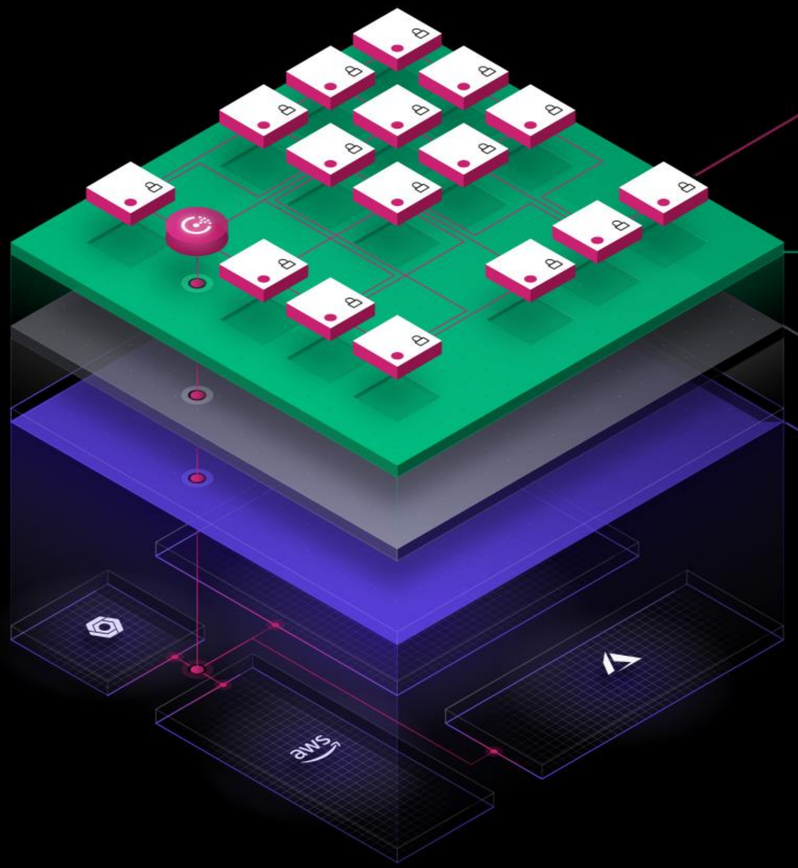
Multi cloud. Hybrid cloud. Any infrastructure. Anywhere.

Enabling safe and efficient provisioning and management.

The 4 essential elements of dynamic infrastructure



Networking

Connect infrastructure and applications

Development

Run applications

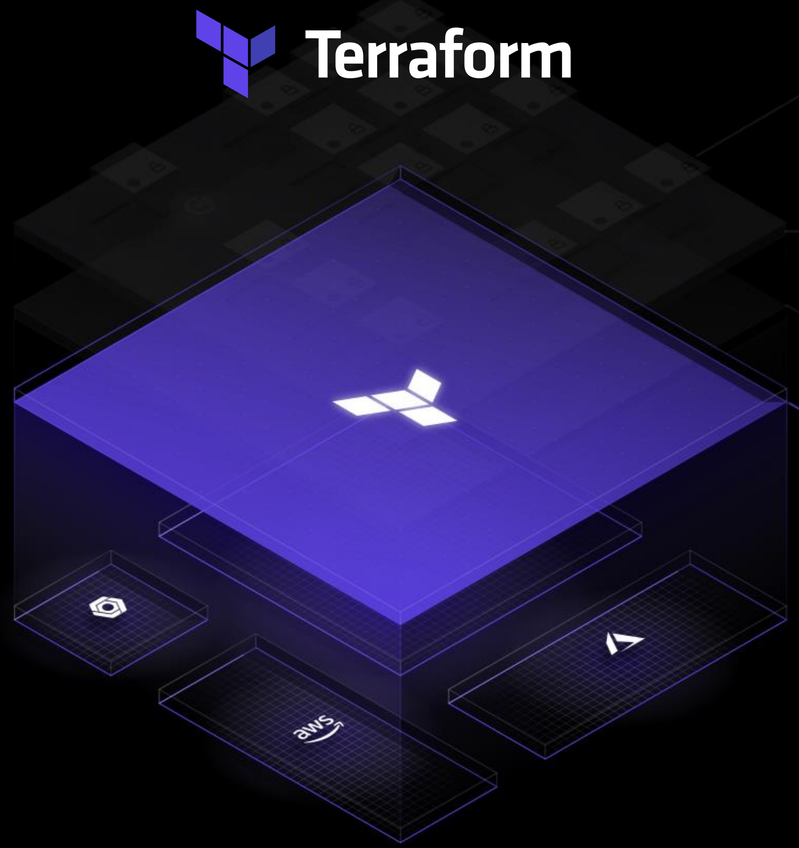
Security

Secure applications and infrastructure

Operations

Provision Infrastructure

The 4 essential elements of dynamic infrastructure



Networking

Connect infrastructure and applications

Development

Run applications

Security

Secure applications and infrastructure

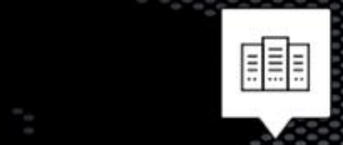
Operations

Provision Infrastructure

The shift to **provisioning dynamic infrastructure**

Static Infrastructure
Homogeneous, Private

Dynamic Infrastructure
Heterogeneous, Distributed



The shift to **provisioning dynamic infrastructure**

Static Infrastructure
Homogeneous, Private

Dynamic Infrastructure
Heterogeneous, Distributed

The shift to **provisioning dynamic infrastructure**

Static Infrastructure

Homogeneous, Private

Homogeneous infrastructure provisioned for long periods of time with dedicated machines.

Traditional Approach

- Manual workflows
- Homogeneous infrastructure
- Delegate with ticketing requests and queues

Dynamic Infrastructure

Heterogeneous, Distributed

Heterogeneous infrastructure frequently provisioned and available on-demand.

Terraform Approach

- Infrastructure as code
- Embrace heterogeneity with extensible providers
- Delegate with self-service modules and automated policy enforcement

Business Challenges of Dynamic Infrastructure

Increased cost.

$10B overspent on cloud annually

Reduced productivity.

Up to 70% of time is used rebuilding, debugging, or waiting.

Increased risk.

Larger surface area to secure, govern, and audit

Business Value of Terraform

Reduce cost.

Eliminate unnecessary spend by 30-40%

Increase productivity.

Engineers spend up to 70% less time on manual tasks.

Reduce risk.

One workflow to secure, govern, and audit.

Terraform Use Cases

Adopt any cloud, infrastructure, or service safely, efficiently, and timely.

Infrastructure as Code

Use infrastructure as code to safely and efficiently provision and manage infrastructure at any scale.

Multi-Cloud Management

Provision and manage public cloud, private infrastructure, and external services holistically while still preserving the uniqueness of each.

Self-Service Infrastructure

Provide a library of approved infrastructure that developers can use to safely and efficiently provision infrastructure on-demand.

Terraform Principles

Automation through Codification

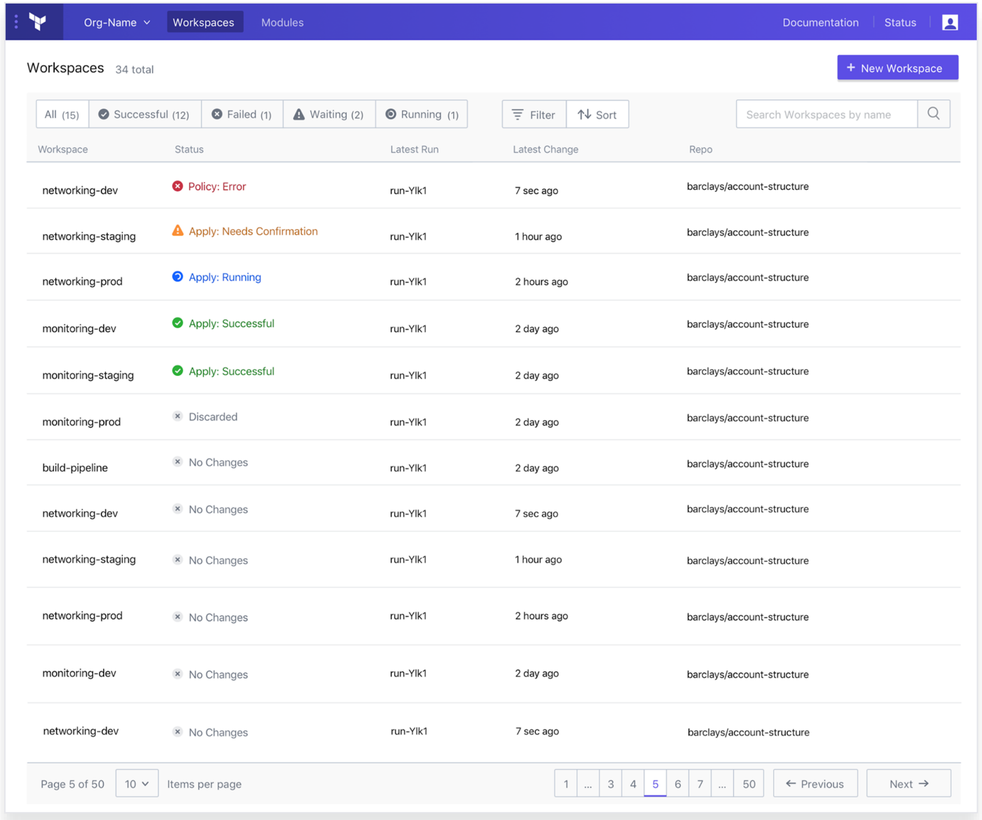
Create, share, collaborate, and provision infrastructure using code.

```
resource "aws_elb" "frontend" {
  name = "frontend-load-balancer"
  listener {
    instance_port     = 8000
    instance_protocol = "http"
    lb_port           = 80
    lb_protocol       = "http"
  }

  instances = ["${aws_instance.app.*.id}"]
}

resource "aws_instance" "app" {
  count = 5

  ami           = "ami-408c7f28"
  instance_type = "t1.micro"
}
```

Terraform Principles

Workflows, not Technology

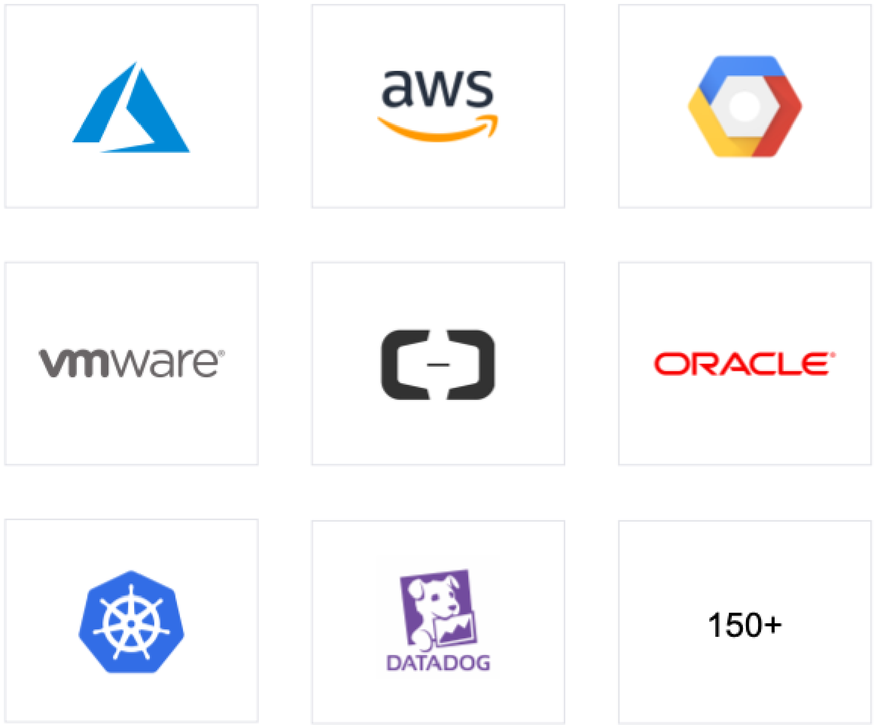
Use a consistent workflow to provision infrastructure while preserving uniqueness of each provider.



Terraform Principles

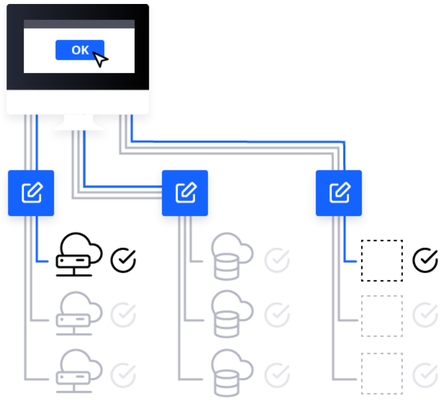
Open and Extensible

Leverage open source providers for broad support of common infrastructure and add your provider to support custom infrastructure.



Use Case
Infrastructure as Code

Use Case: **Infrastructure as Code**



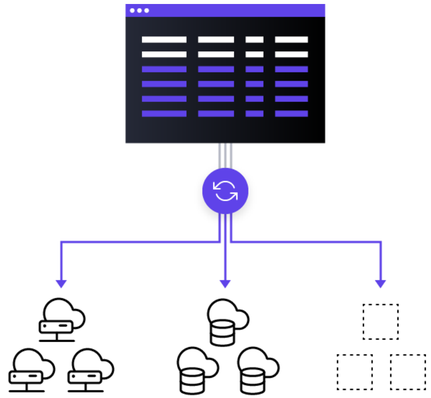
The Challenge

Manual provisioning is error-prone and difficult to scale.

BEFORE

- **Reduced productivity** from manual workflows using Point-and-Click GUIs & APIs
- **Increased cost** with “cloud waste” or over provisioning
- **Increased risk** with more chances for human error and best practices are followed on a “best effort” basis using tribal knowledge

Use Case: **Infrastructure as Code**



The Solution

Automate provisioning using Terraform infrastructure as code.

AFTER

- **Increase productivity** & reduce time to provision from weeks to minutes with automated workflow
- **Control costs** systematically as users and applications scale
- **Reduce risk** and discover errors before they happen with code reviews and embed provisioning guardrails

Features

1 Infrastructure as code

Use HashiCorp Configuration Language (HCL), a human-readable & machine executable DSL, to define all resources needed to run an application.

2 VCS Connection

Write, version, review, collaborate on code.

ENTERPRISE

Automate and trigger runs through connection between Terraform and Major VCS providers.

3 Workspaces

Decompose monolithic infrastructure into logical micro-infrastructures.

ENTERPRISE

Map functional responsibilities to individual workspaces & interlink workspace outputs via APIs.

4 Variables

Granular variables allow easy reuse of code to scale resources, regions, etc.

ENTERPRISE

All variables are securely stored and retrieved as needed during the provisioning process.

5 Runs and State

Two-phased provisioning automation: a plan (dry run) & apply (execution). Output stored in state file.

ENTERPRISE

Remote runs (GUI, CLI, or API executed) and state storage.

6 Policy as Code

Sentinel, a policy as code framework to automate policy controls into workflows.

ENTERPRISE

Create every provisioning run to enforce security, compliance, and operational best practices.

Feature: **Infrastructure as Code**

CHALLENGE SOLUTION RESULTS

Manual Provisioning

Provisioning infrastructure through Point and Click GUI's or custom scripts is slow, error-prone, and inefficient.

Feature: **Infrastructure as Code**

CHALLENGE

SOLUTION

RESULTS

Infrastructure as Code

With HashiCorp Configuration Language (HCL), infrastructure and services from any provider can be provisioned in a codified, secure, and automated fashion.

- HashiCorp Configuration Language (HCL) is human readable and machine executable
- Declarative, Turing-Complete language
- Automate, version, and collaborate on infrastructure

EXAMPLE FOR GOOGLE COMPUTE INSTANCE AND DNSIMPLE RECORD

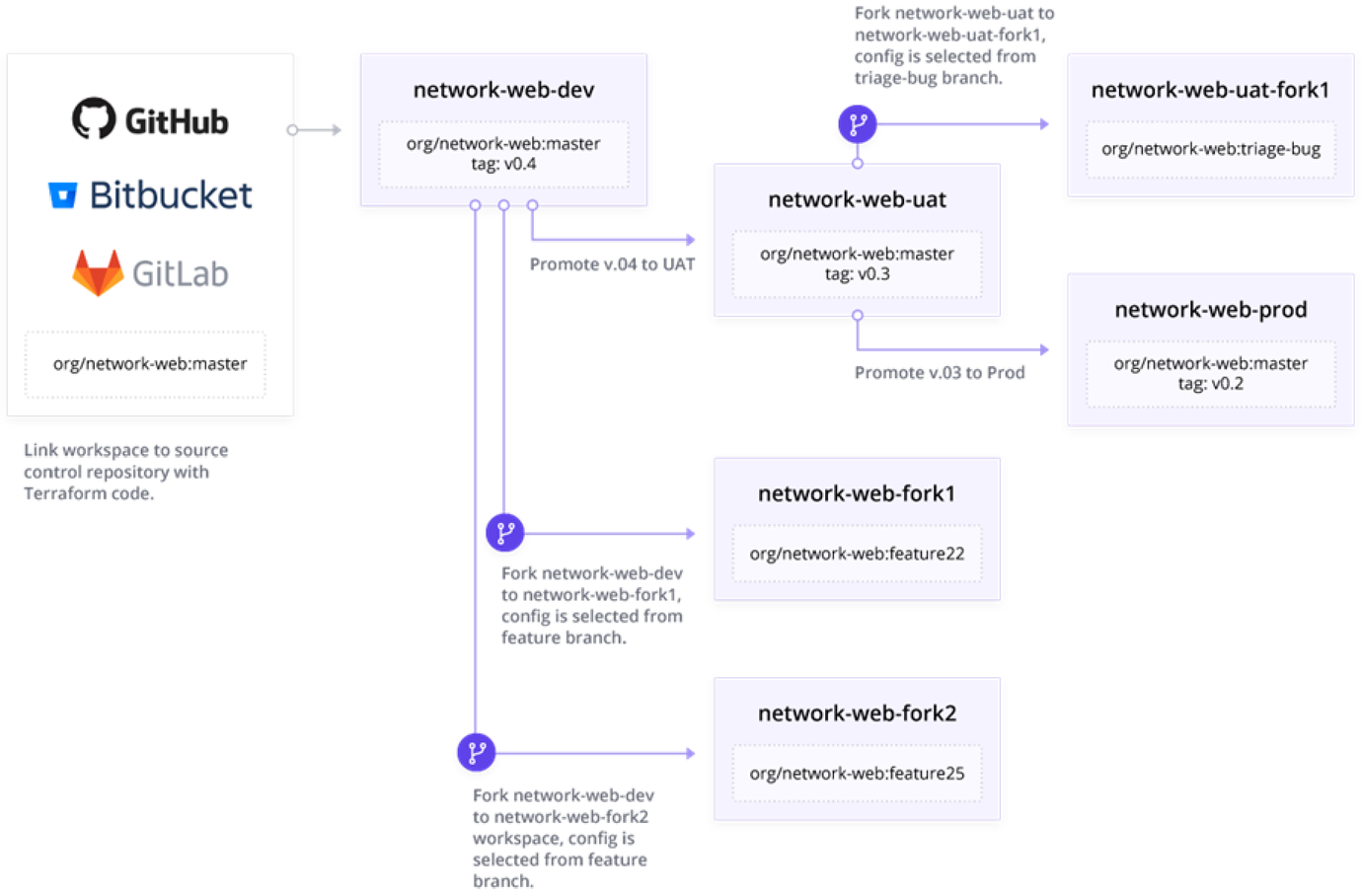
```
resource "google_compute_instance" "server" {
  name         = "server"
  machine_type = "g1-small"
  zone         = "us-central1-a"
  disk {
    image = "ubuntu-1404-trusty-v20160114e"
  }
}

resource "dnsimple_record" "hello" {
  domain = "example.com"
  name   = "server"
  value  = "${google_compute_instance.server.network_interface.0.address}"
  type   = "A"
}
```

Feature: **Infrastructure as Code**

CHALLENGE | **SOLUTION** | RESULTS

Benefits of Infrastructure as Code

- Versioning
- Collaboration
- Promotion
- Forking



Feature: **Infrastructure as Code**

CHALLENGE

SOLUTION


RESULTS

Increase Agility

With Infrastructure as Code the manual effort involved in provisioning infrastructure is significantly reduced. Code can be reused and modified as many times as necessary.

Reduce Risk

Minimize manual, error-prone work and reuse known-working and known-secure infrastructure configurations across an organization.

Reduce Cost

By defining proper infrastructure footprints in modules, teams provision the infrastructure they need without wasteful and costly over provisioning.

Feature: **Workspaces**

CHALLENGE SOLUTION RESULTS BEST PRACTICES

Unbounded Configurations

Though monolithic configurations function for a short time, they quickly grow unwieldy and confusing to new contributors.

Feature: **Workspaces**

Workspaces

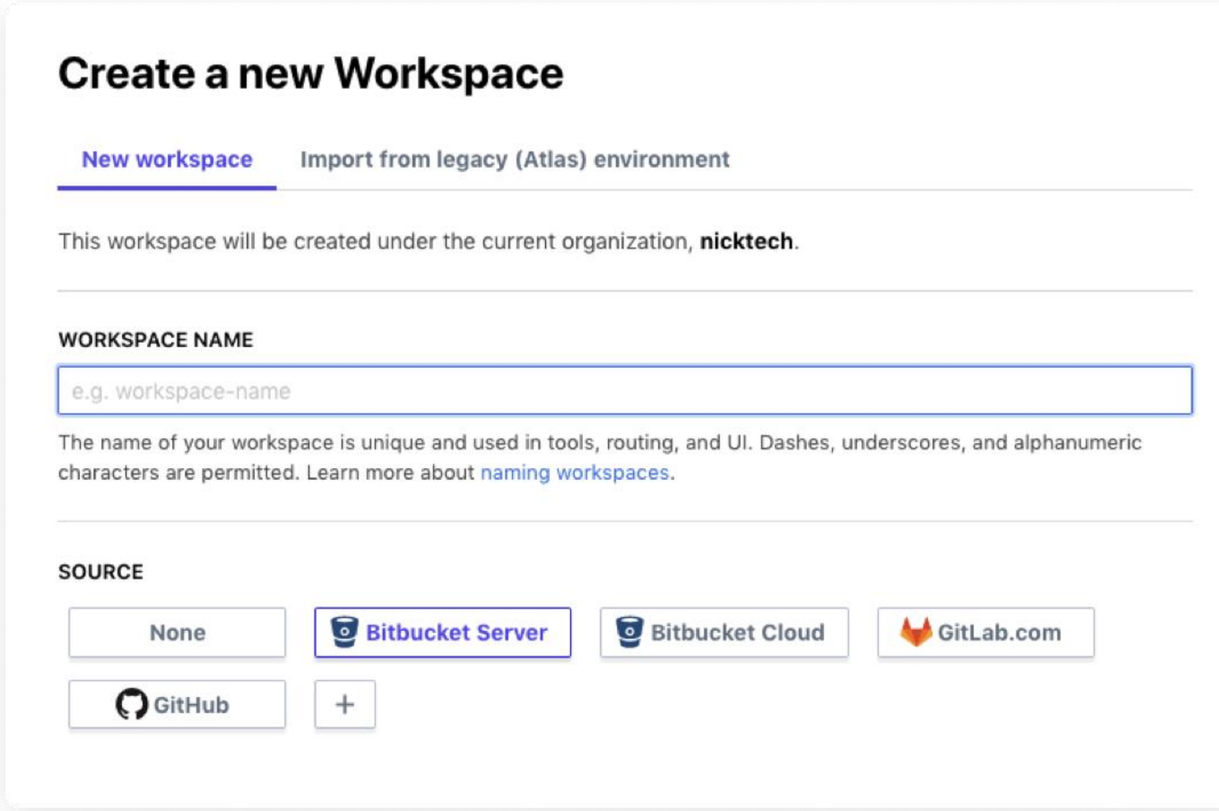
A workspace is a collection of everything Terraform needs to run: a configuration file, variables, and state data.
Workspaces offer powerful decomposition for monolithic configurations to match your organization and application structures.

OSS

- Workspaces are independent state files

ENTERPRISE

- Remote, persistent shared resources
- Access controls:
- Admin, Write, Read-Only, Plan-Only
- VCS Integration

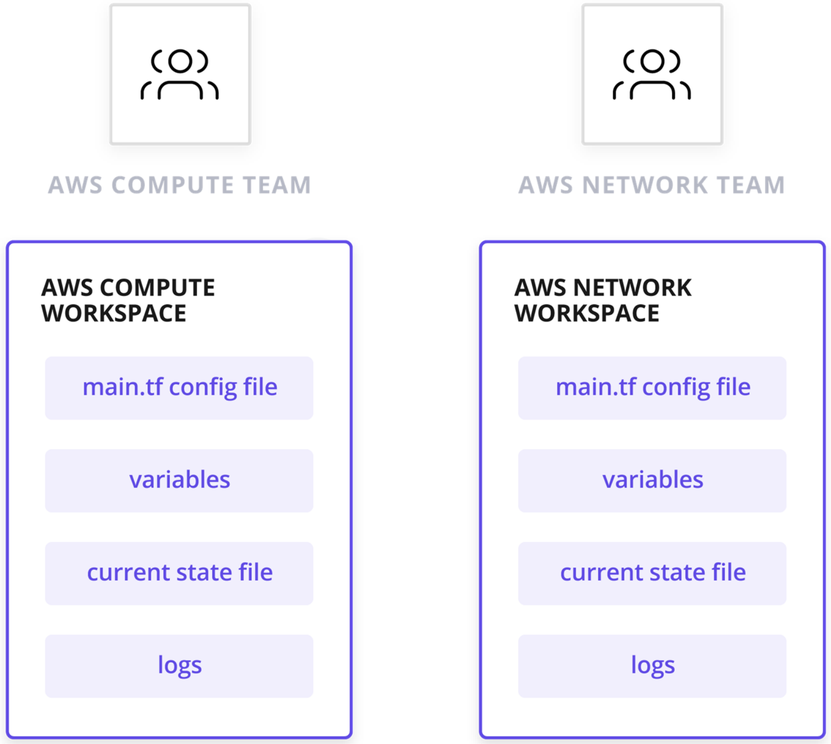


Feature: **Workspaces**

CHALLENGE | **SOLUTION** | RESULTS | BEST PRACTICES

Use Workspaces To:

- Organize and decompose monolithic infrastructure into micro-infrastructures
- Match the organization of your application or teams with your infrastructure



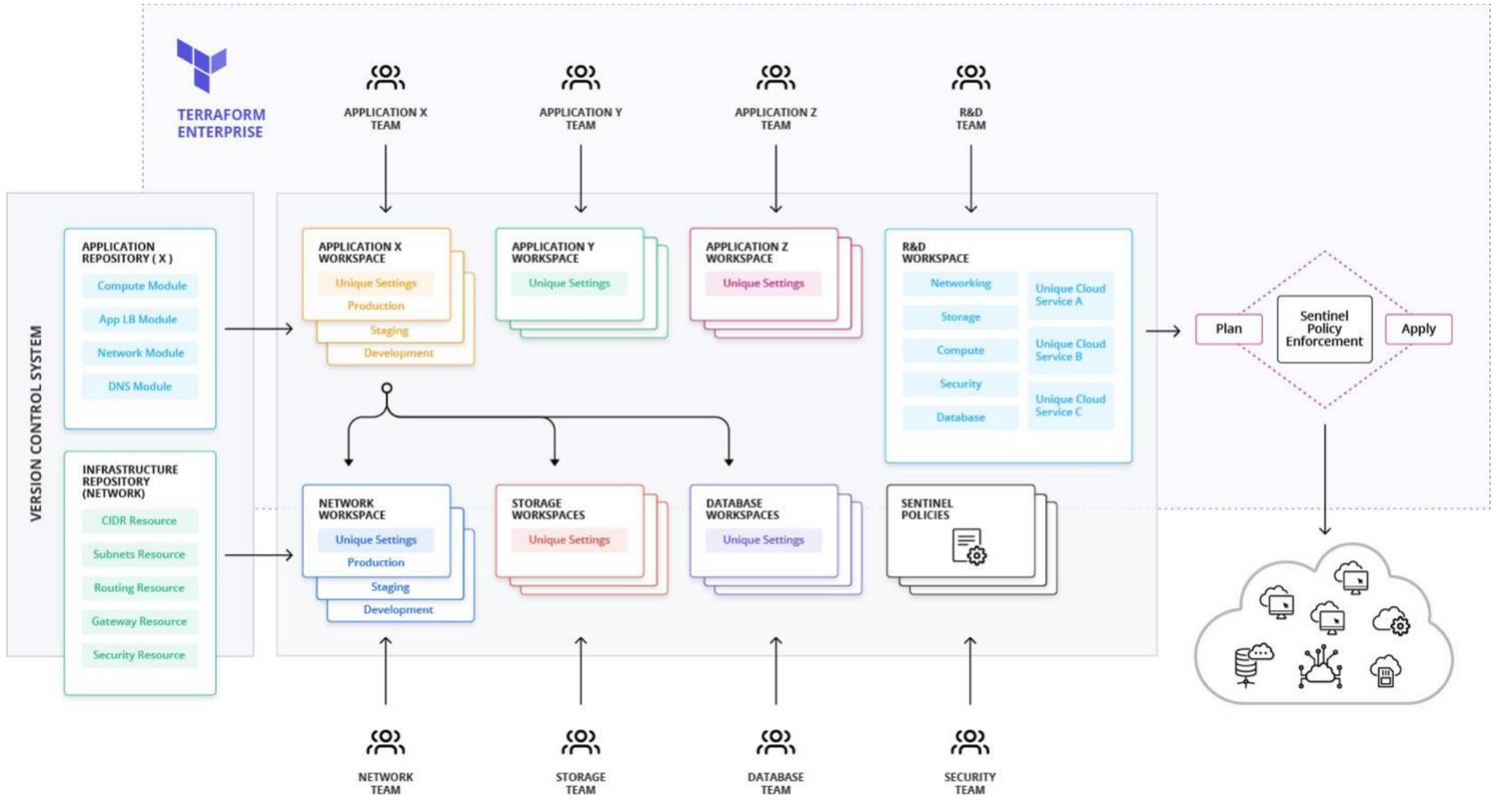
Feature: **Workspaces**

Functional Workspaces

- “Micro-infrastructures” are linked to create the complete infrastructure for the application

ENTERPRISE

- Remote Backend allows workspaces to refer to each other’s output variables



Feature: **Workspaces**

Increase Agility

Decomposing monolithic configurations into logical workspaces allows for in-parallel collaboration on infrastructure and quicker time-to-contribution for new team members.

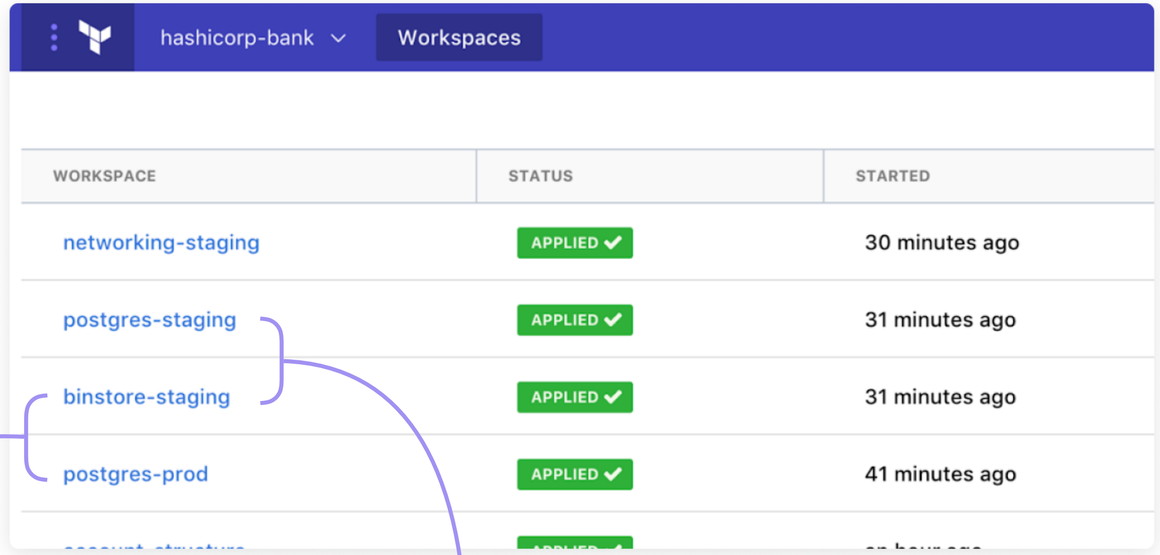
Reduce Risk

Impact from errors or breaks in code can be minimized and isolated to a single workspace as opposed to bringing down an entire configuration.

Feature: **Workspaces**

Workspace Best Practices

- One Workspace Per Environment Per Terraform Configuration
- Name your workspaces with both their component and their environment
- Use per-workspace access controls to delegate ownership of components and regulate code promotion across environments



Create a parallel, distinct copy of a set of infrastructure in order to test a set of changes before modifying the main production infrastructure.

Use multiple separate Terraform configurations that correspond with suitable architectural boundaries within the system so that different components can be managed separately and, if appropriate, by distinct teams.

Feature: **Variables**

CHALLENGE SOLUTION RESULTS

Modifying Code for Reuse

Infrastructure as Code offers powerful reusability, however, manipulating configuration files for even small changes is a slow process that exposes the opportunity for large errors.

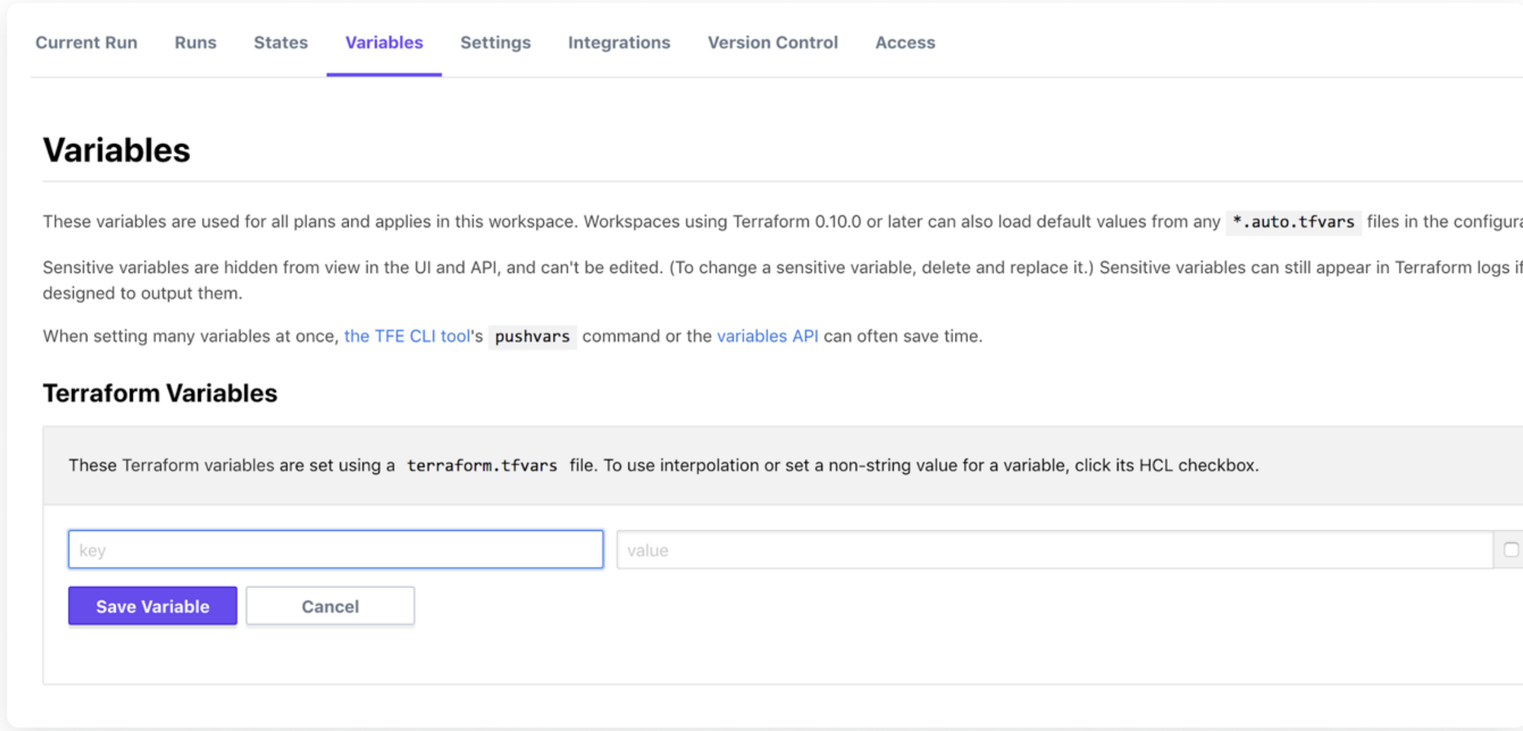
Feature: **Variables**

CHALLENGE | **SOLUTION** | RESULTS

Variables

By writing infrastructure as code with variables, operators and developers can easily customize infrastructure as code without opening a text editor.

- Input variable parameters for Terraform configurations or modules

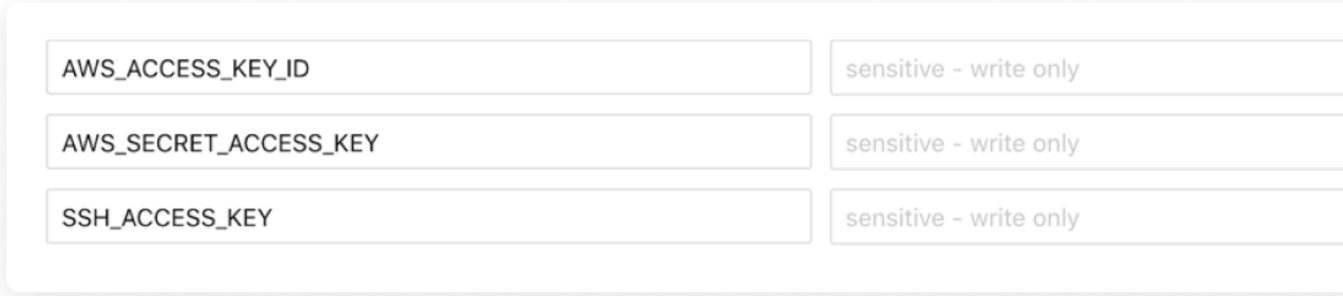


Feature: **Variables**

CHALLENGE SOLUTION RESULTS

Secure Variable Storage

- Share and manage access to variables in your organization
- Connect Terraform to other applications or services without compromising credential security
- Terraform Vault provider integration for secrets management



Feature: **Variables**

CHALLENGE SOLUTION **RESULTS**

Increase Agility

Quickly reusing infrastructure as code files to suit a variety of purposes and deployments increases developer and operator agility to deploy new applications and infrastructure.

Reduce Risk

Minimize manual editing of Infrastructure as Code files without sacrificing reusability.

Feature: **VCS Connection**

CHALLENGE SOLUTION RESULTS

Manual Version Control

Version control offers powerful iteration and forking for Infrastructure as Code configurations. However, manual versioning is cumbersome and error prone.

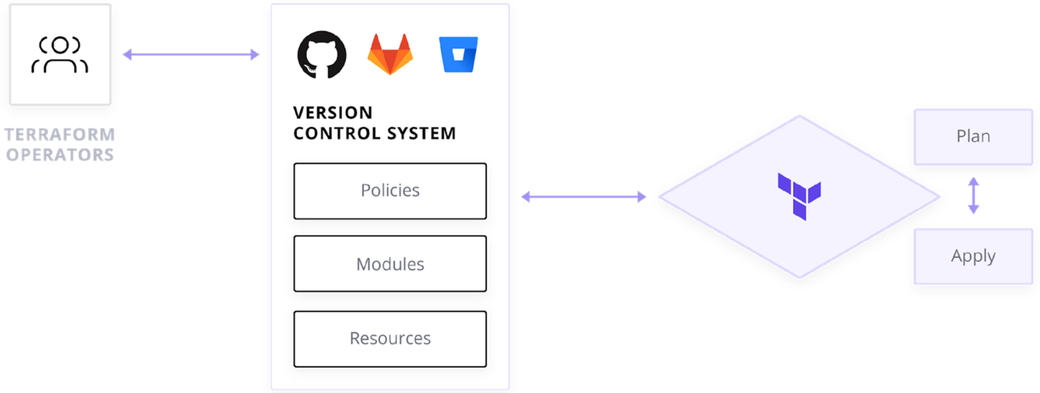
Feature: **VCS Connection**

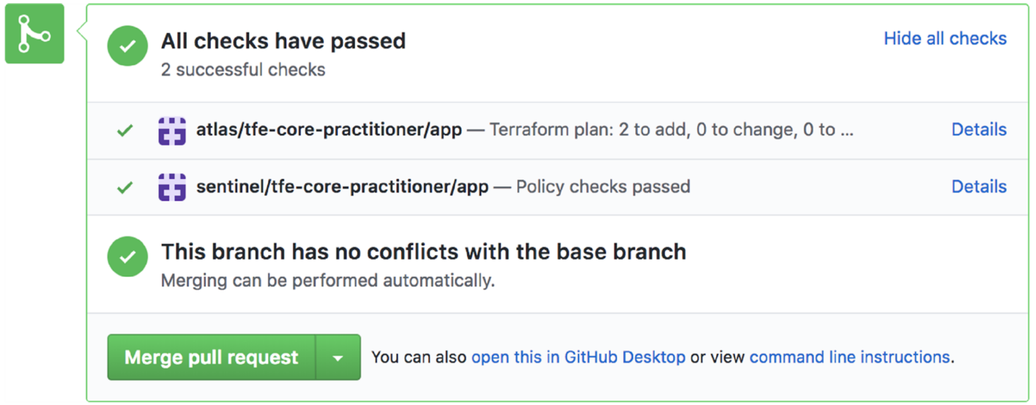
CHALLENGE | **SOLUTION** | RESULTS

VCS Connection

Terraform connects to the major VCS providers allowing for automated versioning and running of configuration files.

- Integrates into existing VCS workflow
- Automatically trigger runs and policy checks upon pull requests
- Support for BitBucket, Gitlab, and Github
- Default Workspaces to VCS pairing





Feature: **VCS Connection**

CHALLENGE

SOLUTION


RESULTS

Increase Agility

Versioned code allows for continual improvement without breaking changes, while tight integration with Terraform means that code can be automatically tested against policy and ran.

Reduce Risk

Reducing manual effort of versioning and running those versions results in both fewer errors and quicker addressing of errors when they occur.

Feature: **Runs and State**

CHALLENGE SOLUTION RESULTS

Knowing the State of Infrastructure

Provisioning infrastructure as code is fast and efficient, but rapid changes can be difficult to predict and track.

Feature: **Runs and State**

CHALLENGE **SOLUTION** RESULTS

Runs

With a two-phase run (plan & apply) terraform gives insight into the state of infrastructure before it is provisioned.

- Terraform runs include separate plan and apply phases.
- Plan allows users to see the changes Terraform will make and review them before they are made.
- Apply command makes those changes

State

State files are managed by Terraform, providing understanding of infrastructure as it is currently as well as all previously provisioned configurations.

- State is Terraform's understanding of infrastructure given the most recent apply

```
Terraform will perform the following actions:

  + aws_instance.i-010eae58aa1e15108
      id:                           <computed>
      ami:                          "ami-46e1f226"
      associate_public_ip_address:  <computed>
      availability_zone:            "us-west-1c"
      ebs_block_device.#:           <computed>
      ephemeral_block_device.#:     <computed>
      get_password_data:            "false"
      instance_state:               <computed>
      instance_type:                "t2.micro"
      ipv6_address_count:           <computed>
      ipv6_addresses.#:             <computed>
      public_ip:                    <computed>
      root_block_device.#:          <computed>
      security_groups.#:            <computed>
      source_dest_check:            "true"
      subnet_id:                    "subnet-0a531dcc52013e269"
      tags.%:                       "2"
      tags.TTL:                     "1"
      tags.owner:                   "Anthony D"
      tenancy:                      <computed>
      volume_tags.%:                <computed>
      vpc_security_group_ids.#:     <computed>


Plan: 1 to add, 0 to change, 0 to destroy.
```

Terraform understands which values need to be defined before the run, versus which values will be computed at runtime

```
Terraform v0.11.7

Initializing plugins and modules...
aws_instance.i-010eae58aa1e15108: Creating...
  ami:                          "" => "ami-46e1f226"
  associate_public_ip_address:  "" => "<computed>"
  availability_zone:            "" => "us-west-1c"
  ebs_block_device.#:           "" => "<computed>"
  ephemeral_block_device.#:     "" => "<computed>"
  get_password_data:            "" => "false"
  instance_state:               "" => "<computed>"
  instance_type:                "" => "t2.micro"
  ipv6_address_count:           "" => "<computed>"
  ipv6_addresses.#:             "" => "<computed>"
  key_name:                     "" => "<computed>"
  network_interface.#:          "" => "<computed>"
  network_interface_id:         "" => "<computed>"
  password_data:                "" => "<computed>"
  placement_group:              "" => "<computed>"
  primary_network_interface_id: "" => "<computed>"
  private_dns:                  "" => "<computed>"
  private_ip:                   "" => "<computed>"
  public_dns:                   "" => "<computed>"
  public_ip:                    "" => "<computed>"
  root_block_device.#:          "" => "<computed>"
  security_groups.#:            "" => "<computed>"
  source_dest_check:            "" => "true"
  subnet_id:                    "" => "subnet-0a531dcc52013e269"
  tags.%:                       "" => "2"
  tags.TTL:                     "" => "1"
  tags.owner:                   "" => "Anthony D"
  tenancy:                      "" => "<computed>"
  volume_tags.%:                "" => "<computed>"
  vpc_security_group_ids.#:     "" => "<computed>"
aws_instance.i-010eae58aa1e15108: Still creating... (10s elapsed)
aws_instance.i-010eae58aa1e15108: Still creating... (20s elapsed)
aws_instance.i-010eae58aa1e15108: Creation complete after 23s (ID: i-04df1

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Feature: **Runs and State**

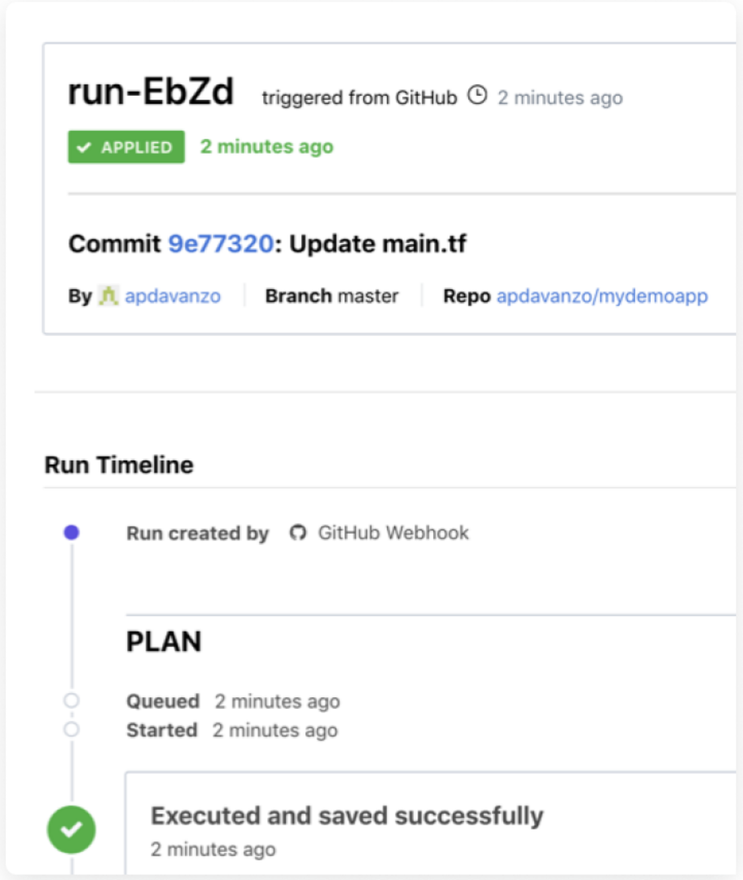
CHALLENGE | **SOLUTION** | RESULTS

Remote Operations

- Trigger Terraform runs remotely:
  - Pull request in VCS
  - API call in CI/CD pipeline
  - CLI
- Automatically coordination of runs by multiple users
- Maintain core workflow from OSS

ENTERPRISE

- Enhanced remote runs and state storage empowering collaboration
- Runs triggered from local CLI are shown in UI



Feature: **Runs and State**

CHALLENGE SOLUTION **RESULTS**

Reduce Risk

With the ability to see the changes Terraform will make before they're made, organizations have the ability to confirm changes and reduce errors.

With detailed state data from current and previous configurations organizations have actionable insight into the state of there infrastructure without having to manage it.

Feature: **Policy as Code**

CHALLENGE SOLUTION RESULTS

Guardrails Around Provisioning

Rapid provisioning opens up tremendous possibility, but organizations need to maintain security and prevent over provisioning.

Feature: **Policy as Code**

CHALLENGE


SOLUTION

RESULTS

Policy as Code

With programmatic policy as code, custom governance can be enforced at the same rate as infrastructure is provisioned.

- Codifying policies automates guardrails around provisioning
- Policy checks built into the provisioning workflow
- Use policy to enforce best-practices, security measures, or compliance

Enforcement Levels

- *Advisory*: Warns when a policy breaks
- *Soft Mandatory*: Provision needs to override policy to break it
- *Hard Mandatory*: Provisioning not allowed to break policy

RESTRICTING MACHINE TYPE IN GCP

```
allowed_machine_types = [
  "n1-standard-1",
  "n1-standard-2",
  "n1-standard-4",
]
```

Feature: **Policy as Code**

CHALLENGE SOLUTION **RESULTS**

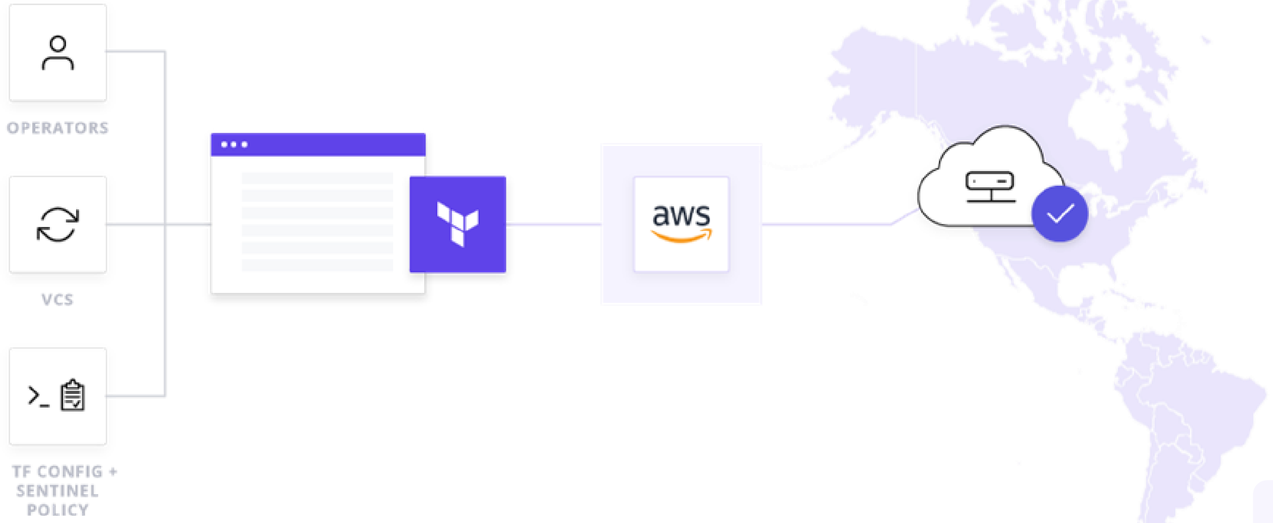
Reduce Risk

Without stunting productivity, policy can be enforced across infrastructure ensuring all infrastructure meets best-practices for security and functionality.

Reduce Cost

Cost-sensitive policies can be written and applied preventing over provisioning.

Workflow after Phase One of adoption

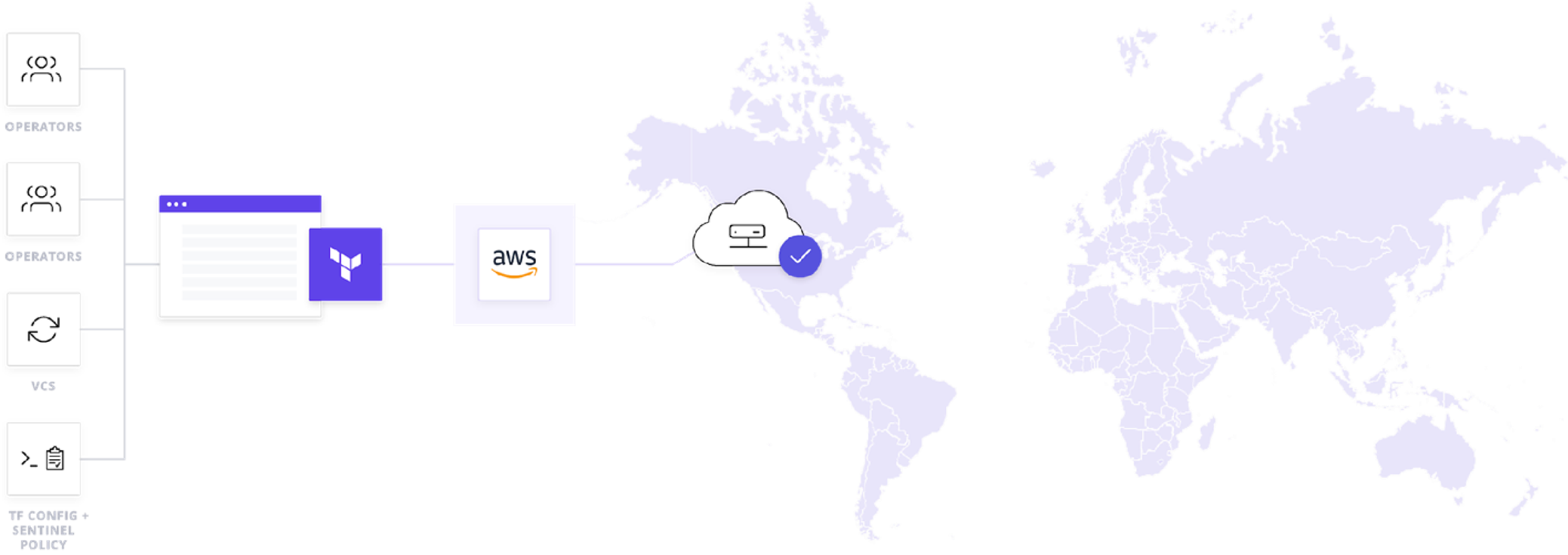


Practitioner Workflow:

Write & Test - An operator responsible for writing infrastructure as code will create a workspace, write code, and run a plan to test that configuration

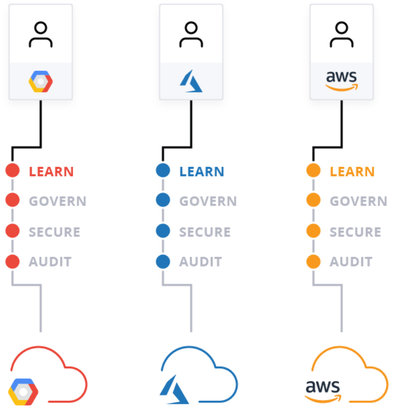
Review & Apply - An operator responsible for QA of infrastructure can review the workspace above, and apply it.

Adoption from one team to multiple teams



Use Case
Multi-Cloud Management

Use Case: **Multi-Cloud Management**



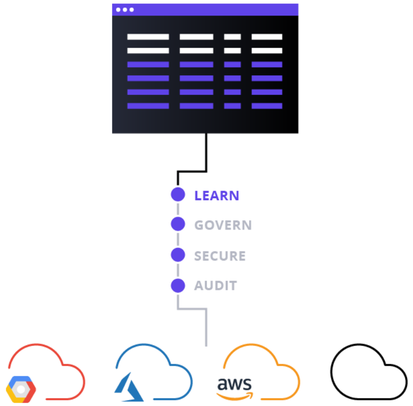
The Challenge

Different infrastructure providers creates silos of teams, tools, and resources.

BEFORE

- **Increased costs** from vendor lock-in and premium pricing
- **Reduced productivity** with multiple workflows/APIs to learn and long lead times to support new LOBs.
- **Increased risk** with multiple workflows to secure, govern, and audit.

Use Case: **Multi-Cloud Management**



The Solution

One workflow to learn, secure, govern, & audit any infrastructure while preserving uniqueness of each.

AFTER

- **Reduce costs** by eliminating vendor lock-in and take advantage of credits & competitive pricing.
- **Increase productivity** with a consistent workflow and reduce time to on-board vendors & support new LOBs.
- **Reduce risk** with a single workflow to secure, govern, and audit.

Features

1 Consistent Workflow

One workflow to provision infrastructure.

ENTERPRISE

One workflow to learn, secure, audit, and govern infrastructure.

2 Providers & Community

150+ available providers for a broad set of common infrastructure. Ability to create new and custom providers. 1200+ community members.

3 Leverage Provider APIs

Providers leverage infrastructure-specific CRUD APIs to preserve unique capabilities for each provider.

4 Multi-Cloud Policy & Governance

ENTERPRISE

Embedded codified policies into TFE and apply during a Terraform run to enforce security, compliance, and operational best practices.

Feature: **Consistent Workflow**

CHALLENGE SOLUTION RESULTS

Heterogenous Workflows

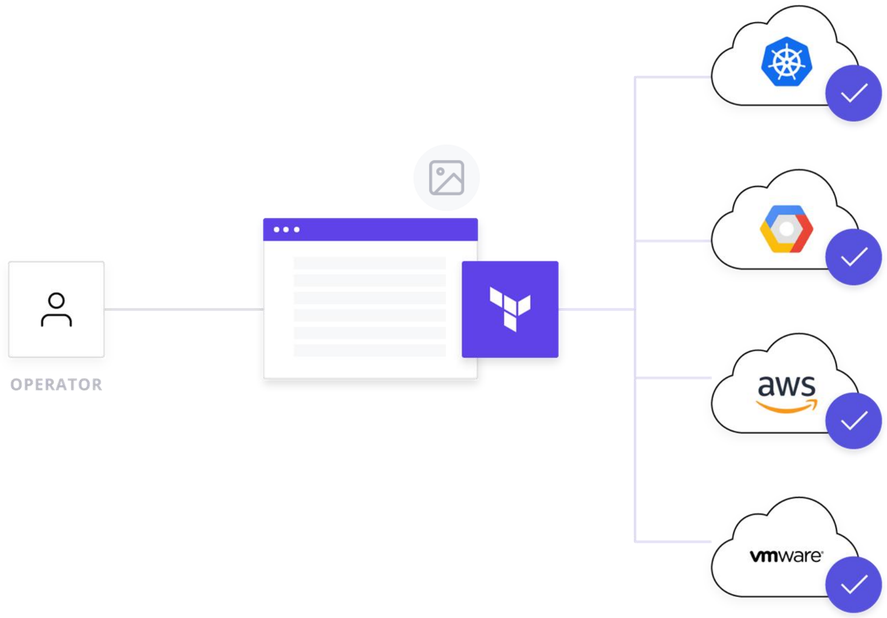
As organizations adopt the technologies that best suit their needs, time-to-value for these technologies is significantly delayed by unique learning and on boarding required for each new technology.

Feature: **Consistent Workflow**

CHALLENGE | **SOLUTION** | RESULTS

Consistent Workflow

Because Terraform is extensible to any cloud provider and service, organizations are able to quickly leverage new technology with the same learned, secured, audited, and governed workflow.



Feature: **Consistent Workflow**

Increase Agility

As operators become fluent in Terraform, they are able to onboard new technologies quicker, resulting in improved time-to-value for new tools.

Reduce Risk

With a single workflow to secure, govern, learn, and audit, organizations develop a better security posture while reducing the likelihood of risk from adopting new tooling.

Reduce Cost

With a consistent workflow organizations significantly reduce the time they are paying for tooling without using it.

Feature: **Providers & Community**

CHALLENGE SOLUTION RESULTS

Adopting Heterogenous Technologies

Organizations want to adopt the technologies that best suit their needs. However, adopting new tooling can be difficult, and requires additional maintenance per tool. Some platforms offer access to multiple tools, but by doing so abstract away the unique functionality of each technology.

Feature: **Providers & Community**

CHALLENGE **SOLUTION** RESULTS

Providers

Providers, extensible to any cloud or service with an API, enable Terraform to provision diverse services without abstracting functionality.

- 160+ Providers and Services
- 1000s of Resources
- Extensibility for custom providers if needed
- Supported by HashiCorp, providers, and the community

AliCloud	Archive	AWS
Azure	Azure Stack	BitBucket
CenturyLinkCloud	Chef	Cloudflare
CloudScale.sh	CloudStack	Cobbler
Consul	Datadog	DigitalOcean
DNS	DNSimple	Docker
Dyn	External	Fastly
GitHub	Gitlab	Google Cloud
Heroku	Kubernetes	MySQL
New Relic	Nomad	OpenStack
Oracle Cloud Platform	Oracle Public Cloud	Packet
Palo Alto Networks	PostgreSQL	RabbitMQ
Rundeck	SoftLayer	UltraDNS
Vault	VMware NSX-T	VMware vCloud
VMware vSphere		

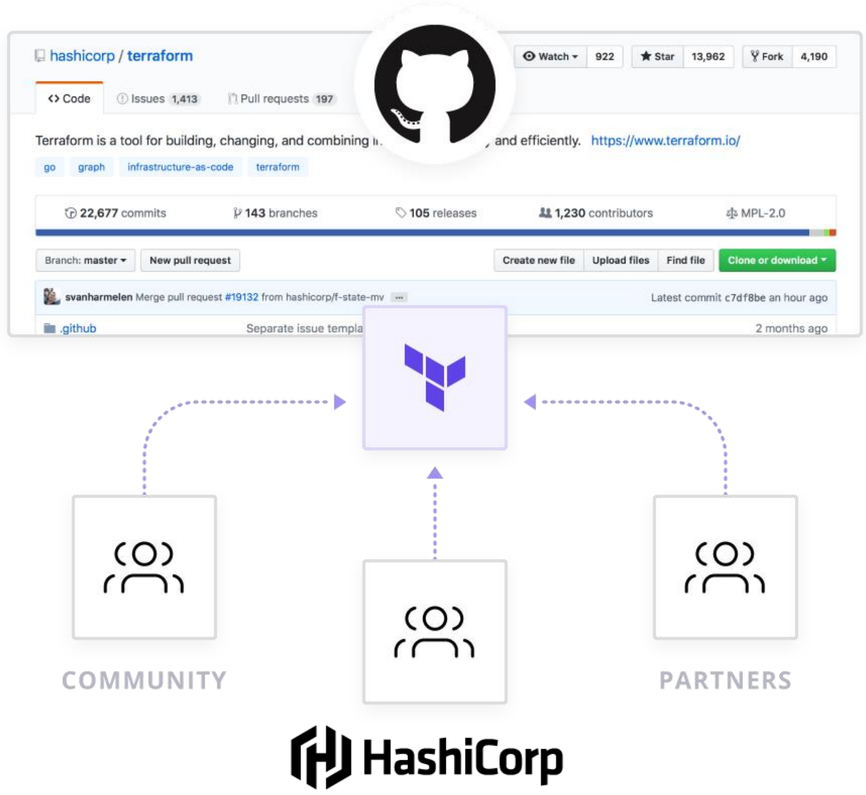
Feature: **Providers & Community**

CHALLENGE **SOLUTION** RESULTS

Community

The large community supporting Terraform Open Source, ranging from independent developers to teams within large technology companies, provides constant improvement to the platform and the provider plugins.

- 90k+ downloads/month
- 1.2k+ Github contributors



Feature: **Providers & Community**

CHALLENGE

SOLUTION


RESULTS

Increase Agility

Terraform frequently has zero-day support for new features from cloud providers and services through the combination of extensible, customizable providers and the actively contributing community enabling organizations to quickly take advantage of the latest technologies.

Reduce Cost

Through quick adoption of new technologies, organizations realize the respective benefits those technologies bring to their business driving revenue and productivity.

Reduce Risk

The large community supporting Terraform constantly flags bugs ensuring powerful hardening against risk and attack.

Feature: **Leverage Provider APIs**

CHALLENGE SOLUTION RESULTS

Interacting with Multiple Clouds and Services

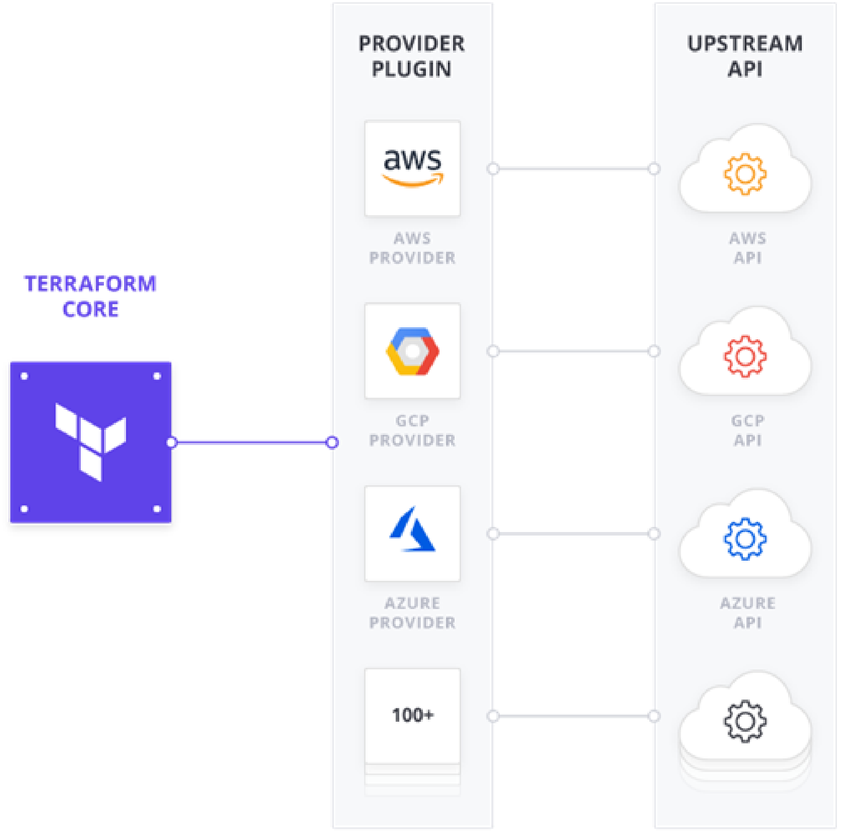
To address the heterogenous set of tooling that powers modern businesses, some organizations are inclined to abstract away differences between providers and services, sacrificing the unique capabilities that each offer.

Feature: **Leverage Provider APIs**

CHALLENGE | **SOLUTION** | RESULTS

Leverage Provider APIs

Terraform directly leverages Provider APIs in order to take advantage of the full robust feature set that clouds and services offer.



Feature: **Leverage Provider APIs**

CHALLENGE SOLUTION **RESULTS**

Increase Agility

Directly leveraging provider APIs without learning new tooling drives development and operations teams to adopt new tooling and technologies with ease.

Feature: **Multi-Cloud Governance**

CHALLENGE SOLUTION RESULTS

Guardrails Around Multi-Cloud Provisioning

Rapidly provisioning multi-cloud infrastructure quickly and efficiently opens up tremendous possibility, but organizations need to maintain security and prevent over provisioning.

Feature: **Multi-Cloud Governance**

CHALLENGE **SOLUTION** RESULTS

Multi-Cloud Governance

With programmatic policy as code, custom governance across all providers and services can be enforced at the same rate as infrastructure is provisioned.

- Use policy to enforce best-practices, security measures, or compliance on every cloud and service

RESTRICTING MACHINE TYPE ACROSS PROVIDERS

```
# Restricting region in AWS
aws_region_valid = rule {
  all region_values as rv {
    rv == "us-east-1"
  }
}

# Restricting machine types in GCP
allowed_machine_types = [
  "n1-standard-1",
  "n1-standard-2",
  "n1-standard-4",
]

# Restricting publisher in Azure
allowed_publishers = [
  "MicrosoftWindowsServer",
  "RedHat",
]
```

Feature: **Multi-Cloud Governance**

CHALLENGE

SOLUTION


RESULTS

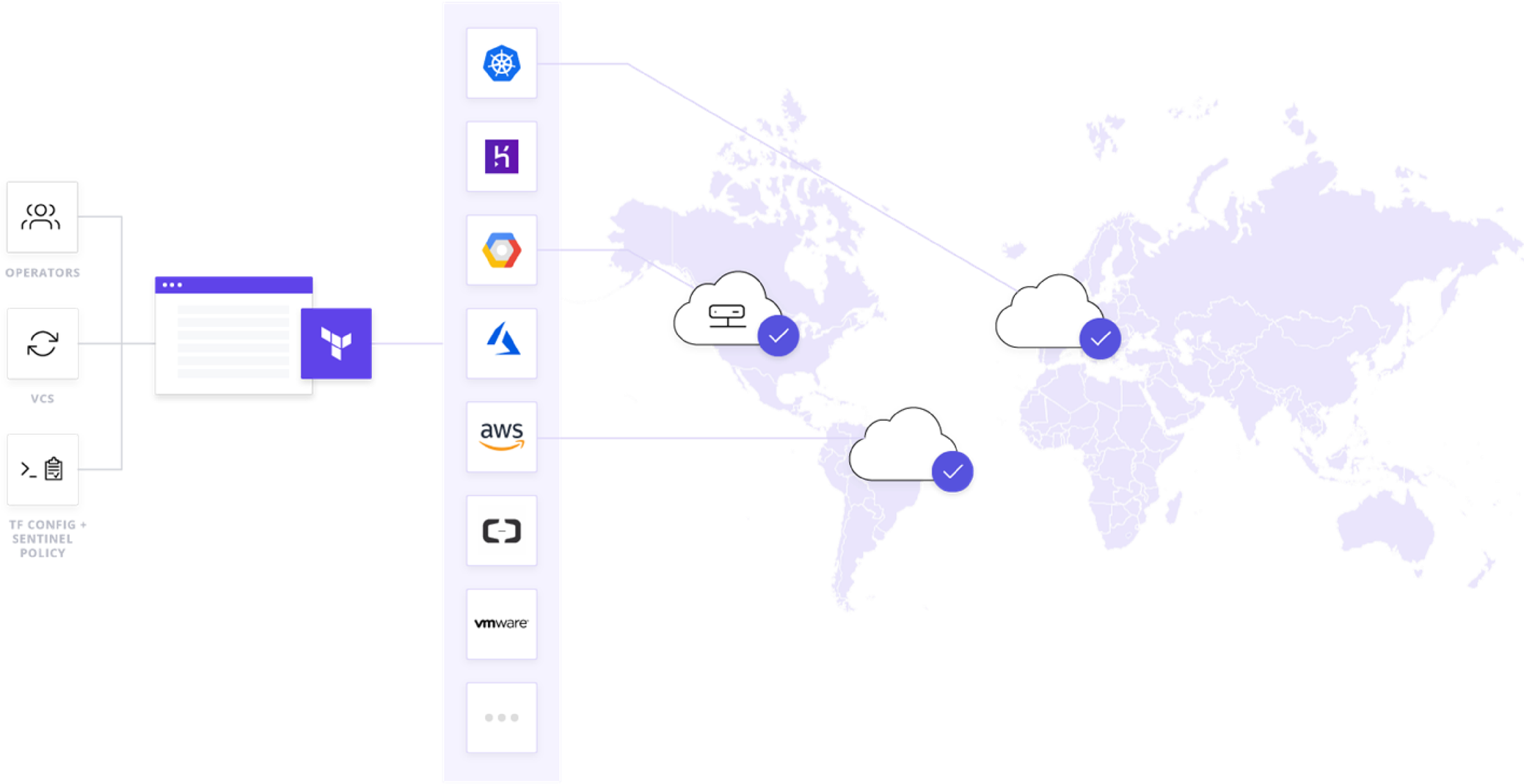
Reduce Risk

Without stunting productivity policy can be enforced across infrastructure ensuring all infrastructure meets best-practices for security and functionality.

Reduce Cost

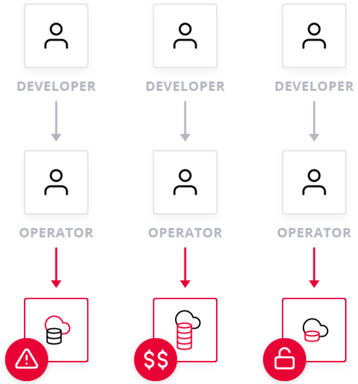
Cost-sensitive policies can be written and applied preventing over provisioning.

Workflow after Phase Two



Use Case
Self-service Infrastructure

Use Case: **Self-Service Infrastructure**



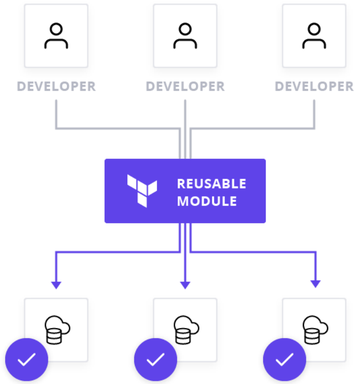
The Challenge

Developers have to wait for operators to provision and assign dedicated infrastructure.

BEFORE

- **Increased costs** from from over-provisioning and unused or orphaned infrastructure
- **Reduced productivity** from Developers waiting on operations to provision/scale infrastructure
- **Increased risk** from many users provisioning infrastructure and not following best practices

Use Case: **Self-Service Infrastructure**



The Solution

Library of versioned and validated infrastructure templates to be consumed for on-demand provisioning.

AFTER

- **Increase revenue** by reducing time developers wait for infrastructure and increase time building apps
- **Increase productivity** of operators with a workflow for developers to discover approved infrastructure
- **Reduce risk** with a single workflow to secure, govern, and audit regardless who provisions

Features

1 Modules

Encapsulate groups of resources to create reusable blocks of infrastructure.

2 Module Registry

A library of community modules from partners, contributors, and HashiCorp.

ENTERPRISE

Private registry of custom modules versioned, validated, and approved for use by the organization.

3 Configuration Designer

ENTERPRISE

A GUI-based workflow to combine modules and define variables for developers to create a custom workspaces without having to write infrastructure as code.

4 Full API

ENTERPRISE

Leverage the Terraform API to integrate the provisioning workflow into an existing CI/CD pipeline.

5 Team Management & SSO

ENTERPRISE

Create and manage an organization of separate business units, teams, and users with assigned roles and privileges.

6 Audit Logging

ENTERPRISE

Audit logs provide a central source of record to track all provisioned infrastructure.

Feature: **Modules**

CHALLENGE SOLUTION RESULTS

Non-Standardized Provisioning

Without templated infrastructure as code, either operators spend time manually fulfilling infrastructure requests or developers provision cloud resources for their applications without oversight or guardrails.

Feature: **Modules**

CHALLENGE **SOLUTION** RESULTS

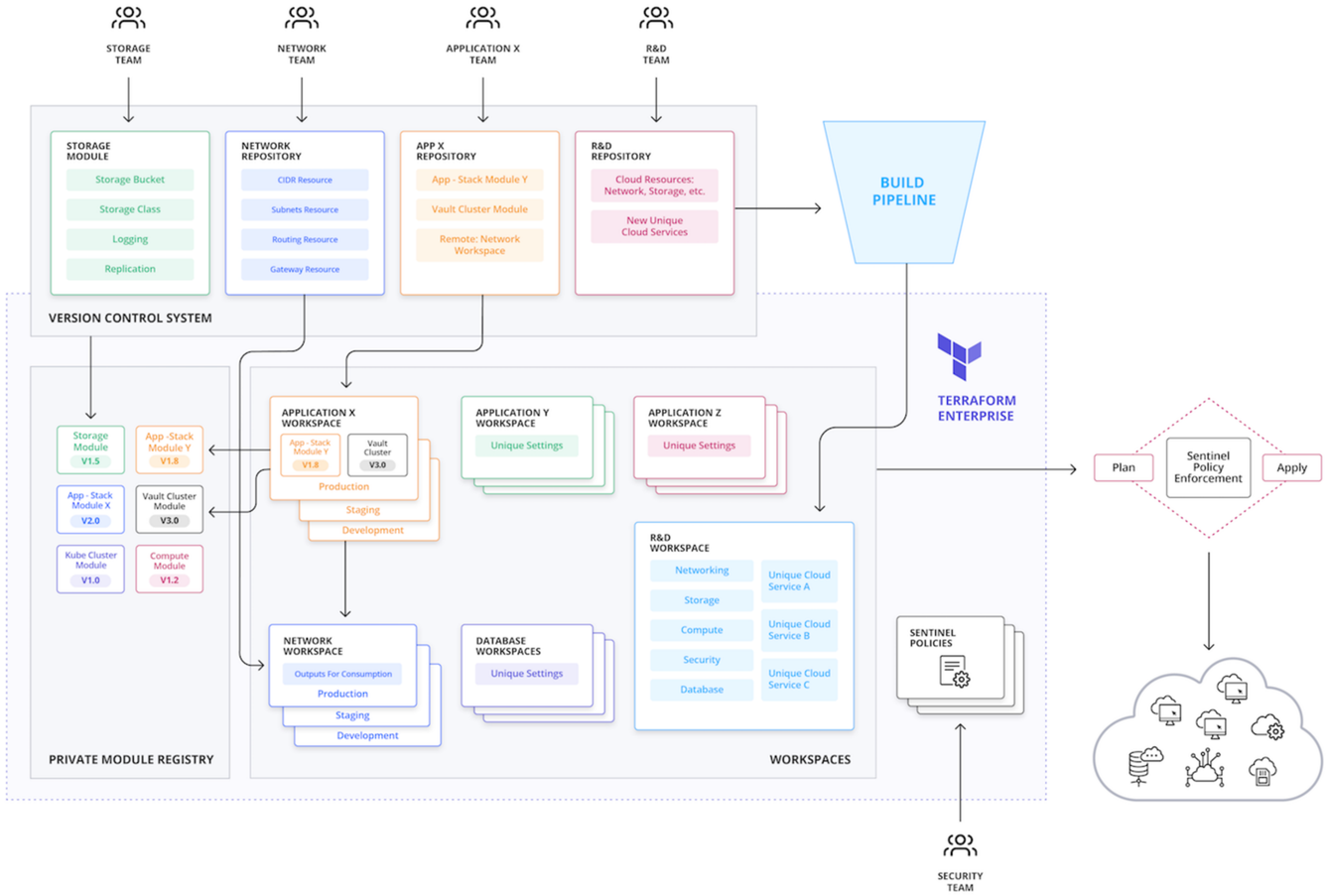
Modules

By creating hardened modules, operations teams empower their organization to efficiently provision vetted, secured, and standardized infrastructure.

- Reusable, templated infrastructure as code
- Customize as needed with variable inputs

Producer / Consumer Workflow

- Producers create modules from functioning workspaces
- Modules published to a registry for discovery
- Consumers leverage registry to create infrastructure as needed for applications



Feature: **Modules**

RESULTS

Increase Agility

By automating the provisioning process users are able to create infrastructure they need without the manual effort of custom scripting or point and click GUIs.

Reduce Risk

By building security practices into reusable modules, operations teams minimize the possibility for insecure infrastructure to be built.

Reduce Cost

By defining proper infrastructure footprints in modules, teams provision the infrastructure they need without wasteful and costly over provisioning.

Feature: **Module Registry**

CHALLENGE

SOLUTION

RESULTS

Storing and Sharing Configurations

Privately storing and sharing modules with peers, collaborators, or even privately across an organization is difficult.

Feature: **Module Registry**

CHALLENGE | **SOLUTION** | RESULTS

Module Registry

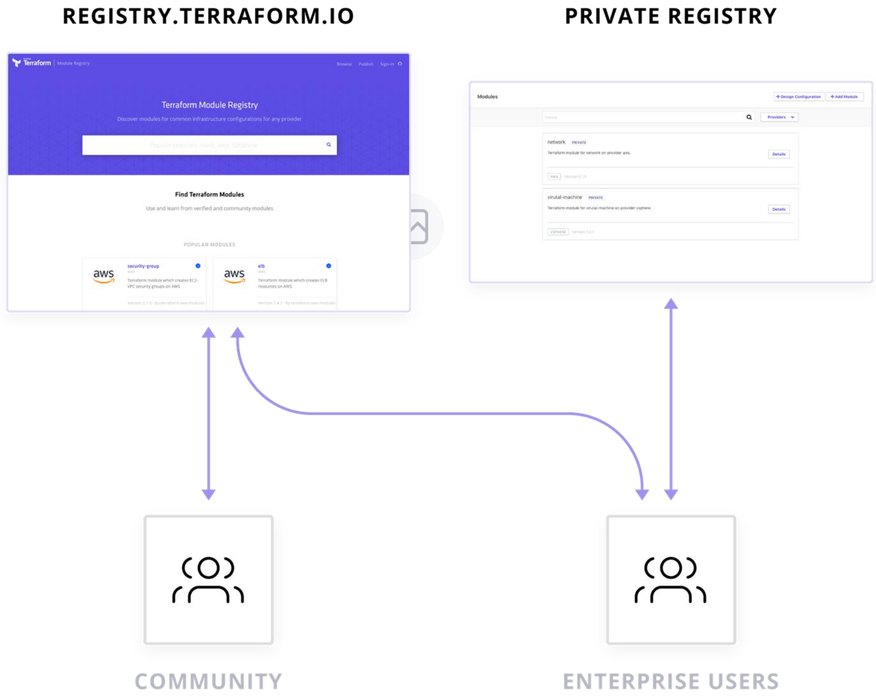
The public module registry offers the large community a repository to store and share modules.

- 700+ Modules
- Build & publish modules for general consumption

Private Module Registry

The private module registry, built into Terraform Enterprise, offers organizations a private repository to store and share modules internally.

- Modules can be created with best practices and operational efficiency built-in, i.e. tagging resources, setting TTL's, etc.



Feature: **Module Registry**

CHALLENGE SOLUTION **RESULTS**

Increase Agility

By leveraging modules built by both the community or privately shared within an organization, operators and developers can quickly customize and provision infrastructure they need.

Reduce Risk

With a repository of already built modules, teams can ensure all infrastructure is provisioned with best-practice security built-in.

Feature: **Configuration Designer**

CHALLENGE SOLUTION RESULTS

Customizing Infrastructure

Many developers and operators can quickly learn and use HCL, however, occasions arise when there is a need to quickly customize and provision infrastructure without knowing infrastructure as code.

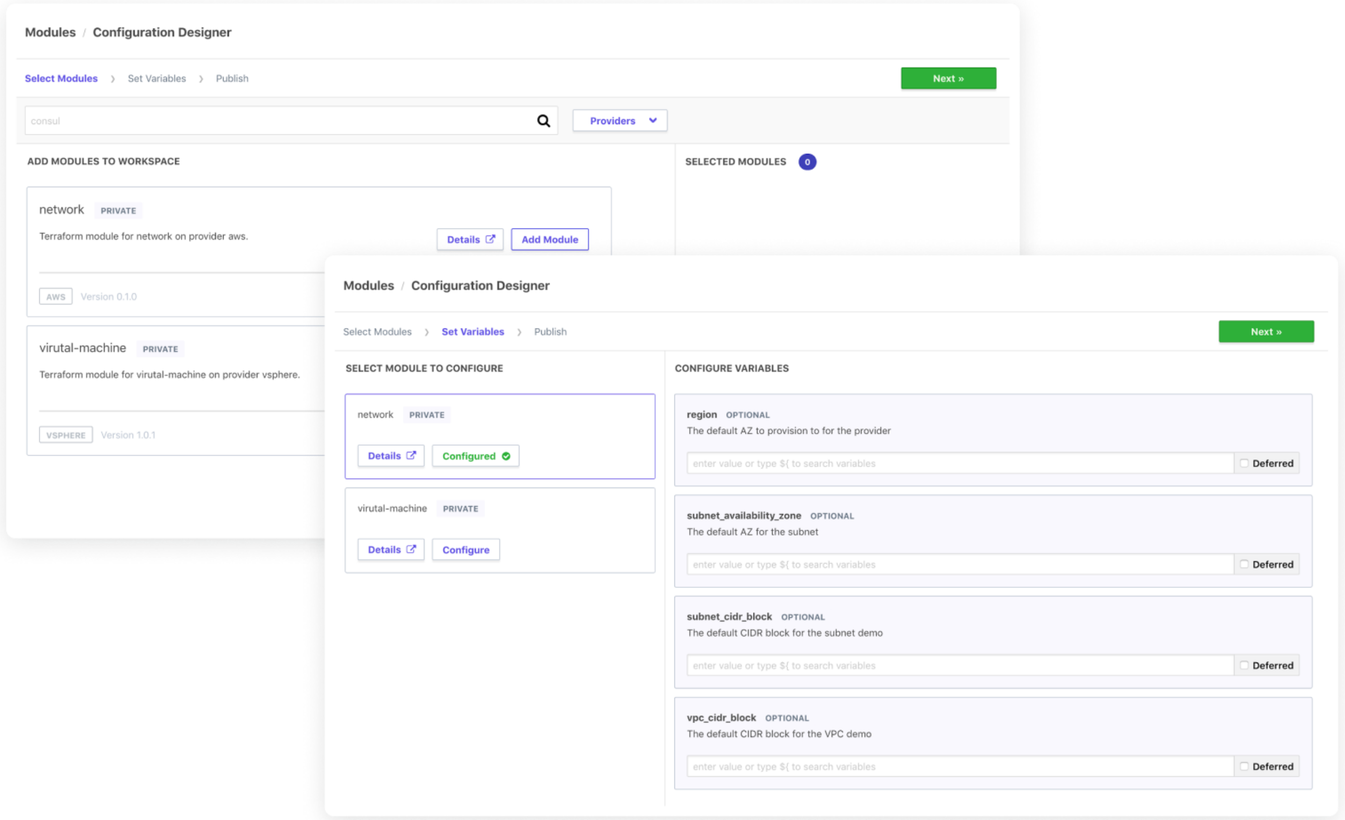
Feature: **Configuration Designer**

CHALLENGE **SOLUTION** RESULTS

Configuration Designer

Configuration designer allows users to choose modules from the public or private registry and customize them to their needs through a graphical user interface.

- Ability to discover, combine, and assign variables to build custom infrastructure as code
- Effectively use Infrastructure as Code without deep provider knowledge



Feature: **Configuration Designer**

CHALLENGE

SOLUTION


RESULTS

Increase Agility

Operators tasked with provisioning infrastructure enable the larger organization to provision and customize infrastructure on-demand improving the speed of application delivery.

Reduce Cost

Enabling more users to provision necessary infrastructure lightens the load on expensive developer and operator teams, resulting in reduced costs for deployment.

Feature: **Full API**

CHALLENGE SOLUTION RESULTS

Integrating into Existing Pipelines

Many organizations want to leverage Terraform through existing CI/CD pipelines.

Feature: **Full API**

CHALLENGE | **SOLUTION** | RESULTS

Full API

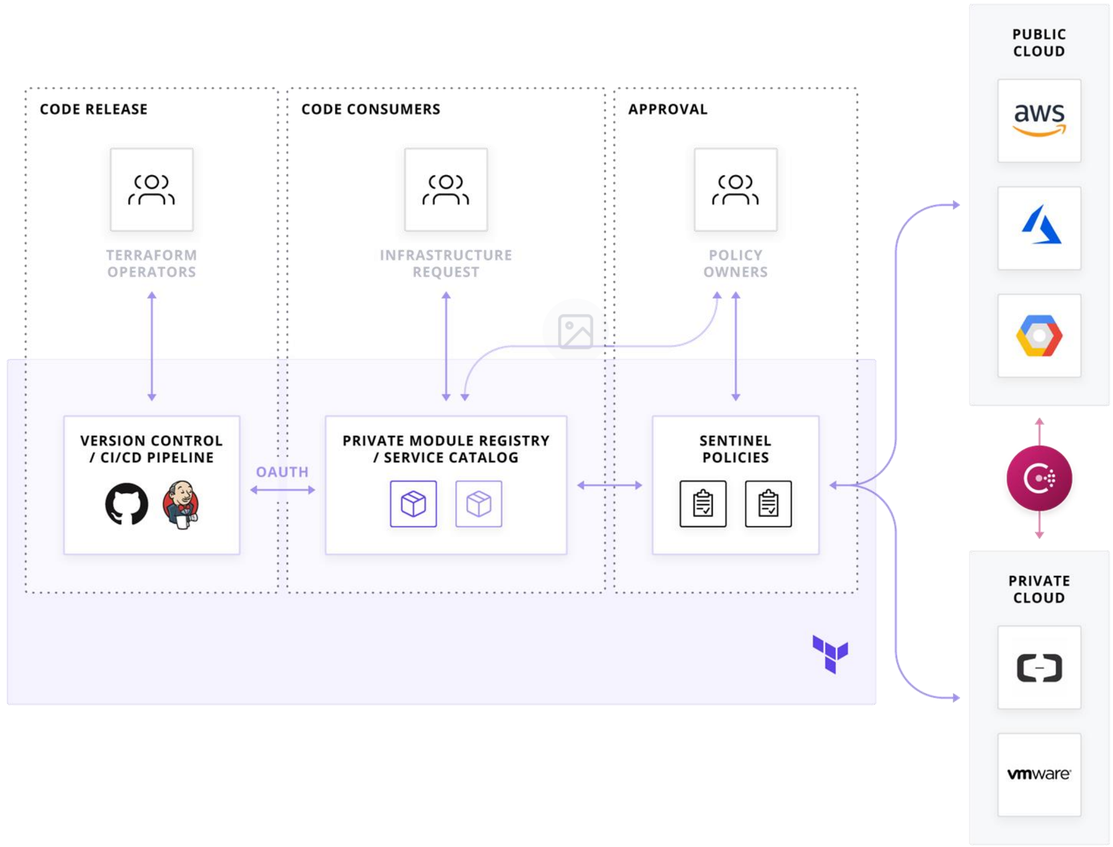
Terraform can be fully operated via API allowing organizations to easily integrate it into their existing application delivery pipelines.

Integrate with existing workflow to minimize process changes:

- CI/CD pipeline integration
- Provision teams and workspaces from Service Now

Use JSON-API endpoints to manage all resources and perform operations, including:

- Environment Variables
- Trigger Plan/apply
- Retrieve state information



Feature: **Full API**

CHALLENGE SOLUTION **RESULTS**

Increase Agility

Incorporating Terraform into an existing automation pipeline extends the automation benefits to infrastructure provisioning.

Feature: **Team Management & SSO**

CHALLENGE SOLUTION RESULTS

Role Based Access Control

To maintain proper security posture, organizations should provide access to configurations and provisioning only as needed by team members.

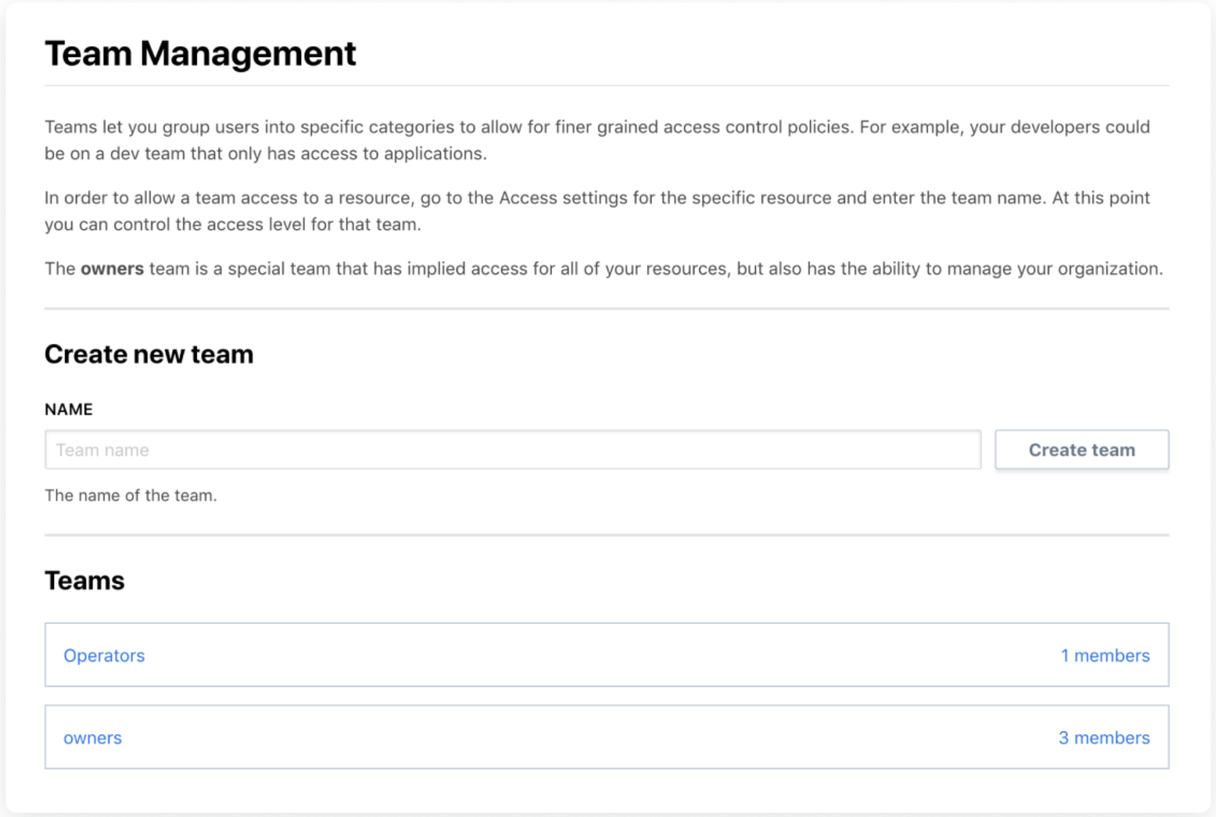
Feature: **Team Management & SSO**

CHALLENGE **SOLUTION** RESULTS

Team Management

Terraform allows organizations to define roles and teams that have access to certain workspaces and environments.

- Full role based access control
- Manage organizations, teams, and privileges of individual users



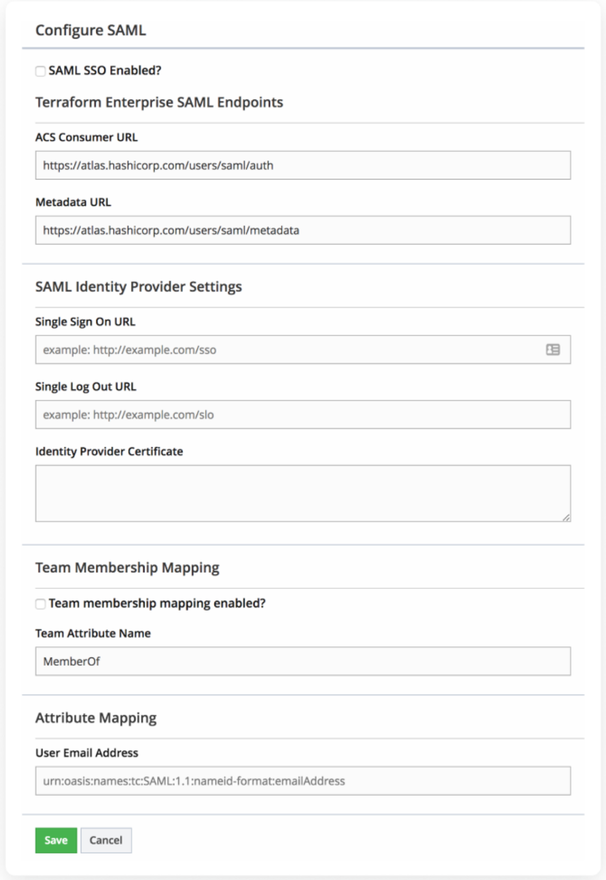
Feature: **Team Management & SSO**

CHALLENGE | **SOLUTION** | RESULTS

SAML SSO Support

With SAML support Terraform integrates with your existing identity provider to authenticate and authorize users.

- Use existing identity provider to authenticate and authorize access to Terraform Enterprise
- Single sign-on capability eliminates all passwords and instead uses standard cryptography and digital signatures to pass a secure sign-in token from an identity provider to Terraform Enterprise
- Authenticate and authorize users



Feature: **Team Management & SSO**

CHALLENGE

SOLUTION


RESULTS

Reduce Risk

Aligning roles with as-needed access to provisioning infrastructure minimizes the opportunity for accidental mistakes to impact the larger application.

Feature: **Audit & Administrate**

CHALLENGE

SOLUTION

RESULTS

Insight Into Infrastructure

Provisioning the infrastructure for an application is necessary for it to function, however, diagnosing an issue when an application malfunctions requires deep insight into the state of that application.

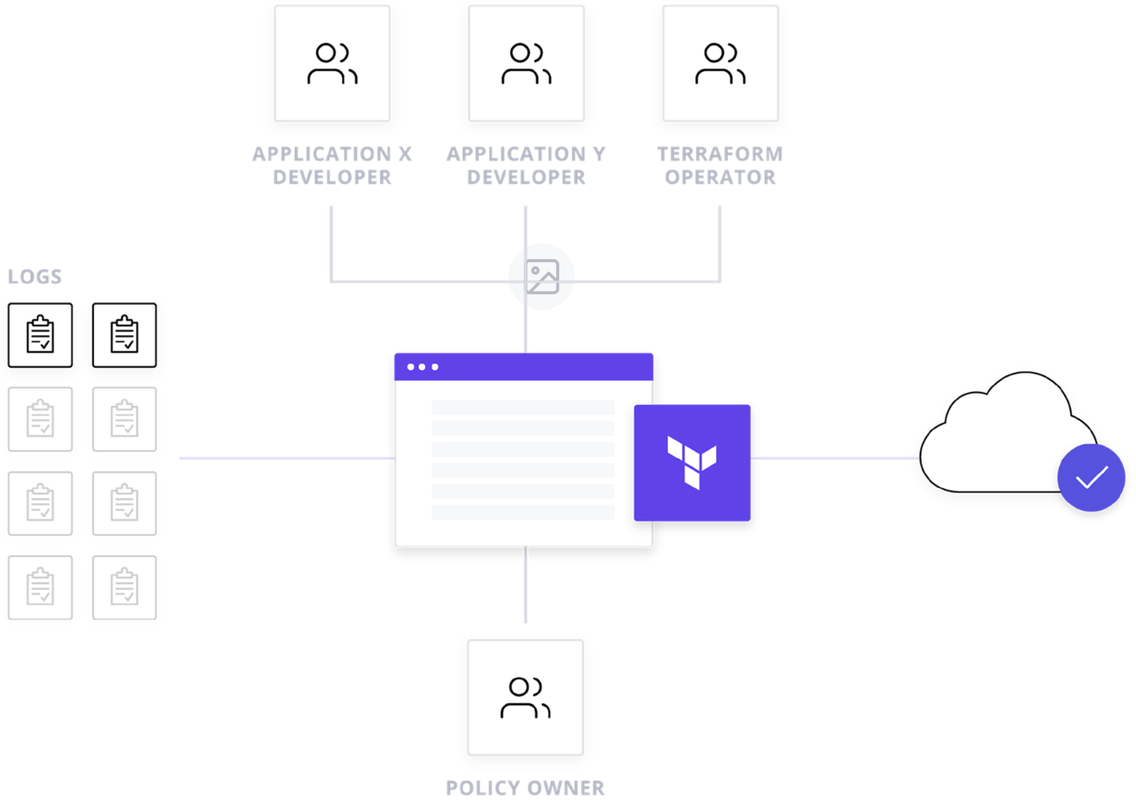
Feature: **Audit & Administrate**

CHALLENGE **SOLUTION** RESULTS

Audit Logs

Terraform Audit Logs provide a trail of every API call made for every provider and service Terraform has provisioned.

- Track operations and identity made across the organization to gain insight into past and present configurations
- Identity action and resource
- Logs include settings, configuration changes, executions, environment updates
- Logs detail every API call made at each plan/apply of Terraform



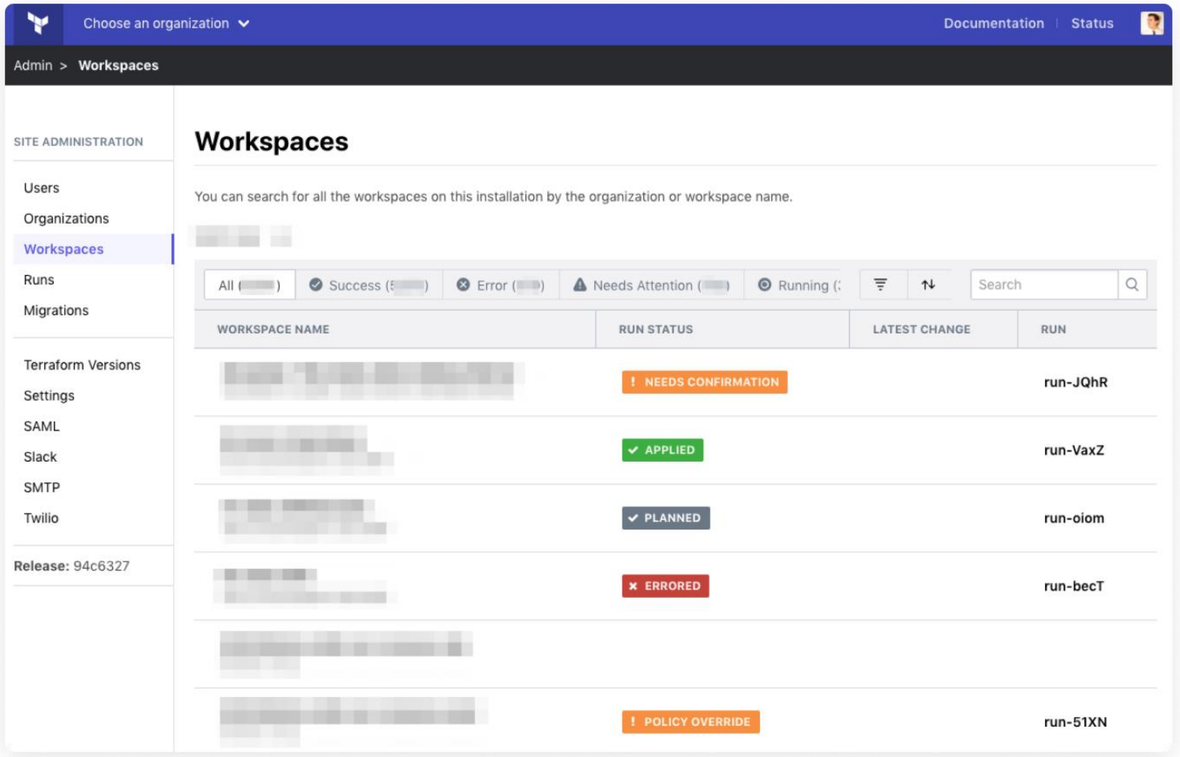
Feature: **Audit & Administrate**

CHALLENGE | **SOLUTION** | RESULTS

Site Admin

The site administrator view in Terraform Enterprise provides a single pane of glass to see the entirety of infrastructure provisioned with Terraform.

- Per workspace run, configuration, and state history
- Site admin gives view of every organization, workspace, user, and run
- Useful for auditing as well as troubleshooting



Feature: **Audit & Administrate**

CHALLENGE

SOLUTION

RESULTS

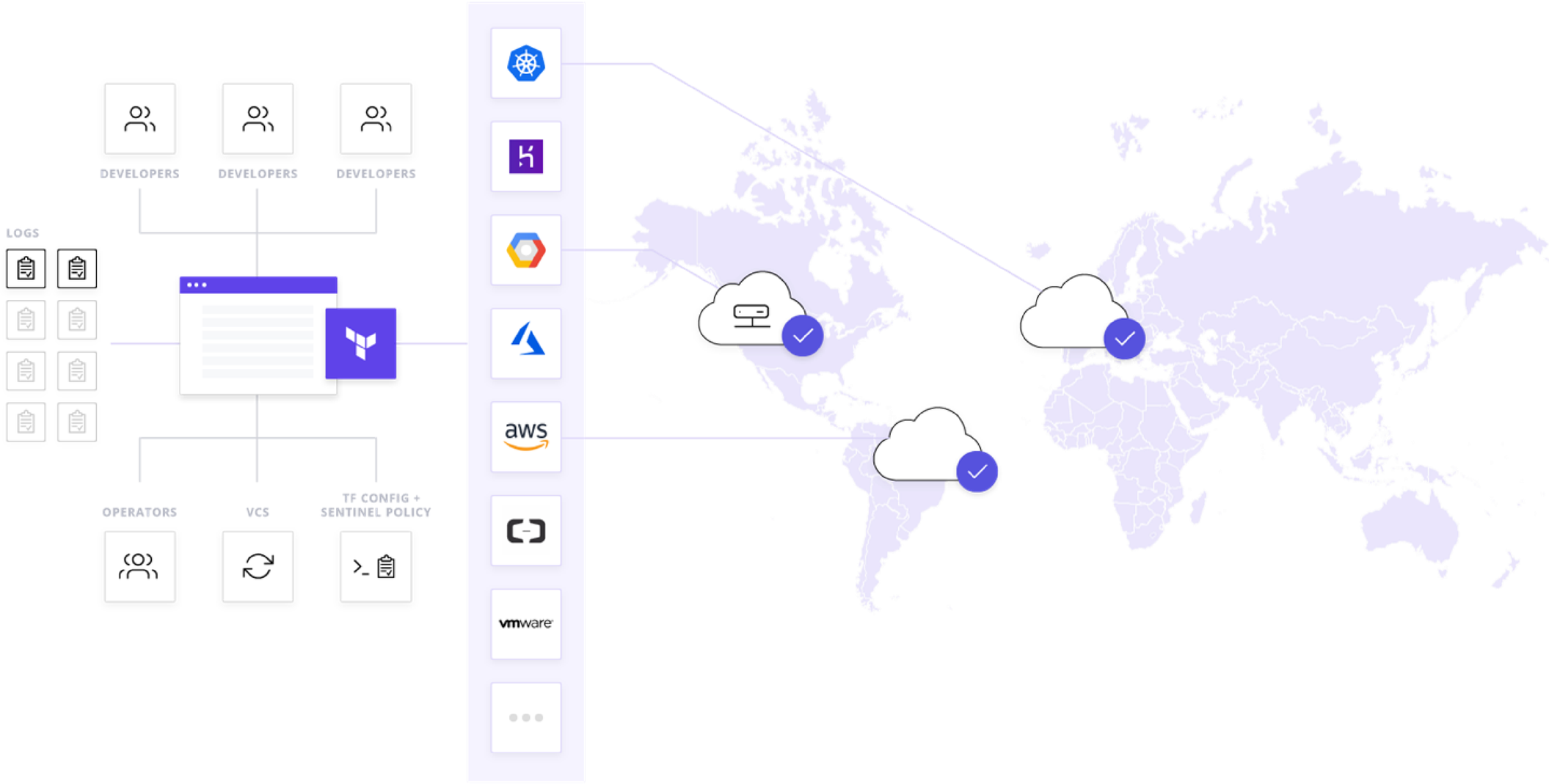
Meet Compliance

Robust audit logs help meet compliance standards for certifications like PCI, HIPAA, and FED RAMP.

Reduce Risk

Knowing exactly who made what change where and when provides powerful insight to mitigate risk. Deep insight into every API call made for all providers and services allows for quicker troubleshooting when applications malfunction.

Workflow for Enterprise Organizations

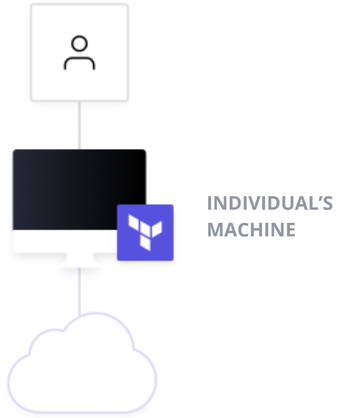


How **Terraform Works**

Delivery Methods

On Premise

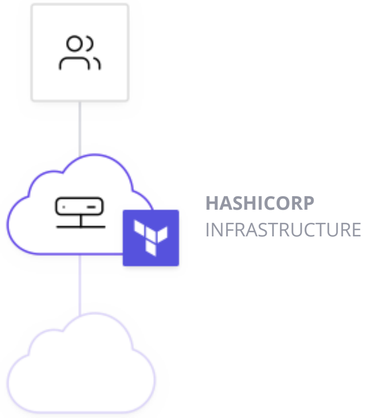
OPEN SOURCE



- No requirements for collaboration
- No requirements for central reusable configs
- No policy or governance requirements

Hosted SaaS

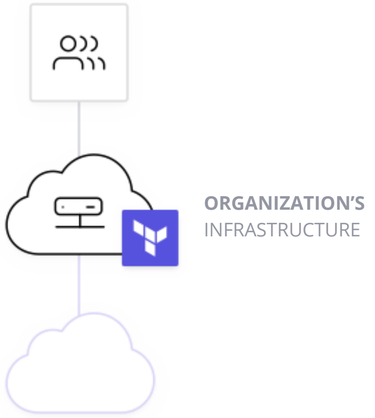
ENTERPRISE



- No special requirements*
- No internal resources to manage Terraform
- No policy or governance requirements

Private Install

ENTERPRISE



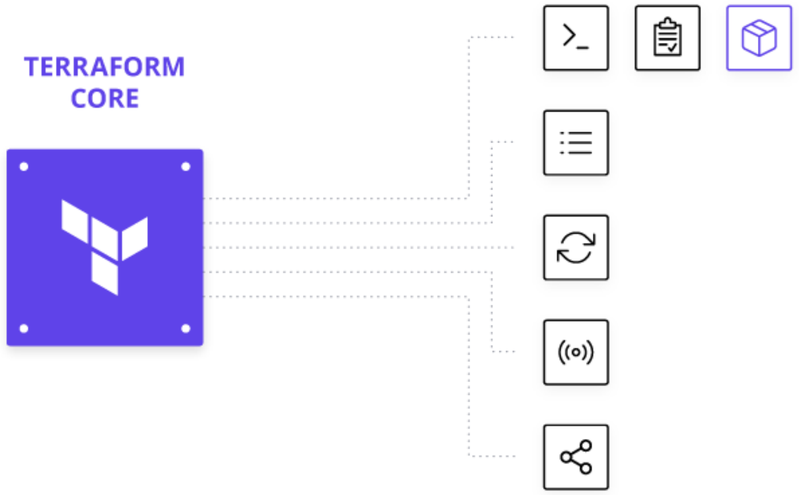
- Infrastructure & applications behind a firewall
- Data sovereignty requirements
- Regulatory compliance requirements
- HA requirements
- Performance requirements

Terraform Core Engine

- OSS hosted at github.com/hashicorp/terraform
- The engine Terraform runs on
- Loads providers as needed

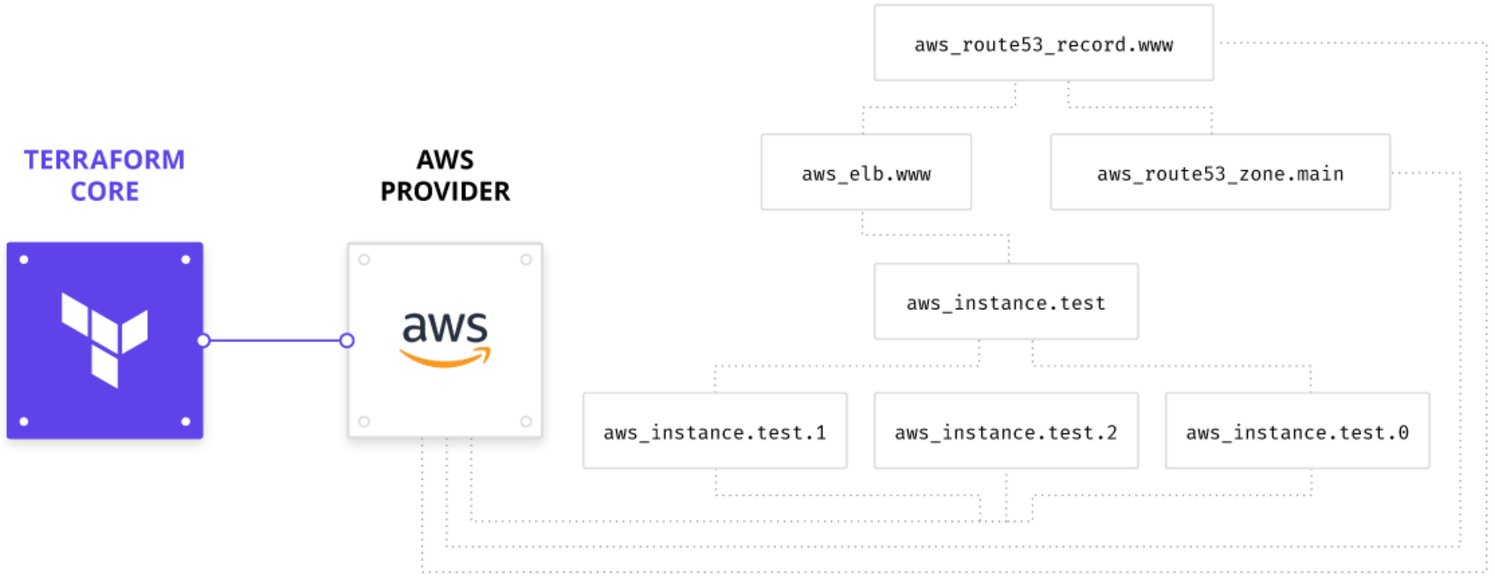
Responsible for:

- Reading and Interpolating configuration files and modules
- State Management
- Executing plan
- Communicating with providers
- Constructing resource graph



Resource Graph

- Safely provision and change infrastructure
- See planned infrastructure changes before execution
- No need to manually coordinate dependent resources

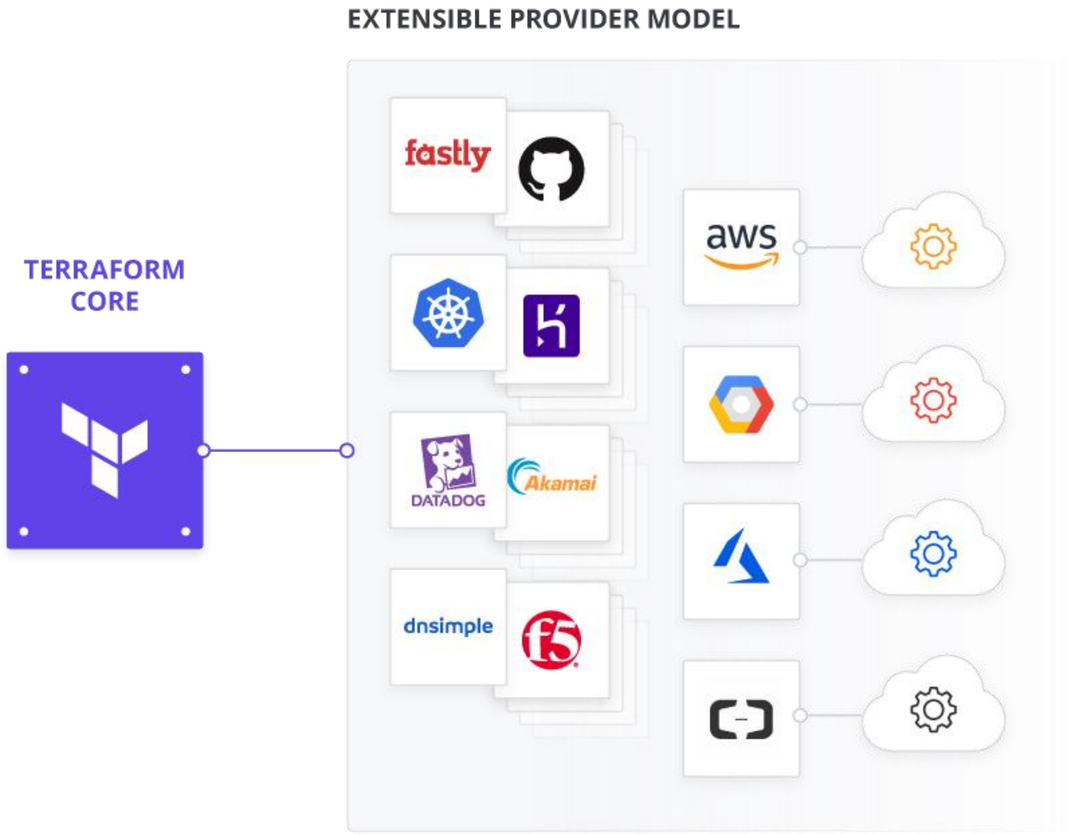


Provider Plugins

- Provider and Provisioner plugins expose implementation for specific services
- Offer extensible layer for ‘Core’ to learn how to talk to anything with an API without any upgrades

Responsible for:

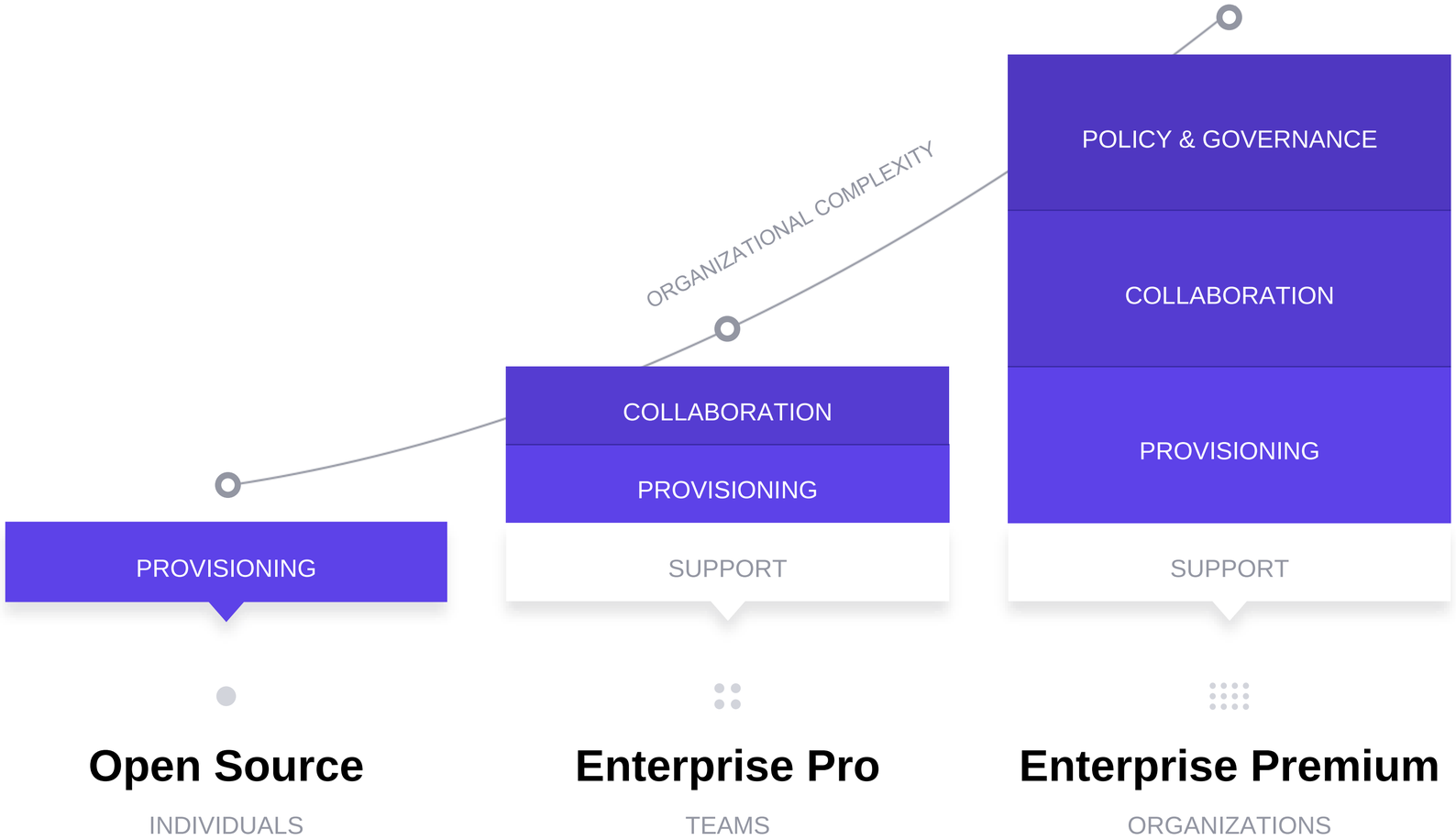
- Initializing libraries for API calls
- Authenticating with Provider
- Defining resources that map to services
- Executing commands or scripts on designated resources



Open Source
and Enterprise

Terraform Packages

Enterprise products build on open source to address organizational complexity.



Compare Packages

Open Source Infrastructure as code, Multi-Cloud Management, Self-Service Infrastructure		Enterprise Pro Collaboration and operations features for teams		Enterprise Premium Governance and policy features for organizations	
Infrastructure as Code (HCL)	✓	All **Open Source** Features	✓	All **Enterprise Pro** Features	✓
Workspaces	✓	VCS Integrated Connection	✓	Sentinel Policy as Code Management	✓
Variables	✓	Workspace Management	✓	Audit Logging	✓
Runs (separate plan and apply)	✓	Secure Variable Storage	✓	SAML for SSO	✓
Resource Graph	✓	Remote Runs & State	✓	Private Install Options	✓
One Workflow to Provision Across Providers	✓	Team Management	✓		
Providers (150+)	✓	Private Module Registry	✓		
Unique Resources (1000s)	✓	Configuration Designer	✓		
Modules	✓	Full API Converage	✓		
Public Module Registry	✓	SaaS	✓		

Competitive **Landscape**

Cloud Native IaC vs HCL

YAML

```
"B5AppSubnet1": {
    "Type": "AWS::EC2::Subnet",
    "Properties": {
        "CidrBlock": { "Fn::Select" : ["0", { "Fn::FindInMap" : [ "SubnetCidr", { "Ref" : "Env" }, "b5app"] }] },
        "AvailabilityZone": { "Fn::Select" : [ "0", { "Fn::GetAZs" : "" } ]},
        "VpcId": { "Ref": "Vpc" },
        "Tags": [
            { "Key" : "Application", "Value" : "B5" },
            { "Key" : "env", "Value": { "Ref" : "Env" } },
            { "Key" : "Name", "Value": { "Fn::Join" : ["-", [ {"Ref" : "Env"}, "b5", "b5app-subnet1"]] } }
        ]
    }
},

"B5AppSubnet2": {
    "Type": "AWS::EC2::Subnet",
    "Properties": {
        "CidrBlock": { "Fn::Select" : ["1", { "Fn::FindInMap" : [ "SubnetCidr", { "Ref" : "Env" }, "b5app"] }] },
        "AvailabilityZone": { "Fn::Select" : [ "1", { "Fn::GetAZs" : "" } ]},
        "VpcId": { "Ref": "Vpc" },
        "Tags": [
            { "Key" : "Application", "Value" : "B5" },
            { "Key" : "env", "Value": { "Ref" : "Env" } },
            { "Key" : "Name", "Value": { "Fn::Join" : ["-", [ {"Ref" : "Env"}, "b5", "b5app-subnet2"]] } }
        ]
    }
},

"B5AppSubnet3": {
    "Type": "AWS::EC2::Subnet",
    "Properties": {
        "CidrBlock": { "Fn::Select" : ["2", { "Fn::FindInMap" : [ "SubnetCidr", { "Ref" : "Env" }, "b5app"] }] },
        "AvailabilityZone": { "Fn::Select" : [ "2", { "Fn::GetAZs" : "" } ]},
        "VpcId": { "Ref": "Vpc" },
        "Tags": [
            { "Key" : "Application", "Value" : "B5" },
            { "Key" : "env", "Value": { "Ref" : "Env" } },
            { "Key" : "Name", "Value": { "Fn::Join" : ["-", [ {"Ref" : "Env"}, "b5", "b5app-subnet3"]] } }
        ]
    }
},
```

HCL

```
resource "aws_subnet" "b5app" {
  count             = "${length(var.subnet_cidr["b5app"])}"
  vpc_id            = "${aws_vpc.b5.id}"
  cidr_block        = "${element(var.subnet_cidr["b5app"],count.index)}"
  availability_zone = "${var.az[count.index]}"

  tags {
    Application = "B5"
    env         = "${var.env}"
    type        = "${var.type}"
    Name        = "${var.env}-b5-b5app-subnet-${count.index}"
  }
}
```

This shows the difference between defining a subnet for an application server in a Cloud Native Infrastructure as Code compared to HCL. More here:

Terraforming 1Password - AgileBits Blog

Cloud Native IaC vs HCL

JSON

```
"StagingInstance": {
  "Type": "AWS::EC2::Instance",
  "Properties": {
    "UserData": {
      "Fn::Base64": {
        "Fn::Join": ["", [
          "#!/bin/bash -v\n",
          "yum update -y aws*\n",
          "yum update --sec-severity=critical -y\n",
          "yum install -y aws-cfn-bootstrap\n",
          "# download data and install file\n",
          "/opt/aws/bin/cfn-init -s ", {
            "Ref": "AWS::StackName"
          }, " -r StagingInstance ",
          "    --region ", {
            "Ref": "AWS::Region"
          },
          " || error_exit 'Failed to run cfn-init'\n"
        ]]
      }
    },
    "SecurityGroupIds": [{
      "Ref": "StagingSecurityGroup"
    }],
    "ImageId": {
      "Ref": "StagingAMI"
    },
    "KeyName": {
      "Ref": "InstancePrivateKeyName"
    },
    "InstanceType": {
      "Ref": "StagingInstanceType"
    },
    "IamInstanceProfile": {
      "Ref": "StagingInstanceProfile"
    },
    "Tags": [{
      "Key": "Name",
      "Value": {
        "Fn::Join": ["-", [
          "staging", {
            "Ref": "AWS::StackName"
          }, "app-instance"
        ]]
      }
    }],
    "SubnetId": {
      "Ref": "PrivateSubnet1"
    }
  }
}
```

HCL

```
# Create the staging instance
resource "aws_instance"
"staging" {
  ami                    = "${var.staging_instance_ami}"
  instance_type          = "${var.staging_instance_type}"
  subnet_id              = "${var.private_subnet_id_1}"
  vpc_security_group_ids = [ "${aws_security_group.staging.id}" ]
  iam_instance_profile   = "${aws_iam_instance_profile.staging.name}"
  key_name               = "${var.instance_private_key_name}"
  tags {
    Name = "staging-${var.stack_name}-instance"
  }
  user_data = "${file("instances / staginguserdatascript.sh ")}"
}
```

This shows the difference between defining an instance in AWS with CloudFormation in JSON versus HCL. More here:

AWS CFT vs Terraform Advantages and Disadvantages

Cloud Native IaC vs HCL

JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "vnetName": {
      "type": "string",
      "defaultValue": "VNet1",
      "metadata": {
        "description": "VNet name"
      }
    },
    "vnetAddressPrefix": {
      "type": "string",
      "defaultValue": "10.0.0.0/16",
      "metadata": {
        "description": "Address prefix"
      }
    },
    "subnet1Prefix": {
      "type": "string",
      "defaultValue": "10.0.0.0/24",
      "metadata": {
        "description": "Subnet 1 Prefix"
      }
    },
    "subnet1Name": {
      "type": "string",
      "defaultValue": "Subnet1",
      "metadata": {
        "description": "Subnet 1 Name"
      }
    },
    "subnet2Prefix": {
      "type": "string",
      "defaultValue": "10.0.1.0/24",
      "metadata": {
        "description": "Subnet 2 Prefix"
      }
    },
    "subnet2Name": {
      "type": "string",
      "defaultValue": "Subnet2",
      "metadata": {
        "description": "Subnet 2 Name"
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "variables": {},
  "resources": [
    {
      "apiVersion": "2018-06-01",
      "type": "Microsoft.Network/virtualNetworks",
      "name": "[parameters('vnetName')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[parameters('vnetAddressPrefix')]"
          ]
        }
      },
      "resources": [
        {
          "apiVersion": "2018-06-01",
          "type": "subnets",
          "location": "[parameters('location')]",
          "name": "[parameters('subnet1Name')]",
          "dependsOn": [
            "[parameters('vnetName')]"
          ],
          "properties": {
            "addressPrefix": "[parameters('subnet1Prefix')]"
          }
        },
        {
          "apiVersion": "2018-06-01",
          "type": "subnets",
          "location": "[parameters('location')]",
          "name": "[parameters('subnet2Name')]",
          "dependsOn": [
            "[parameters('vnetName')]"
          ],
          "properties": {
            "addressPrefix": "[parameters('subnet2Prefix')]"
          }
        }
      ]
    }
  ]
}
```

HCL

```
# Configure the Azure Provider
provider "azurerm" { }

# Create a resource group
resource "azurerm_resource_group" "network" {
  name     = "production"
  location = "West US"
}



# Create a virtual network within the resource group
resource "azurerm_virtual_network" "network" {
  name                = "production-network"
  address_space       = ["10.0.0.0/16"]
  location            = "${azurerm_resource_group.network.location}"
  resource_group_name = "${azurerm_resource_group.network.name}"



  subnet {
    name           = "subnet1"
    address_prefix = "10.0.1.0/24"
  }

  subnet {
    name           = "subnet2"
    address_prefix = "10.0.2.0/24"
  }

  subnet {
    name           = "subnet3"
    address_prefix = "10.0.3.0/24"
  }
}
```

This shows the difference between defining an virtual network in Azure with Arm Templates in JSON versus HCL.

[Azure ARM Github]

[Azure Terraform Provider]

Thank you

hello@hashicorp.com

www.hashicorp.com