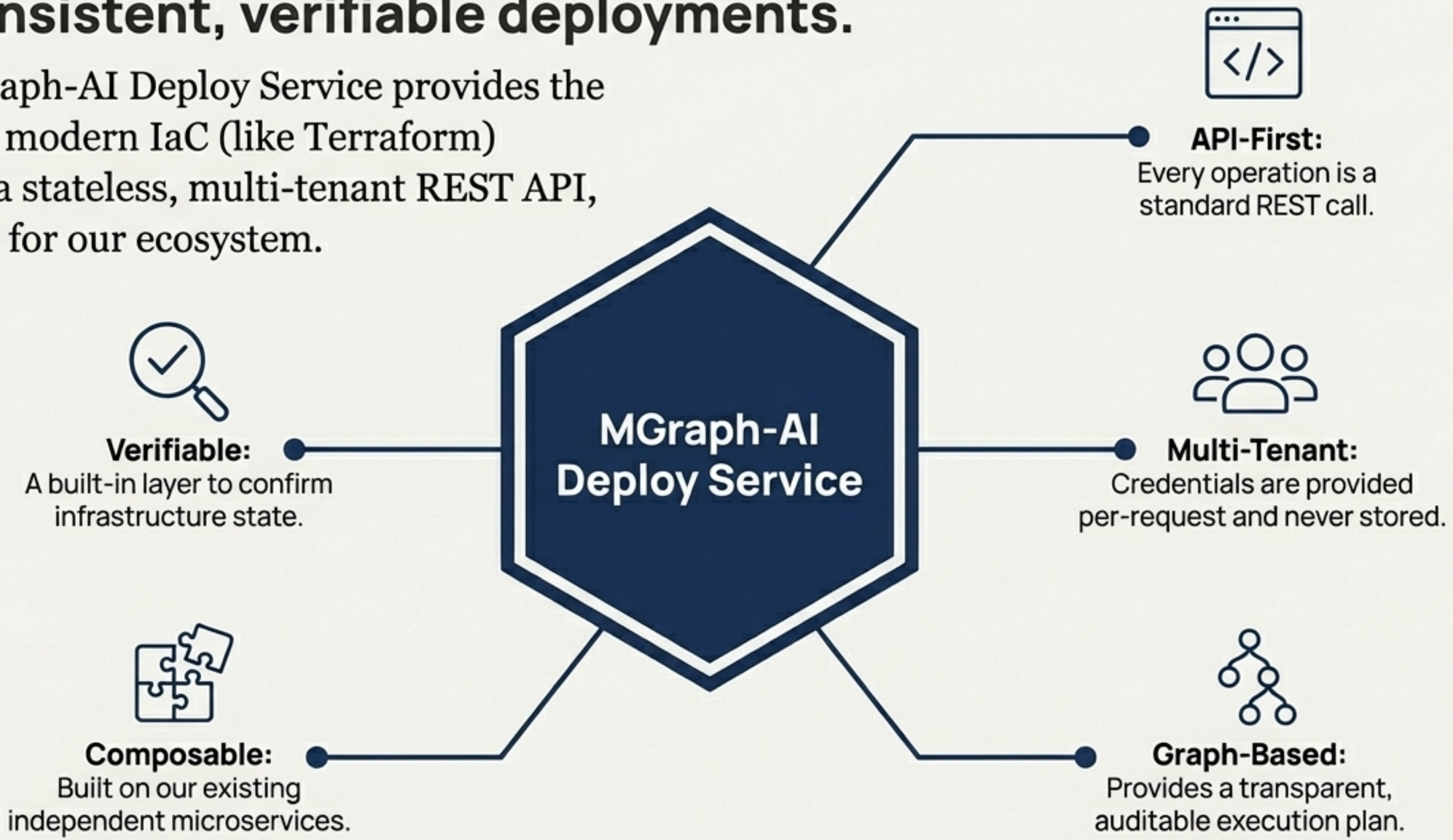
Difficult to Reproduce: Deployments are inconsistent across environments.

No Audit Trail: We cannot reliably answer "who deployed what, and when?"

High Migration Cost: Moving resources between regions is a manual, error-prone project.

We are building an API-driven orchestrator for consistent, verifiable deployments.

The MGraph-AI Deploy Service provides the power of modern IaC (like Terraform) through a stateless, multi-tenant REST API, designed for our ecosystem.



The service provides a complete lifecycle toolkit for infrastructure management.

Plan

Preview changes without execution (dry run)

Deploy

Create new resources across AWS and GitHub

Update

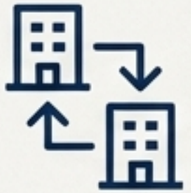
Modify existing resources efficiently

Verify

Test that infrastructure matches the expected state

Delete

Remove resources and their dependencies cleanly



Move

Migrate resources between regions or accounts

Resume

Continue a deployment from the point of failure

Step-Through

Execute one operation at a time for debugging

Multi-Region

Deploy identical resources to multiple regions simultaneously

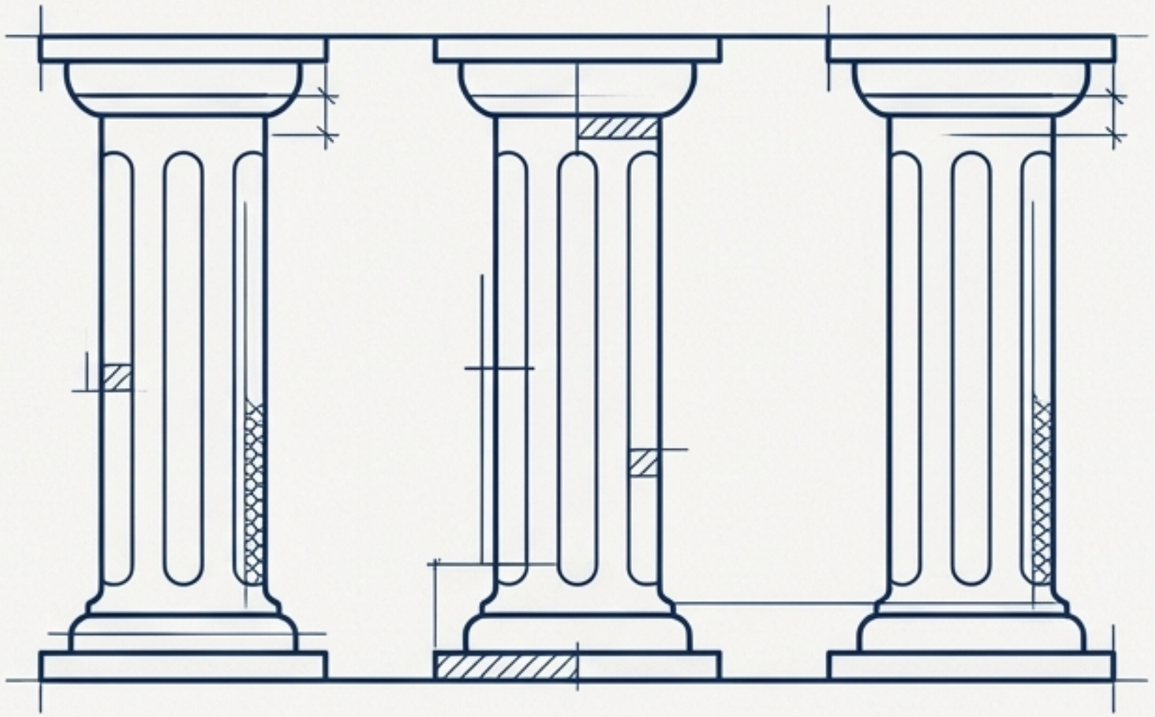
Three core principles make this a fundamentally different approach.

Our design is not an iteration on existing tools; it's a re-imagining of how infrastructure state and operations should be managed. Understanding these three decisions is key to understanding the service.

1. The Execution Graph IS the State

2. Resources Discover Each Other via Tags

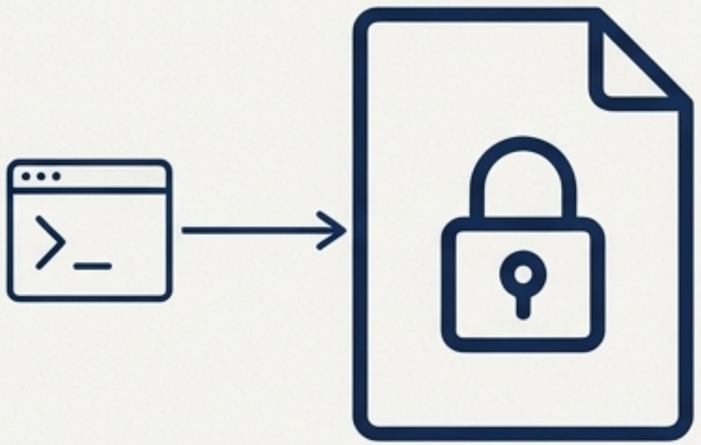
3. Orchestration is Separate from Implementation



Principle 1: The Execution Graph IS the State.

We eliminate the concept of a separate, opaque ‘state file’. The **MGraph execution graph**, which is inherently auditable and versioned, becomes the single source of truth for the deployment’s history and status.

The Old Way



terraform.tfstate

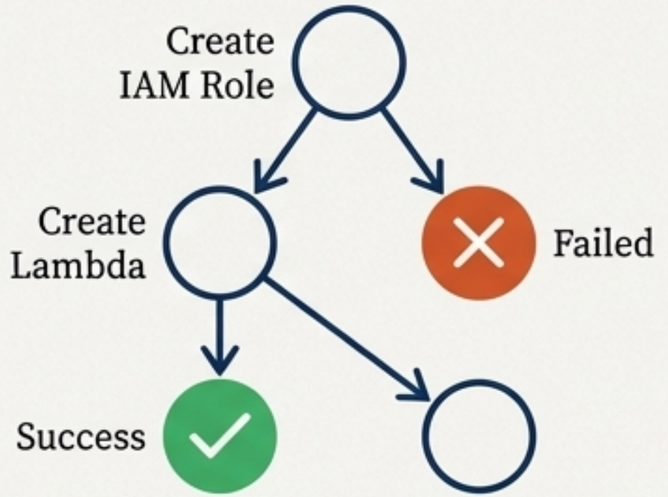
Opaque

Localised

Prone to drift

The New Way

MGraph Execution Graph



Transparent Audit Trail

Resumable from Failure

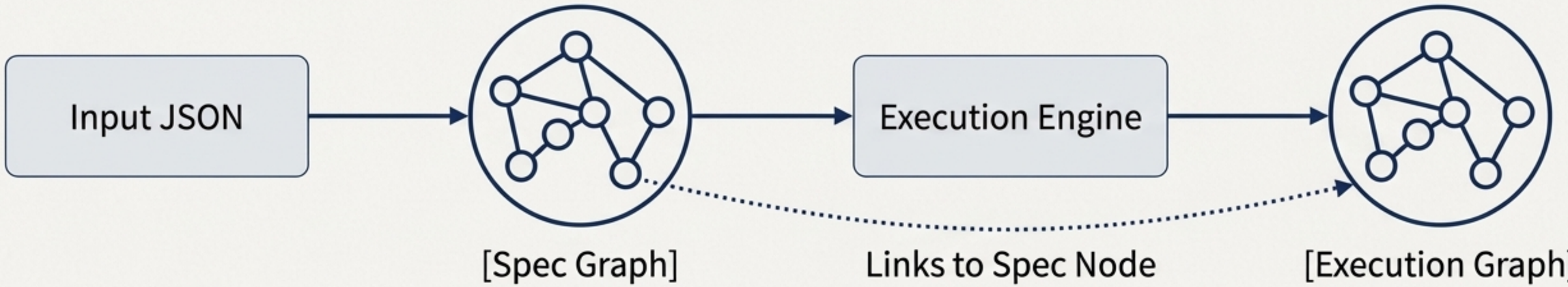
Inherent Parallelism

Explicit Dependencies

We use two distinct but linked graphs: one for the plan, one for the result.

This separation provides a clear distinction between the immutable deployment specification and the mutable record of what actually occurred during execution.

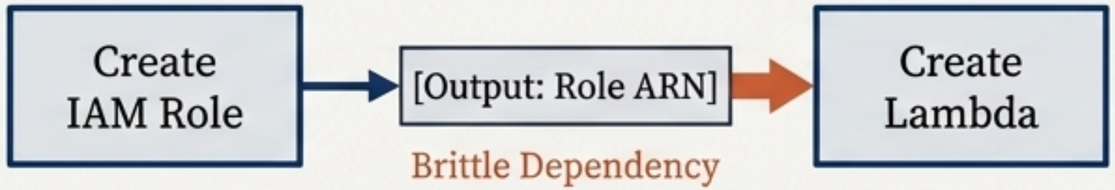
Aspect	Spec Graph	Execution Graph
Purpose	What to deploy	What happened during deployment
Mutability	Immutable once created	Updated live during execution
Contents	Resource definitions, properties	Status (Success, Fail), timing, outputs, errors



Principle 2: Resources Discover Each Other via Tags.

Operations are designed to be atomic and independent. Instead of passing resource IDs (like ARNs) from one step to the next, subsequent operations find their dependencies by querying for resources with specific, predictable tags.

Tightly Coupled



Decoupled Discovery

- ✓ Keeps operations truly atomic and independent.
- ✓ Enables state to be reconstructed by simply scanning existing infrastructure.
- ✓ Works seamlessly for multi-region and verification workflows.

We embed the graph's identity directly into the infrastructure itself.

Every resource created in AWS receives a standard set of tags linking it back to the exact graph node that created it. For services without tags, like GitHub, we use a consistent naming convention.

AWS Standard Tags

```
"Tags": {
  "mgraph:graph-id": "65b..
  "mgraph:node-id": "65b...
  "mgraph:edge-id": "65b...
}
```

This provides a complete audit trail directly on the resource.

GitHub Naming Convention

Resource:
`Repository Secret`

Pattern:
`{secret_name}__{node-id}`

The unique `node-id` ensures no name collisions.

S3 Global Namespace

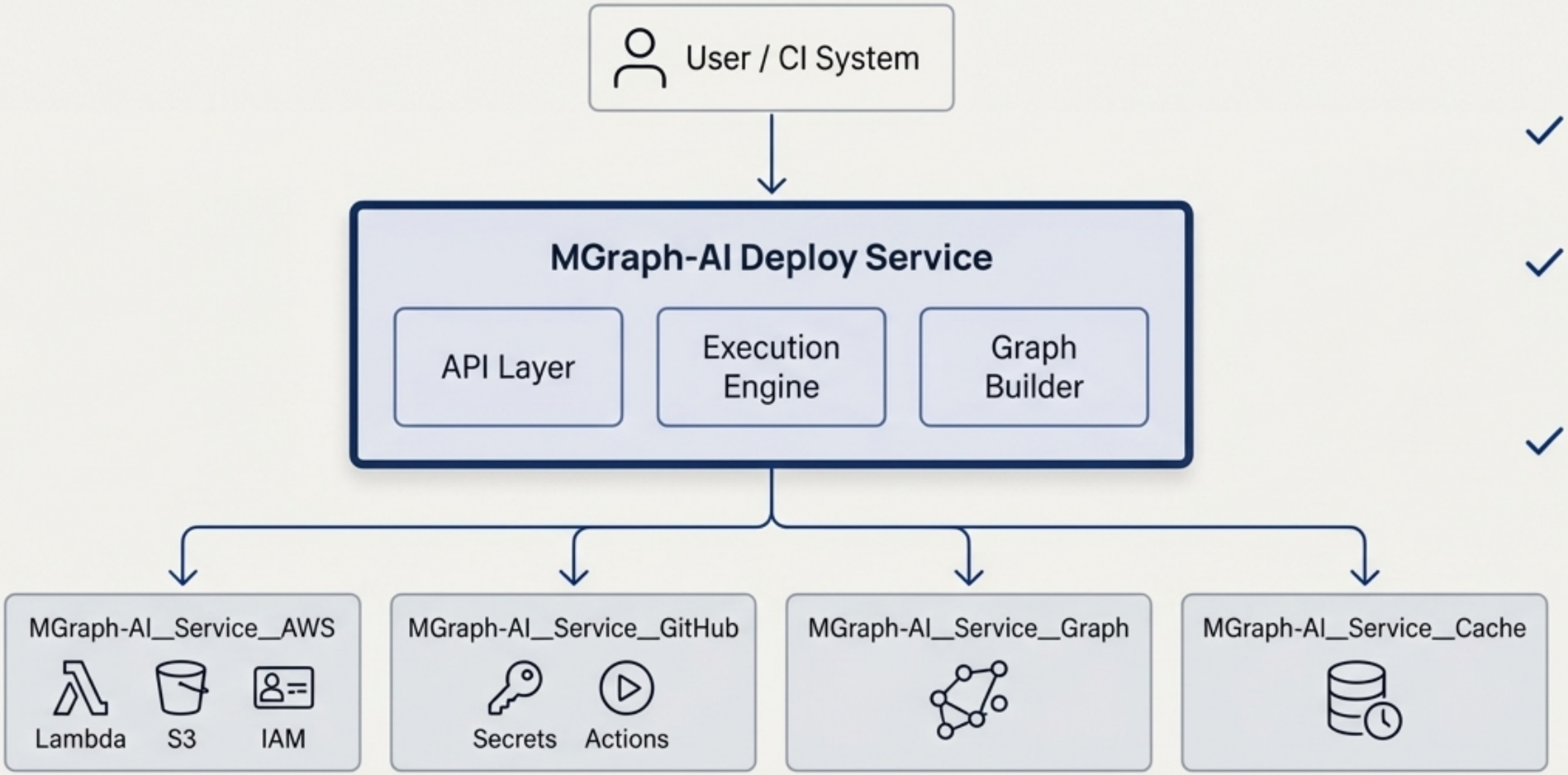
Resource:
`S3 Bucket`

Pattern:
`{bucket-name}-{node-id}`

Appending the unique `node-id` solves for the globally unique bucket name requirement.

Principle 3: Orchestration is Separate from Implementation

The Deploy Service is a pure orchestrator. It focuses entirely on executing the graph logic—determining what runs when. The actual work of interacting with cloud providers is delegated to specialised "sister services".



- ✓ Clean separation of concerns.
- ✓ Enables provider services to be tested and used independently.
- ✓ Simplifies maintenance and extension of the Deploy Service.

A typical deployment follows a simple, three-step API workflow.

The main visual is a clean, horizontal flow diagram with three numbered stages, moving from left to right.

1

PLAN: `POST /deploy/plan`

Action: User submits a declarative JSON configuration.

Service Response: Validates the config, builds the Spec and Execution graphs, and returns a detailed execution plan for review. This is the “dry run”.

2

APPLY: `POST /deploy/apply`

Action: User submits the `graph_id` from the plan phase.

Service Response: The Execution Engine begins processing the graph, calling provider services to create real infrastructure. Returns an immediate acknowledgement.

3

MONITOR:

`GET /deploy/status/{graph_id}`

Action: User polls this endpoint with the `graph_id`.

Service Response: Returns the real-time status of the Execution Graph, showing completed, failed, and in-progress nodes.

The engine provides flexible execution modes and a clear storage strategy.

This slide details the operational flexibility and secure data handling within the engine.

Execution Modes

	Mode	Description	Use Case
	**FULL**	Execute all operations in parallel where possible.	Production deployments
	**STEP**	Execute one operation, then wait for a continue call.	Debugging & manual review
	**DRY_RUN**	Validate config and build plan; no execution.	Previewing changes

Storage Strategy

Data Type	Storage Service(s)	Rationale
Spec & Execution Graphs	Graph Service & Cache Service	Dual storage for different access patterns
Execution Logs & Outputs	Cache Service	Persistent, per-node audit and data
Deployment Metadata	Cache Service	Quick lookup of deployments
Credentials	In-Memory Only	**Never persisted to disk**
Execution State	Cache Service	Transient state for active executions

We will deliver the service incrementally over six distinct phases.

The main visual is a high-level horizontal timeline graphic.

Phase 1 (Wk 1-2): Foundation

Project setup, schemas, and basic health routes.

Phase 2 (Wk 3-4): Graph Building

Convert input JSON into a persisted Spec Graph.

Phase 3 (Wk 5-6): Execution Engine

Implement `FULL`, `STEP`, and `DRY_RUN` modes and the `/apply` route.

Phase 4 (Wk 7-8): Operations Library

Build out the full set of AWS & GitHub operation primitives.

Phase 5 (Wk 9-10): Advanced Features

Implement Pause/Resume, Destroy, and Move capabilities.

Phase 6 (Wk 11-12): Testing & Docs

Achieve production quality with comprehensive test suites and documentation.

Success is defined by these 14 measurable outcomes.

The implementation will be considered complete when all of the following criteria are met and verified.

Core API Lifecycle

- ✓ `/plan` validates and returns an execution plan.
- ✓ `/apply` creates resources across AWS and GitHub.
- ✓ `/verify` confirms infrastructure state.
- ✓ `/destroy` cleanly removes all resources.

Engine & State

- ✓ Tag-based discovery is fully functional.
- ✓ Spec & Execution graphs are correctly linked and persisted.
- ✓ `FULL`, `STEP`, `DRY_RUN` modes are working.
- ✓ Pause/Resume with `/continue` is operational.

Advanced Workflows

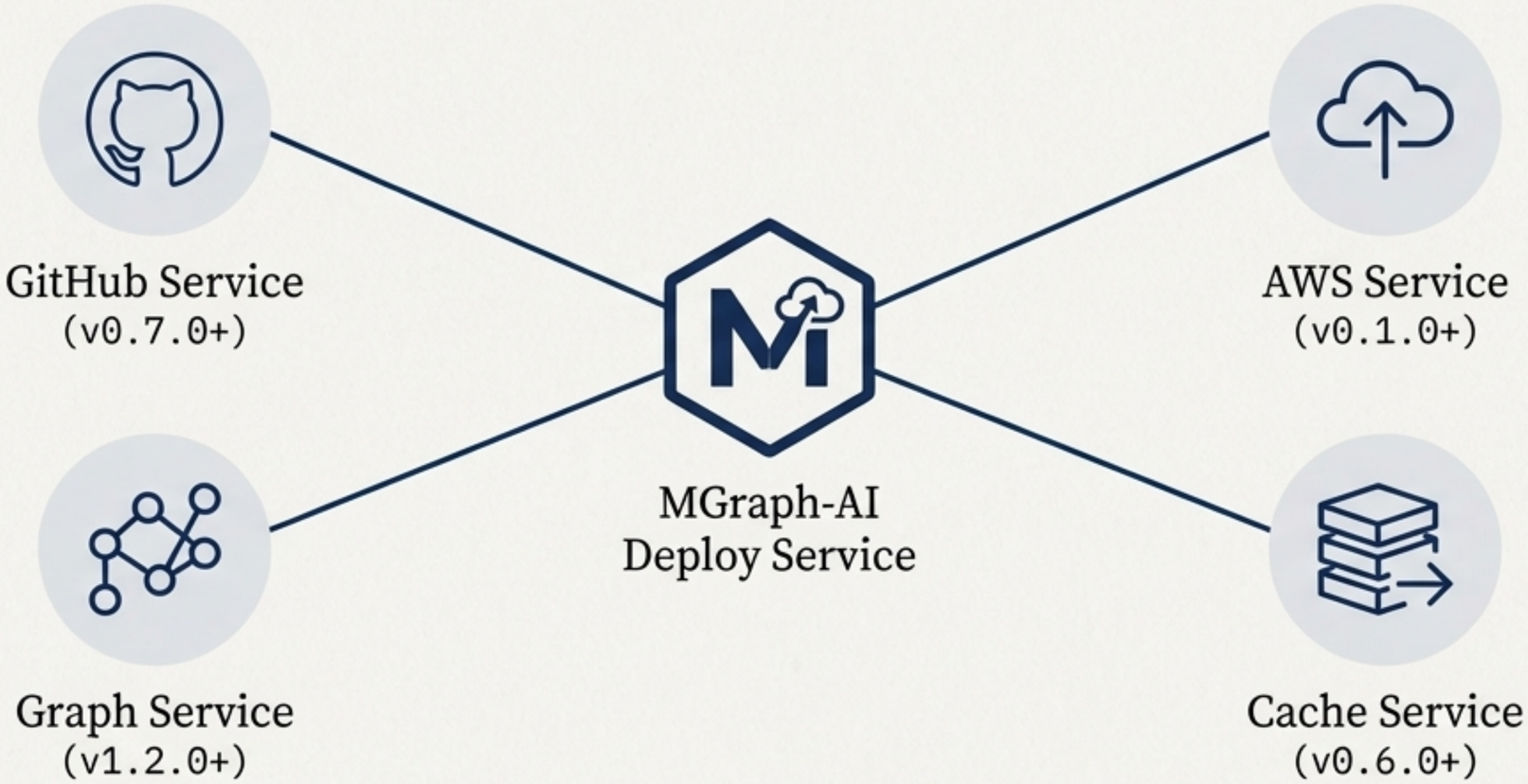
- ✓ Multi-region deployment is supported.

Quality & Readiness

- ✓ All state persisted with discoverable paths in Cache Service.
- ✓ Unit, Integration, and QA tests are all passing.
- ✓ OpenAPI documentation is complete and auto-generated.

The Deploy Service is a key orchestrator within the MGraph-AI ecosystem.

It leverages and enhances our existing service-oriented architecture, relying on these core dependencies to perform its work.



Target Repository: `the-cyber-boardroom/MGraph-AI__Service__Deploy`