Terraform Enterprise Workshop

Ancil McBarnett – Snr. Solutions Engineer

Kawsar Kamal – Snr. Solutions Engineer

Who we are

Founded 2012 by Mitchell Hashimoto and Armon Dadgar

Mission We enable organizations to Provision, Secure, Connect, and Run any infrastructure for any application

Key Products

Agenda

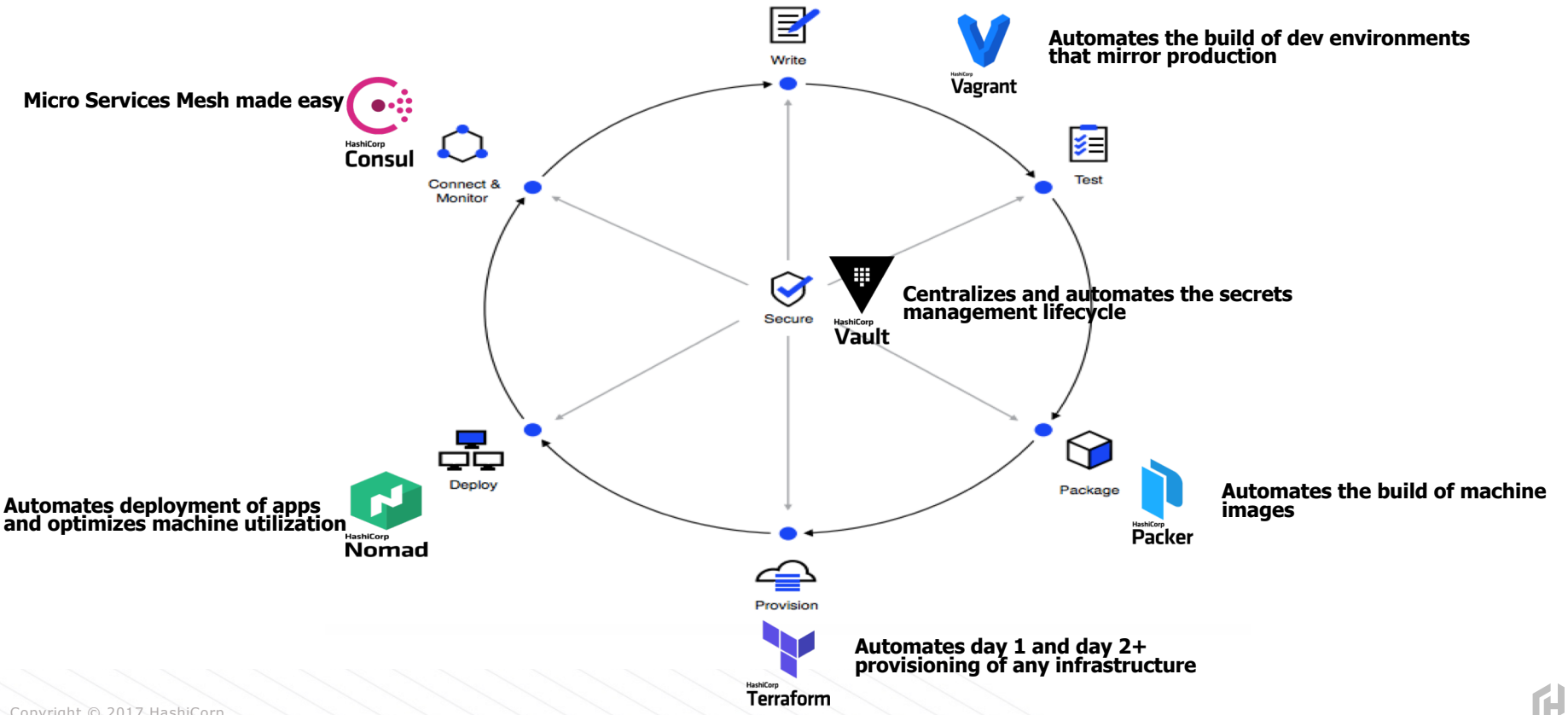
- HashiCorp Dev Ops Overview
- Terraform Introduction
- Terraform Open Source vs. Enterprise
- **Demo:** Terraform Enterprise
- Policy as Code - Sentinel
- **Labs:** TFE Sas Labs

Lunch

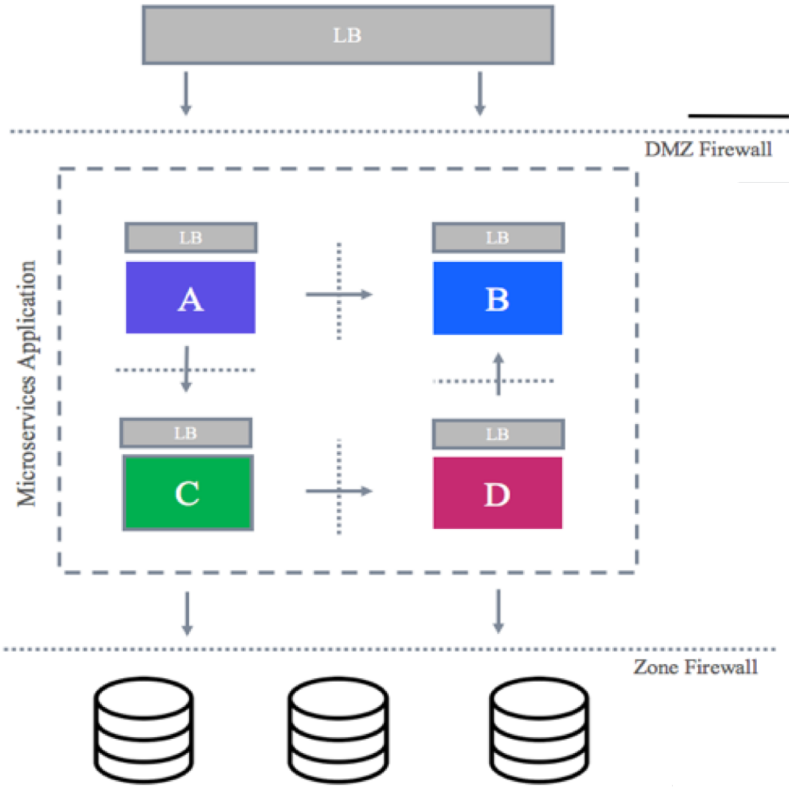
- Collaboration & Private Registry
- Producer/ Consumer best practices
- CICD pipelines & TFE APIs
- Private TFE Deployment Architecture
- Intro to Vault, Consul, Nomad
- Terraform with Vault, Consul, Nomad
- Closeout & Questions

Automating Application Delivery w/ HashiCorp Tools

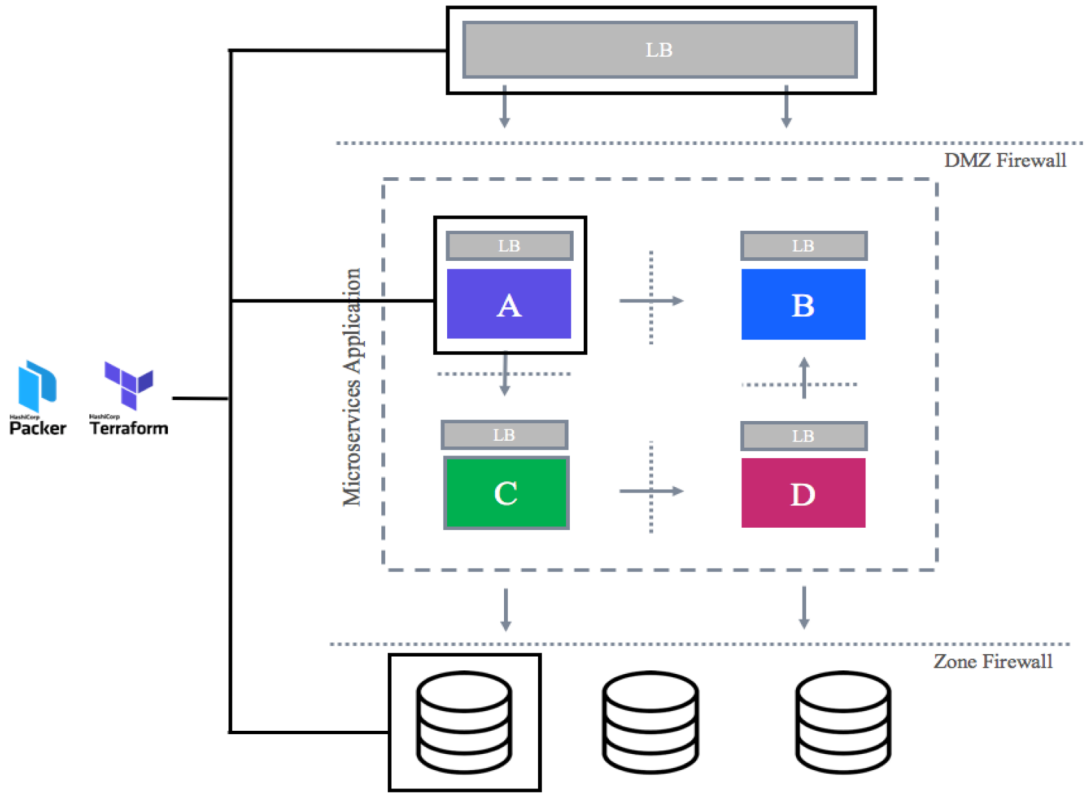
www.hashicorp.com/resources/application-delivery-hashicorp



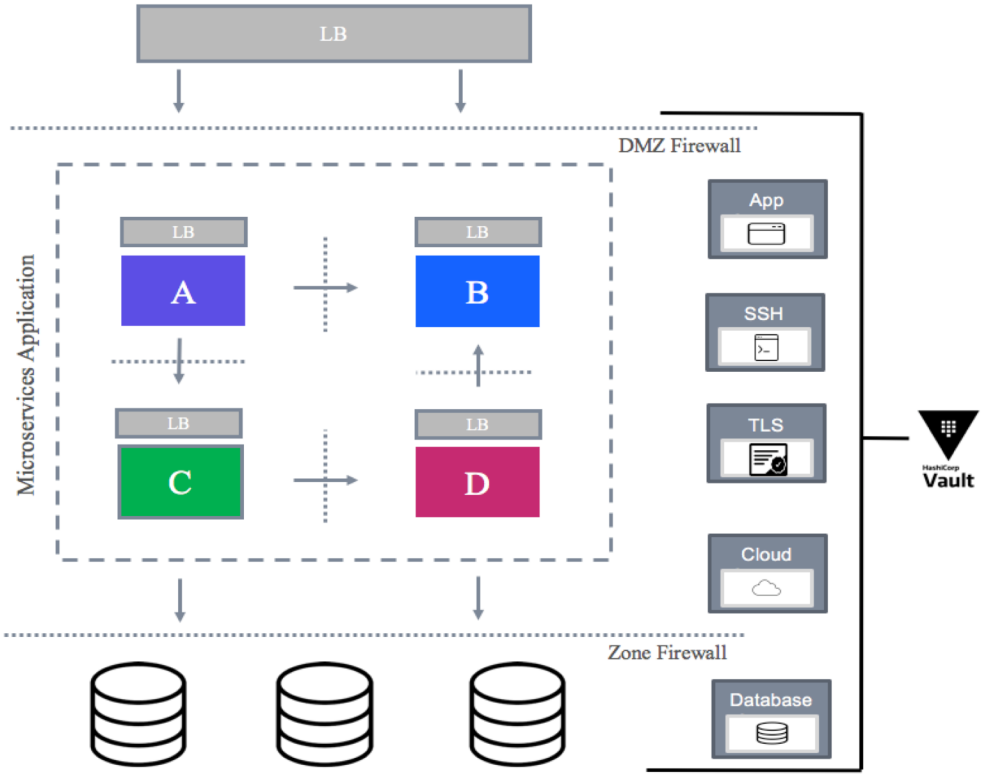
Applied Concepts for Delivering a Microservices Application



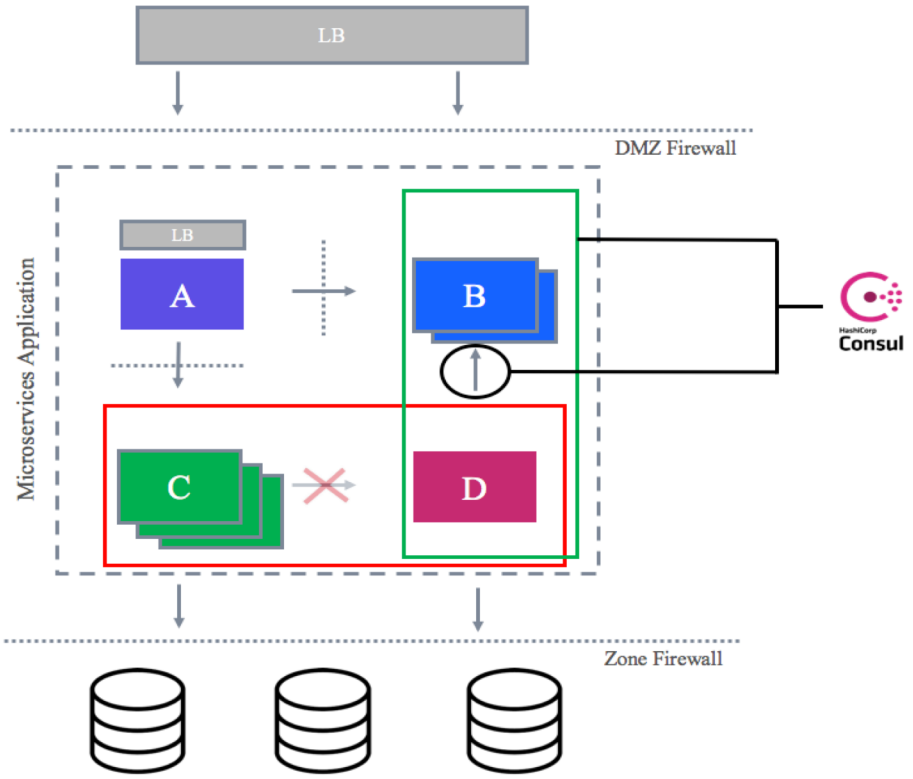
Packer and Terraform provide the “recipe”. Packer codifies golden images while Terraform codifies, governs, and provisions approved infrastructure



Vault dynamically creates and revokes secrets based on centralized policy, identity, and time to live.



Consul referees and secures service to service communication and vastly reduces network complexity



The largest open source community of its kind

843 Contributors
11,273 Commits
17,067 Stars

718 Contributors
10,282 Commits
7,903 Stars

1,227 Contributors
21,618 Commits
13,074 Stars

259 Contributors
12,318 Commits
3,759 Stars

428 Contributors
8,823 Commits
13,092 Stars

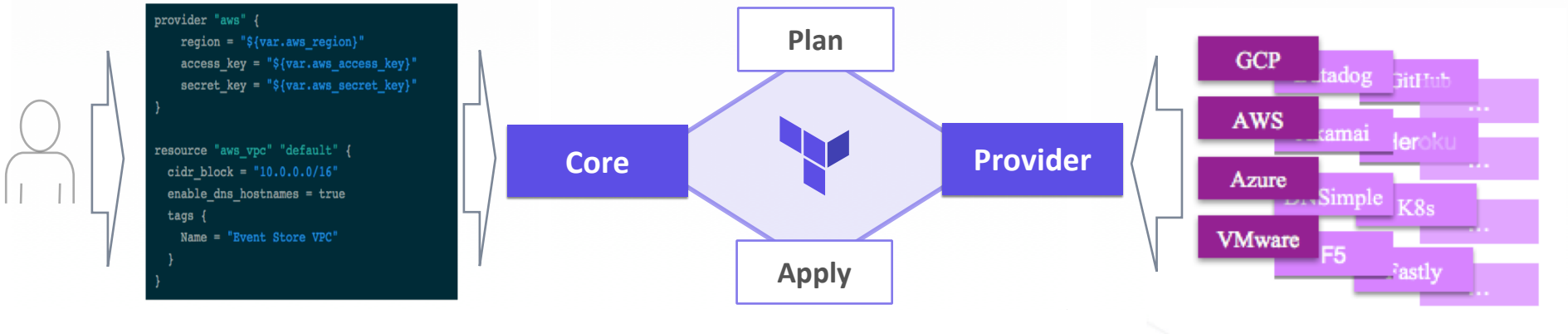
513 Contributors
8,823 Commits
9,760 Stars

Terraform Overview

Provision any infrastructure for any application

Terraform Differentiators

USER PRODUCTIVITY	CONSISTENT WORKFLOWS	UNIVERSAL PROVISIONING
Manage infrastructure at scale	Safe, consistent workflows	Distributed Infrastructure Strategy
• *Common Language - HCL* • *Resource Graph* • *Best Practice Templates available for reuse*	• *Safely provision and manage infrastructure (Plan and Apply)* • *Enforce Policy with Sentinel* • *Reduce errors and costs*	• *Extensible providers to provision any infrastructure* • *Module registry - self serve* • *Hedge against technology or vendor lock-in*

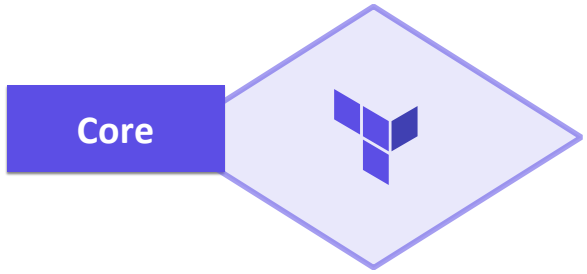


User Productivity

User Productivity: Increased Productivity with HCL and TF Graph

- Terraform can leverage HCL or JSON. However HCL is typically easier to adopt and maintain for users.
- HCL knowledge is also transferable to a wide set of technologies by virtue of Terraform providers

Example is making use of parameters and variables on Azure




```
resource "azurerm_public_ip" "basicvm" {
  name                         = "${local.publicIpAddressName}"
  location                     = "${var.resource_group_location}"
  ...
  public_ip_address_allocation = "${local.publicIpAddressType}"
}
```

Vs.

```
{
      "name": "[variables('publicIpAddressName')]",
      "type": "Microsoft.Network/publicIpAddresses",
      "apiVersion": "2016-09-01",
      "location": "[parameters('location')]",
      "properties": {
        "publicIpAllocationMethod": "[variables('publicIpAddressType')]"
      }
}
```


User Productivity: Increased Productivity with HCL and TF Graph

- The simplicity of HCL is really valuable as "real" infrastructure takes shape

Example is is creating a NIC using HCL vs. JSON on Azure

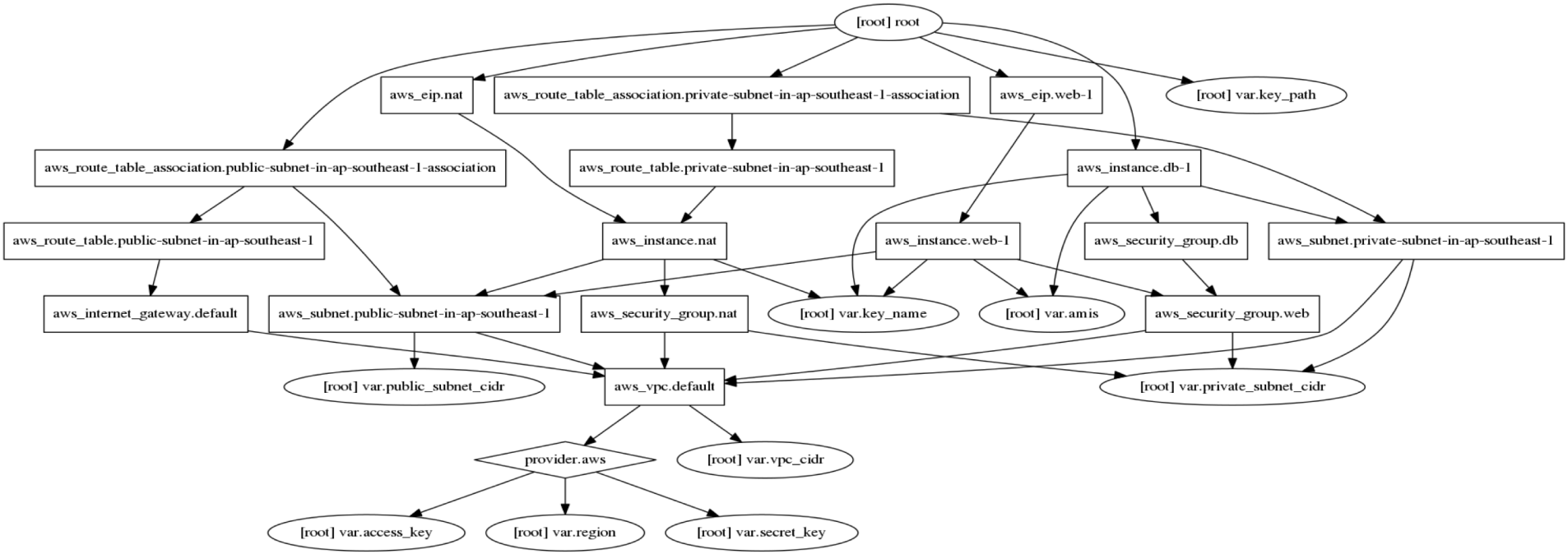
```
resource "azurerm_network_interface" "basicvm" {
  name                = "${local.networkInterfaceName}"
  location            = "${var.resource_group_location}"
  resource_group_name = "${azurerm_resource_group.basicvm.name}"

  ip_configuration {
    name                          = "ipConfig"
    subnet_id                     = "${azurerm_subnet.basicvm.id}"
    private_ip_address_allocation = "dynamic"
    public_ip_address_id          = "${azurerm_public_ip.basicvm.id}"
  }
}
```

Vs.

```
{
      "name": "[variables('networkInterfaceName')]",
      "type": "Microsoft.Network/networkInterfaces",
      "apiVersion": "2016-09-01",
      "location": "[parameters('location')]",
      "dependsOn": [
        "[concat('Microsoft.Network/virtualNetworks/', variables('virtualNetworkName'))]",
        "[concat('Microsoft.Network/publicIpAddresses/', variables('publicIpAddressName'))]",
        "[concat('Microsoft.Network/networkSecurityGroups/', variables('networkSecurityGroupName'))]"
      ],
     "properties": {
        "ipConfigurations": [
          {
            "name": "ipconfig1",
            "properties": {
              "subnet": {
                "id": "[variables('subnetRef')]"
              },
              "privateIPAllocationMethod": "Dynamic",
              "publicIpAddress": {
                "id": "[resourceId(resourceGroup().name,'Microsoft.Network/publicIpAddresses', variables('pu
              }
            }
          }
        ],
        "networkSecurityGroup": {
          "id": "[resourceId(resourceGroup().name, 'Microsoft.Network/networkSecurityGroups', variables('ne
        }
}
```


User Productivity: Increased Productivity with HCL and TF Graph

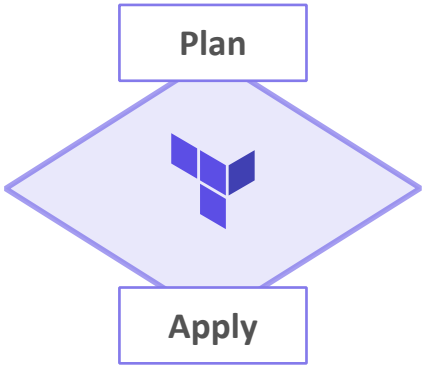
- Terraform's graph is a provisioning "easy button" as it reasons over dependencies and automates the necessary order of operations.



Consistent Workflows

Consistent Workflows: TF State and Plan and Apply

- Terraform manages state and applies changes against that state
- ”Plan” provides of view of what will happen
- “Apply” executes the desired state

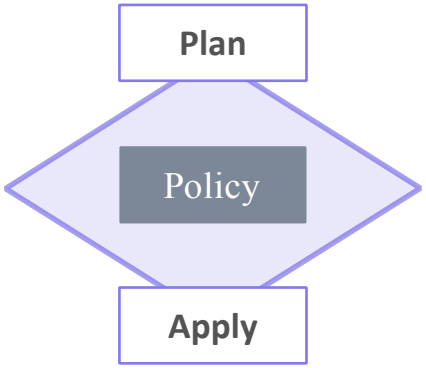



```
$ terraform plan -out=terraform.tfplan terraform/
Refreshing Terraform state prior to plan...


The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed.

Your plan was also saved to the path below. Call the "apply" subcommand
with this plan file and Terraform will exactly execute this execution
plan.

Path: terraform.tfplan

+ digitalocean_droplet.app
    backups:                "" => "1"
    image:                  "" => "ubuntu-14-04-x64"
    ipv4_address:           "" => "<computed>"
    ipv4_address_private:   "" => "<computed>"
    ipv6:                   "" => "1"
    ipv6_address:           "" => "<computed>"
    ipv6_address_private:   "" => "<computed>"
    locked:                 "" => "<computed>"
    name:                   "" => "app-01"
    region:                 "" => "lon1"
    size:                   "" => "512mb"
    status:                 "" => "<computed>"

+ digitalocean_droplet.hosting
    backups:                "" => "1"
    image:                  "" => "ubuntu-14-04-x64"
    ipv4_address:           "" => "<computed>"
    ipv4_address_private:   "" => "<computed>"
    ipv6:                   "" => "1"
    ipv6_address:           "" => "<computed>"
    ipv6_address_private:   "" => "<computed>"
    locked:                 "" => "<computed>"
    name:                   "" => "hosting-01"
    region:                 "" => "lon1"
    size:                   "" => "512mb"
    status:                 "" => "<computed>"


Plan: 2 to add, 0 to change, 0 to destroy.
```

Consistent Workflows: TF State and Plan, Policy Check*, and Apply

- Sentinel is HashiCorp's Policy as Code framework and part of Terraform Enterprise*
- Applies policy w/n path to provision

Example is a for allowing machine types

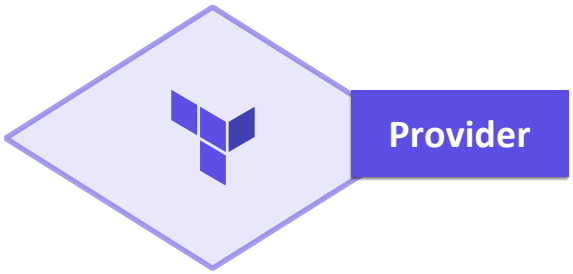



```
import "tfplan"

allowed_machine_types = [
    "n1-standard-1",
    "n1-standard-2",
    "n1-standard-4",
    "n1-standard-8",
]

main = rule {
    all tfplan.resources as type, resources {
        all resources as r {
            r.applied.machine_type in allowed_machine_types
        }
    }
}
```

Universal Provisioning

Universal Provisioning: Providers and Module Registry

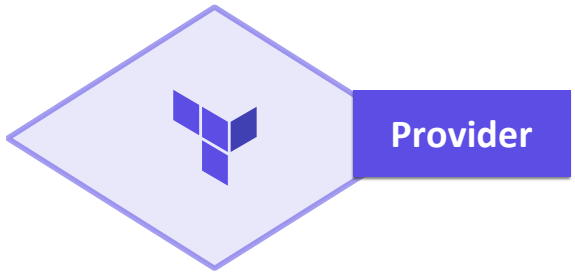
- Providers are what allow Terraform to be the “common language” for provisioning
- Not a least common denominator
- 125+ providers and growing eliminates blind spots and manual automation

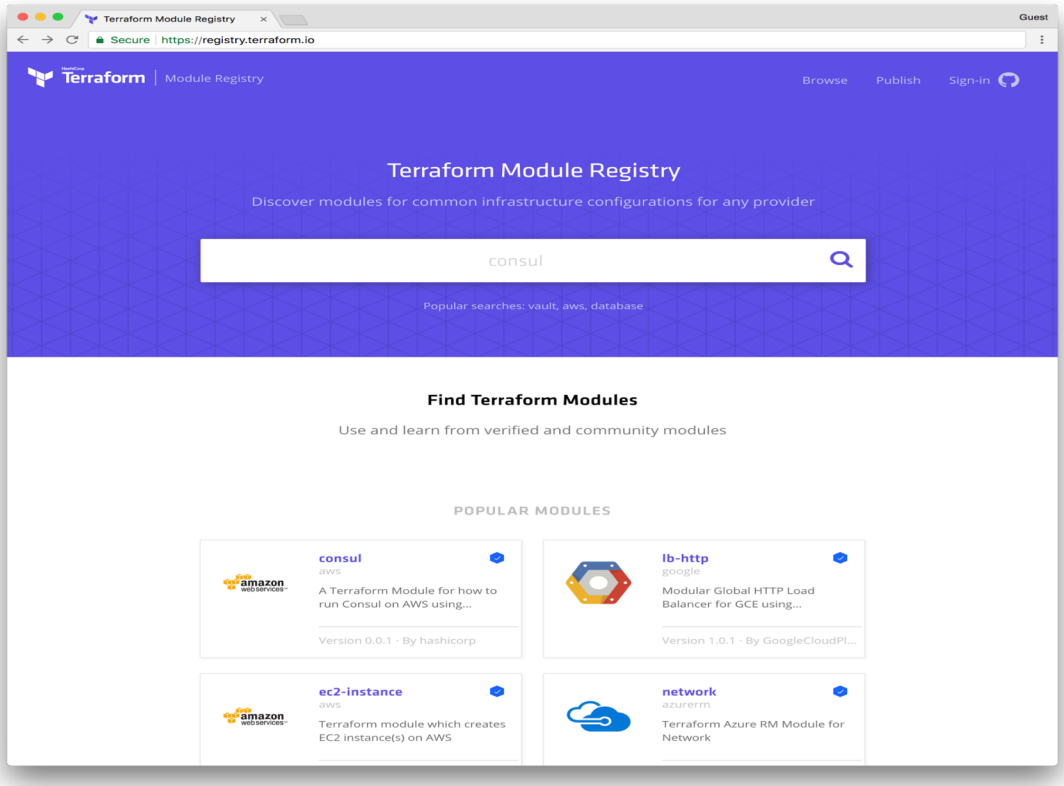


ACME	Alicloud	Archive
Arukas	AWS	Azure
Azure Stack	Bitbucket	Brightbox
CenturyLinkCloud	Chef	Circonus
Cloudflare	CloudScale.ch	CloudStack
Cobbler	Consul	Datadog
DigitalOcean	DNS	DNSMadeEasy
DNSimple	Docker	Dyn
External	Fastly	FlexibleEngine
GitHub	Gitlab	Google Cloud
Grafana	Heroku	Hetzner Cloud
HTTP	HuaweiCloud	Icinga2
Ignition	InfluxDB	Kubernetes
Librato	Local	Logentries
LogicMonitor	Mailgun	MySQL
New Relic	Nomad	NS1
Null	1&1	OpenStack
OpenTelekomCloud	OpsGenie	Oracle Public Cloud
Oracle Cloud Platform	OVH	Packet
PagerDuty	Palo Alto Networks	PostgreSQL
PowerDNS	ProfitBricks	RabbitMQ
Rancher	Random	Rundeck
Runscope	Scaleway	SoftLayer
StatusCake	Spotinst	TelefonicaOpenCloud
Template	Terraform	Terraform Enterprise
TLS	Triton	UltraDNS
Vault	VMware vCloud Director	VMware NSX-T
VMware vSphere		

Universal Provisioning: Providers and Module Registry

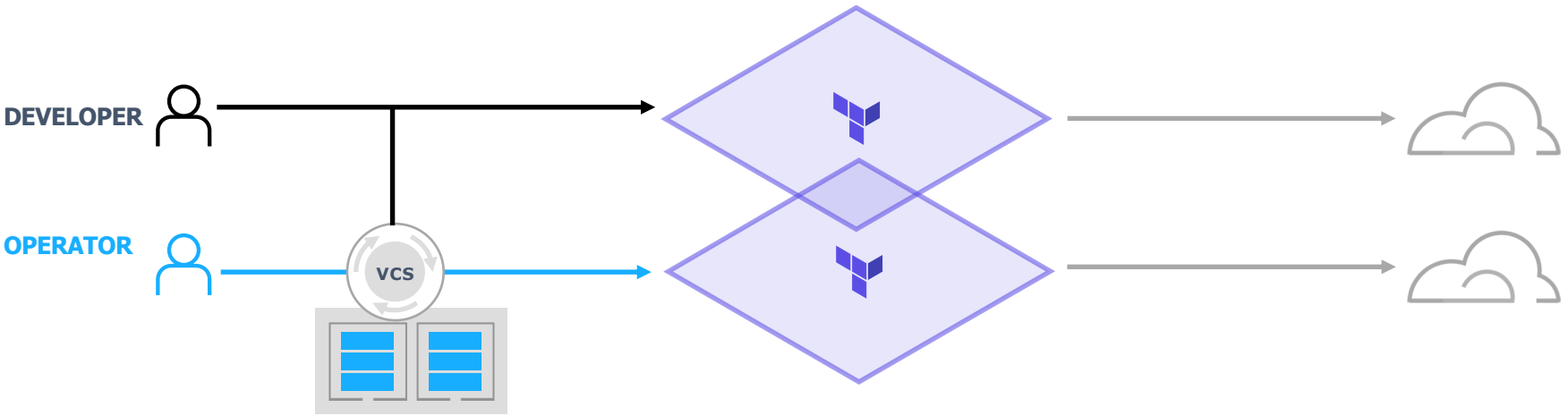
- Modules enable producers to productize approved infrastructure
- Registry allows consumers to discover and reuse these assets
- Supporting a self service model



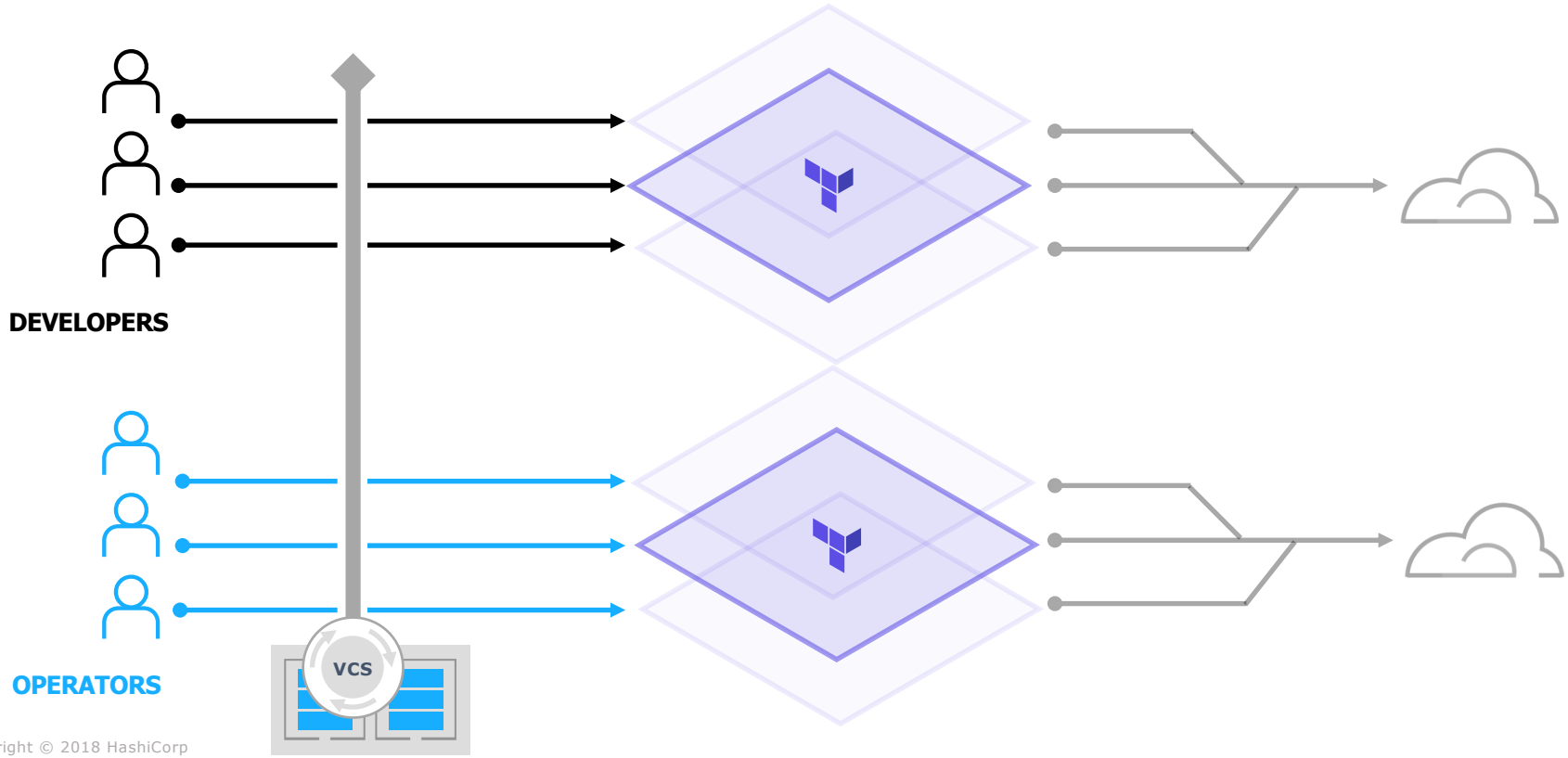


Terraform Enterprise

Terraform OSS workflow for individuals



How can teams safely collaborate on infrastructure



Adopting enterprise IaC is more than just great technology

- How do we manage state easily across the company and account for varying lifecycles across teams?
- How can we safely enable these infrastructure assets to be discovered and reused?
- How do we incorporate Terraform into our CICD pipeline?
- How does all of this fit with our corporate standards/policies?
- How do we NOT end up with a MONOLITH?

Adopting enterprise IaC is more than just great technology

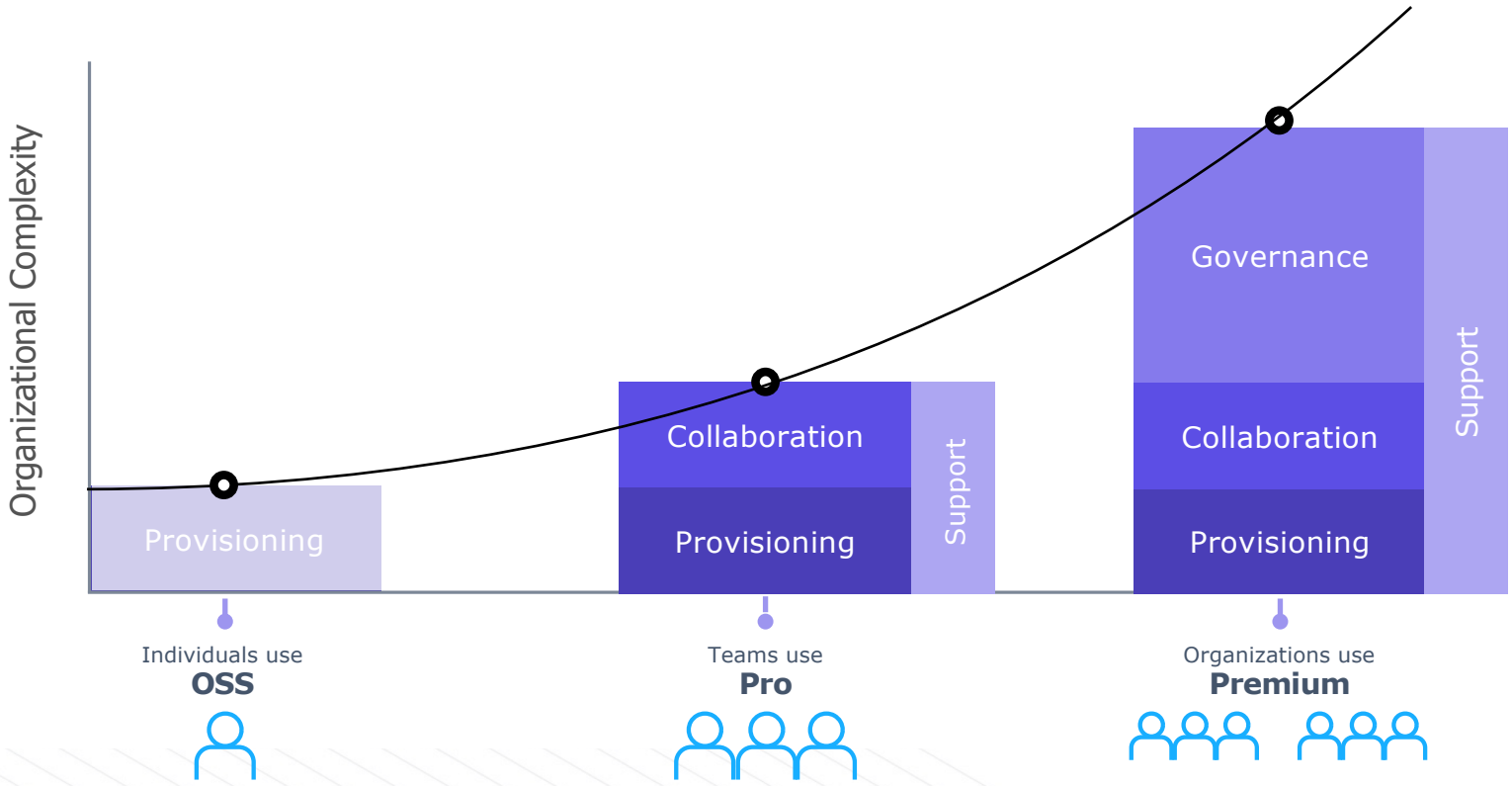
- How you manage TF states for single and multiple accounts
  - Terraform File structure – single vs multi account

- How to work in collaboration as a team –
  - Locking state, validation
  - Live Infrastructure vs Blueprints
  - Consumer/Subscriber model

Terraform Whiteboard

Terraform Open Source and Terraform Enterprise

OSS for technical complexity & Enterprise for organizational complexity



Terraform Ecosystem

Partnerships with 25+ technology providers, community support for many more

VERSION-CONTROL SYSTEMS

Bitbucket

GitLab

GitHub

INFRASTRUCTURE PROVIDERS

Google Cloud Platform

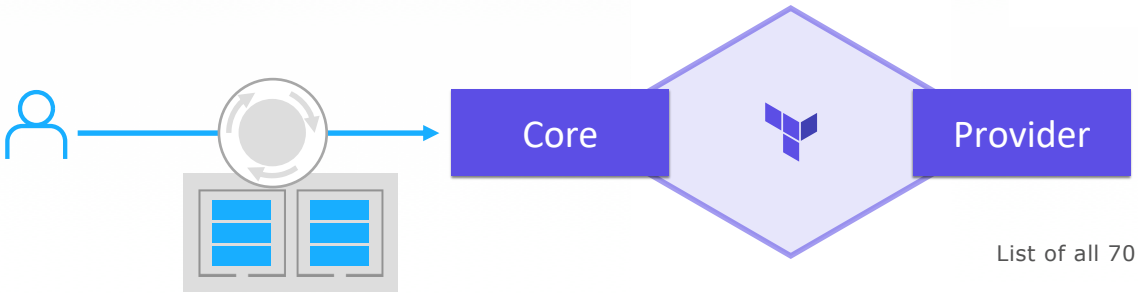
Microsoft Azure

amazon web services™

ORACLE

Alibaba Cloud

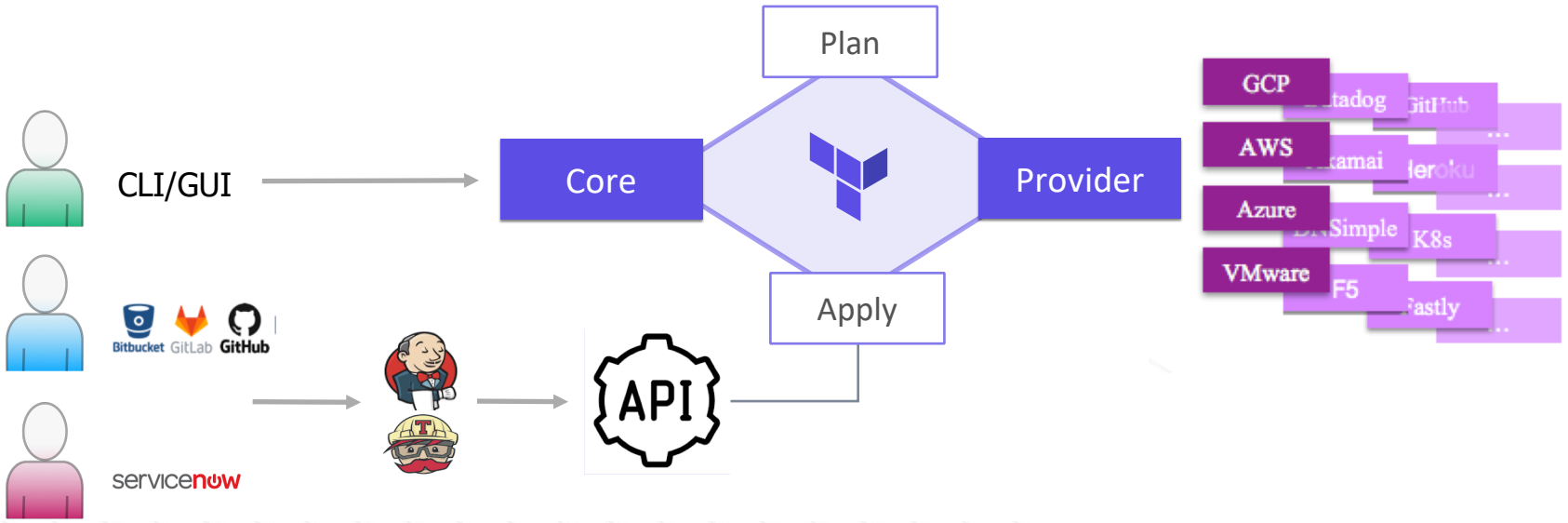
vmware



List of all 70+ providers www.terraform.io/docs/providers/

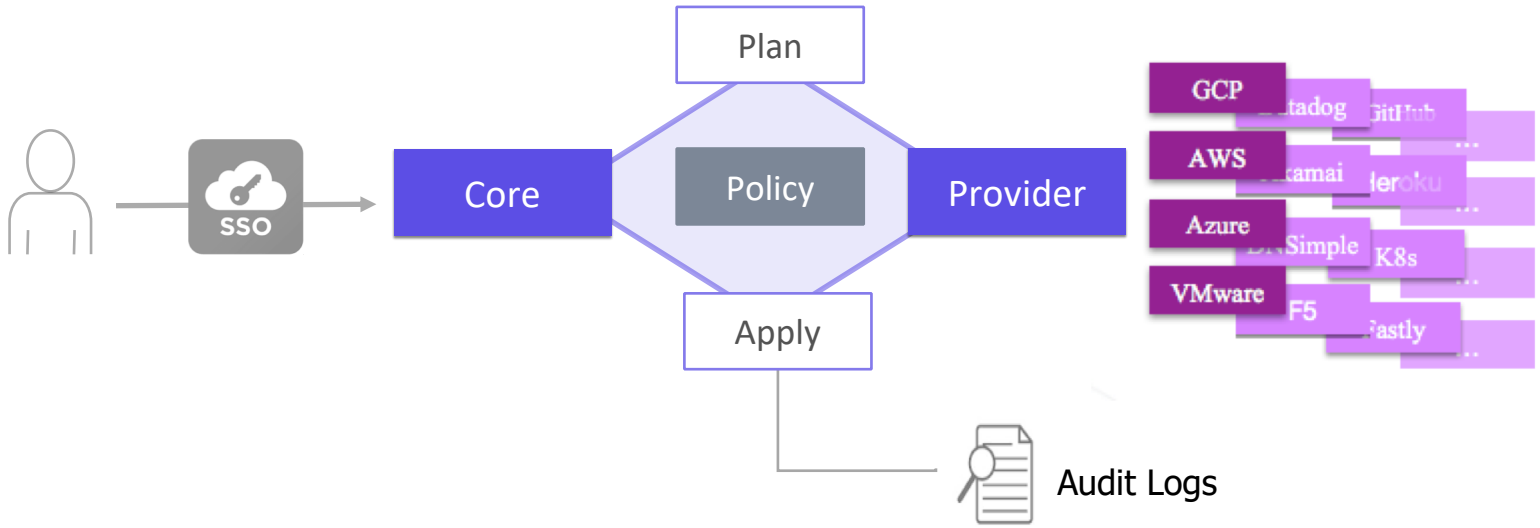
Terraform Enterprise: Operational Efficiency Benefits

- Organizations – Establish Multitenancy
- Workspaces – Centrally organize and manage access to infrastructure by teams/groups
- Remote State Management – Store, version, and collaborate on state
- VCS Connection – Execute TF runs as you commit changes
- API – Integrate Terraform with existing toolchain or workflows



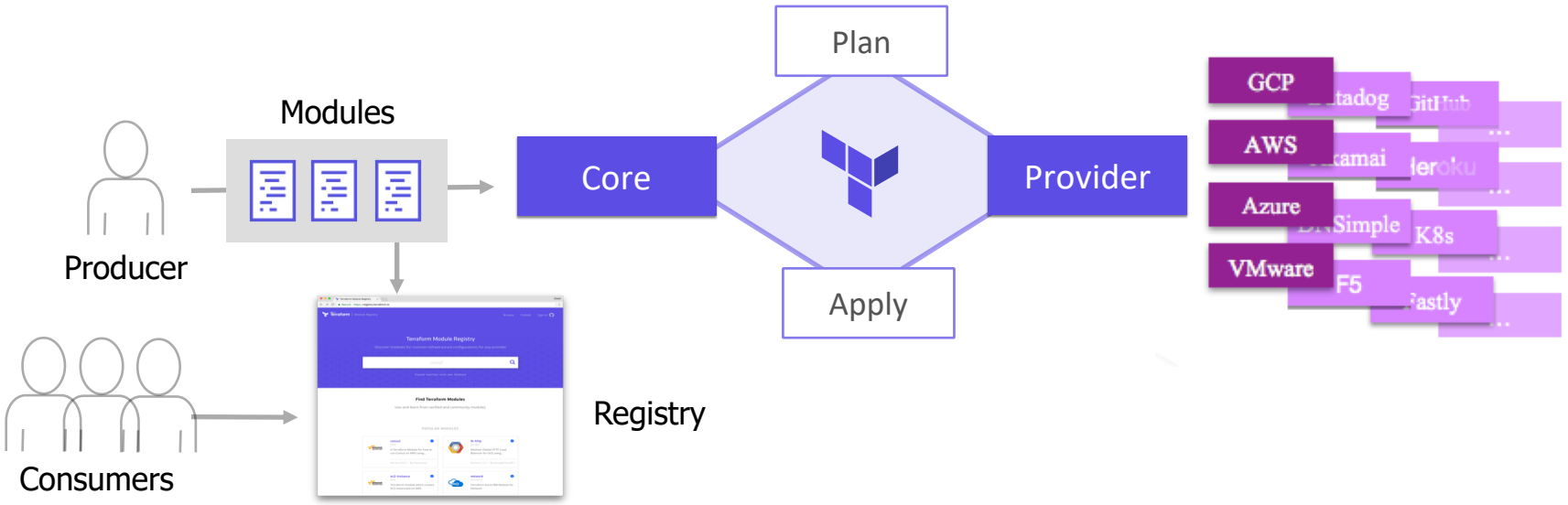
Terraform Enterprise: Governance and Security Benefits

- Policy as code – Sentinel enforces virtually any policy within the path to provision
- Audit Logs – Single pane of glass to track infrastructure changes
- Secure variable management – add, edit, delete all variables from view
- SSO with SAML – Centralize user management



Terraform Enterprise: Self Service Benefits

- Private Module Registry
- Producers: Publish ”approved” modules for consumption by wider audience
- Consumers: Leverage best practice templates developed by SME’s



Ways a Project Grows

Larger Team

More resources to manage (subnets, storage, compute, databases, user accounts)

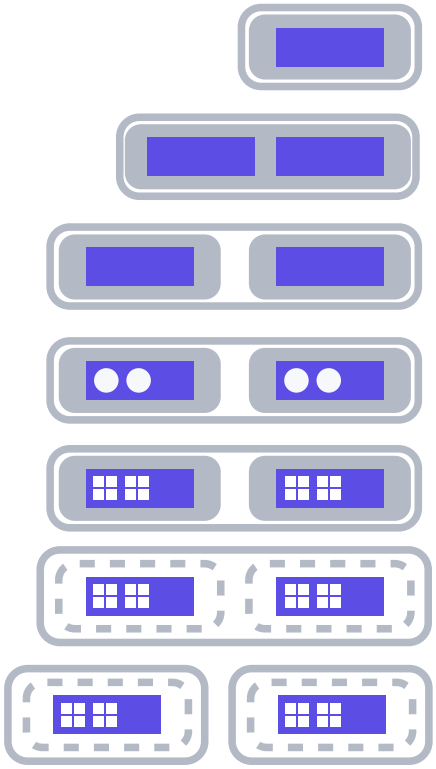
More Resources

More environments in which to duplicate resources (dev, staging, test, prod)

Dev, Test, Staging, Prod

More people building infrastructure (ops, devs, security)

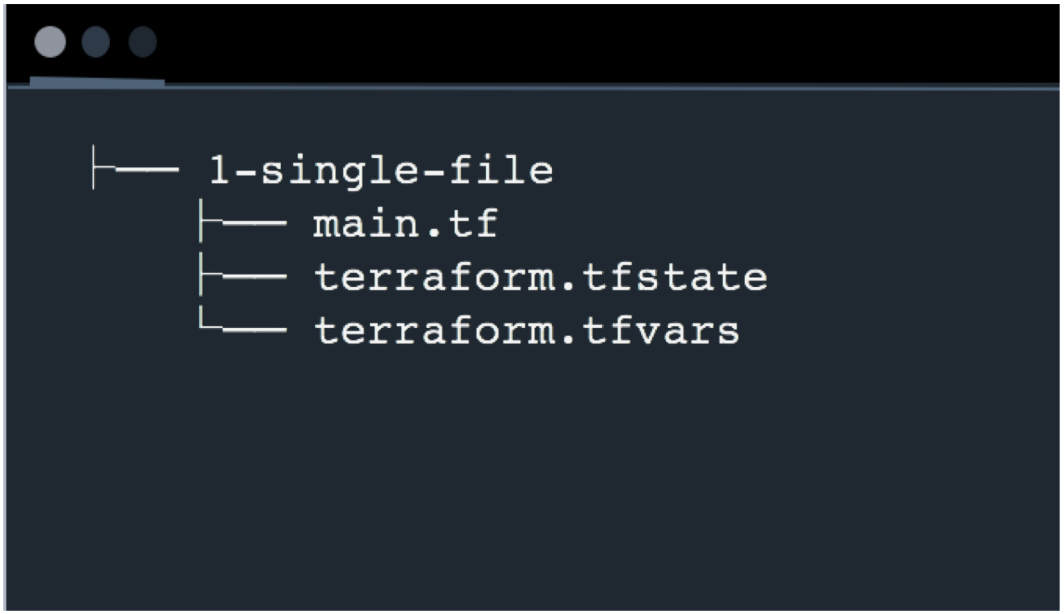
Evolution of a Terraform Project



1. Single File
2. Separate Environments
3. State per Environment
4. Modules
5. Nested Modules
6. Distributed State
7. Repository Isolation

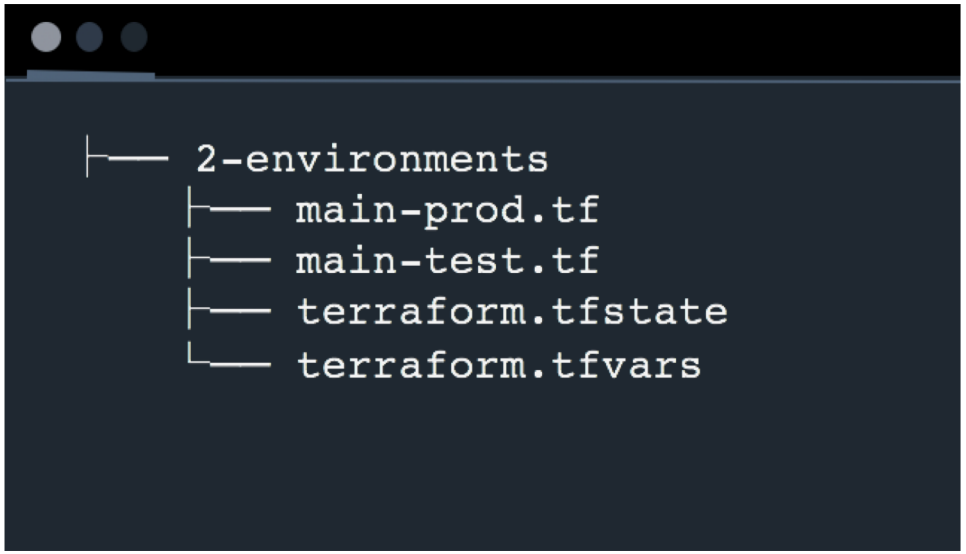
Single File

- main.tf



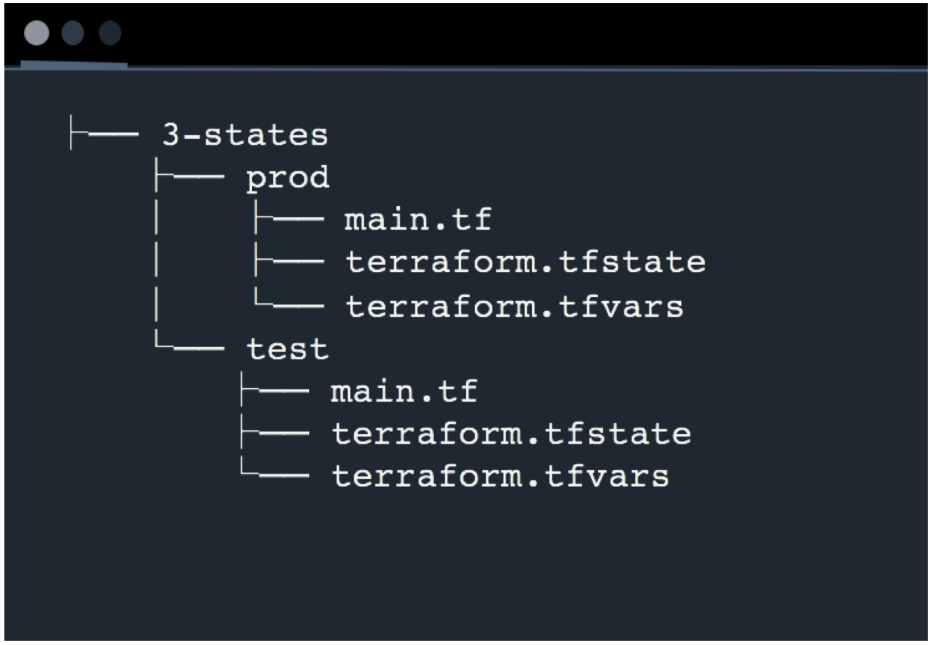
Multiple Environments

- May contain test and prod ("web_test", "web_prod")
- Possibly in different files with a single repo



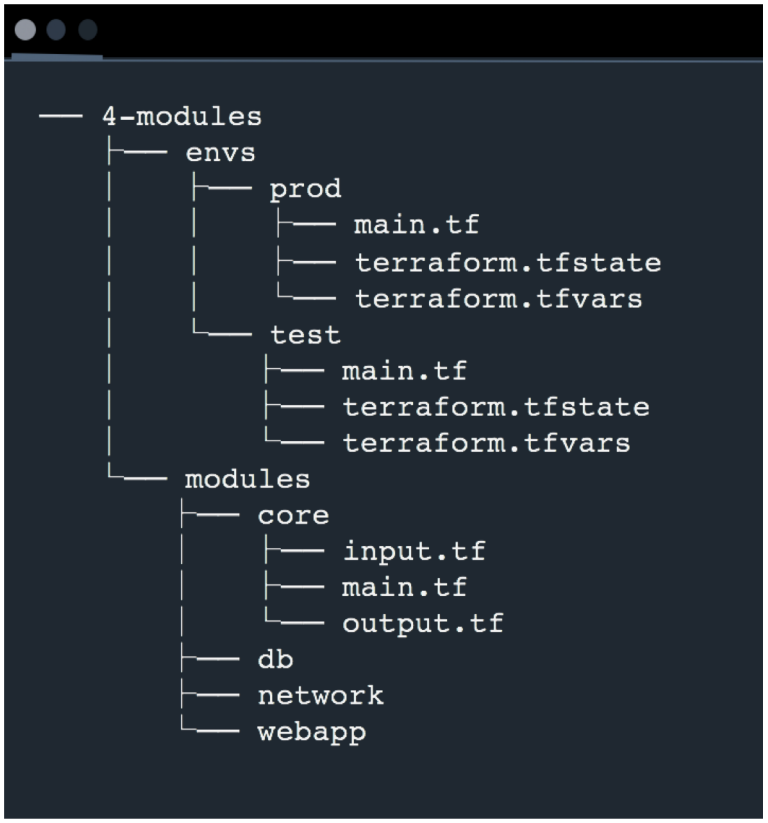
Separate States

- Limit the impact of provisioning one configuration
- Control timing and release schedule of provisioning



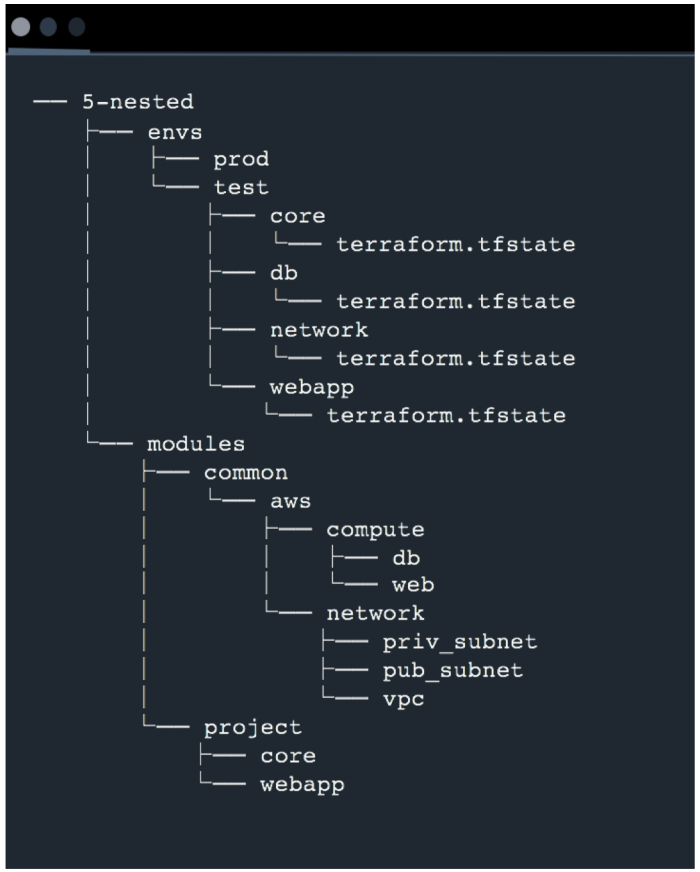
Resources in Modules

- Improves long term maintenance
- Eases reuse
- Encourages consistency in design



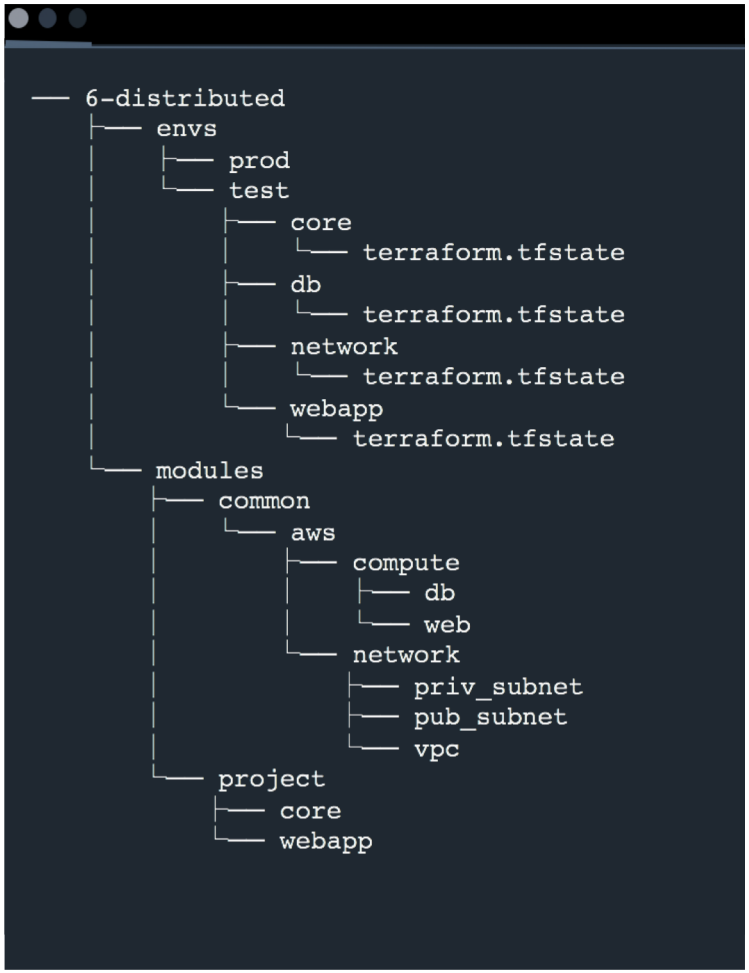
Nested Modules

- A more complex system outgrows single modules
- Able to isolate parts of the system that co-exist but don't depend on each other



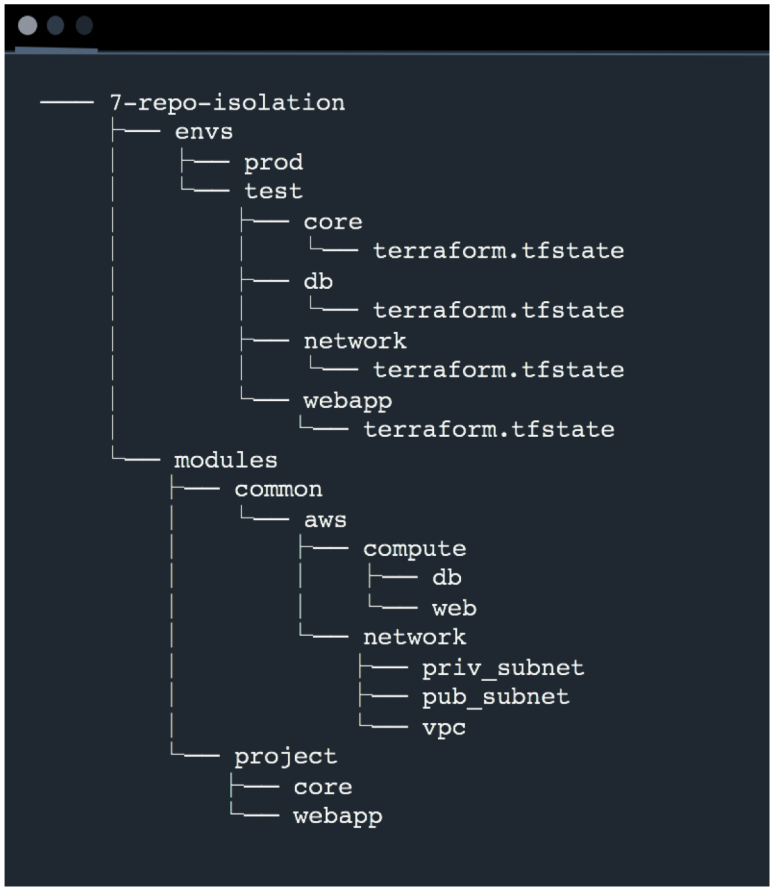
Distributed State

- Necessary for teams
- Further isolates individual environments and deployments



State/ Repo per Component

- Full isolation
- Control team read/write permissions on each repo
- Full control over authorization per state
- Limit Terraform Enterprise apply actions



Terraform Collaboration Answers

- Is this config owned by one team?
- Make a separate repository. Producing infrastructure
- Is this config used by more than one team?
- Publish a module. Creating infrastructure
- Does this config create resources?
- Publish outputs. Existing within infrastructure

Terraform Collaboration Answers

- Is this config owned by one team?
- Make a separate repository. Producing infrastructure

- Is this config used by more than one team?
- Publish a module. Creating infrastructure

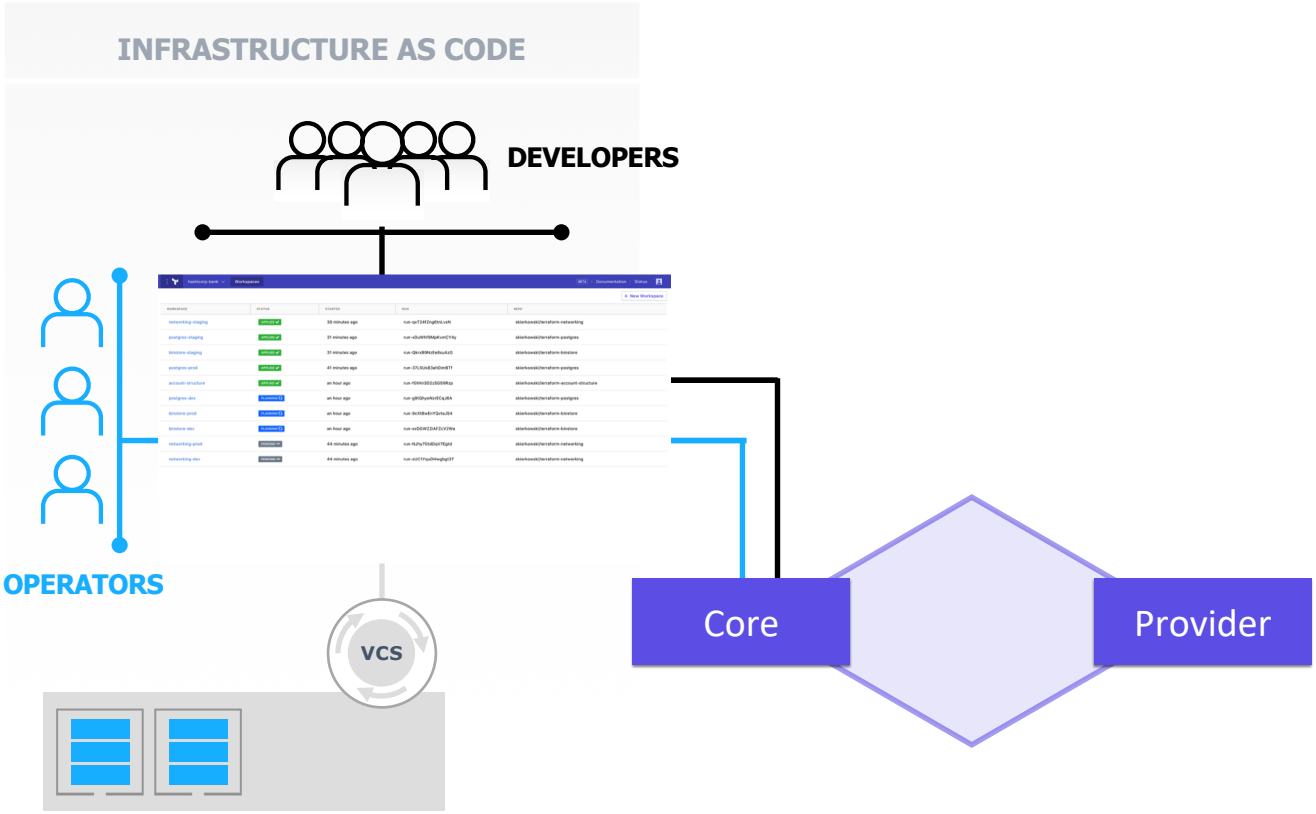
Terraform Collaboration Answers

- Do you need this component to be updated independently from another?
- Separate config in its own repo

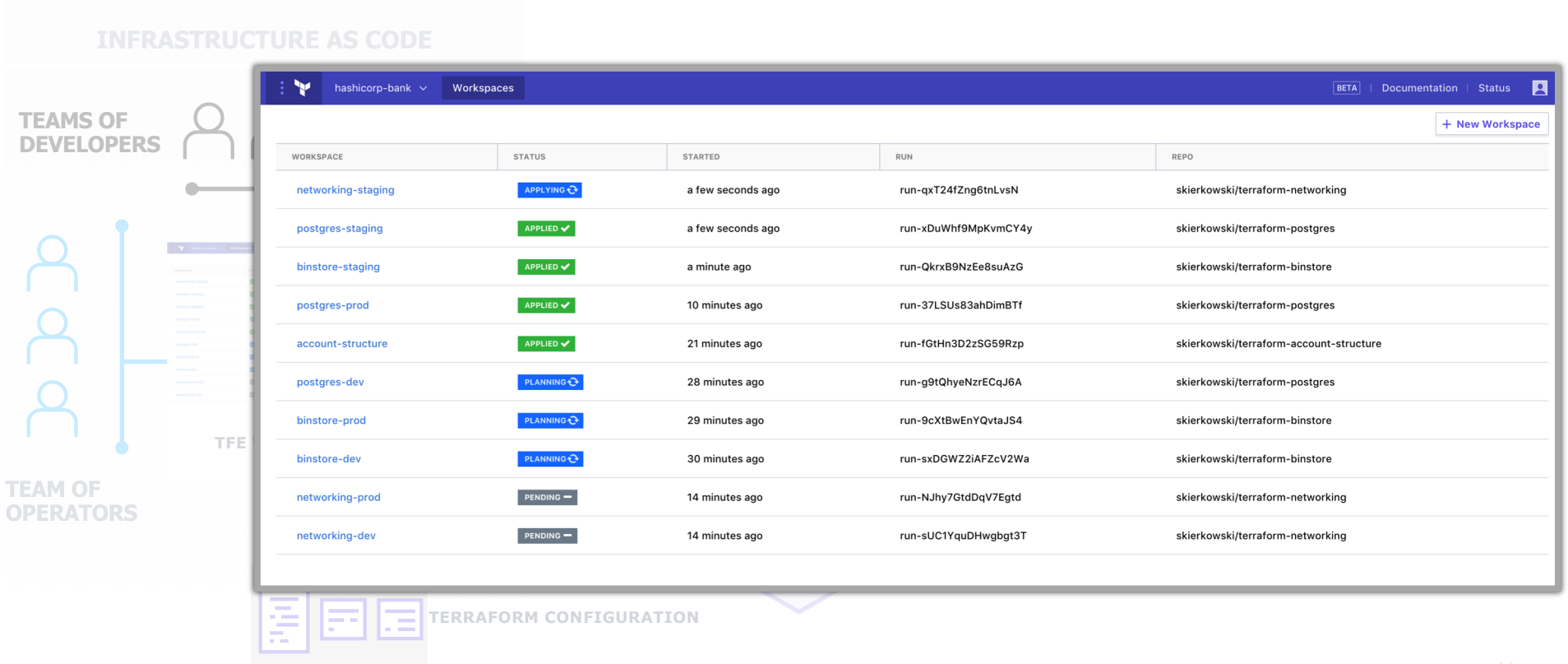
- Does this config create resources?
- Publish outputs. Existing within infrastructure

Workspaces

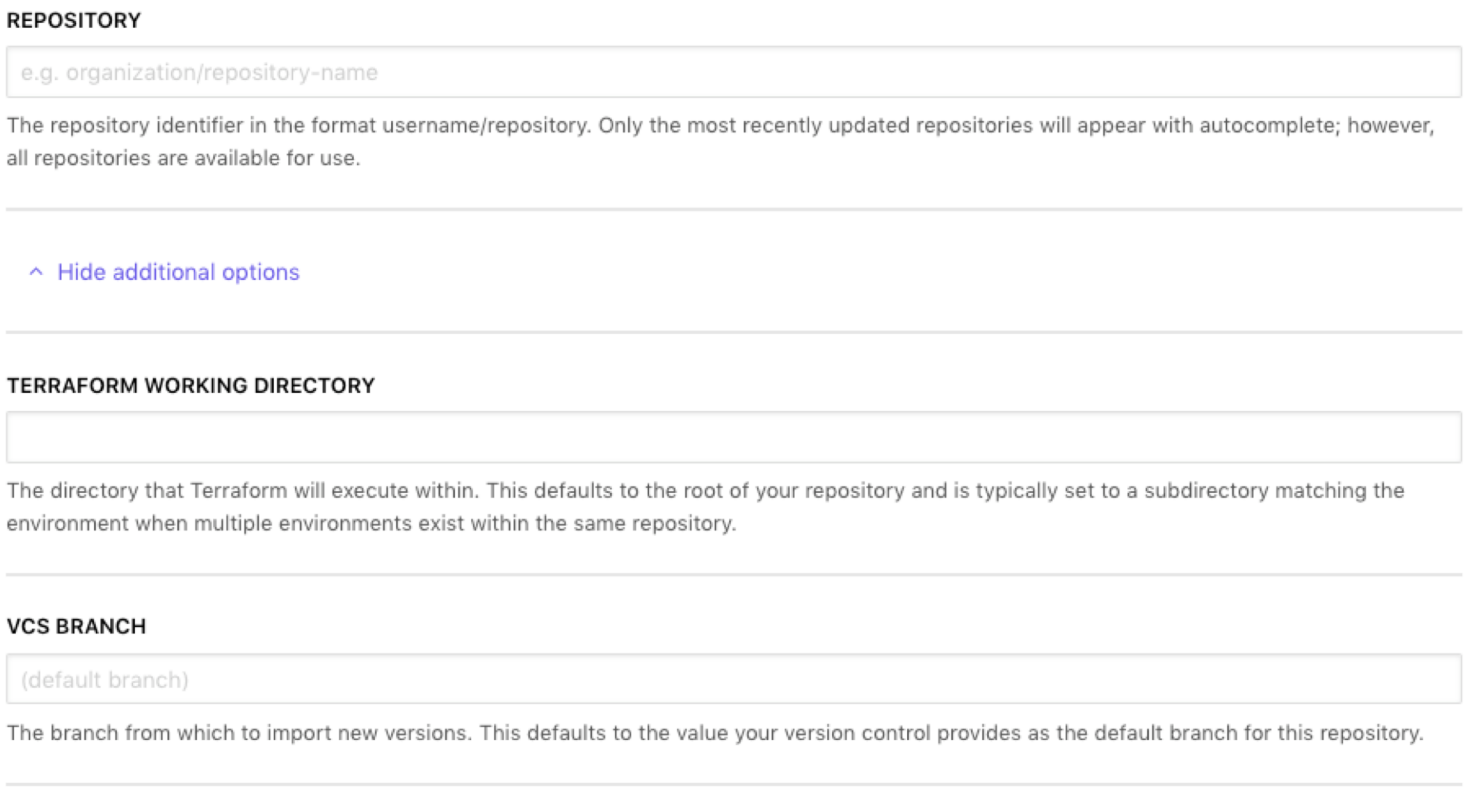
Workspace management and VCS connection



Workspace management and VCS connection

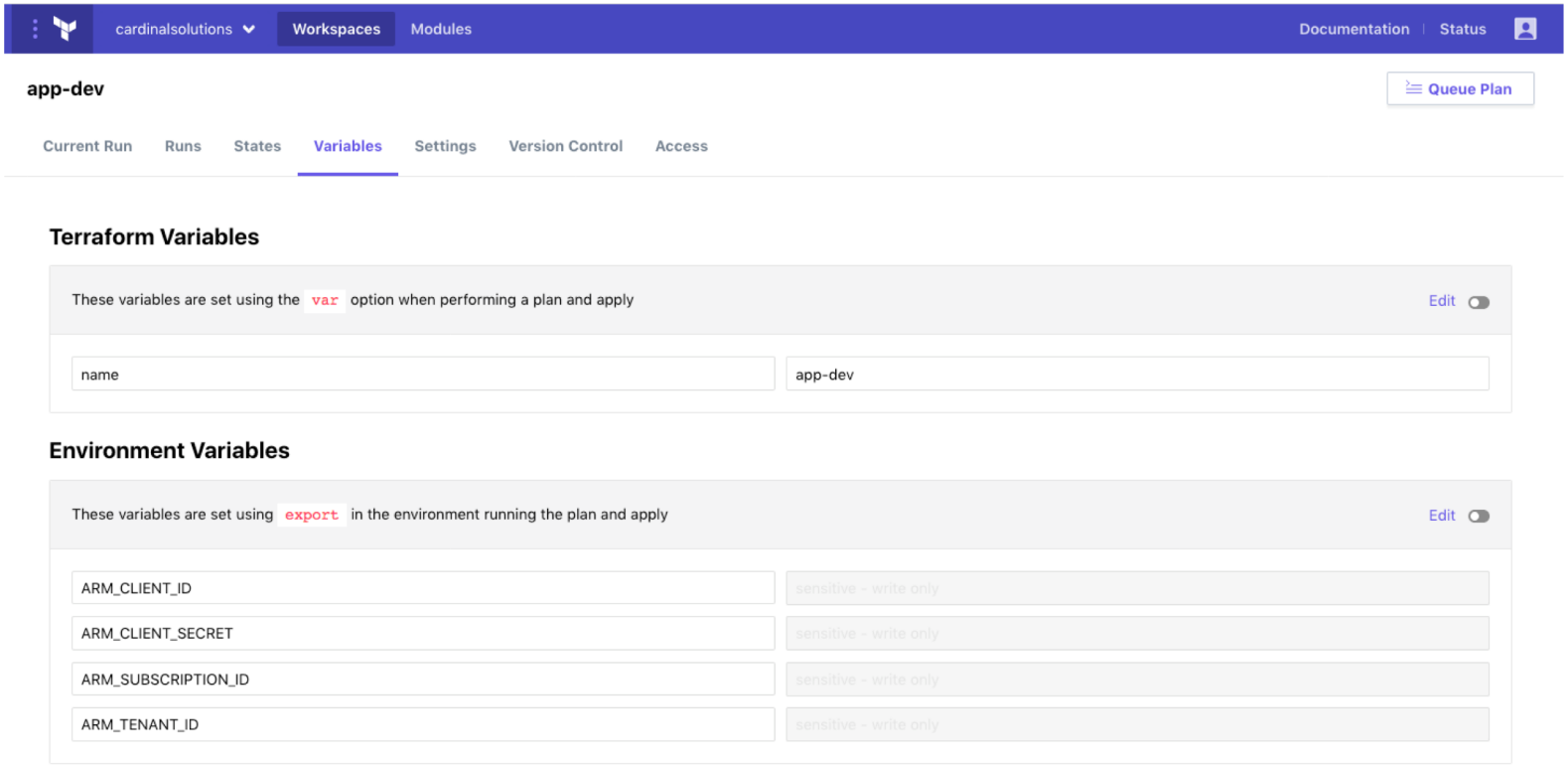


Workspace – VC Connection



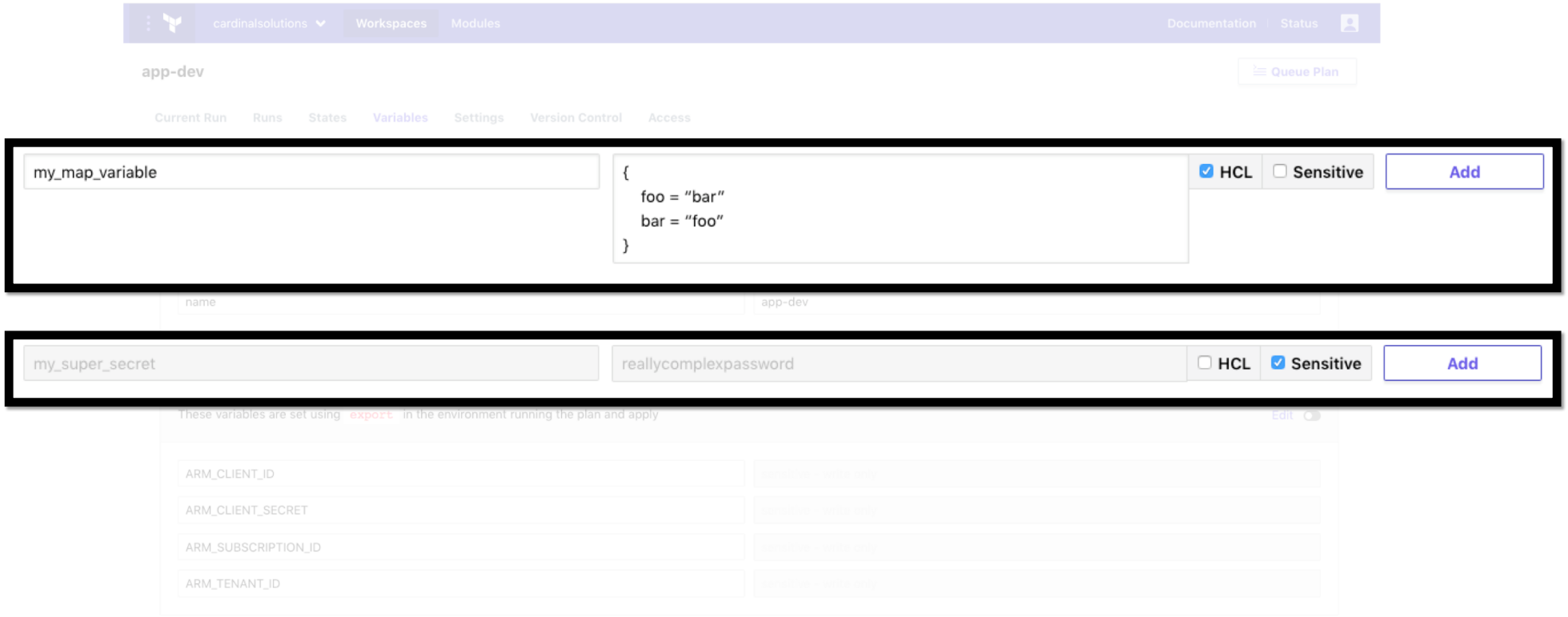
https://www.terraform.io/docs/enterprise/vcs

Workspace – Variables



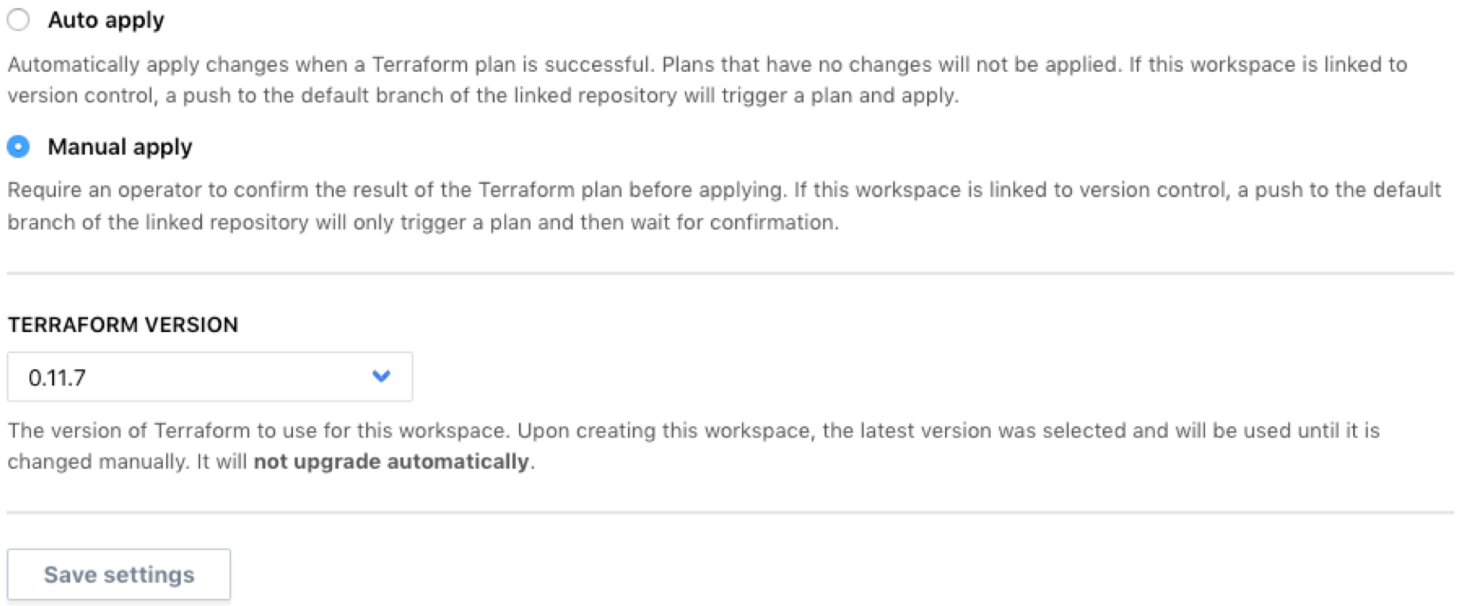
https://www.terraform.io/docs/enterprise/workspaces/variables.html

Workspace – Variables



https://www.terraform.io/docs/enterprise/workspaces/variables.html

Workspace – Settings



https://www.terraform.io/docs/enterprise/workspaces/variables.html

Workspace – Settings

Workspace Delete

There are two independent steps for destroying this workspace and any infrastructure associated with it. First, any Terraform infrastructure should be destroyed. Second, the workspace in Terraform Enterprise, including any variables, settings, and alert history can be deleted.

Queueing a destroy Plan will redirect to a new Plan that will destroy all of the infrastructure managed by Terraform. This requires confirmation before Apply. It is equivalent to running `terraform plan -destroy -out=destroy.tfplan` followed by `terraform apply destroy.tfplan` locally.

Queueing a destroy Plan will be disabled until there is an environment variable set named `CONFIRM_DESTROY` with a value of `1` . You can use the variables page to set it.

Queue destroy Plan

Delete from Terraform Enterprise

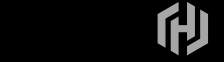
https://www.terraform.io/docs/enterprise/workspaces/variables.html

Labs

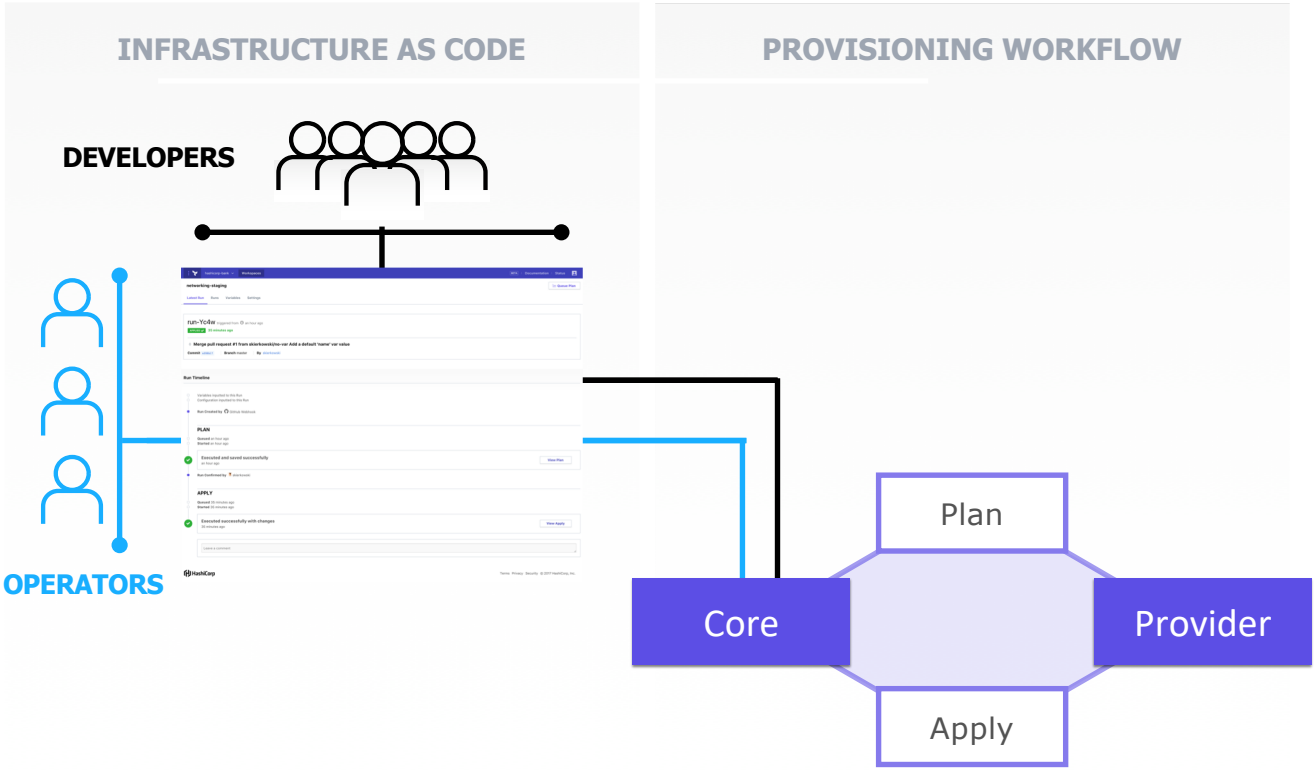
Lab Github and Instructions

- Have your AWS Credentials Handy
- Sign up for a new account at http://app.terraform.io/account/new?trial=terraform
- Send your new account user name to Lab Instructor. He/she will add you to the lab organization
- Go to https://github.com/kawsark/aws-terraform-workshop to lab instructions

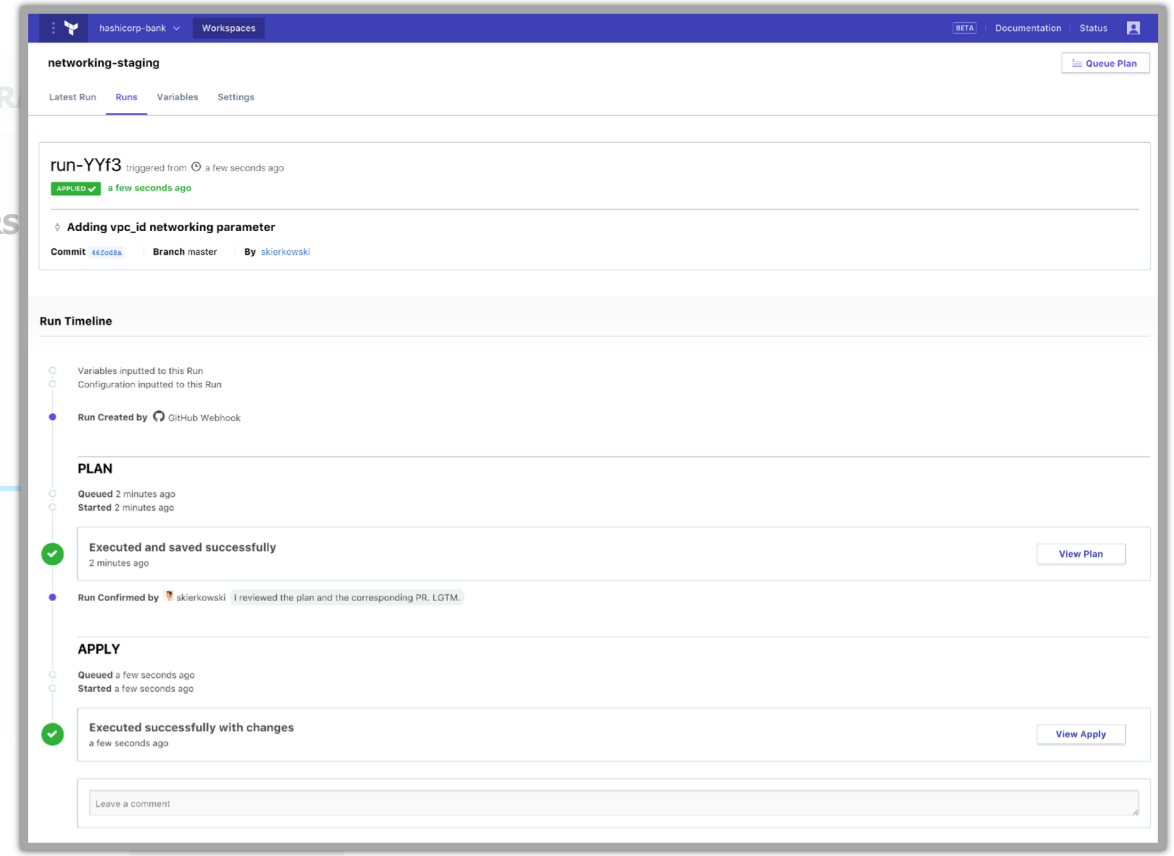
Remote State



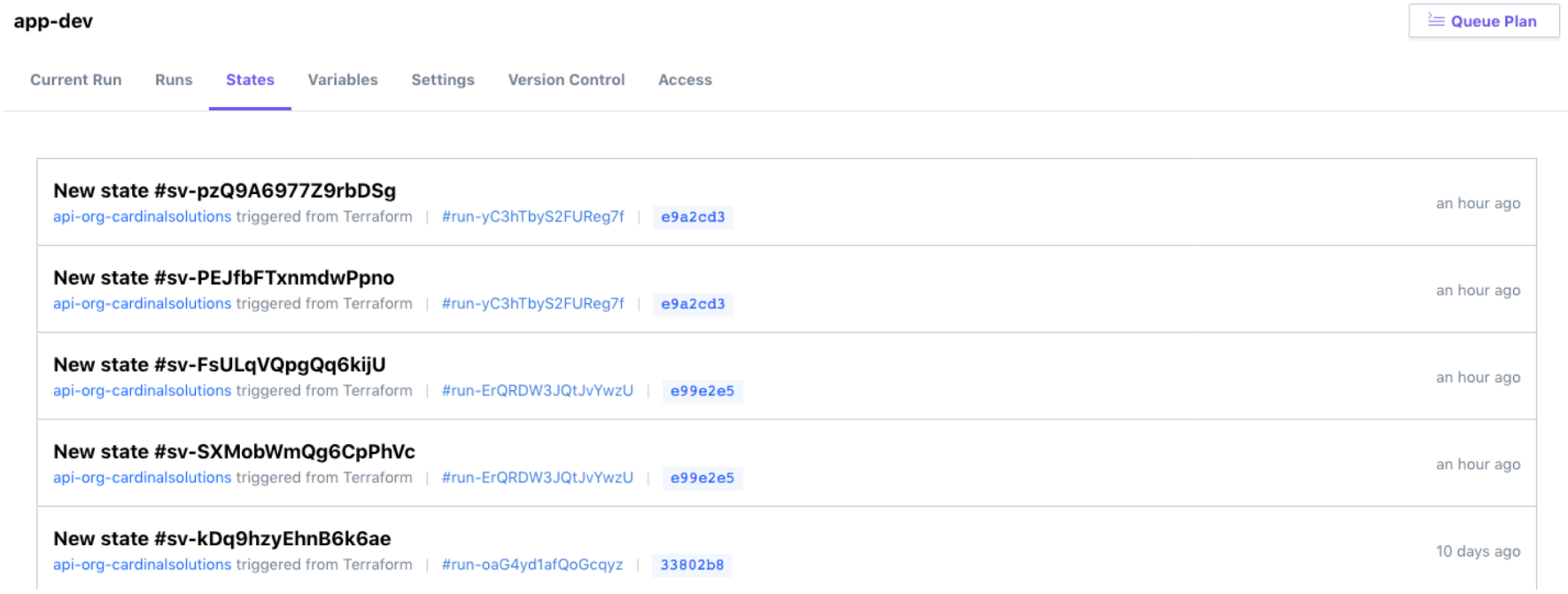
Remote Terraform runs and state management



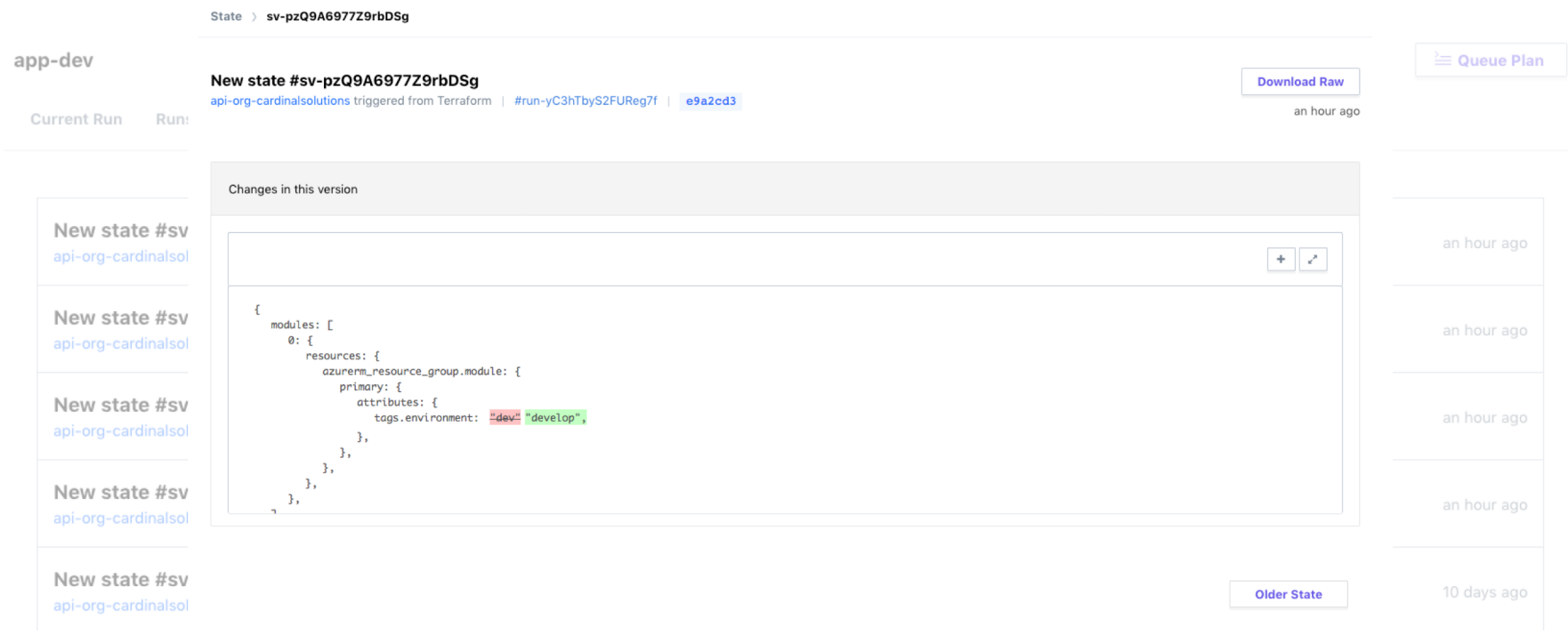
Remote state management and Terraform runs



Workspace – State Management



Workspace – State Management



Referencing Remote State

```
data "terraform_remote_state" "k8s_cluster" {
          backend = "atlas"
          config {
          name = "${var.tfe_organization}/${var.k8s_cluster_workspace}"
          }
}
```

```
provider "kubernetes" {
host = "${data.terraform_remote_state.k8s_cluster.k8s_endpoint}"
client_certificate = "${base64decode(data.terraform_remote_state.k8s_cluster.k8s_master_auth_client_certificate)}"
client_key = "${base64decode(data.terraform_remote_state.k8s_cluster.k8s_master_auth_client_key)}"
clusterPca_certifcate="${base64decode(data.terraform_remote_state.k8s_cluster.k8s_master_auth_cluster_ca_certificate)}"
} }
```


Require AWS tags on all instances

```
main = rule {

  // All resources in the plan
  all tfplan.resources as r {

    // Must not be an instance
    r.type != "aws_instance" or

    // Or if they are, must have a billing-id tag
    r.tags contains "billing-id"

  }
}
```

Collaboration

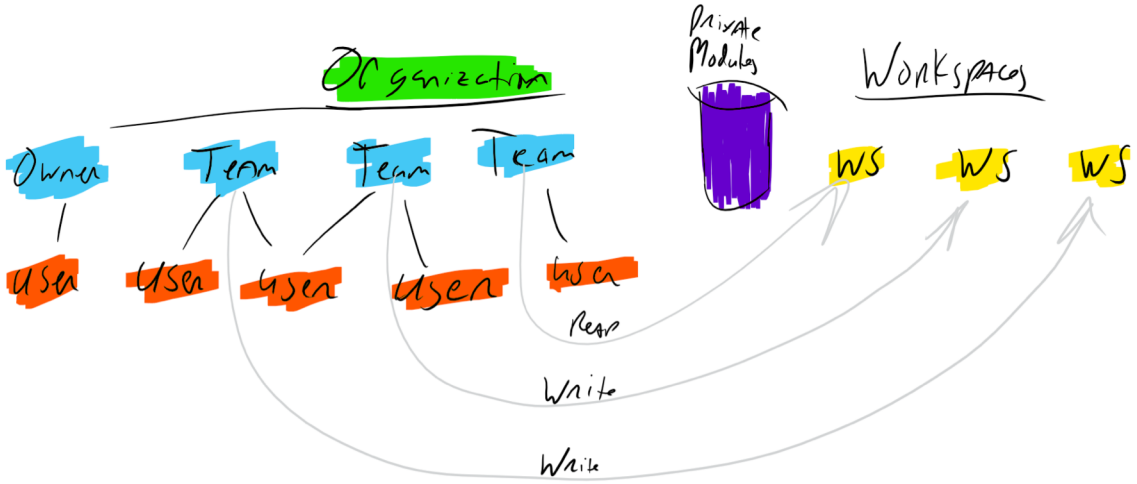
Collaboration

Role Based Access Controls

Organization

Teams

Users



Service Accounts

API Tokens

- User
- Team
- Organization

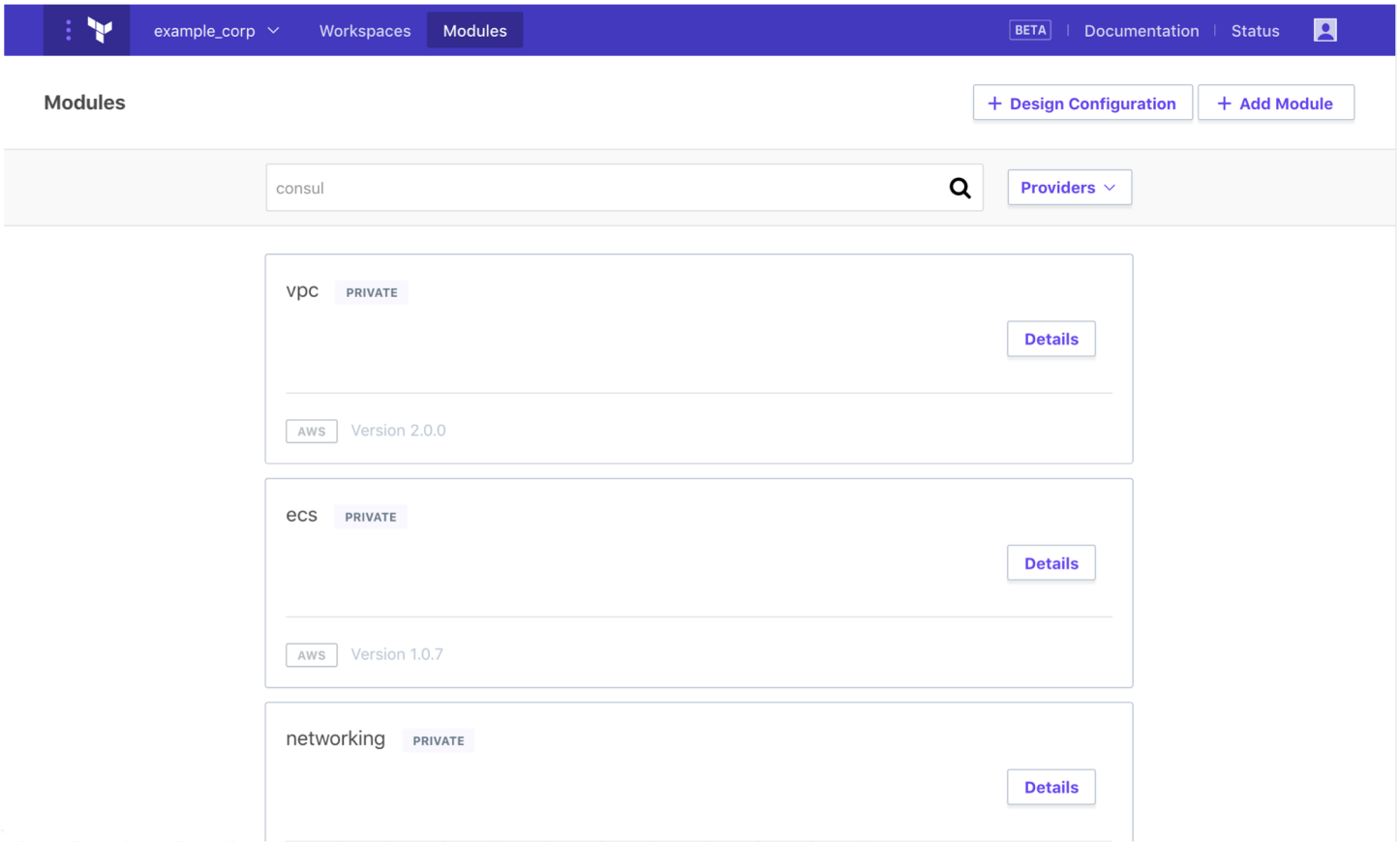
Private Module Registry

Private Module Registry

Modules for sharing across organization

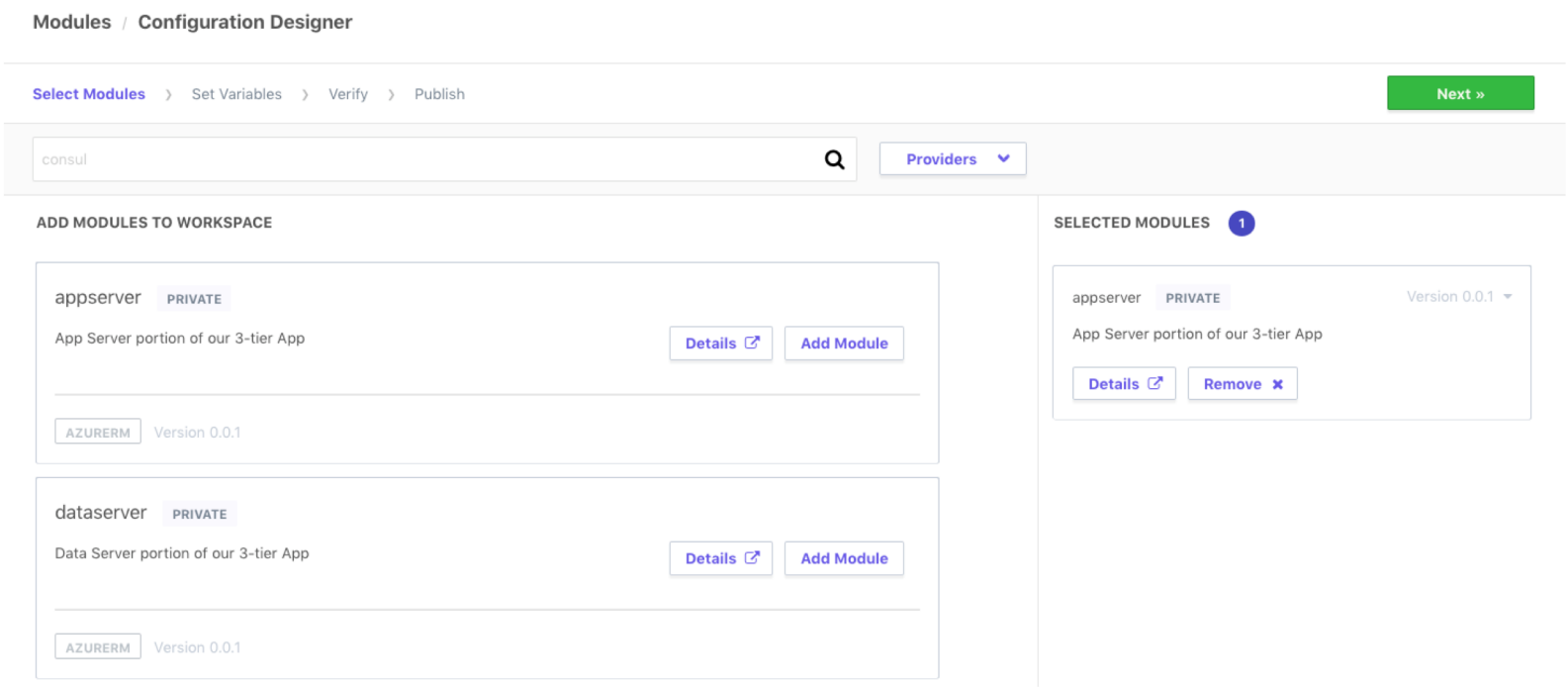
Versioning

Private Module Registry

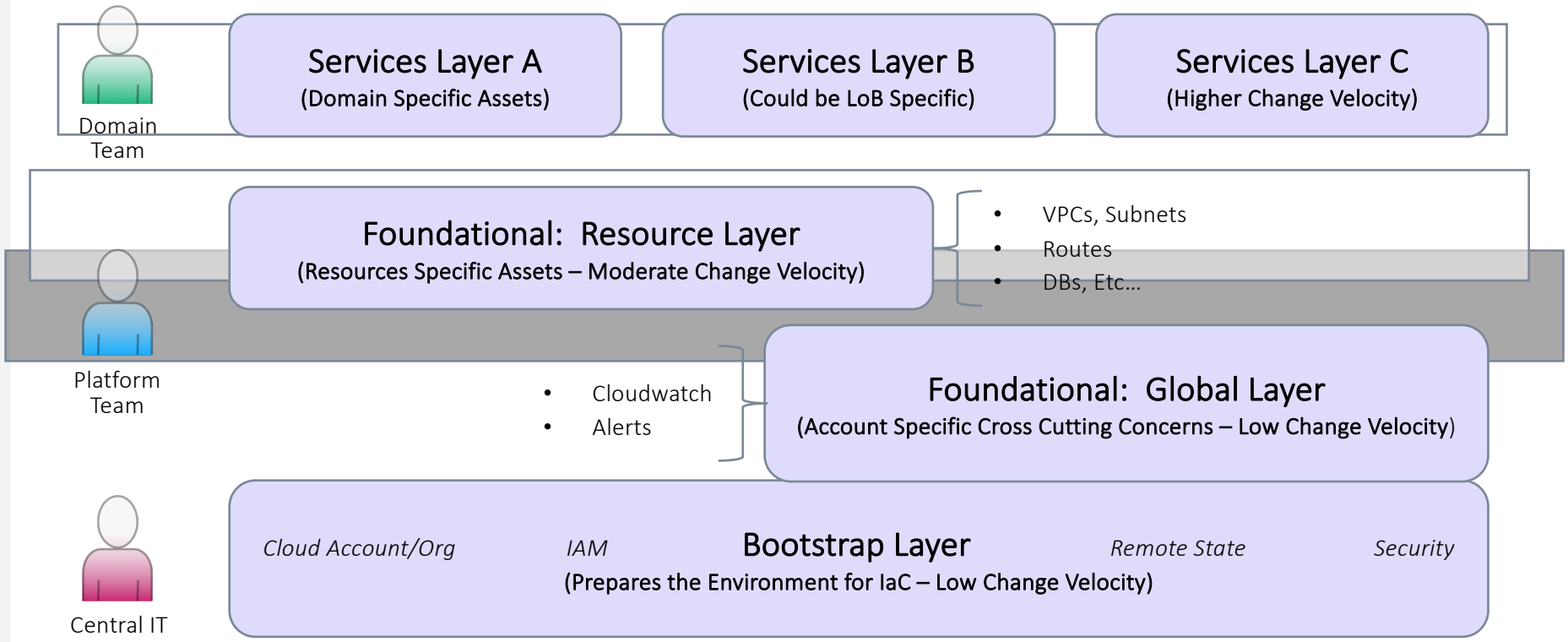


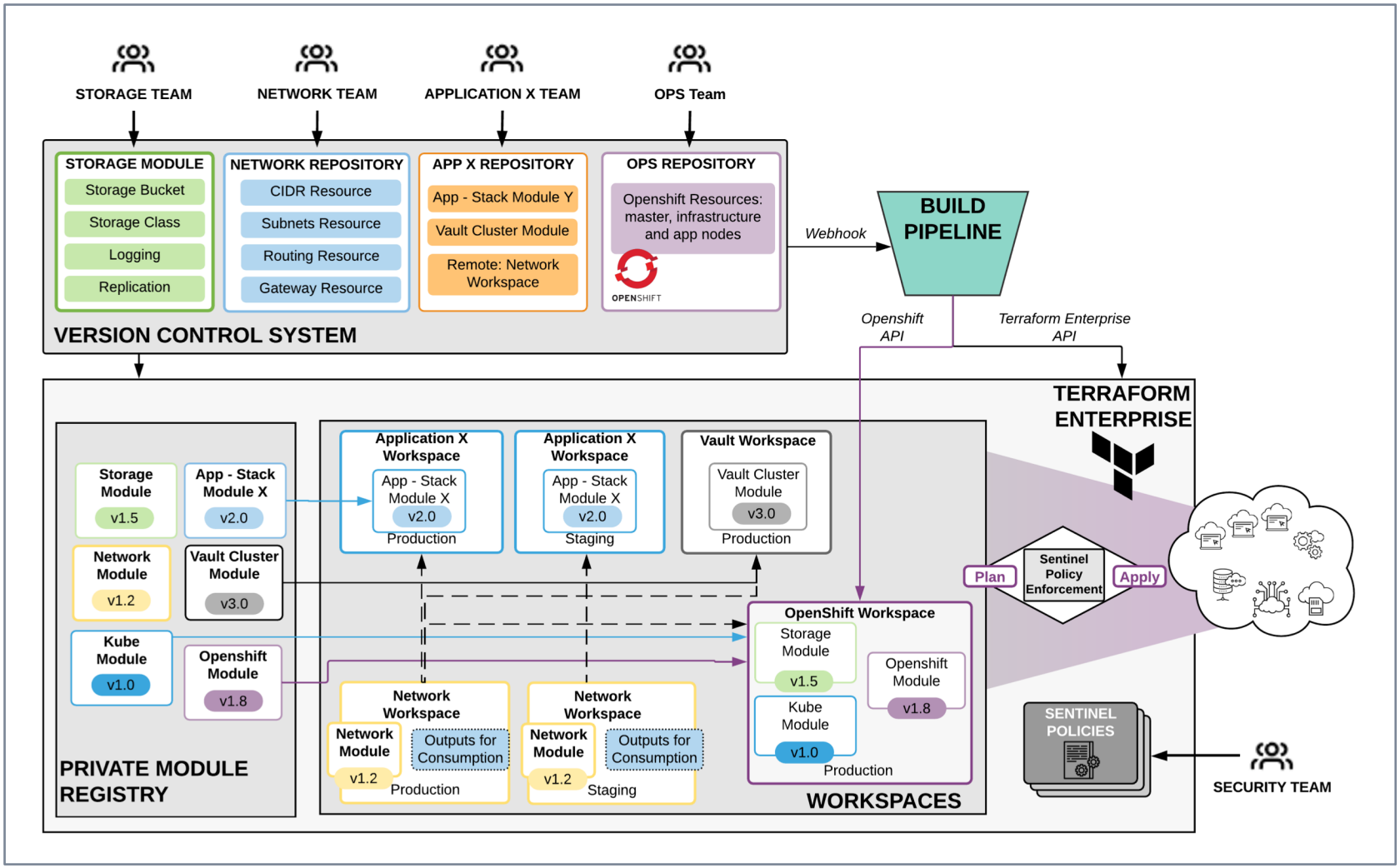
https://www.terraform.io/docs/enterprise/registry

Design Configuration



Layered approach to Infrastructure as Code





Governance

What is Sentinel

Sentinel is an *embeddable* policy as code framework to enable *fine-grained*, *logic-based* policy decisions that can be *extended* to source external information to make decisions.

What is Sentinel

Policy as Code

Treat policy like an application — version control, pull review, and automate tests. Use programming constructs to determine policy decisions beyond the limited constraints of typical ACL systems.

Multiple Enforcement Levels

Advisory, soft-mandatory, and hard-mandatory levels allow policy writers to warn on or reject offending behavior.

Fine-grained, condition-based policy

Reject actions on any available input rather than coarse-grained read, write, and admin policies. Make policy decisions based on the condition of other values.

External Information

Advisory, soft-mandatory, and hard-mandatory levels allow policy writers to warn on or reject offending behavior.

Embedded

Sentinel is embedded to enable policy enforcement in the data path to actively reject violating behavior instead of passively detecting.

Multi-Cloud Compatible

Ensure infrastructure changes are within business and regulatory policy on every infrastructure provider.

Policy as Code

- **Use real programming constructs to determine policy decisions** beyond the limited constraints of typical ACL systems.
- **Treat policy like an application:** version control, pull review, automate tests

Fine-grained, condition-based policy

- **Reject actions on any available input** rather than coarse-grained read, write, admin policies
- **Make policy decisions based on the condition of other values.** For example, disallow volumes in Nomad only if the “docker” driver is being used

Embedded

- **Enables policy enforcement in the data path** to actively reject violating behavior instead of passively detecting

Source external information

- **Plugin system to source external information** allows for holistic policy decisions.

Example: Enforce Terraform decisions based on data in Consul — don't allow Terraform to execute while Consul health checks are failing

- **Easy to write new plugins** so organizations can source data in their own systems — ServiceNow, Datadog, etc

Multiple Enforcement Levels

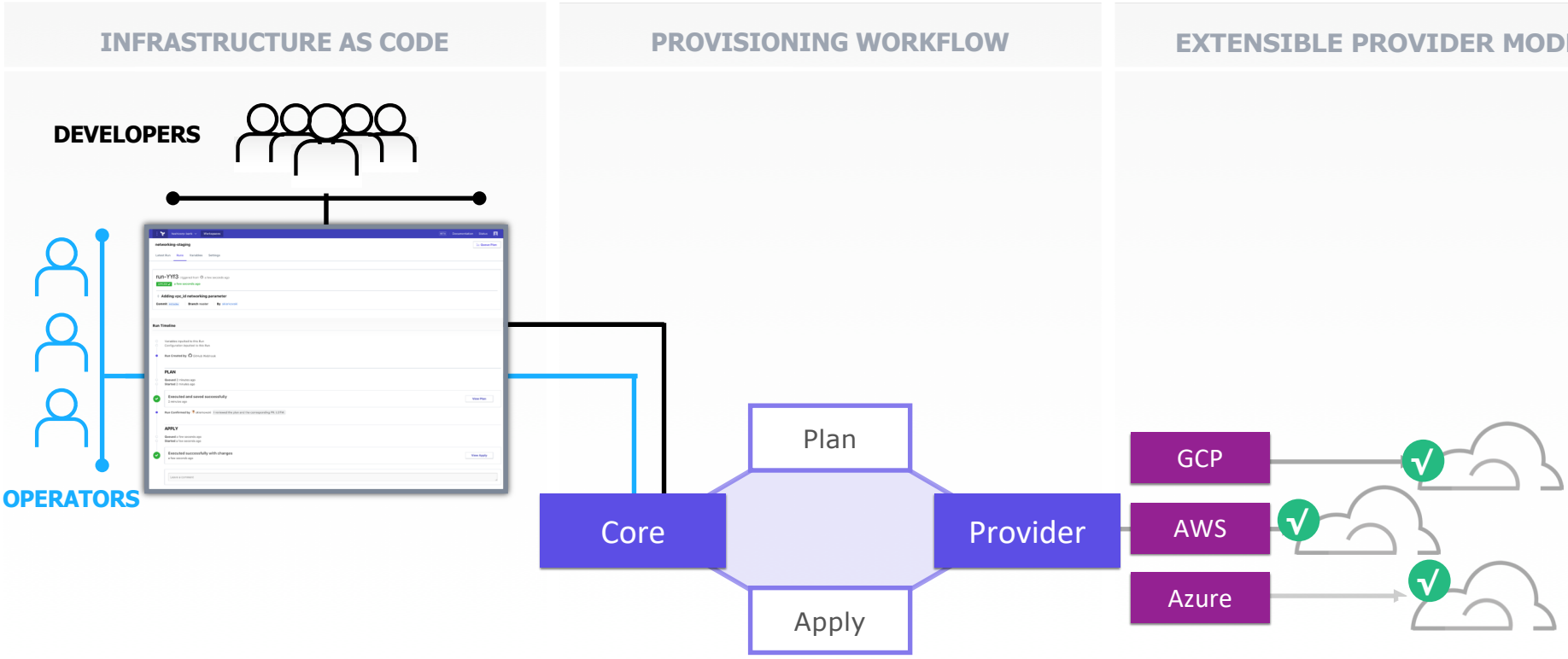
- **Advisory, soft-mandatory, and hard-mandatory** levels allow policy writers to warn or reject offending behavior.
- **Advisory** can be used as a tool to educate new users. For example, warn a user if they aren't following internal infrastructure patterns.

Multi-cloud compatible

- Ensure infrastructure changes are within business and regulatory policy on every infrastructure provider.

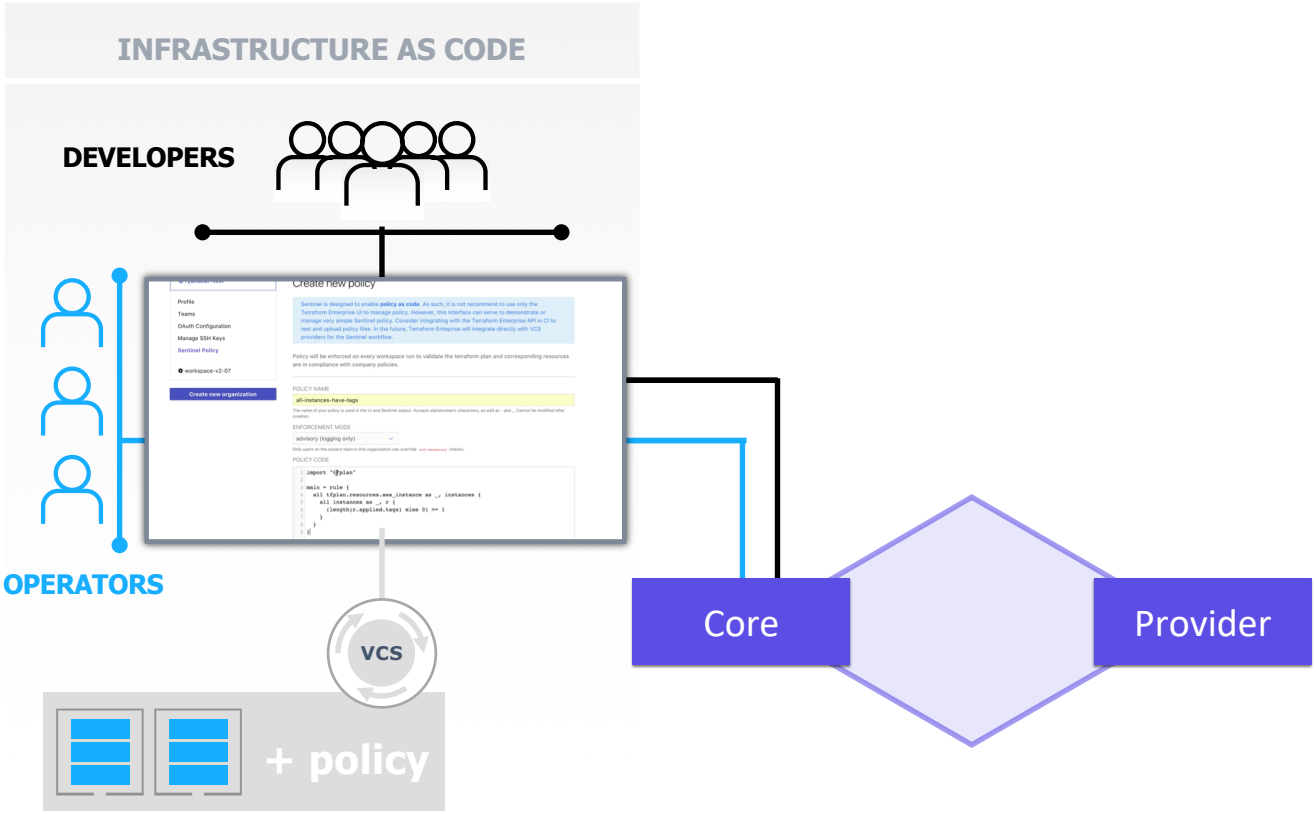
Enable Organization to have governance over infrastructure

Ensure provisioning policy is being maintained to minimize risk to the business



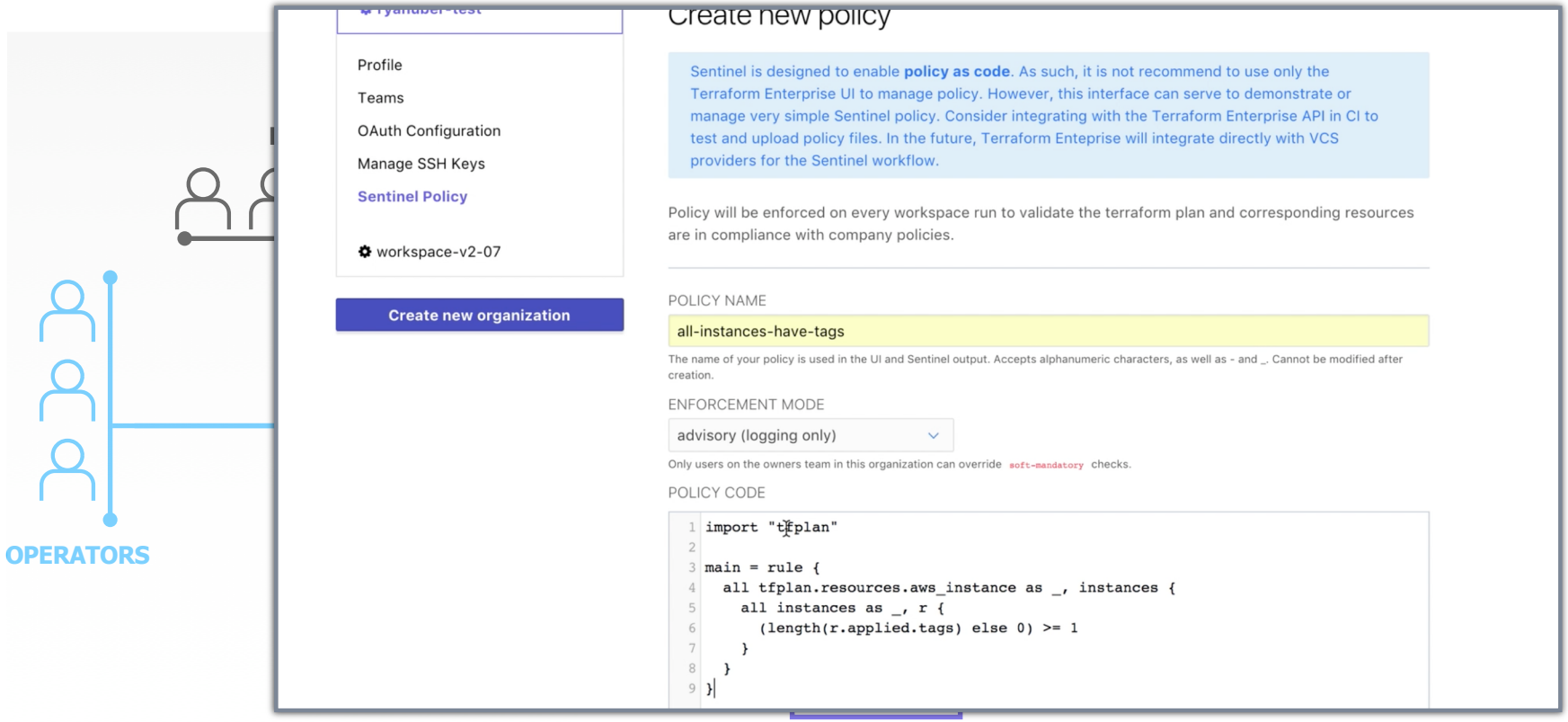
Enable Organization to have governance over infrastructure

Automate policy with Sentinel policy as code management



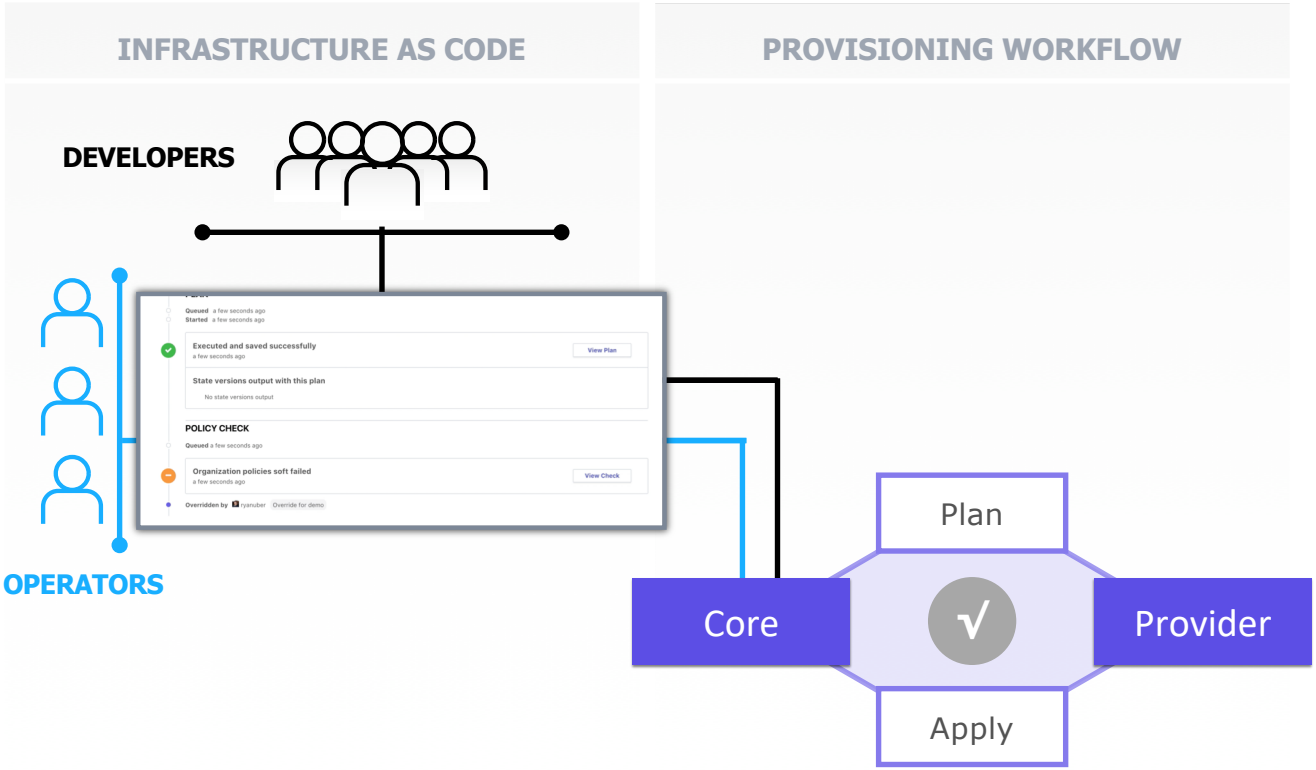
Enable Organization to have governance over infrastructure

Ensure provisioning policy is being maintained to minimize risk to the business



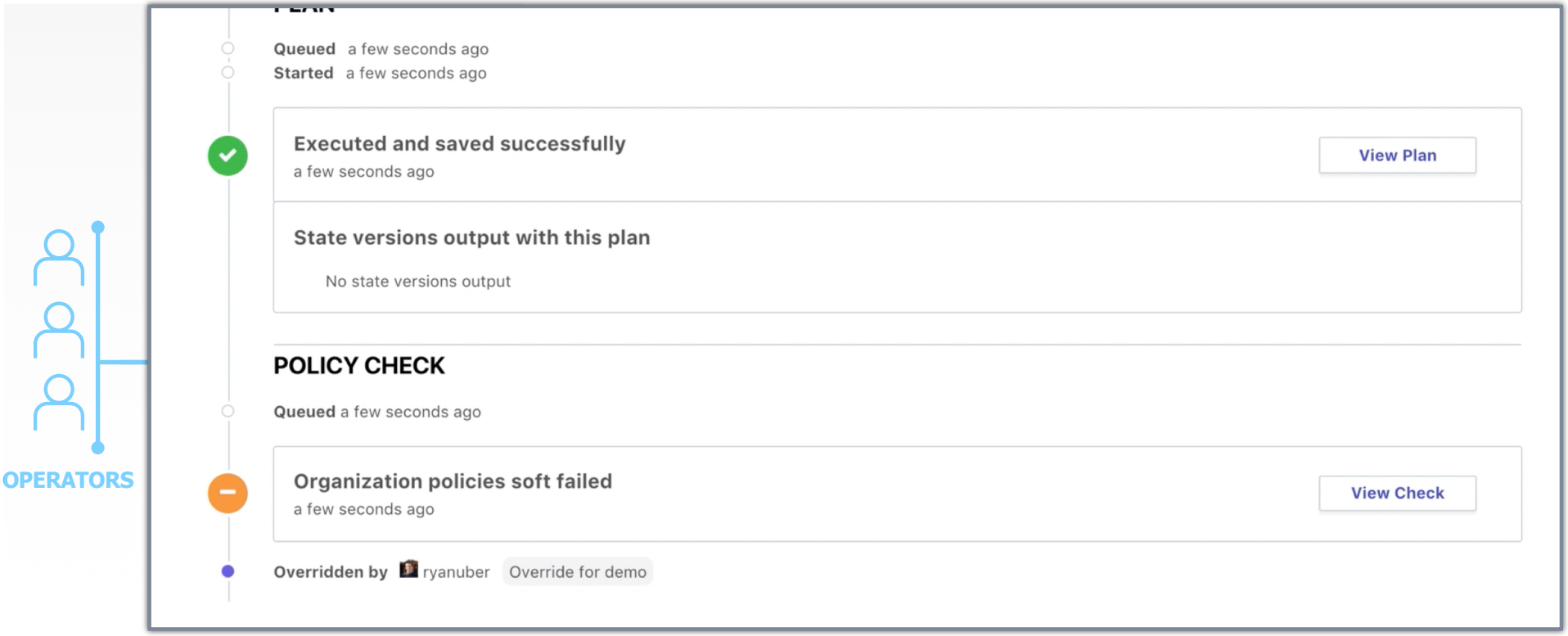
Enable Organization to have governance over infrastructure

Automate policy to be enforced on every Terraform run



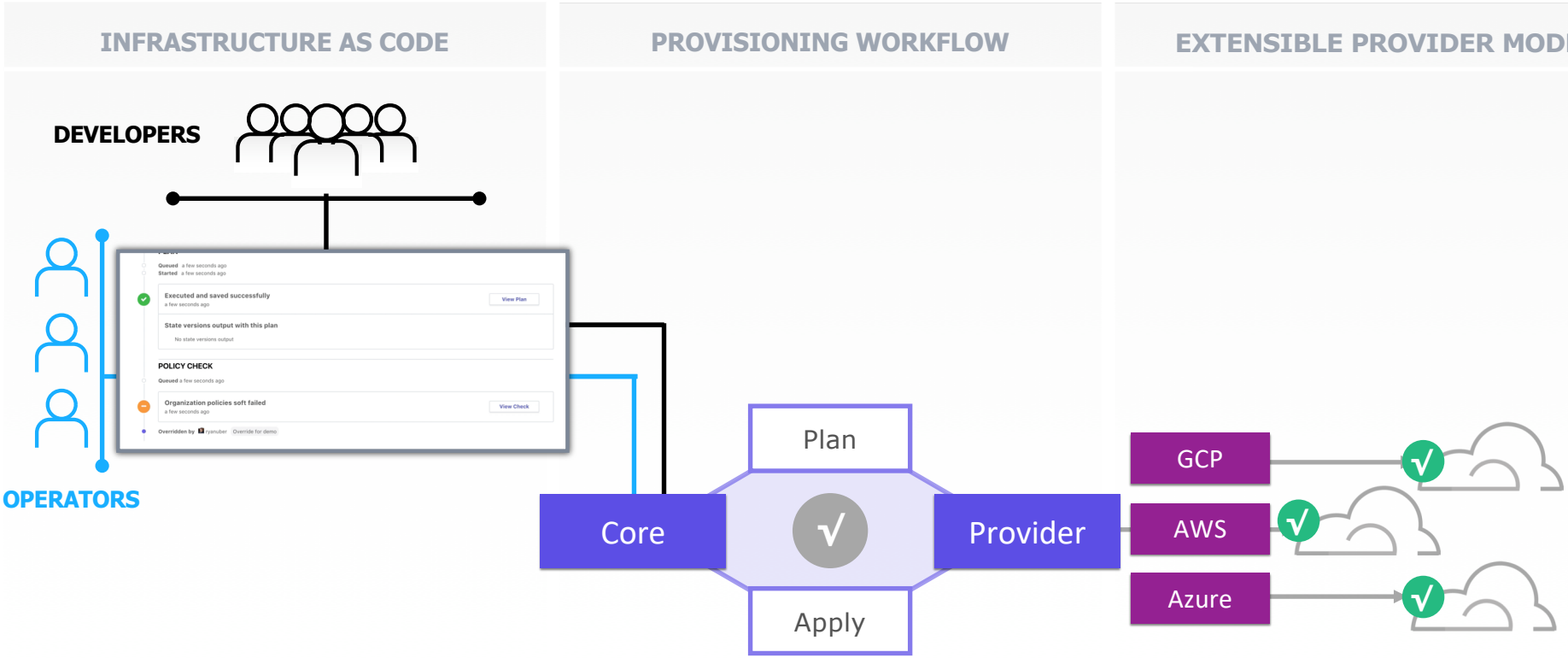
Enable Organization to have governance over infrastructure

Ensure provisioning policy is being maintained to minimize risk to the business



Enable Organization to have governance over infrastructure

Automate policy to be enforced on every Terraform run



Applied Sentinel

Example Policies for Terraform

Terraform

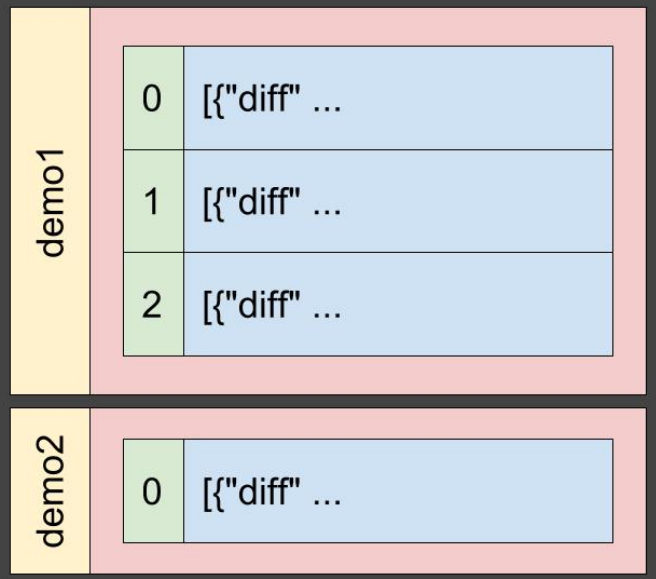
- Do not allow resources to be provisioned outside of business hours
- Require AWS tags on all instances
- Only allow "staging" resources to be provisioned in `us-west-1` and "production" resources in `us-east-1`
- Deployments must be to more than one AZ
- No security groups can have 0.0.0.0/0 egress

How to write Sentinel policies

- Start with a plain English policy
  - Require AWS tags on all instances
- Determine rules by splitting it by logic operator
  - Require AWS tags on all instances
- Map rules to the embedded data structure
  - All resources: `all tfplan.resources`
    - All instances: `r.type != “aws_instance”`
    - Must have a tag: `r.tags contains “billing-id”`

```
resource "aws_instance" "demo1" {
  count         = 3
  ami           = "${data.aws_ami.ubuntu.id}"
  instance_type = "t2.nano"
  subnet_id     = "${aws_subnet.public.id}"
}

resource "aws_instance" "demo2" {
  ami           = "${data.aws_ami.ubuntu.id}"
  instance_type = "m4.large"
  subnet_id     = "${aws_subnet.public.id}"
}
```

```
import "tfplan"

allowed_instance_types = [
    "t2.nano",
    "t2.micro",
    "t2.small",
]

main = rule {
  all tfplan.resources.aws_instance as name, resource {
    all resource as index, instance {
      instance.applied.instance_type in allowed_instance_types
    }
  }
}
```



Require AWS tags on all instances

```
main = rule {

  // All resources in the plan
  all tfplan.resources as r {

    // Must not be an instance
    r.type != "aws_instance" or

    // Or if they are, must have a billing-id tag
    r.tags contains "billing-id"

  }
}
```

No resources provisioned outside of business hours

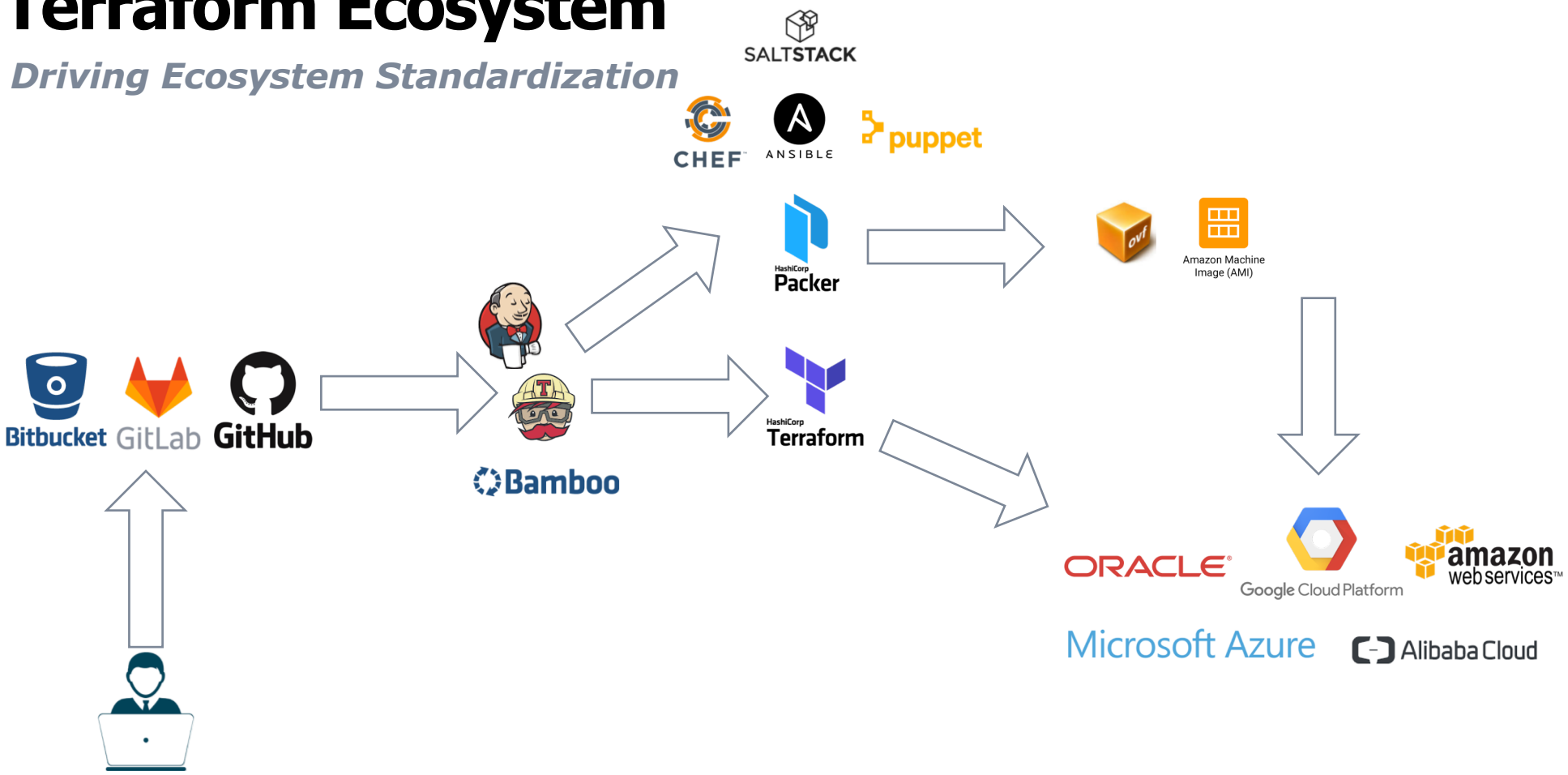
```
import "env"
import "time"

// Validate time is between 8 AM and 4 PM PST
valid_time = rule { time.hour > 8 and time.hour < 16 }

// Validate day is M - Th
valid_day = rule { time.day in ["monday", "tuesday",
"wednesday", "thursday"] }

main = rule {
  (valid_time and valid_day) or
}
```

main.tf

```
import "tfplan"

clusters = tfplan.resources.azurerm_container_service

agent_node_count_limit = rule {
  all clusters as name, instances {
    all instances as index, r {
      int(r.applied.agent_pool_profile[0].count) < 10
    }
  }
}

master_node_count_limit = rule {
  all clusters as name, instances {
    all instances as index, r {
      int(r.applied.master_profile[0].count) <= 3
    }
  }
}

vm_size_allowed = rule {
  all clusters as name, instances {
    all instances as index, r {
            r.applied.agent_pool_profile[0].vm_size matches "Standard_A1"
    }
  }
}
main = rule {
  (master_node_count_limit and agent_node_count_limit and vm_size_allowed)
else true
}
```

Terraform Ecosystem

Driving Ecosystem Standardization

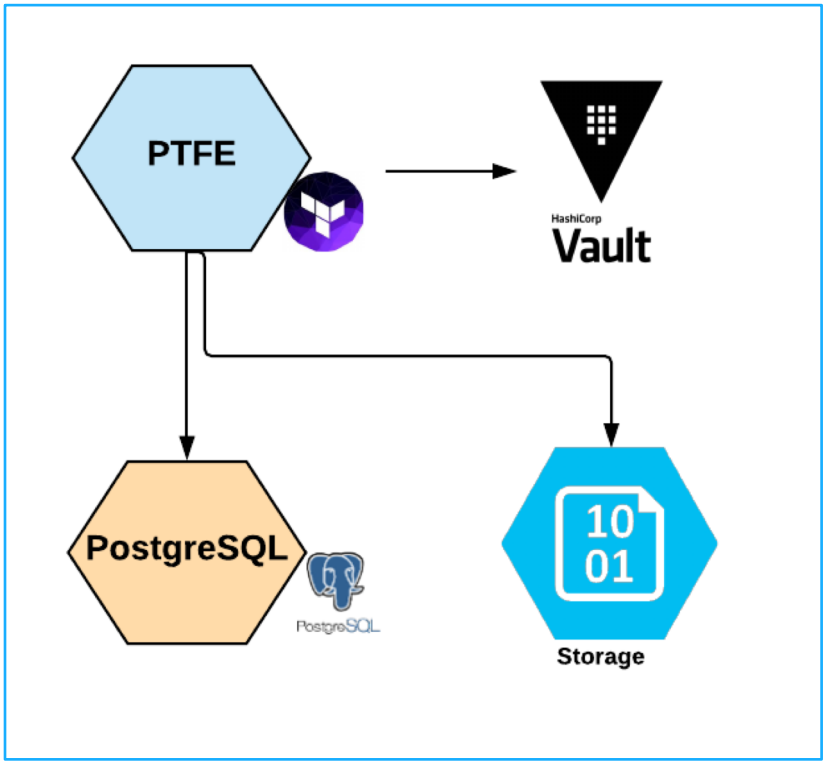


TFE API Example Demo

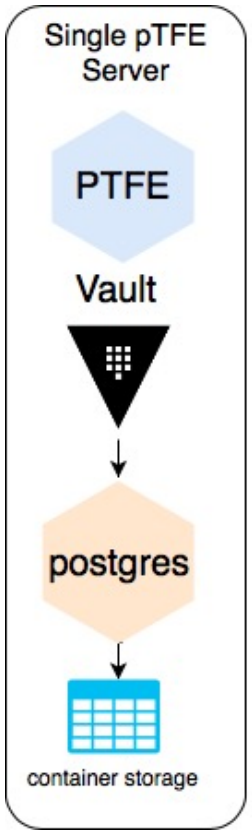
Private Terraform Enterprise Install (PTFE) questions

Platform and Providers	
Region	Single Region or Datacenter / Multiple
Environments	Production, Staging etc.
Resiliency goals	RPO, RTO etc.
Storage type	Mounted disk / External Services (PostgreSQL, Object store)
VCS integration	

Terraform Enterprise Components

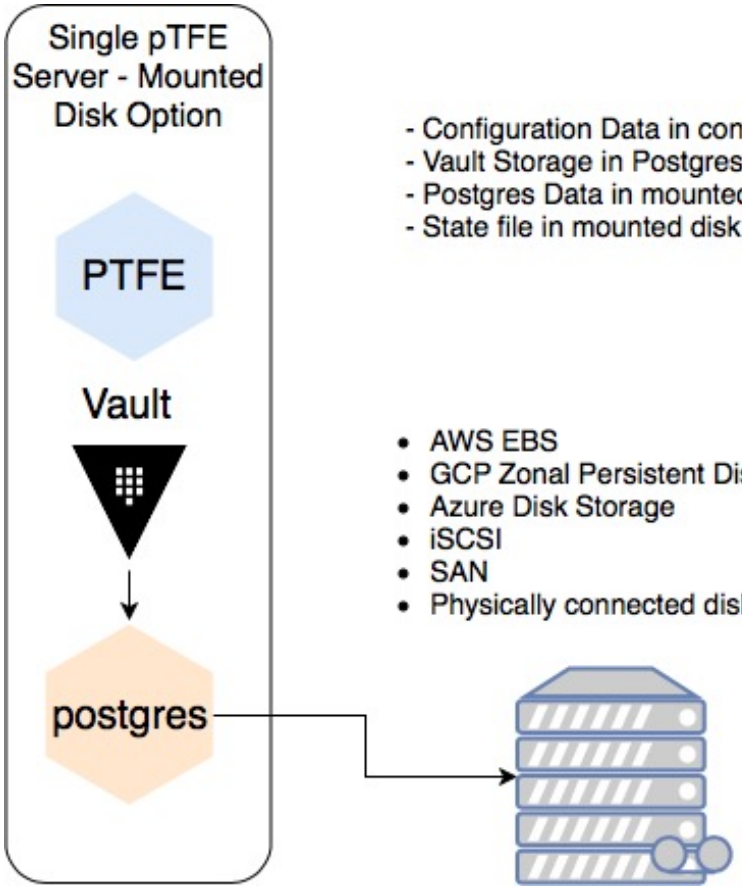


Terraform Enterprise Deployment (Demo Mode)



- Configuration Data in container
- Vault Storage in Postgres
- Postgres Data in container storage
- State file in container

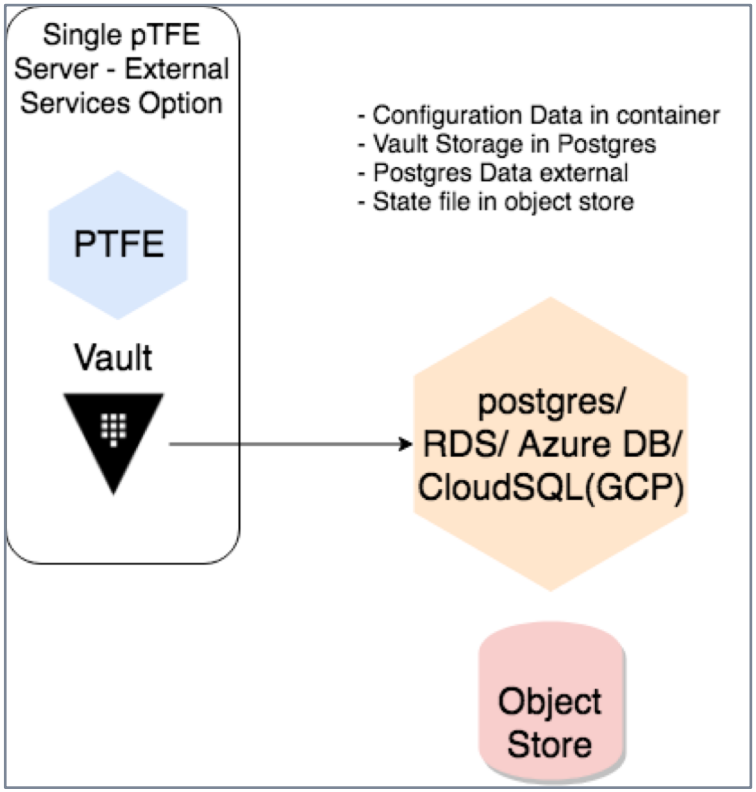
Terraform Enterprise Deployment (Mounted Disk)



- Configuration Data in container
- Vault Storage in Postgres
- Postgres Data in mounted disk
- State file in mounted disk

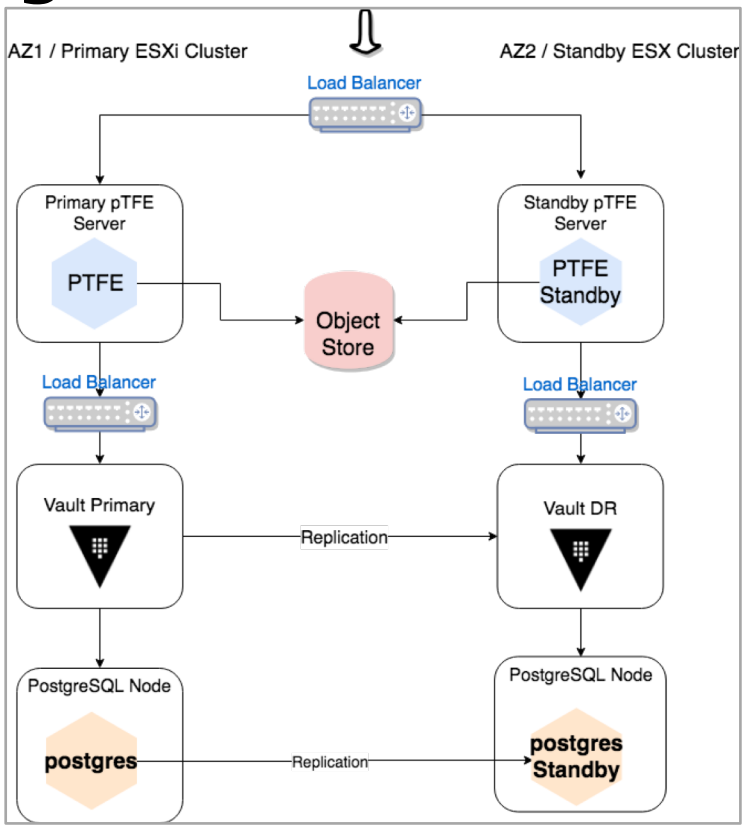
- AWS EBS
- GCP Zonal Persistent Disk
- Azure Disk Storage
- iSCSI
- SAN
- Physically connected disks as in non-cloud hardware

Terraform Enterprise Deployment (External with Local Vault)

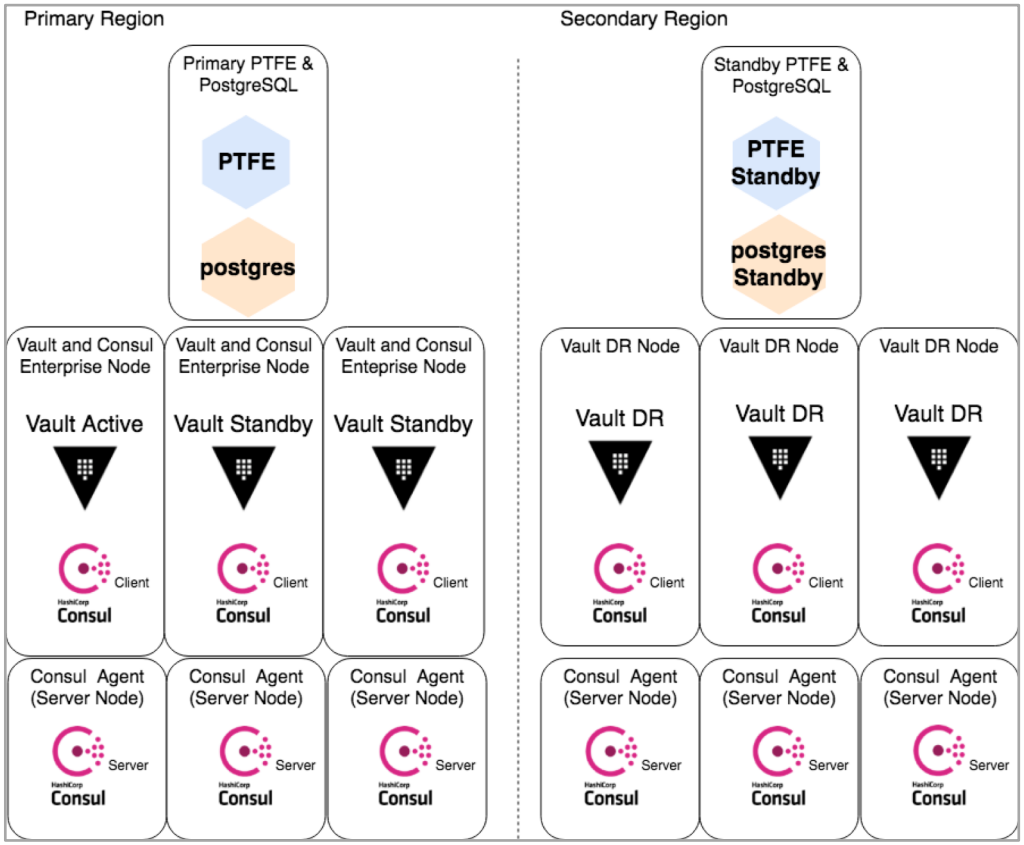


Terraform Enterprise External Services with External Vault – Single Region

Primary Region

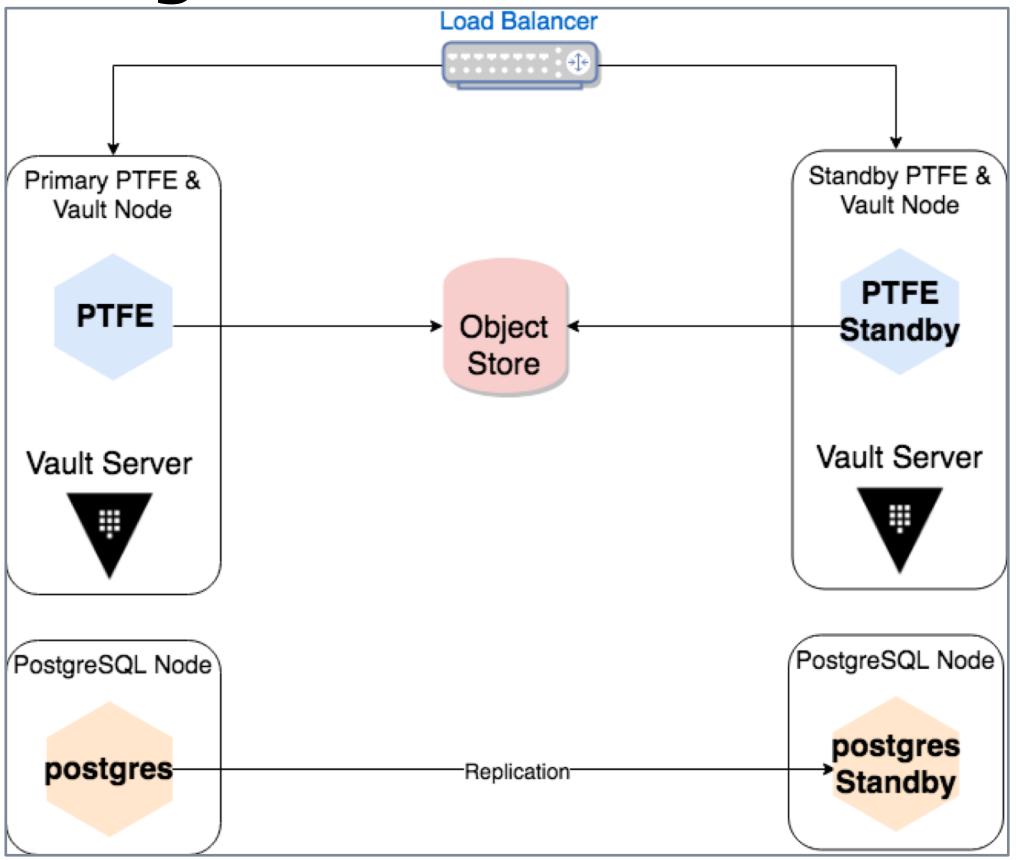


Terraform Enterprise Production - Multi-Region

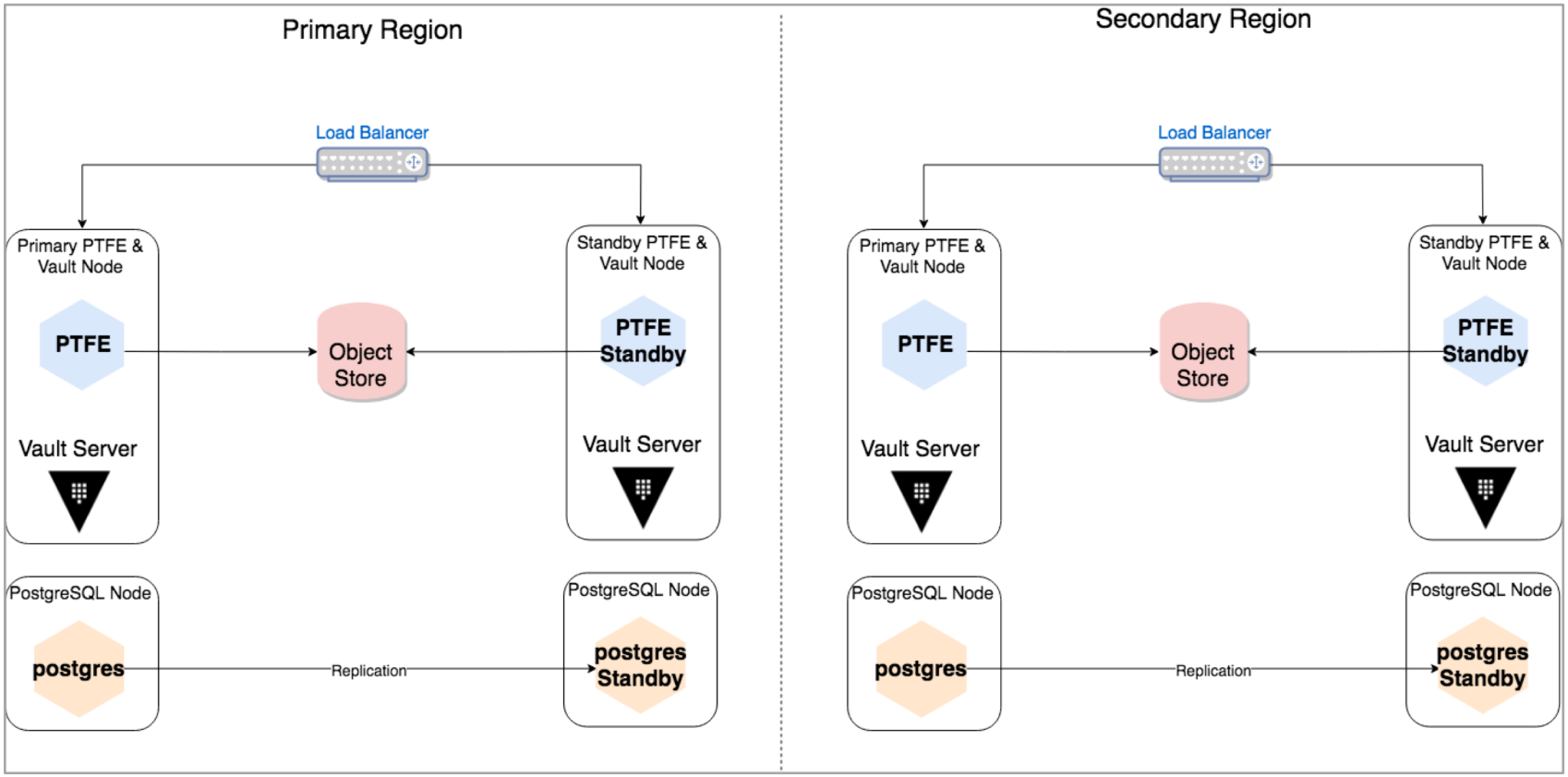


Terraform Enterprise External Services with Internal Vault – Single Region

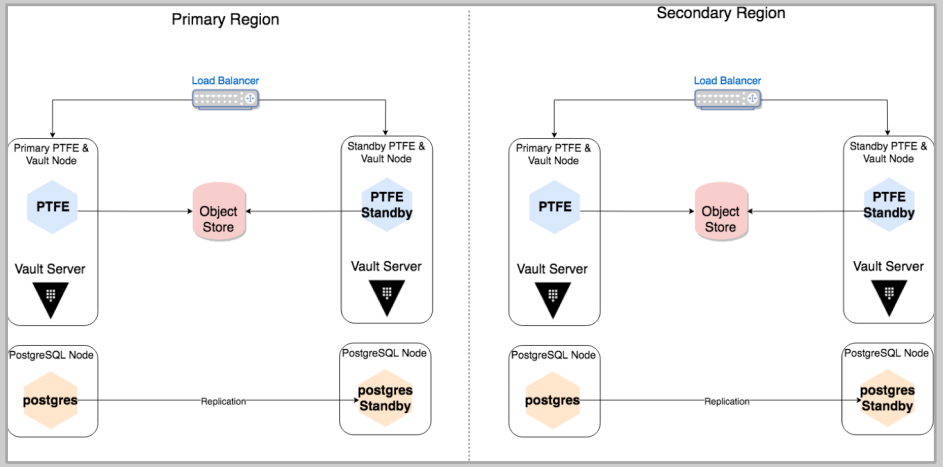
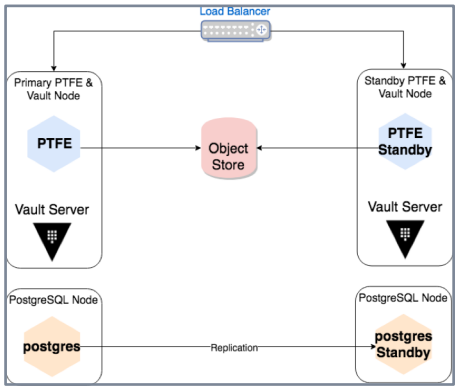
Primary Region



Terraform Enterprise Production – Multi-Region



Terraform Enterprise Environments

Production and Staging		x2
Dev and QA		x2

Terraform Enterprise Storage and Security

Private Terraform Enterprise - Security

Private Terraform Enterprise (PTFE) takes the security of the data it manages seriously. This table lists which parts of the PTFE app can contain sensitive data, what storage is used, and what encryption is used.

Object	Storage	Encrypted
Ingressed VCS Data	Blob Storage	Vault Transit Encryption
Terraform Plan Result	Blob Storage	Vault Transit Encryption
Terraform State	Blob Storage	Vault Transit Encryption
Terraform Logs	Blob Storage	Vault Transit Encryption
Terraform/Environment Variables	PostgreSQL	Vault Transit Encryption
Organization/Workspace/Team Settings	PostgreSQL	No
Account Password	PostgreSQL	bcrypt
2FA Recovery Codes	PostgreSQL	Vault Transit Encryption
SSH Keys	PostgreSQL	Vault Transit Encryption
User/Team/Organization Tokens	PostgreSQL	HMAC SHA512
OAuth Client ID + Secret	PostgreSQL	Vault Transit Encryption
OAuth User Tokens	PostgreSQL	Vault Transit Encryption
Twilio Account Configuration	PostgreSQL	Vault Transit Encryption
SMTP Configuration	PostgreSQL	Vault Transit Encryption
SAML Configuration	PostgreSQL	Vault Transit Encryption
Vault Unseal Key	PostgreSQL	*encryption password*

Note on PTFE recent changes

- Note: **Vault Unseal Key** will now be stored in PostgresSQL and Encrypted with an encryption password setting supplied during install.

- Reference (to be updated soon with above): https://www.terraform.io/docs/enterprise/private/data-security.html

Terraform Enterprise Managed PostgreSQL Requirements

PostgreSQL Requirements

Sizing Requirements

Type	CPU	Memory	Storage
Minimum	2 core	8 GB RAM	50GB
Recommended	4-8 core	16-32 GB RAM	50GB

Port Requirements

- Default port is 5432
- Can be changed using the PGPORT env variable.

PostgreSQL Requirements

- See https://www.terraform.io/docs/enterprise/private/install-installer.html#network-requirements
- PostgreSQL version 9.4 or greater. Go to https://www.postgresql.org/download/ to install.
- Any user with schema create access will do. Default user is “ptfe”

Terraform Enterprise PTFE Instance requirements

Private TFE server (VM)

Example Sizing Requirements (VMWare)

Type	CPU / Total Cores	Memory	Storage
Minimum	2 / 2	8 GB RAM	40GB
Recommended	2 / 4	16-32 GB RAM	50GB

Network Requirements

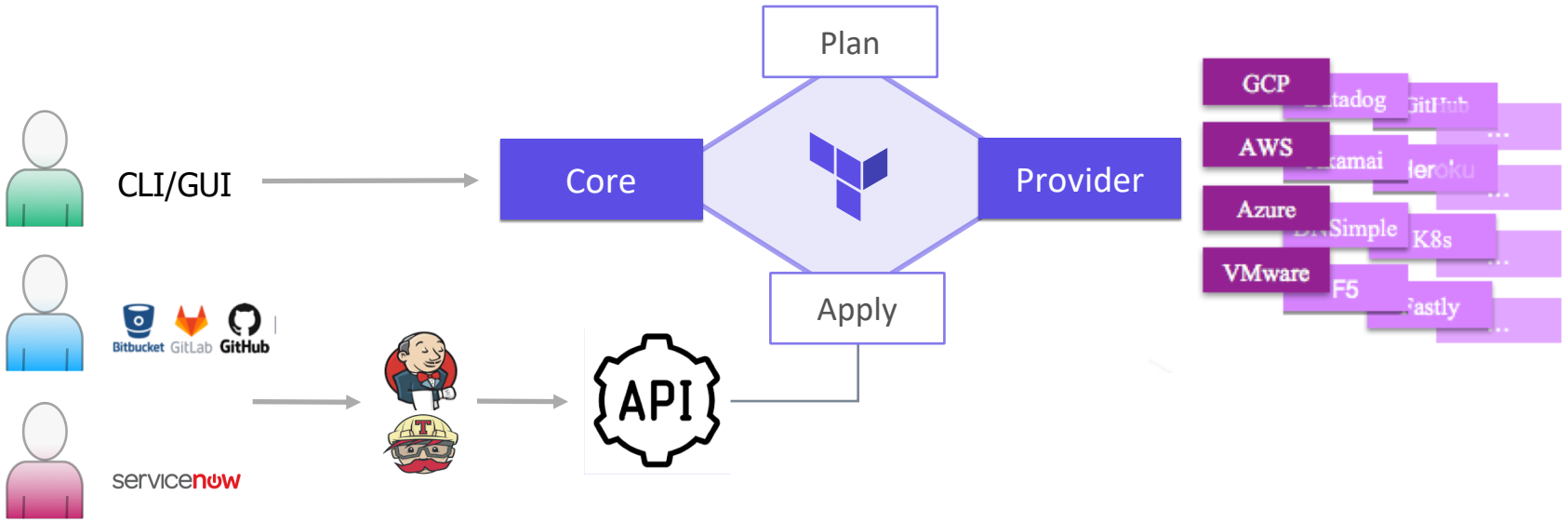
- 22, 8800, 80 / 443 and 9870-9880
- TLS Private key and certificate
- Proxy configuration
- Trusting SSL/TLS certificates

OS Requirements

- Linux OS version and Kernel
- Docker version
- Disable SELinux
- Trusting SSL/TLS certificates

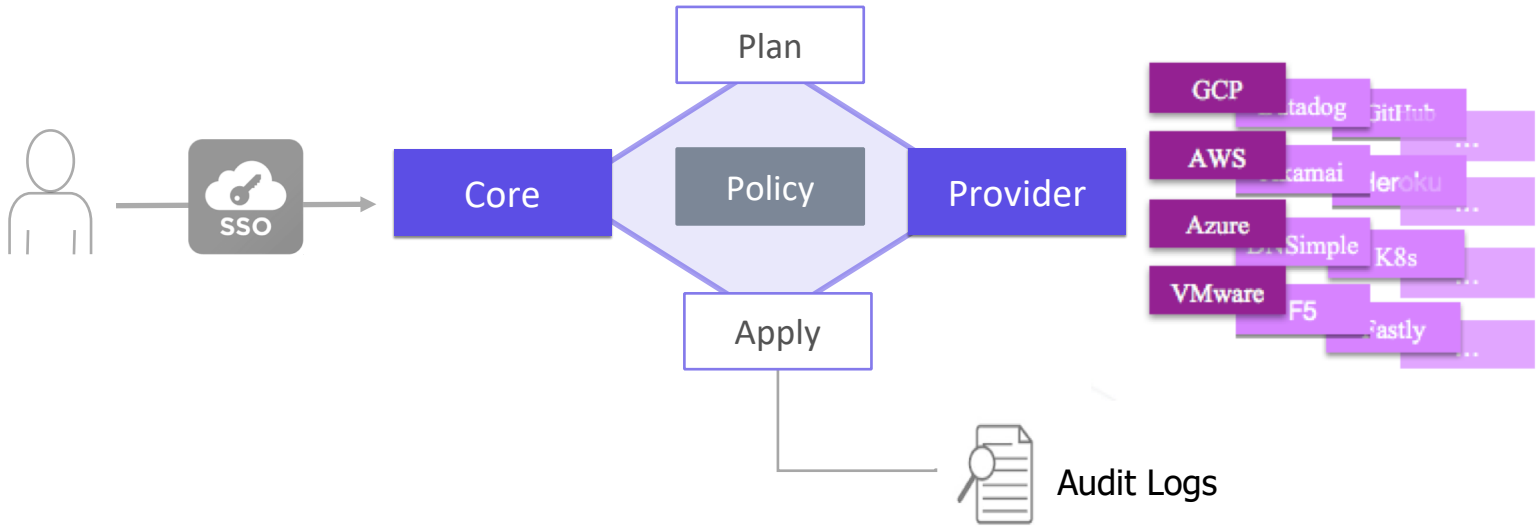
Terraform Enterprise: Operational Efficiency Benefits

- Organizations – Establish Multitenancy
- Workspaces – Centrally organize and manage access to infrastructure by teams/groups
- Remote State Management – Store, version, and collaborate on state
- VCS Connection – Execute TF runs as you commit changes
- API – Integrate Terraform with existing toolchain or workflows



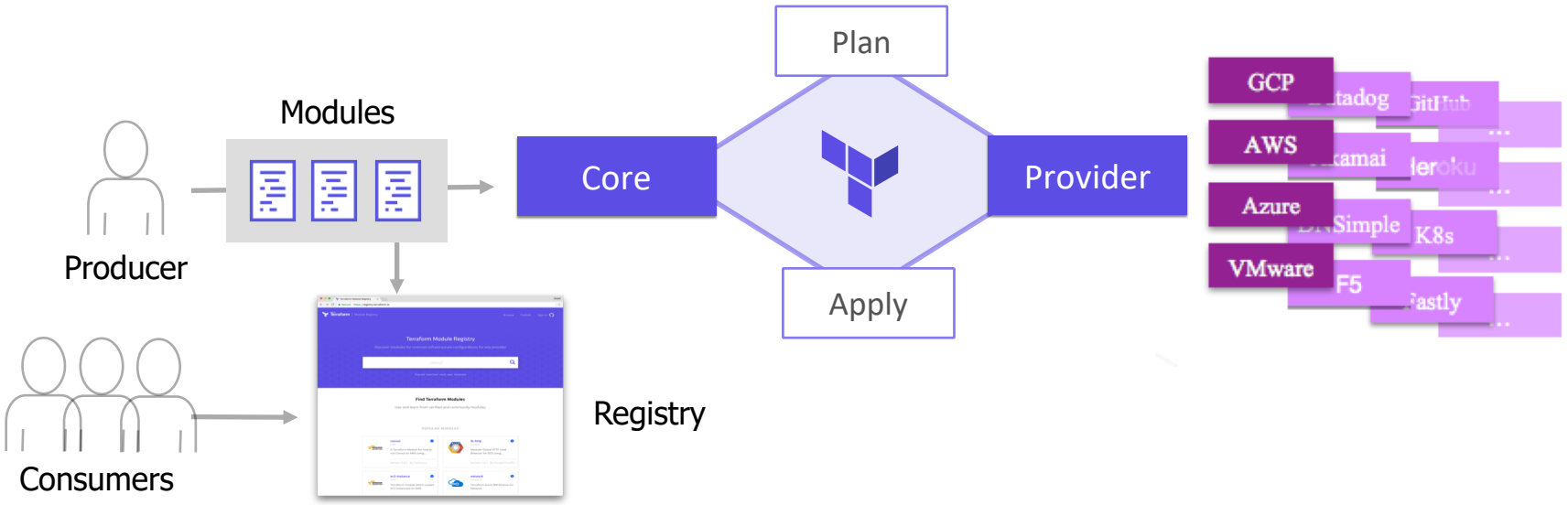
Terraform Enterprise: Governance and Security Benefits

- Policy as code – Sentinel enforces virtually any policy within the path to provision
- Audit Logs – Single pane of glass to track infrastructure changes
- Secure variable management – add, edit, delete all variables from view
- SSO with SAML – Centralize user management



Terraform Enterprise: Self Service Benefits

- Private Module Registry
- Producers: Publish ”approved” modules for consumption by wider audience
- Consumers: Leverage best practice templates developed by SME’s



Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com