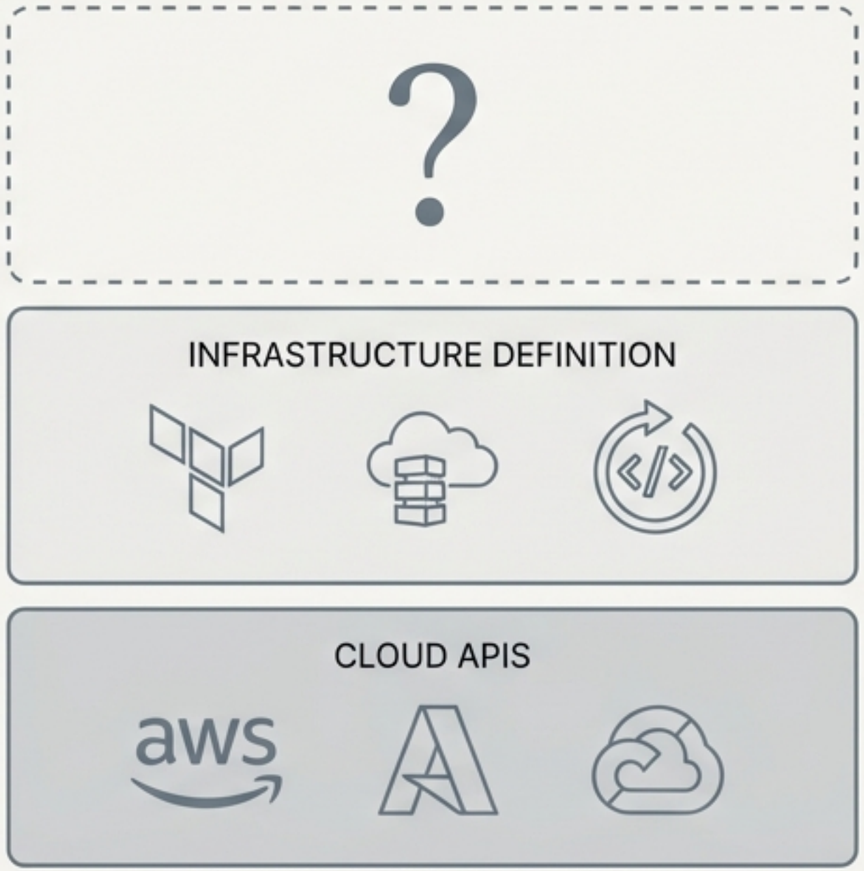
Your Infrastructure Stack is Missing a Layer

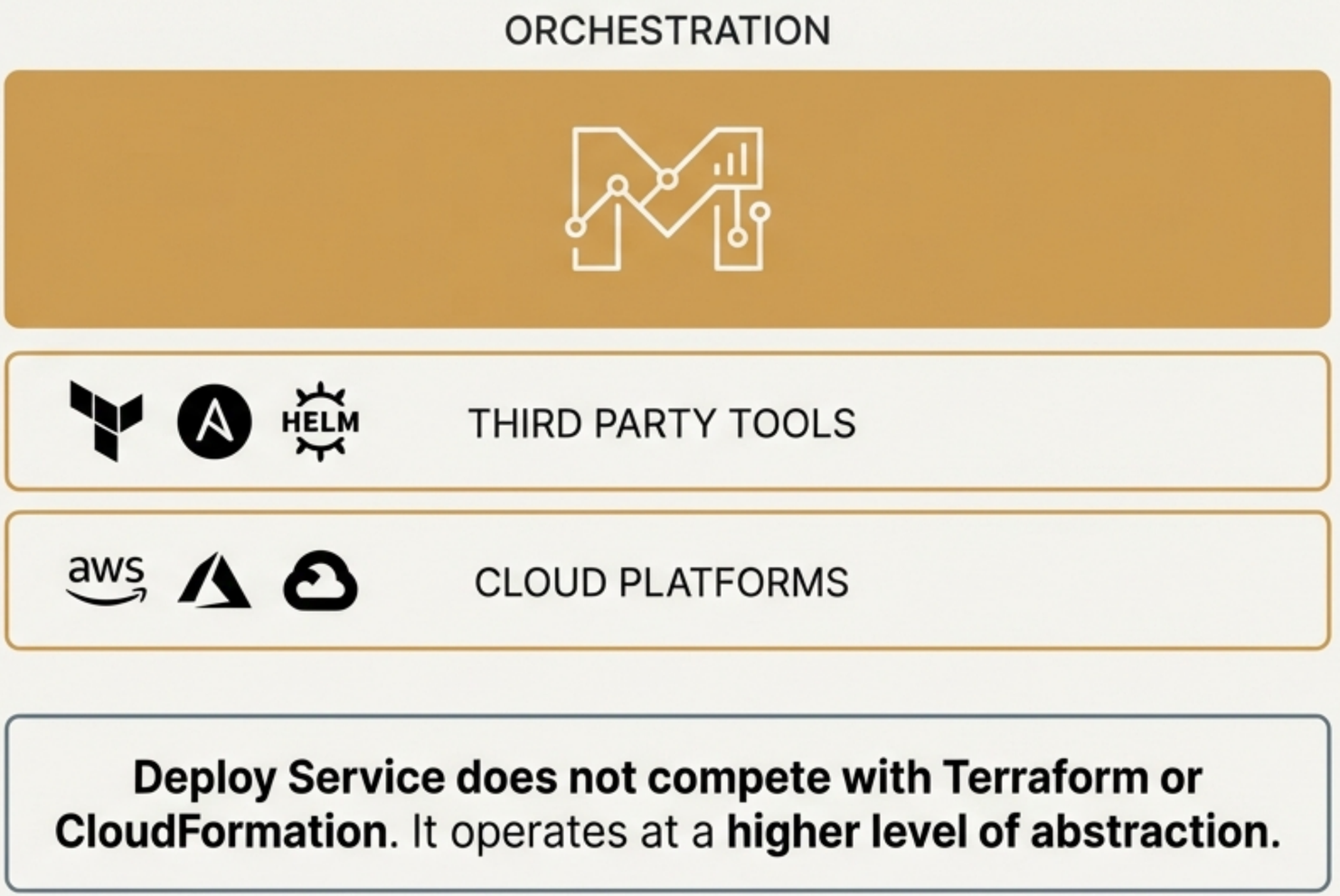
You have mastered defining infrastructure with code. But as systems grow, a new challenge emerges: orchestrating the complex workflows that deploy, connect, and verify that code.

This is the missing layer.



Deploy Service is the Orchestration Layer

- A **single API interface** to drive any deployment workflow.
- A **single, unified execution graph** for visibility across all tools.
- A **single audit trail and verification framework** for compliance and reliability.

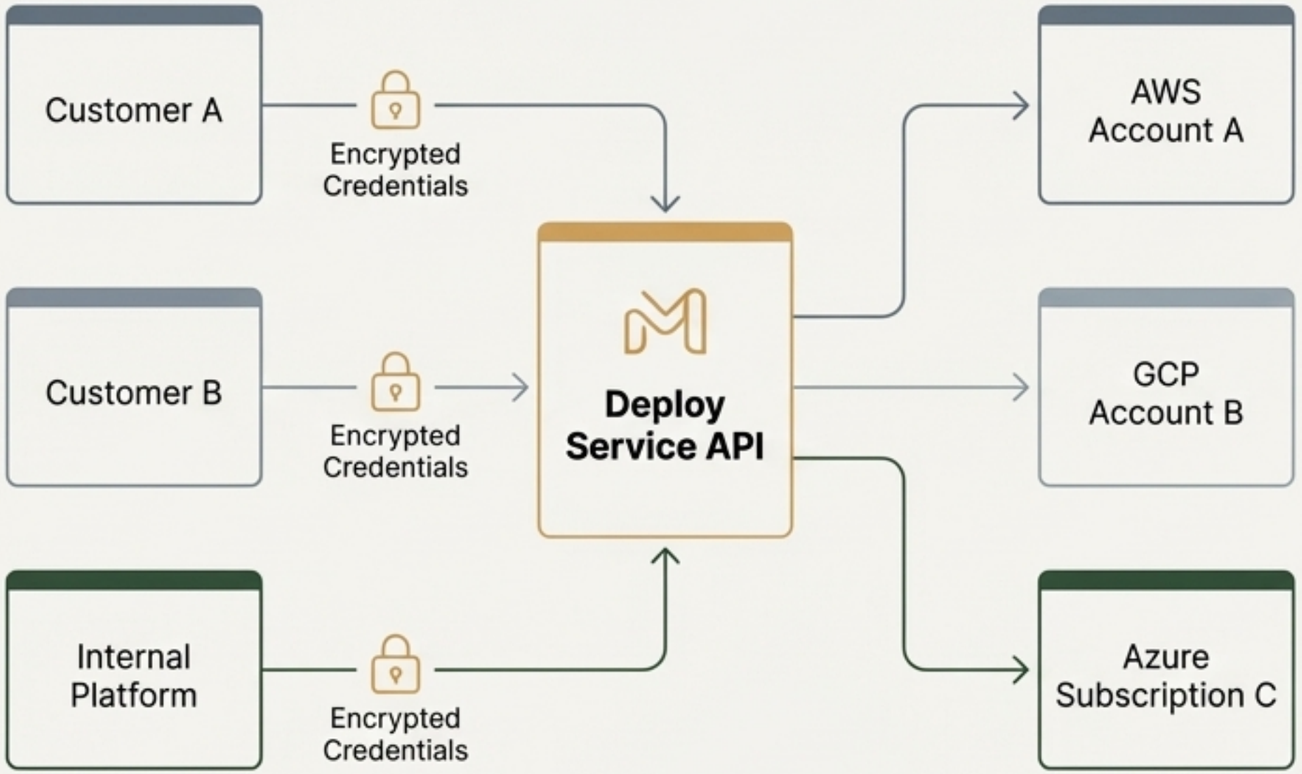


Deploy Service does not compete with Terraform or CloudFormation. It operates at a **higher level of abstraction.**

The API-First Architecture Unlocks New Capabilities

Capability	Traditional IaC (CLI-centric)	Deploy Service (API-first)
LLM/AI Agent Integration	Difficult; requires fragile shell wrappers	**Native**; direct API calls
ChatOps & Interfaces	Complex; requires custom backend integration	**Native**; a single, simple API call
Embedding in Applications	Complex; requires managing CLI binaries & state	**Native**; simple, stateless HTTP requests
No Client-Side Tooling	Requires CLI installation and configuration	**True**; access from any HTTP client

True Multi-Tenancy is a Native Feature, Not an Afterthought



Key Architectural Differentiators

Per-Request Credentials: Credentials are provided and encrypted with each API call, not stored in environment variables or local files.

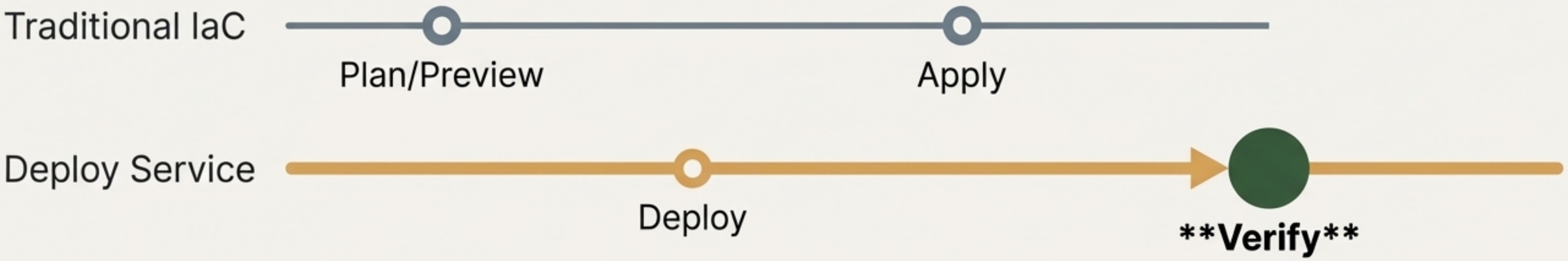
Native Tenant Isolation: The system is designed to handle requests for thousands of distinct tenants simultaneously and securely.

Stateless Execution: No local state files to manage or isolate between tenants. State is managed centrally in MGraph.

Ideal For

- Platforms deploying infrastructure on behalf of customers.
- Managed Service Providers (MSPs) managing multiple client accounts.
- SaaS applications provisioning customer-owned infrastructure.

Verification is a First-Class Citizen



Traditional Tools

Offer pre-flight checks (`terraform plan`, `pulumi preview`) or delayed, separate drift detection. They check the *_code_*, not the *_reality_* after deployment.

Deploy Service

Provides a dedicated `/verify` endpoint and per-operation verification. It answers the question: "Did the deployment achieve the desired state in reality?"

This Enables

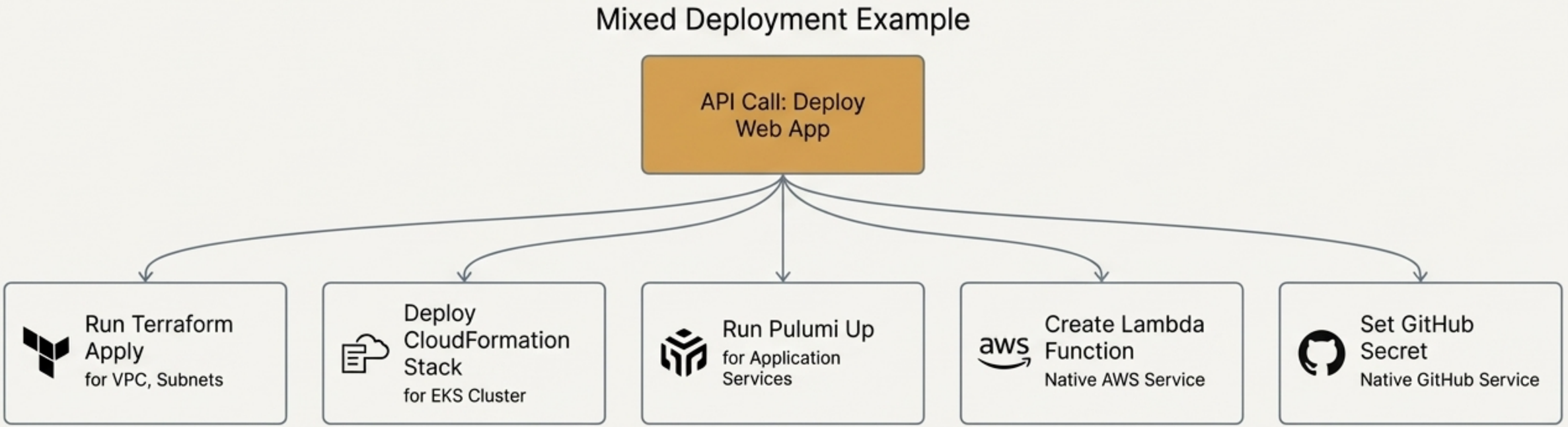
- Immediate, post-deployment proof that resources exist and match spec.
- Scheduled verification runs for continuous infrastructure health checks.
- Compliance evidence for auditors ("Show me that all production S3 buckets have logging enabled").

Built on a Foundation of Mature Ecosystems

We acknowledge the giants. It would take years to match the resource coverage of existing IaC tools, so we don't try to. Instead, we orchestrate them.

Strength	Where It Excels	Our Approach
Ecosystem Maturity	Terraform has 3,000+ providers covering every cloud service imaginable.	**Orchestrate It**: We wrap Terraform, leveraging its entire ecosystem.
Drift Detection	`terraform plan` automatically shows drift from a known state file.	**Verify It**: We use verification to compare reality against the desired spec.
IDE & Developer Experience	Rich IDE plugins for HCL and Pulumi provide validation and autocomplete.	**Abstract It**: We provide a consistent JSON API, regardless of the underlying tool.
Resource Import	Mature workflows exist to import existing infrastructure into state.	**Integrate It**: We plan to import existing state to generate deployment graphs.

A Single Deployment Graph Can Orchestrate Any Tool



All of these disparate operations are executed as part of a single deployment, providing:

- A single, atomic transaction with a unified execution graph.
- Consistent logging, credential flow, and audit trail for every step.
- One verification pass to confirm the end-to-end success of the entire deployment.

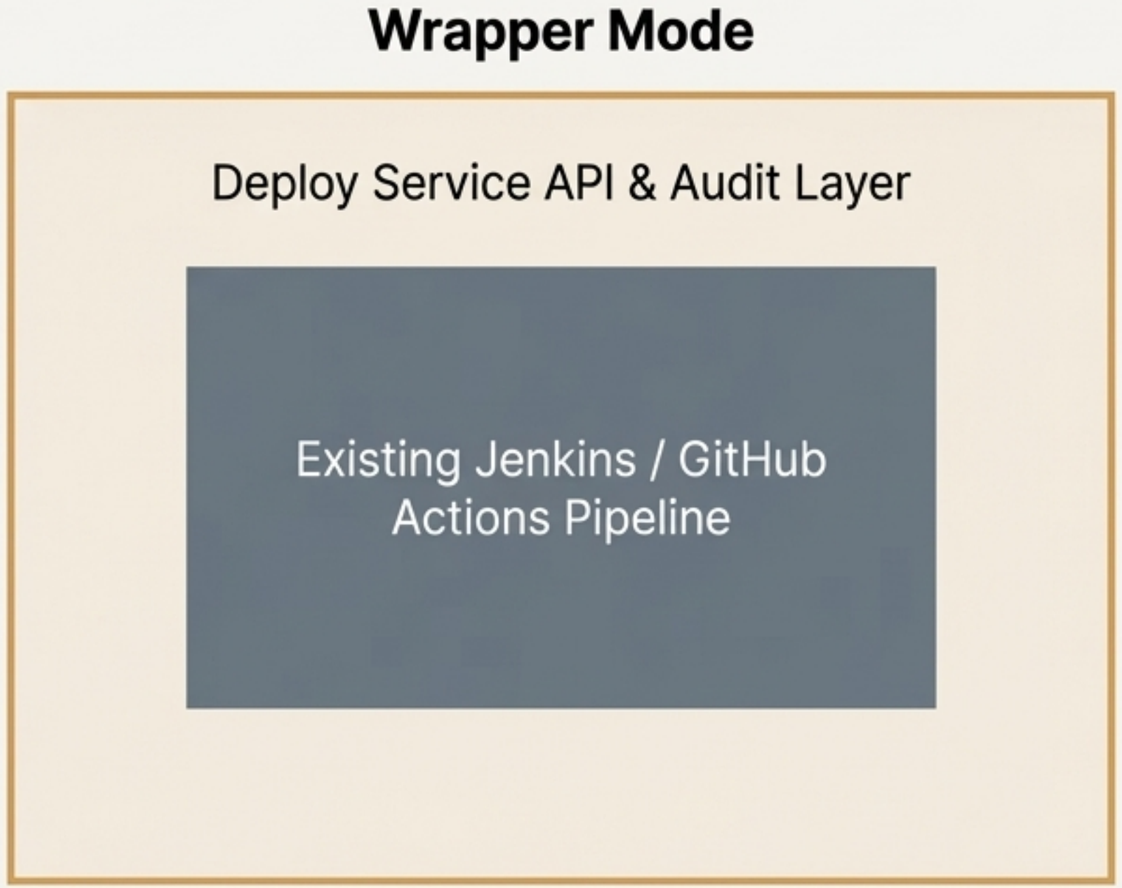
The Path to Adoption is an Evolution, Not a Revolution

The Challenge

Organisations have years of refinement invested in existing CI/CD pipelines. They work, the team understands them, and they cannot be replaced overnight. A rip-and-replace strategy is not viable.

The Solution

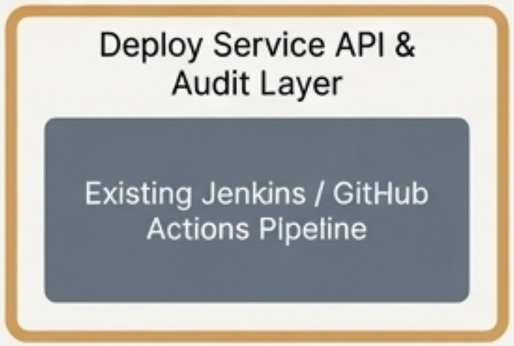
We provide a gradual migration path that delivers value from day one, without disrupting existing workflows. You start by wrapping what you already have.



Gaining Value at Every Step of the Migration Journey

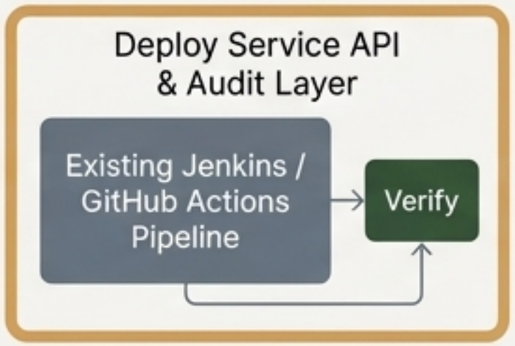
Phase 1: Wrapper

Wrapper Mode



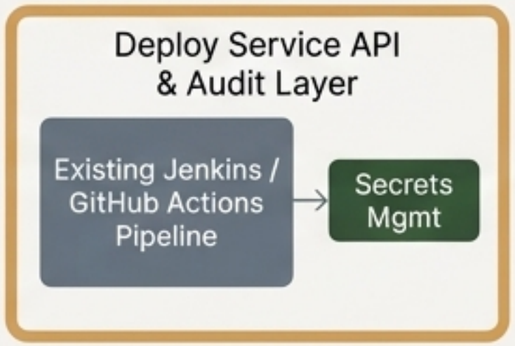
API access, unified logging, audit trail.

Phase 2: Add Verification



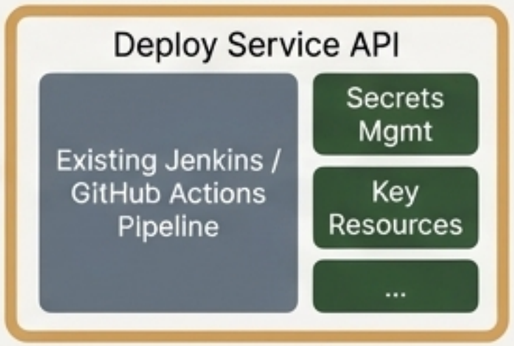
Proof of success, health checks.

Phase 3: Extract Secrets



API-driven secrets, no UI clicks.

Phase 4: Incremental Extraction



Full graph visibility, tag-based discovery.

Coexistence is a feature, not a limitation. Many organisations will permanently operate in a hybrid mode, and this is a fully supported and powerful model.

A Clear Return on Investment at Each Phase

Phase	Relative Effort	Key Benefits Gained
1. Wrapper	**Low**	API access to existing pipelines, centralised logging, unified audit trail.
2. + Verification	**Low**	Proof of deployment success, automated infrastructure health checks.
3. + Secrets Mgmt	**Low**	API-driven secret management, changes tracked in the graph, no manual UI clicks.
4. + Key Resources	**Medium**	Tag-based resource discovery, native verification, granular control.
5. Full Native	**High**	Complete graph control and visibility, maximum orchestration power.

Choosing the Right Tool for the Job

Scenario Guidance Table

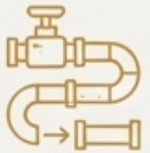
If your scenario is...	Recommended Approach
A team with existing Terraform	Keep Terraform, wrap with Deploy Service for API, audit, and verification.
A platform deploying for customers	**Deploy Service** (for its native multi-tenancy and API).
AI/LLM-driven automation	**Deploy Service** (for its API-first architecture).
A compliance-heavy environment	**Deploy Service** (for its first-class verification and audit trails).
A mixed-tool environment	**Deploy Service** (for unified orchestration across all tools).

Deploy Service Sweet Spots

- API-driven infrastructure automation

- Multi-tenant SaaS platforms

- Unifying disparate IaC and CI/CD tools

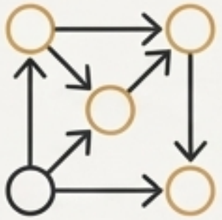


- Gradual modernization of legacy pipelines

What an Orchestration Layer Makes Possible

Enhanced Integrations

- **Natural Language Interface**: “Deploy my app to three regions” -> API call.
- **Policy Engine**: Enforce rules like “Never deploy without verification” at the API layer.
- **Approval Workflows**: Pause deployment graphs pending Slack/Teams approval.



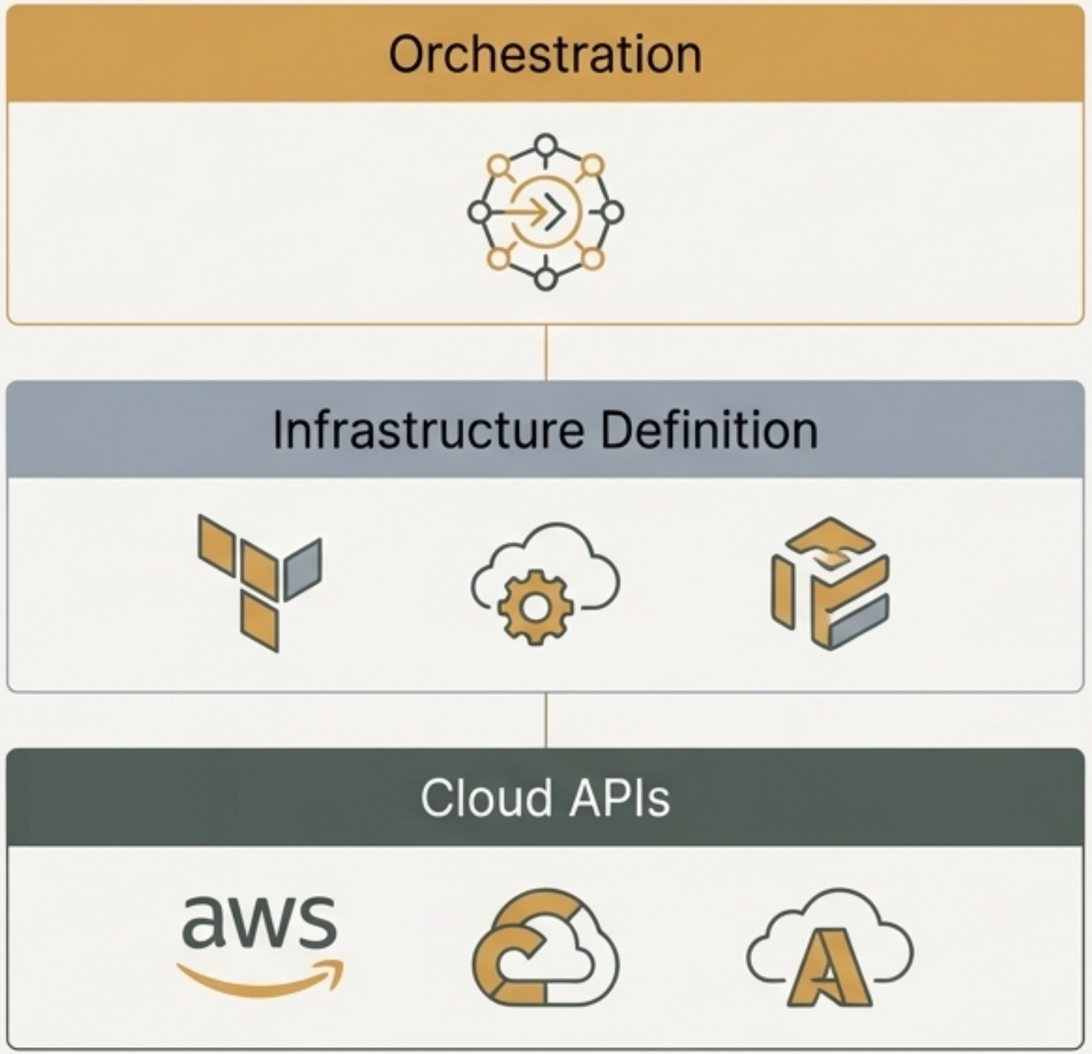
Advanced Orchestration

- **Canary & Blue-Green Deployments**: Define complex rollout strategies as a single graph.
- **Automated Rollback**: On verification failure, execute a pre-defined reverse graph.
- **Scheduled & Cross-Account Deployments**: A single graph to orchestrate complex, timed operations across an entire organisation.

Visualization & Reporting

- **Drift Dashboard**: A unified view of drift across all orchestrated resources, regardless of the underlying tool.
- **Dependency Visualization**: Automatically generate Mermaid/DOT graphs from any deployment.
- **Cross-Deployment Queries**: “Show me all Lambdas created last week.”

The Modern Infrastructure Stack: Definition and Orchestration



The Value Proposition

1. **Unification**: A single API and audit trail across all your tools.
2. **Multi-tenancy**: Native, secure, per-request credentials for platform use cases.
3. **Verification**: First-class, post-deployment verification for any resource.
4. **Gradual Adoption**: Start as a wrapper around existing pipelines and evolve.
5. **Future-Proof**: An architecture built for AI agents, ChatOps, and complex workflows.

“Terraform is for developers deploying infrastructure.

Deploy Service is for **systems** deploying infrastructure—including Terraform.”

Orchestrate, unify, and enhance your existing workflows.