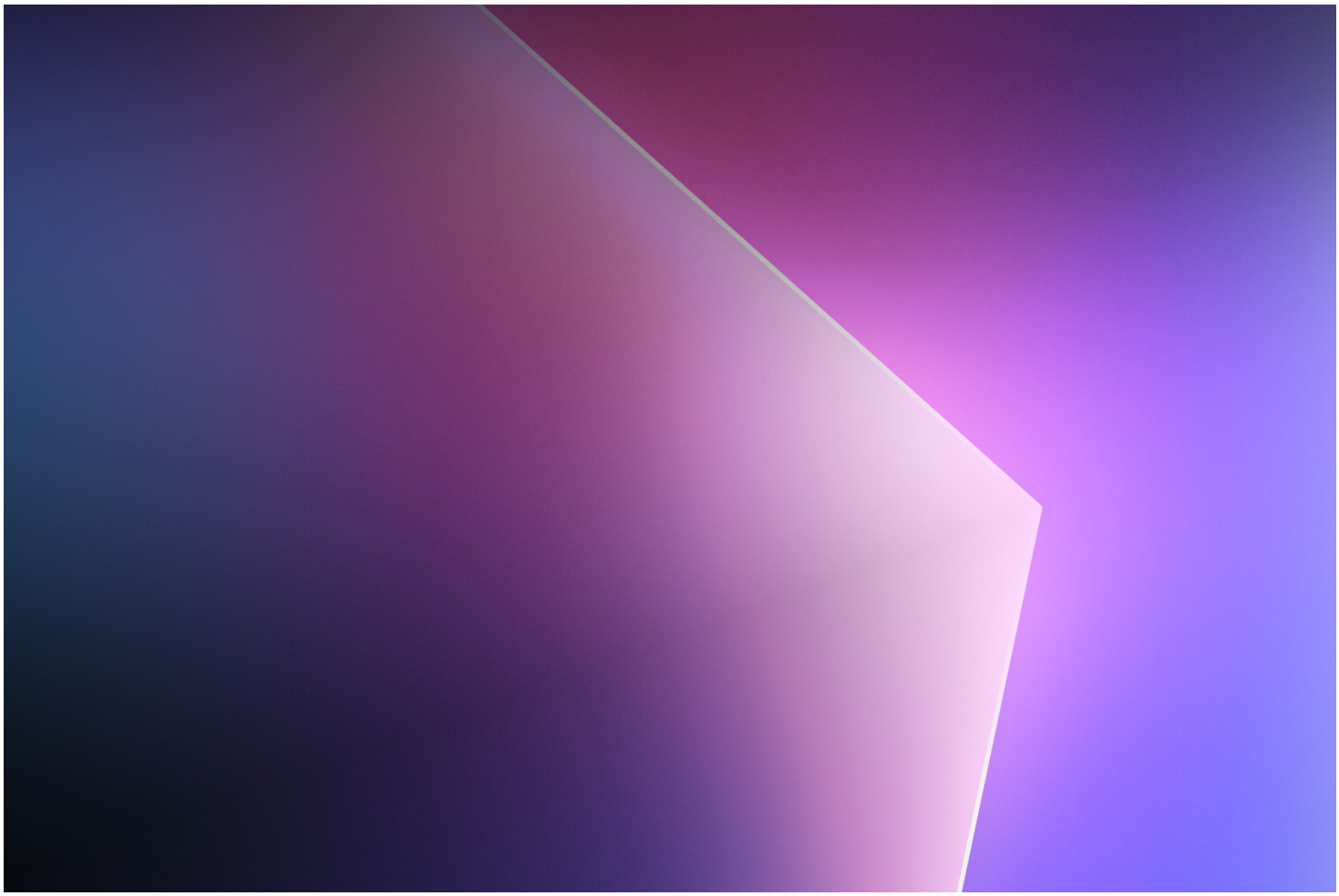

Operating infrastructure at scale

Terraform for Day 2 operations on AWS

HashiCorp
an IBM Company

aws

Operating infrastructure at scale: Terraform for Day 2 on AWS

Organizations operating on AWS reach a point where provisioning infrastructure is no longer the primary challenge. The harder problem is operating infrastructure over time as environments grow, teams change, and complexity increases. The 2025 HashiCorp Cloud Complexity Report shows that 97% of organizations admit they're struggling with their cloud infrastructure, while 98% also admit facing associated cost challenges as well. The report also states that 52% of organizations cite the complexity of operating a hybrid/multi-cloud environment to be their biggest challenge.

Practitioners and technology business decision makers consistently face the same Day 2 issues:

- Limited visibility into resources created outside infrastructure as code
- Configuration drift between intended and actual state
- Increasing cloud spend driven by orphaned or idle resources
- Governance and policy controls that are difficult to apply consistently

Built for AWS environment
HashiCorp collaborates closely with AWS to ensure Terraform integrates directly with AWS services, APIs, and architectural best practices. This alignment enables customers to adopt new AWS capabilities while maintaining a consistent infrastructure control plane.

HashiCorp Terraform addresses these challenges by extending beyond Day 0 and Day 1 provisioning into Infrastructure Lifecycle Management. This white paper explains how three Terraform capabilities work together to support Day 2 operations on AWS:

- Terraform search and import discovers unmanaged AWS resources and brings them into Terraform-managed workflows
- Terraform Stacks allow teams to operate complex AWS and multi-environment systems across multiple accounts, regions, and environments as a single unit
- Terraform actions standardize AWS operational tasks, such as invoking Lambda functions or creating CloudFront invalidations, within Terraform runs

Together, these capabilities position Terraform as a control plane for operating AWS infrastructure at scale. For practitioners, this means safer changes, clearer ownership, and reduced manual work. For technology business decision makers, it means improved governance, lower operational risk, and better cost control across the AWS estate.

Introduction

HashiCorp, an IBM company, works with organizations at every stage of cloud adoption. Across industries and regions, one pattern consistently emerges: most infrastructure challenges do not appear during deployment. They surface later, as environments grow, teams expand, and infrastructure must be operated over time.

Terraform is widely adopted for Day 0 and Day 1 provisioning because it provides a declarative, version-controlled, and reproducible workflow for infrastructure as code. As cloud estates scale, however, customers need more than provisioning. They need help discovering what already exists, operating infrastructure safely at scale, and standardizing ongoing operational work.

This white paper explains how HashiCorp Terraform extends beyond provisioning into Day 2 operations on AWS. It focuses on three capabilities introduced or significantly expanded by HashiCorp to address operational challenges at scale.

Day 0, Day 1, Day 2 at a glance
Day 0 — Design and foundational architecture

Day 1 — Provision and configure infrastructure

Day 2 — Operate, govern, optimize, and evolve over time

Across industries and regions, one pattern consistently emerges: most infrastructure challenges do not appear during deployment. They surface later, as environments grow, teams expand, and infrastructure must be operated over time.

To understand where these challenges arise, it is useful to frame infrastructure work across three stages of the lifecycle: Day 0, Day 1, and Day 2.

Day 0 refers to initial infrastructure creation. This includes:

– Selecting cloud providers

– Defining foundational architecture

– Establishing networking and identity

– Choosing tooling

Decisions made at Day 0 shape everything that follows, but at this stage, little infrastructure actually exists.

Day 1 begins when infrastructure is provisioned. Resources are created, configured, and deployed to support applications and platforms. Terraform is widely adopted at this stage because it provides a version-controlled and reproducible workflow for infrastructure as code.

Day 2 represents the longest and most complex phase of the lifecycle. It includes infrastructure:

– Operations

– Security

– Goverance

– Optimization

– Decommissioning

Day 2 work includes:

– Responding to drift

– Managing cost

– Enforcing policy

– Coordinating changes across environments

– Handling operational tasks that extend beyond provisioning

While Day 0 and Day 1 establish a foundation, most operational risk and cost accumulate during Day 2. As cloud estates scale, teams need more than provisioning tools. They need systems that help them continuously reconcile desired state with reality and operate infrastructure safely at scale.

This white paper explains how HashiCorp Terraform extends beyond provisioning into Day 2 operations on AWS. It focuses on three capabilities introduced by HashiCorp to address operational challenges at scale:

– Terraform search and import

– Terraform Stacks

– Terraform actions

Together, these capabilities position Terraform as a consistent control plane for operating AWS infrastructure across its full lifecycle.

Why Day 2 operations break down at scale

As organizations scale AWS usage, infrastructure complexity grows quickly. Resources are created directly through the AWS console or CLI. Temporary environments remain long after their purpose ends. Configuration changes accumulate outside infrastructure as code workflows.

These patterns are common, especially during early cloud adoption, lift-and-shift migrations, or incident response. Over time, however, they introduce real risk:

– Cloud waste from orphaned or idle resources

– Configuration drift between desired and actual state

– Gaps in governance and policy enforcement

– Reduced confidence in infrastructure state

Terraform state accurately reflects what Terraform manages, but it does not automatically reflect everything that exists in the cloud. As this gap widens, teams lose a reliable source of truth.

Day 2 operations require more than provisioning. They require a control plane that can continuously reconcile desired state with reality. Terraform's newer workflows were designed to address this need.

Terraform as the infrastructure control plane

Terraform acts as a unifying control plane that coordinates infrastructure lifecycle management across tools, teams, and environments.

The Terraform AWS provider
Terraform integrates with AWS through the Terraform AWS provider, which maps Terraform resources directly to AWS APIs and services. The provider supports a broad set of AWS services and evolves alongside new AWS releases.

This integration enables teams to manage networking, compute, storage, identity, and application services using infrastructure as code while maintaining alignment with AWS capabilities.

In this role, Terraform provides:

– A consistent workflow for infrastructure changes

– A shared source of truth through state

– Version-controlled governance and auditability

– Integration points for discovery, orchestration, and automation

Extending Terraform into Day 2 operations means expanding what the control plane can see and influence. This includes unmanaged resources, multi-configuration systems, and operational tasks that traditionally live outside provisioning workflows.

Discover and govern existing AWS resources with Terraform search and import

The operational problem
Many AWS environments contain resources that were never provisioned by Terraform. Early migrations, manual fixes, and legacy workflows often result in infrastructure created outside infrastructure as code.

These unmanaged resources increase operational risk. Without centralized visibility and governance, organizations struggle with cloud cost control, security posture, and compliance. Configuration drift becomes inevitable when infrastructure exists outside Terraform managed workflows.

Terraform search was built to address these challenges by allowing teams to query their cloud environments and bring unmanaged resources into Terraform managed state.

How Terraform search works
Terraform search allows users to define provider-specific queries using list blocks. These queries describe which resources Terraform should search for, using filters such as region, tags, and resource state.

How to get started
– Use Terraform Search to discover unmanaged AWS resources

– Introduce Stacks for multi-account coordination

– Standardize operational workflows with Actions

Running terraform query executes the search remotely and returns matching resources in both the CLI and the UI. This enables discovery without requiring prior knowledge of resource IDs or manual inventory work.

From discovery to managed infrastructure
After reviewing query results, users can select which resources should be imported. Terraform generates starter configuration that includes the required resource and import blocks.

This configuration can be reviewed, adjusted if needed, and applied using the standard terraform apply workflow. During apply, Terraform imports the selected resources into state and begins managing them.

This approach replaces manual, one-by-one imports with a repeatable and auditable process. It also allows teams to immediately normalize imported resources, such as updating tags or enforcing standard configuration.

Operational considerations
Importing existing infrastructure is an operational activity and should be approached carefully. For some complex resources, the generated configuration may require modification before it can be applied successfully.

Terraform search currently supports a subset of AWS resources, with coverage expanding over time. HashiCorp expects adoption to be incremental, starting with the resources that present the highest operational risk or cost.

Operate infrastructure systems with Terraform Stacks

Why Terraform Stacks exist
As Terraform adoption grows in AWS environments, infrastructure is often split across many configurations and workspaces. Teams separate networking, IAM, shared services, Kubernetes clusters, and application workloads to limit blast radius and divide responsibilities. While this improves modularity, it introduces new challenges. Dependencies between configurations must be tracked manually, and coordinating changes across AWS accounts and regions becomes increasingly complex.

AWS users frequently describe the same operational pain points:

- Updating a shared VPC module requires coordinated applies across multiple workspaces
- Rolling out IAM or policy changes across dozens of AWS accounts is slow and error-prone
- Promoting infrastructure changes from development to staging to production requires manual sequencing
- Repeating the same architecture across regions leads to copy-and-paste configuration drift

Before Terraform Stacks, there was no built-in way to provision and manage multiple interdependent Terraform configurations as a single unit. Customers relied on custom scripting, CI pipelines, and internal runbooks to coordinate changes, increasing operational overhead and risk.

Terraform Stacks were introduced to solve this problem for large-scale AWS environments.

Components and deployments
Terraform Stacks introduce a higher-level configuration layer that sits above Terraform modules. Stacks are composed of:

- Components, which reference existing Terraform modules such as VPC, EKS, RDS, or shared IAM foundations
- Deployments, which define where and how many times those components should be instantiated across AWS accounts and regions

This model allows teams to describe complete AWS systems, such as a regional application stack with networking, compute, storage, and identity, and deploy them consistently across multiple environments without duplicating configuration.

For example, a platform team can define a Stack that includes:

- A VPC component
- An EKS cluster component
- A namespace or workload component

Deployments can then instantiate that system in us-east-1, us-west-2, and eu-west-1, or across separate AWS accounts for development, staging, and production.

Managing change at scale
When a Stack configuration changes, Terraform generates plans for each deployment. Teams can review and approve changes selectively, enabling controlled rollouts across AWS environments.

Deployment groups allow organizations to apply different approval and automation rules for different environments. For example:

- Auto-approve changes in development accounts when no resources are destroyed
- Require manual approval in staging
- Enforce stricter checks in production

This approach reduces the risk of broad, uncoordinated changes across AWS accounts while still allowing teams to move quickly.

Operational capabilities for large AWS estates
Terraform Stacks include additional capabilities designed for complex AWS environments, including deferred changes for dependency-heavy workflows such as Kubernetes provisioning, linked Stacks for managing cross-stack relationships between networking and application layers, self-hosted agents for restricted VPC or private datacenter networks, and unified CLI workflows.

Together, these features allow Terraform to manage AWS infrastructure systems as cohesive units rather than isolated configurations, directly addressing one of the most common scaling complaints from AWS operators.

Standardize Day 2 operations with Terraform actions

The Day 2 operations gap
Not all infrastructure work involves creating or modifying resources. Cache invalidations, notifications, remediation workflows, and operational triggers often occur outside Terraform, leading to fragmented workflows and limited visibility.

Terraform actions were introduced to bring these Day 2 operations into Terraform workflows without turning Terraform into a general-purpose automation engine.

How Terraform actions work
Terraform actions are provider-defined operations that can be invoked during Terraform runs. Actions can be triggered automatically during resource lifecycle events or invoked explicitly using the Terraform CLI.

Because Actions execute within Terraform workflows, they appear in run output and audit logs. This provides visibility into operational tasks that were previously manual or opaque.

Actions in AWS environments
In AWS environments, Terraform actions can invoke Lambda functions, create CloudFront cache invalidations, send SNS notifications, and trigger other operational workflows.

By standardizing these tasks within Terraform, organizations reduce manual effort and ensure operational steps are executed consistently and are auditable.

From provisioning to Infrastructure Lifecycle Management

Infrastructure risk rarely appears at the moment resources are created. It accumulates over time as environments grow and change.

HashiCorp Terraform has evolved to address this reality. By extending Terraform beyond provisioning into discovery, system-level management, and operational orchestration, HashiCorp enables organizations to manage AWS infrastructure across its full lifecycle.

Terraform search and import make unmanaged infrastructure visible and governable. Terraform Stacks provide a scalable model for operating complex systems. Terraform Actions standardize Day 2 operations within Terraform workflows.

Together, these capabilities allow Terraform to serve as a long-term control plane for Infrastructure Lifecycle Management at scale.

Get started today:

- Explore Terraform on AWS
- Review Terraform search and import documentation
- Learn more about Terraform Stacks use cases

Visit hashicorp.com/terraform to begin.

February 2026