harness

The Practical Guide to

Modernizing Infrastructure Delivery

harness

Contents

Introduction

In today's AI native software landscape, velocity is everything. Code moves from idea to production faster than ever. Continuous integration and deployment have become standard. Teams iterate rapidly to meet rising user expectations.

But infrastructure delivery has not kept pace.

While development workflows have evolved, infrastructure remains a common bottleneck. Developers are expected to move quickly, but too often they are waiting—for environments to be provisioned, configurations to be applied, or approvals to clear. And when they are not waiting, they are navigating a maze of fragmented tools and inconsistent processes.

Tooling has improved, but not in unison. Provisioning is often managed through Infrastructure as Code frameworks like Terraform or OpenTofu, executed via custom scripts or isolated CI jobs. Configuration is handled separately, sometimes through automated tools like Ansible, other times via manual post-provision steps. CI/CD pipelines operate independently of infrastructure state, unaware of what was provisioned, whether it conforms to policy, or if it is even ready to receive code. The result is a delivery pipeline that appears modern but lacks cross-layer coordination.

Increasingly, infrastructure complexity demands more than automation. It demands systems that are intelligent, consistent, and adaptive.

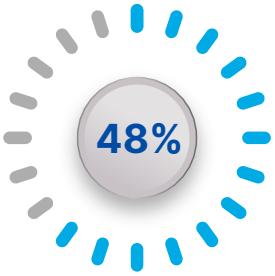
AI is beginning to reshape how infrastructure is managed. From surfacing cost anomalies to recommending remediations for drift or security violations, machine learning and AI agents are emerging as assistive layers that complement human workflows—not replace them. As part of a governed system, AI can help platform teams scale decisions, enforce consistency, and reduce operational overhead.

This guide is for platform engineers, DevOps leaders, and infrastructure architects who are working to close the gap between what their teams need and what traditional infrastructure delivery can support. It offers a framework grounded in unified workflows, reusable components, policy-driven guardrails, and self-service models that scale. Whether you are just beginning your journey or rethinking a mature platform, the goal is the same: deliver infrastructure that moves as fast as your code while remaining secure, compliant, and increasingly intelligent over time.

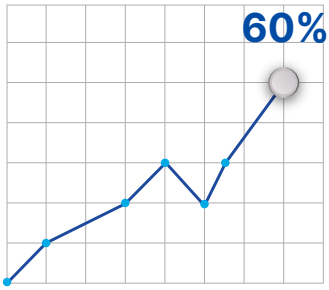
Key Statistics

48% of outages in cloud environments are caused by misconfigurations, many of which stem from inconsistently applied infrastructure definitions.

– Gartner, 2024



Outages due to misconfigurations



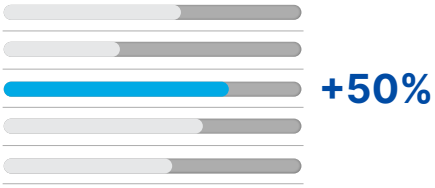
Developers blocked each sprint

60% of developers say they are blocked waiting for infrastructure at least once per sprint.

– Harness IDP, 2023

Over 50% of organizations using Terraform have no system in place for reusing modules across teams leading to drift, duplication, and inconsistent security practices.

– OpenTofu Community Survey, 2024



Organizations **not reusing** modules

The State of Infrastructure Delivery Today

Challenges Facing Teams

Software teams today want to move fast but infrastructure often doesn't. Despite advancements in CI/CD, DevOps practices, and cloud-native tooling, the way infrastructure is provisioned, configured, and deployed remains a patchwork of disconnected workflows.

Across many organizations, infrastructure delivery is still defined by:

- **Multiple handoffs** between platform, security, operations, and development teams at every stage including provisioning, configuration, and deployment.
- **Inconsistent environments** caused by configuration drift, manual changes, and untracked infrastructure definitions.
- **Delayed access to infrastructure**, which blocks developers from shipping features or testing changes when they need to.
- **Siloed tooling** that lacks integration and visibility like Terraform for provisioning, Ansible for config, Jenkins for deployment that are all managed in isolation.
- **Governance gaps**, where policy enforcement, access control, and cost oversight are bolted on as afterthoughts rather than built into the delivery workflow.

As engineering organizations scale, these inefficiencies compound. Manual processes become brittle. Platform teams become overburdened. Developers seek workarounds. And security, compliance, and cost control suffer as a result.

Why a New Approach Is Needed

The demands on infrastructure teams have changed, but the systems many rely on have not.

- **Cloud and hybrid environments are growing more complex,** with resources spread across multiple regions, accounts, and architectures. Managing them through scripts or spreadsheets no longer scales.
- **Platform engineering teams are now tasked with enabling self-service** providing reusable templates, golden paths, and automation that empowers developers without compromising control.
- **Compliance and cost accountability are no longer optional.** Teams must prove that every provisioned resource is secure, policy-compliant, and cost-justified before it's deployed.

Meeting these needs requires more than better automation. It requires a shift in mindset from managing infrastructure as a set of tools, to delivering infrastructure as a product.

A Simpler Model: Provision, Configure, Deploy

A growing number of high-performing engineering organizations are consolidating these workflows into a more cohesive model. Instead of treating provisioning, configuration, and deployment as distinct processes, they are orchestrating them as a single, integrated system. This approach enables greater predictability, stronger governance, and higher developer velocity.

Provision: Codify and Allocate Infrastructure

Provisioning defines the structure of cloud environments. It involves translating architectural intent into code that declares compute, storage, networking, IAM policies, and other foundational resources. Most teams use frameworks such as Terraform or OpenTofu to achieve this, enabling infrastructure to be version-controlled, reviewed, and automated.

However, provisioning at scale introduces significant challenges. Organizations managing dozens or hundreds of teams often contend with duplicated modules, inconsistent variables, and fragile automation. As the number of workspaces grows, so does the operational overhead.

A platform-driven provisioning approach addresses these issues by introducing:

- **Shared module registries that enforce consistent patterns and ownership**
- **Variable sets that standardize inputs across teams and environments**
- **Policy controls that validate infrastructure before it is applied**
- **Pull request automation that integrates provisioning into the broader software development lifecycle**
- **Cost estimation and drift detection to maintain accuracy and efficiency over time**

Provisioning infrastructure is no longer just about codifying resources. It is about embedding infrastructure creation into governed, automated systems that are resilient to scale and change.

Configure: Make Infrastructure Operational

Provisioning delivers the underlying resources. Configuration makes those resources functional.

Post-provision configuration includes installing agents, setting environment-specific parameters, enforcing security hardening, and applying network and runtime policies. These tasks are procedural, often requiring ordered execution and conditional logic. Declarative IaC frameworks are not built to handle this layer effectively.

Some teams attempt to bridge the gap with custom shell scripts or manual steps. Others rely on separate configuration management systems that drift over time or fall out of sync with provisioning pipelines.

Tools like Ansible provide a programmatic way to manage configuration through structured playbooks and inventory-driven automation. By codifying configuration in playbooks, teams can:

- **Ensure consistency across environments**
- **Avoid drift caused by manual changes**
- **Reduce onboarding time for new infrastructure patterns**

Looking ahead, the configuration layer will become increasingly intelligent. With the rise of AI-assisted agents, teams can detect misconfigurations, auto-generate procedural playbooks for common use cases, and suggest configuration changes based on historical patterns where all before infrastructure reaches production. These capabilities are not hypothetical. They are already appearing in modern platforms like Harness, with beta-stage infrastructure agents designed to proactively recommend or apply remediations based on observed drift or policy violations.

Deploy: Release Applications with Infrastructure Context

Application deployment typically relies on CI/CD pipelines to build, test, and release code. However, these pipelines often operate without visibility into the environments they deploy to. They may not know if an environment is provisioned correctly, if required services are running, or if compliance policies have been met.

This lack of infrastructure awareness creates a fragile deployment surface. Failures occur late in the process. Rollbacks are incomplete. Troubleshooting spans multiple systems and teams.

Integrating infrastructure delivery with application deployment creates a more reliable release lifecycle. When provisioning, configuration, and deployment are orchestrated as a single workflow, the system gains shared context across all layers. This allows teams to:

- **Validate that deployments target environments in a known and compliant state**
- **Link infrastructure and application changes in the same versioned history**
- **Improve traceability across pipeline runs, infrastructure changes, and application behavior**
- **Reduce manual coordination between platform and application teams**

Aligning application delivery with infrastructure readiness results in faster recovery times, fewer production issues, and a more consistent release process across environments.

The provision, configure, deploy model has been discussed for years. What has changed is the maturity of the tools and platforms now available to support it. Organizations no longer need to build this integration from scratch. They can implement it consistently and at scale, using systems designed to treat infrastructure delivery as a first-class concern within the broader software lifecycle.

harness

What Modern Teams Do Differently

What Modern Teams Build	What It Enables
Reusable, versioned modules for compute, networking, storage, IAM, and more	Consistency across teams, faster onboarding, and reduced duplication
Environment templates codified in IaC and configuration tools	Rapid environment provisioning without ticket-based workflows
Policy-as-code guardrails enforced at every stage of the pipeline	Safe self-service for developers with built-in security and compliance checks
Integrated pipelines that connect provisioning, configuration, and deployment	A unified delivery flow with traceability and reduced handoffs
Internal registries and templates exposed through self-service platforms	Developer access to infrastructure as a product, with minimal operational overhead
Automated validation gates for cost, security, and architectural conformance	Risk reduction before resources are provisioned or deploy

Rethinking Tooling: From Scripts to Systems

Most infrastructure teams have adopted automation in some form. Provisioning scripts exist. CI/CD pipelines run. Configuration management tools are in place. But despite this, the workflows remain brittle. Tasks are automated, but systems are not integrated. The result is a collection of tools that execute steps in isolation rather than a delivery system that operates end-to-end.

What Matters More Than the Tools Themselves

The tools used for provisioning, configuration, and deployment are often the same across organizations. What sets high-performing teams apart is not the tooling itself, but how they build systems that scale, enforce consistency, and remain adaptable over time. Rather than focus on which tool is being used, modern platform teams focus on the outcomes the system can guarantee. They ask different questions that shift attention from automation to architecture, and from speed to sustainability.

Can you enforce standards without slowing developers down?

Governance should not be a separate track. When policies, security checks, and architectural guidelines are embedded into the delivery process, compliance becomes a default, not a delay. Developers can deploy infrastructure through self-service workflows, confident that what they ship aligns with internal controls without needing to navigate a separate review process.

Can you update infrastructure across environments with a single change?

Without modularization, teams are forced to replicate effort. Updating environments means managing a patchwork of scripts or isolated pipelines. By investing in versioned modules and environment templates, teams enable repeatability. They can apply changes globally, understand the impact locally, and reduce the surface area for error.

Can you see what is being provisioned, changed, and deployed at any given moment?

Provisioning logs and deployment status tell only part of the story. Understanding how infrastructure evolves over time like who made changes, when they occurred, and whether they conform to policy requires deeper observability. Teams need real-time insight into workflows, not just artifacts. This visibility enables better incident response, tighter access control, and more informed decisions about change.

What Modern Platforms Need

Infrastructure tooling has matured, but maturity does not come from using more tools. It comes from building systems that integrate those tools into a cohesive delivery experience. Modern platforms are not collections of automation; they are structured environments that enforce standards, support scale, and remain flexible across teams and technologies.

They treat infrastructure components as reusable assets.

With a module registry and workspace templates, infrastructure patterns can be defined once and consumed many times. Modules are versioned and controlled, enabling consistency without limiting flexibility. Templates abstract complexity while still allowing input customization where required.

They unify provisioning and deployment into a single workflow.

Modern pipelines do more than deliver application code. They orchestrate the creation and configuration of infrastructure alongside the software that runs on it. This alignment ensures that environments are ready, compliant, and validated before deployment ever begins.

They validate changes before execution.

Policy enforcement and cost estimation are built into the pipeline. Changes are evaluated against rules and budgets before resources are created. Security, compliance, and financial guardrails are applied early, not after the fact, making the delivery process safer and more predictable.

They support the tools your teams already use.

Standardization should not require consolidation. Modern platforms accommodate diverse teams and tooling, from Terraform and OpenTofu to Ansible and beyond. As infrastructure evolves, the system evolves with it without requiring a full rebuild of your workflows.

The shift is not from one tool to another. The shift is from toolchains to systems. Systems that are observable, repeatable, and reliable by default. That is the foundation for modern infrastructure delivery.

harness

Developer Self-Service Is Not Optional Anymore

As organizations scale, infrastructure bottlenecks become a constraint on velocity. When developers depend on ticket-based provisioning or manual reviews to access environments, delivery slows down, and cognitive load increases. Modern teams recognize that self-service is not a luxury and that it is a foundational requirement for scalable software development.

Internal developer platforms (IDPs) play a central role in this shift. By abstracting infrastructure complexity behind simple interfaces, they empower developers to deploy, test, and iterate independently, without compromising on governance or security.

The Role of the IDP

An IDP provides a structured interface between developers and the underlying platform. Rather than expose raw scripts or infrastructure definitions, it offers workflows that are safe, reusable, and aligned with organizational policy.

- **Developers interact with standardized templates rather than authoring infrastructure code from scratch.**
- **Access is role-based, ensuring the right teams can provision the right resources.**
- **Guardrails are built-in, so teams do not need to memorize compliance or security requirements.**
- **Observability is provided by default, so developers understand the impact of what they deploy.**

The result is a clear boundary: platform teams define the rules, and developers consume the infrastructure productively within them.

harness

Powering Infrastructure Catalogs

A critical enabler of self-service is the infrastructure catalog. Catalogs allow developers to select from pre-defined environment templates, module combinations, or workload types that are each pre-approved by the platform team.

- **Templates are created using Infrastructure as Code and configuration management tools.**
- **Versions are controlled and published to internal registries.**
- **Metadata and documentation guide developers in choosing the right patterns for their use case.**

Catalogs reduce duplication, increase reliability, and shift responsibility for provisioning away from centralized teams without giving up control.

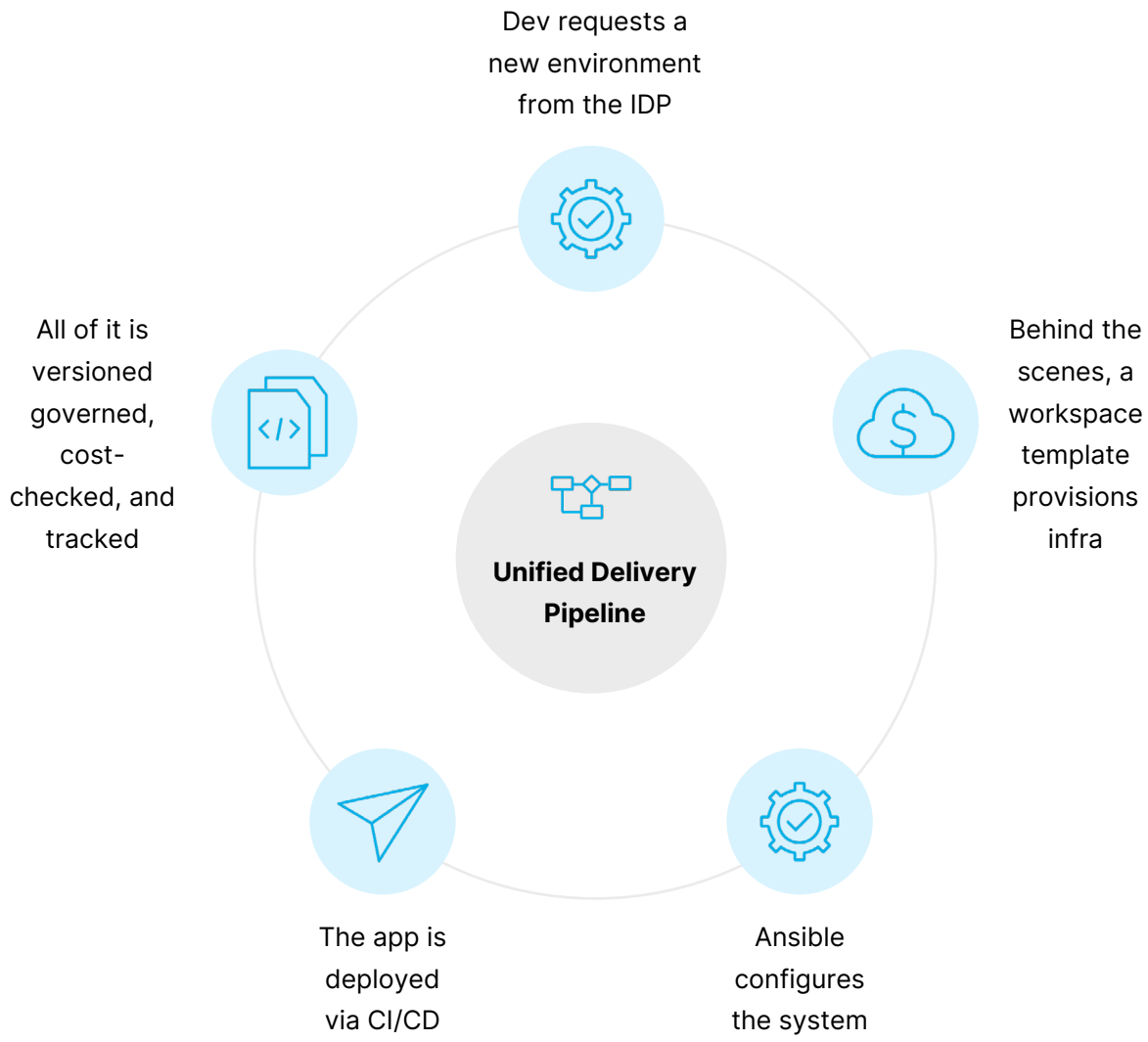
Integration into Developer Workflows

Self-service only works when it fits naturally into how developers already build and ship software. That means:

- **Infrastructure requests are embedded into pull requests or CI/CD pipelines.**
- **Environments can be created on-demand for feature branches, test cases, or deployments.**
- **Developers receive feedback in the same systems they use every day such as version control, issue trackers, or chat tools.**

When infrastructure is embedded into existing workflows, adoption increases and friction decreases. More importantly, the boundaries between application and platform teams become clearer and more collaborative. Developer self-service is not just about speed. It is about scale, consistency, and autonomy. As software systems grow more complex, the ability for developers to provision and manage infrastructure safely becomes essential, not optional.

Putting It Together: A Unified Delivery Pipeline



Key Principles for Scaling Infrastructure Delivery

Scaling infrastructure is not just about adopting new tools or increasing automation. It requires a deliberate shift in how teams think about system design, governance, and collaboration. The following principles are foundational for building infrastructure platforms that grow with the organization.

Think in Platforms, Not Tools

Tools evolve. Platforms last.

Rather than expose developers to the details of provisioning or configuration tools, modern platforms define clear interfaces for requesting infrastructure. Portals, templates, and workflows become the surface area for developers abstracting away complexity while preserving flexibility behind the scenes. This separation enables platform teams to evolve their tooling over time without disrupting developer workflows.

Enforce Standards Through Code, Not Checklists

Manual reviews do not scale.

Policies and approvals must be codified. Policy-as-code frameworks allow teams to define compliance, security, and architectural rules in a repeatable, testable format. These rules are then enforced automatically as part of infrastructure workflows removing the need for out-of-band reviews and reducing time to delivery. Codifying standards also improves transparency and ensures that enforcement is consistent across teams and environments.

Measure Infrastructure Like Software

Infrastructure delivery should be observable and measurable.

Treat infrastructure workflows with the same rigor as application code. Track version history. Measure time-to-provision. Monitor for drift and unintended changes. Metrics like deployment frequency, lead time for changes, and change failure rate apply equally to infrastructure systems. This mindset turns infrastructure from a support function into an engineering discipline.

Make Reuse the Default

Avoid solving the same problem more than once.

Platform teams should invest in module registries, workspace templates, and paved paths that encode best practices. When developers can start from proven patterns, they move faster with less risk. Reuse also accelerates iteration. Improvements made to a shared module benefit every team using it.

Start Simple, Scale Thoughtfully

Not every team needs every feature from day one.

Infrastructure platforms should grow iteratively. Begin with foundational patterns and evolve based on usage and feedback. Introduce self-service incrementally. Add guardrails and policies as needed. Scaling is not about building everything at once, it is about building the right things at the right time.This approach keeps the platform manageable and aligned with real-world needs.

What Modernized Looks Like

Modern infrastructure delivery is not measured by the number of tools adopted or scripts automated. It is defined by clarity, consistency, and the ability to scale governance without sacrificing speed. The most effective teams do not just build pipelines, they build platforms for their teams.

From the Field

Across industries, leading infrastructure teams are applying shared principles in different ways:

- **A fintech engineering org on Harness has scaled to provision over 4,000 workspaces across teams using fewer than ten reusable templates by reducing drift while maintaining team autonomy.**
- **Another DevOps team implements automated cost estimation and Open Policy Agent (OPA) policies to block misconfigured infrastructure before it reaches production by ensuring financial and operational safeguards without manual intervention.**

These teams do not rely on brute-force automation. They build systems that encode organizational knowledge, enforce controls programmatically, and create a better developer experience by design.

Consistent Patterns, Regardless of Scale

While implementation details vary, the foundational practices of high-performing infrastructure teams remain consistent:

They optimize for trust and autonomy.
Developers are empowered through self-service, but every workflow is governed by code-defined guardrails.

They automate governance early.
Policy enforcement, cost control, and compliance validation are integrated from the start—not added retroactively.

They treat infrastructure as a platform, not a pipeline.
The platform is reusable, composable, and maintained with the same discipline as production software.

What modernized looks like is not perfection. It is clarity, reuse, and confidence at every stage of the Software Delivery Lifecycle. As teams scale, AI will play a larger role in helping platform teams manage infrastructure proactively. The most modernized infrastructure systems will not just be automated. They will be adaptive by leveraging data, feedback, and AI-driven insights to evolve alongside the applications they support.

The Role of AI in Infrastructure Delivery

As infrastructure systems grow more complex, platform teams face an increasing need for automation that goes beyond static pipelines and hardcoded policies. AI introduces a new layer of adaptability by enabling infrastructure systems that learn, respond, and improve over time.

Early use cases already emerging in modern platforms like Harness to include:

Intelligent remediation
AI agents analyze drift, failed plans, or out-of-policy changes and recommend or apply remediations based on prior decisions and organization rules.

Cost anomaly detection
AI models can identify infrastructure changes that significantly deviate from normal usage patterns and flag potential overspending before it becomes a budget issue.

Automated policy suggestion
Machine learning models trained on historical provisioning activity can suggest new OPA policies or updates to existing rules when violations recur.

Configuration optimization
AI can assist in recommending optimal configuration patterns based on runtime metrics, cloud provider data, and internal SLOs.

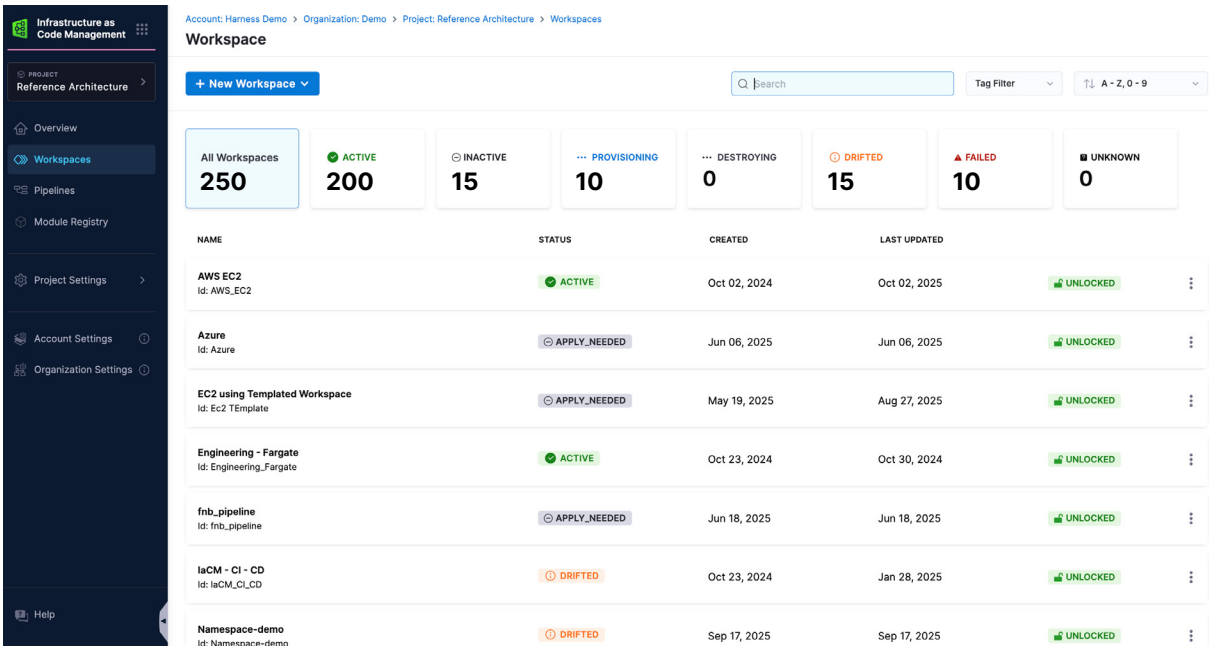
The value of AI in infrastructure is not in replacing platform teams, but in augmenting them. By offloading repetitive analysis and enabling proactive decision support, AI shifts the platform team's role from firefighting to system design and continuous improvement.

Harness Infrastructure Management Capabilities at a Glance

Harness provides infrastructure delivery systems that enable platform teams to manage complexity, enforce consistency, and expose infrastructure to developers through self-service interfaces. The core capabilities are delivered through two integrated modules: **Infrastructure as Code Management (IaCM)** and the **Internal Developer Platform (IDP)**.

Infrastructure as Code Management (IaCM)

Harness IaCM is designed to bring consistency, control, and scalability to how infrastructure is defined, tested, approved, and deployed. It enables platform teams to standardize infrastructure delivery while giving developers safe, self-service access to provisioned environments. IaCM workflows feel familiar to application teams. Infrastructure changes are executed through pipelines, pull requests, or API calls that mirror CI/CD patterns. At the same time, platform teams retain centralized control over modules, policies, and infrastructure state.



harness

Centralized pipeline orchestration

All workspaces execute through a shared pipeline engine that handles plan, apply, approvals, and rollback. No external orchestrators or custom automation layers are required.

Workspace templates

Define repeatable, parameterized environments with guardrails baked in. Platform teams maintain a few golden templates while enabling thousands of workspaces across teams.

Module registry and integrated testing

Store, version, and test reusable modules. Developers pull only trusted infrastructure components, while platform teams retain control over changes and conformance.

OPA-based policy enforcement

Validate plans against security, compliance, and operational policies before execution. Approvals can be triggered based on policy evaluation outcomes, ensuring governance at scale.

Cost estimation before apply

Preview cost impact during the plan phase. Identify misconfigurations early and align infrastructure usage with financial expectations.

Drift detection and remediation

Monitor for configuration drift between declared and actual state. Future capabilities include an AI Infrastructure Agent that analyzes the nature of the drift and automatically recommends or applies remediations based on organizational policies and prior behavior.

GitOps-native delivery

Infrastructure updates can be triggered via pull requests with integrated policy checks and approval gates. PRs serve as the single source of truth for both infrastructure and application changes.

Support for multiple IaC tools

Manage Terraform and OpenTofu natively, with support for Ansible,Terragrunt, and other tooling to provide teams with flexibility as they evolve.

Fine-grained access control and auditability

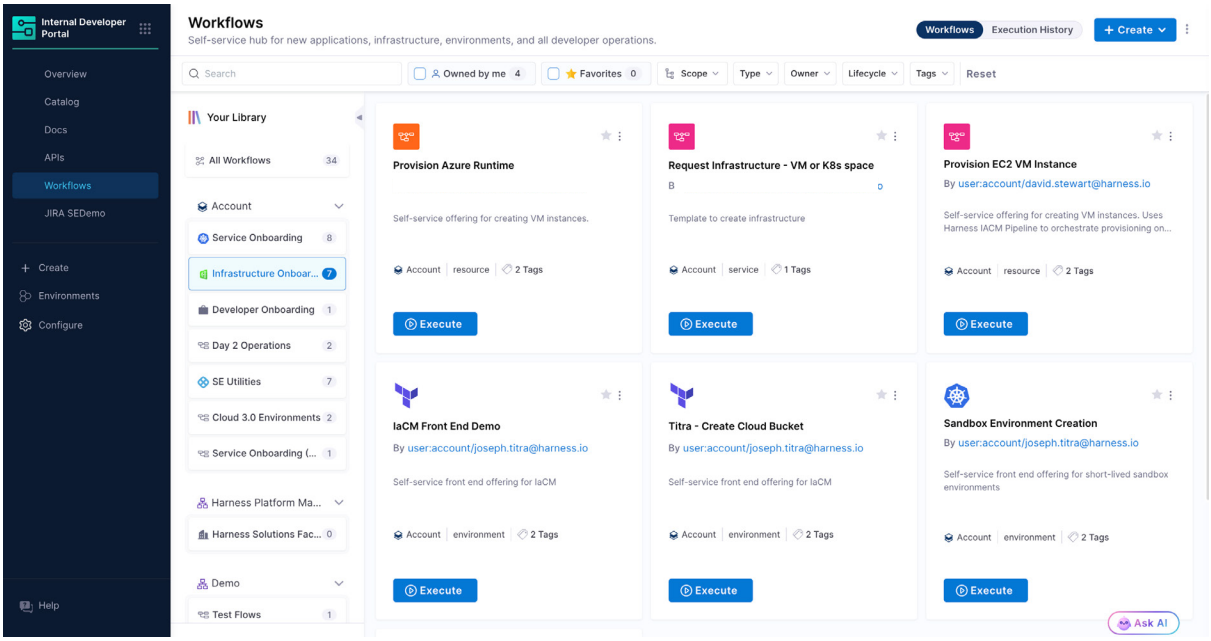
Define role-based permissions for modifying pipelines, applying changes, and viewing infrastructure state. Every change is tracked, providing full audit trails for platform and security teams.

Custom dashboards for infrastructure visibility

Build tailored views for platform, security, or finance stakeholders. Monitor usage, cost, drift, and change activity across environments in one place.

Internal Developer Platform (IDP)

Harness IDP provides the control plane for delivering infrastructure and services to developers through self-service. It exposes infrastructure workflows as productized experiences—codified by platform teams, consumed by developers, and governed by policy. The goal is not to hide complexity, but to abstract it behind safe interfaces. Developers interact with templates, not pipelines. They provision environments, services, or infrastructure from curated catalogs. Platform teams control how these catalogs are built, how policies are enforced, and how workflows are executed.



Infrastructure and service catalogs

Curated catalogs expose approved templates for compute, networking, environments, and application services. Catalog entries are backed by IaC and configuration automation but abstracted into developer-friendly forms.

Workflow embedding

Developers invoke infrastructure workflows from pull requests, issue trackers, or chat interfaces—no need to manually configure pipelines or request access. Workflows are event-driven and Git-integrated.

Self-service provisioning with built-in controls

Developers request environments through forms or APIs. Platform teams define inputs, default values, and constraints. Approvals, cost estimates, and policy validations are enforced automatically.

Pipelines as a product

Infrastructure delivery pipelines are versioned and maintained by platform teams. Developers do not build their own pipelines; they select and use paved paths defined for their use case.

Role-based access control (RBAC)

Permissions are fine-grained and environment-aware. Platform teams can control who can provision what, in which environment, and with what level of access.

Audit trails and reporting

Every provisioning, deployment, and configuration action is logged. Audit data is queryable and exportable for compliance, cost analysis, or incident investigations.

Integration with external systems

IDP connects to secret managers, SCM systems, observability platforms, and ticketing tools—reducing context switching and making infrastructure part of the broader developer workflow.

Onboarding and discoverability

Templates include metadata, documentation, and tags. Developers can search, filter, and request the infrastructure they need without waiting on ops.

The Modern Software Delivery Platform™

Follow us on

/harnessio

/harnessinc

Contact us on

www.harness.io