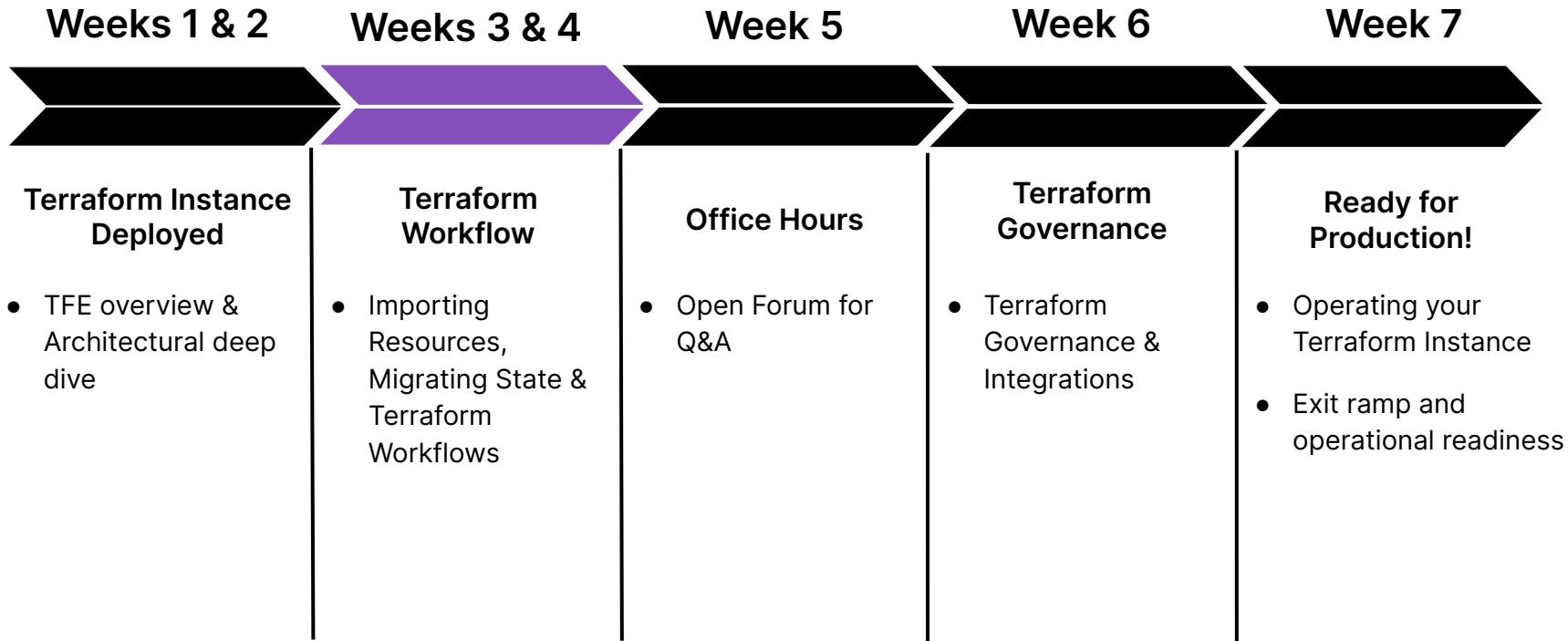


Terraform Workflows



TFE Path to Production



Agenda

- Run Workflows 01

- Terraform Modules 02

- Private Module Registry 03

- Workspaces 04

- Variables 05

- Git Repo Structure 06



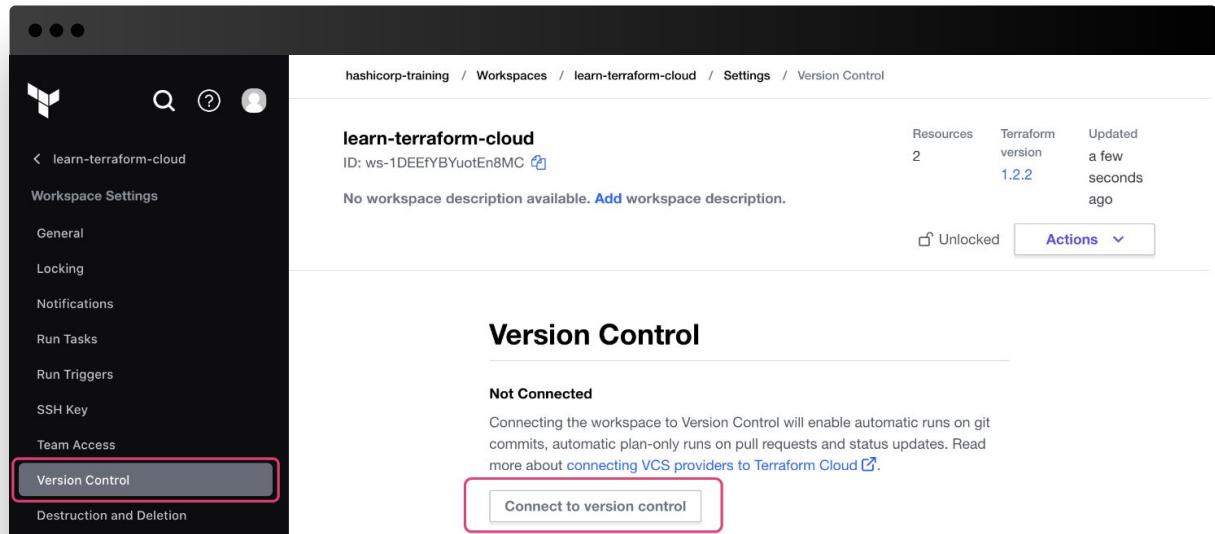
01

Run Workflows



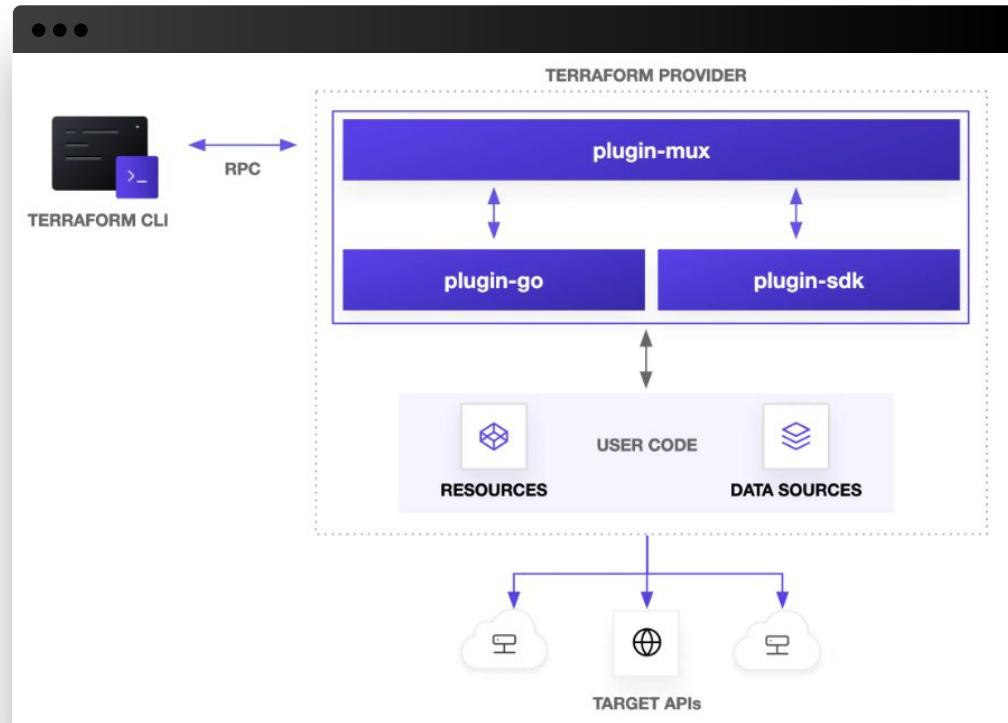
Terraform Run Workflows

- [UI-Driven Runs](#) - manually trigger runs from the TFC web UI
- [VCS-Driven Runs](#) - easiest integration, directly connects a Git Repo to a Terraform Workspace, with automatic runs on Git Commit and Pull Request code changes
- [CLI-Driven Runs](#) - easy to use, single CLI command to trigger runs, takes files in the local folder, creates a .zip file, and sends the contents to the TFC API



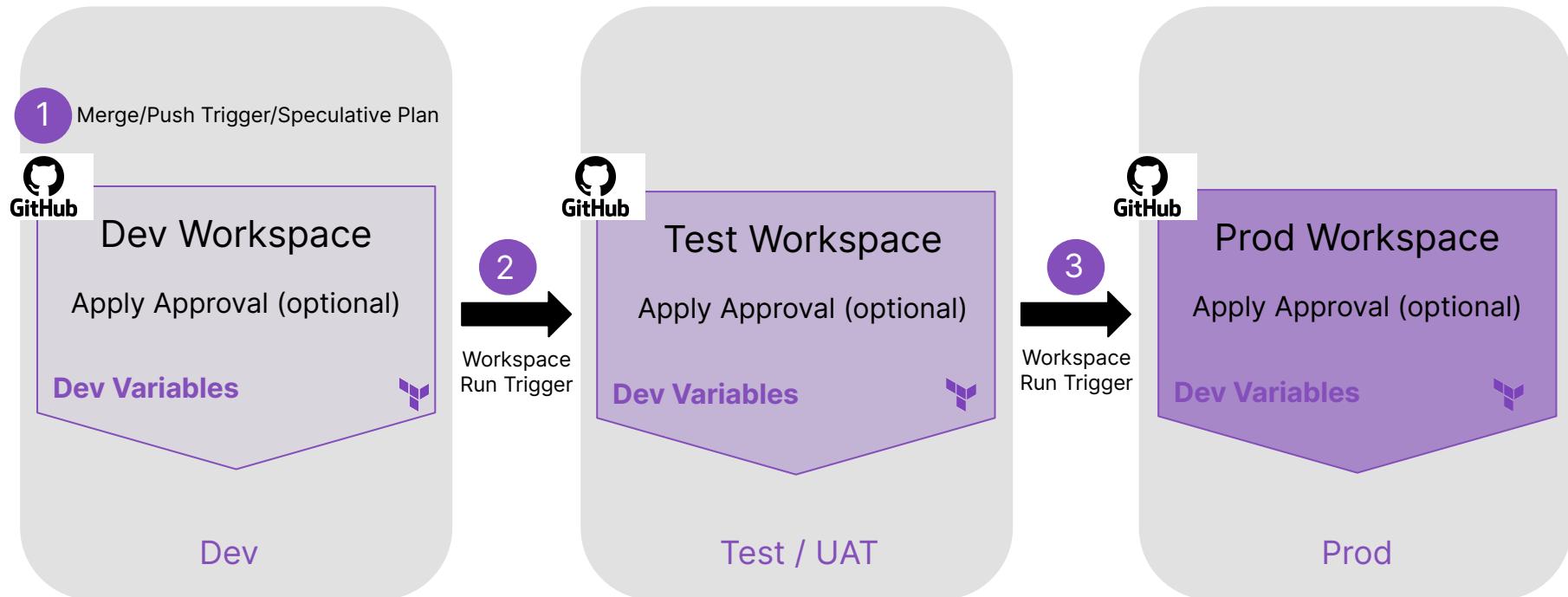
Terraform Run Workflows

- SDK-Driven Runs - calls to the TFC API, using a Language Specific integration, available for Golang, Python, and .NET
- API-Driven Runs - full control, all features available to the web UI have an API call, but requires custom coding JSON REST HTTP API calls



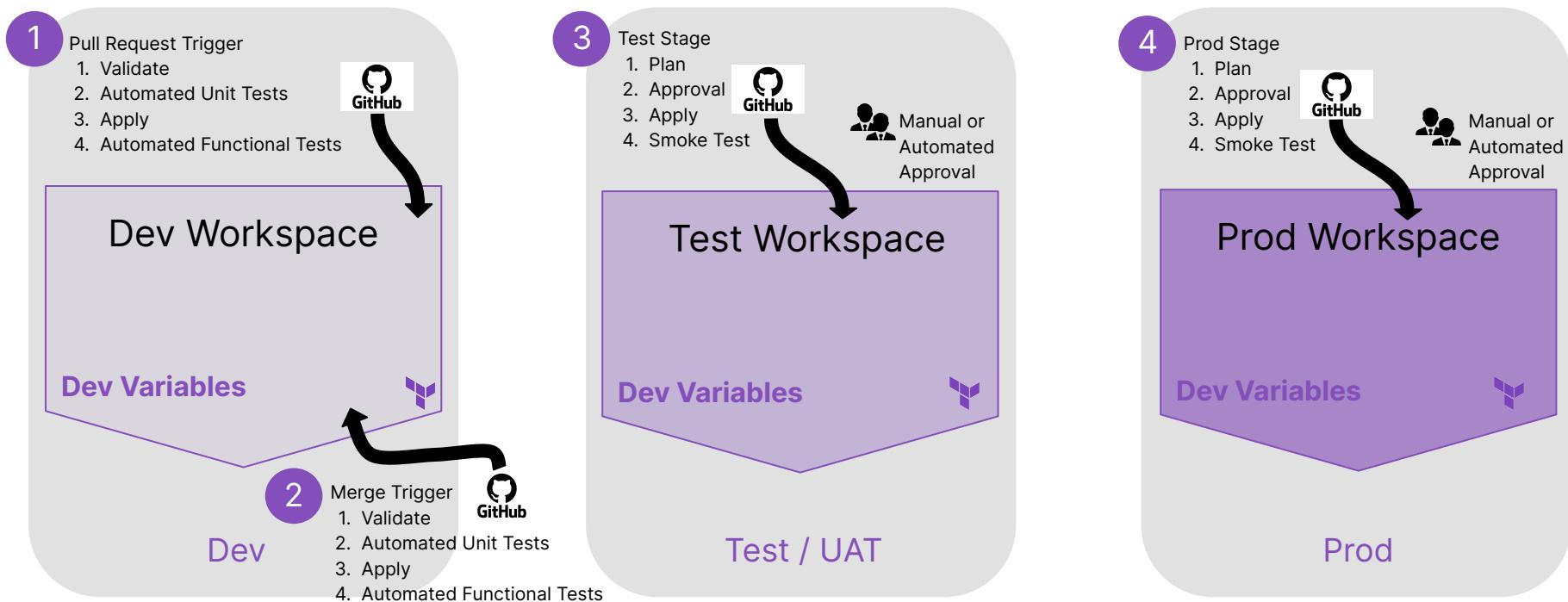
Workflow Types

VCS Trigger

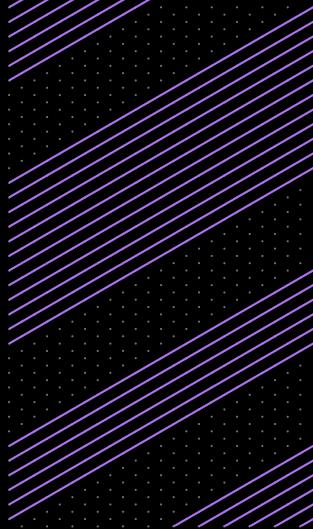


Workflow Types

API in CD Tool

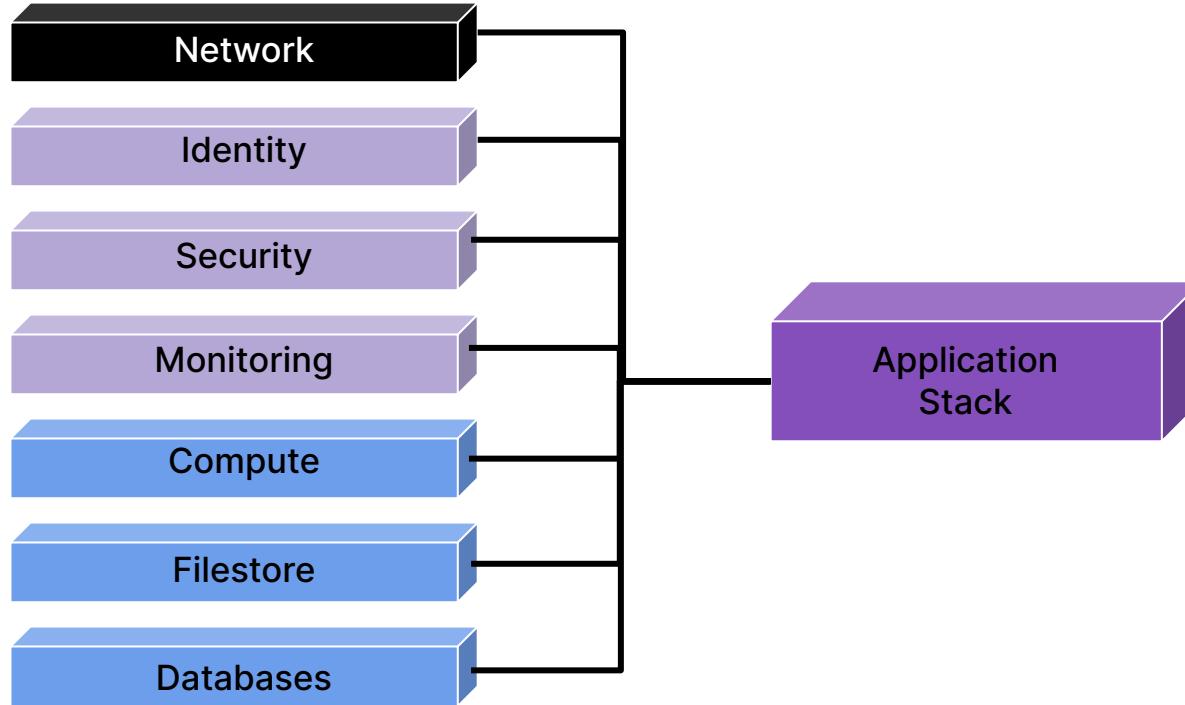


02



Terraform Modules

Architecture



Code Layout

Static Variables and Dynamically Generate Outputs can be passed between Modules

```
# ./main.tf
variable "vpc_cidr" {
  default    = "10.0.0.0/16"
}

module "network" {
  source          = "./network"
  vpc_cidr        = var.vpc_cidr
  public_subnet_cidr = var.public_subnet_cidr
  region          = var.region
  availability_zones = var.availability_zones
}

module "security-groups" {
  source = "./security-groups"
  vpc_id      = module.network.vpc_id
  vpc_cidr    = var.vpc_cidr
  public_subnet_ids = module.network.pub_sub_ids
}
```

```
# ./network/vpc.tf
resource "aws_vpc" "default" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = true
}

output "vpc_id" {
  value = "${aws_vpc.default.id}"
}

resource "aws_subnet" "subnet_public" {
  vpc_id           = aws_vpc.default.id
  cidr_block       = var.public_subnet_cidr
  availability_zone = var.availability_zones
}

output "pub_sub_ids" {
  value = [ "${aws_subnet.subnet_public.*.id}" ]
}
```



Network



Route 53 DNS, TLS/SSL Certs, Regions, Availability Zones, VPC, Internet Gateway, Public Subnet, Private Subnet, Route Table, Network ACL



VNet, Network Gateway, NAT Gateway, Route Table, Express Route (on-prem), Public IP, Application Gateway



Google Cloud Platform

VPC, Subnet, Cloud NAT, Compute Route, Cloud Interconnect (on-prem), Public IP, API Gateway



Infoblox DNS / BIND, Verisign / Microsoft AD / Cloud Foundry CA TLS/SSL certs, Regions, Availability Zones, VLAN, Palo Alto / Checkpoint Firewall, DMZ, Internal VLANs, Cisco / Juniper / HP / Dell Route Table, Network ACL, WAN Link / Dark fiber, VMware ESXi / Tanzu NSX Firewall Rules, VMware vLAN



Security



AWS Config (resource), AWS GuardDuty (NIDS), AWS Macie (S3), VPC Flow Logs



Azure PolicySets, Network Security Groups, Azure AD Policies



Google Cloud Platform

GCP Security Command Center



Palo Alto Prisma (resource), Splunk (NIDS), SFlow / NetFlow / Cisco Network Flow Logs, Qualys / Tenable Nessus / Rapid7 Nexpose / Checkpoint (VM, container), Tripwire / OSSEC (FIM)

Identity



IAM Group, IAM Role, IAM User,
IAM Policy (customer-managed)



Azure AD (Active Directory),
Azure Resource Group



Google Cloud Platform

Service Account, Folder, Roles, Policy



Microsoft Active Directory, LDAP, SAML, Okta

Monitoring



AWS CloudTrail (cli/sdk), CloudWatch,
CloudWatch Metrics



Azure Network Watcher Flow Log,
Monitor



Google Cloud Platform

Network Telemetry, VPC Flow Logs,
Cloud Audit Logs



DataDog / SignalFX / Nagios / SolarWinds,
Splunk / ELK / SumoLogic, HP OpenView

Compute



Load Balancer (ALB, ELB, NLB),
Auto-scaling Group + Launch Config +
Resource Group + EC2, EKS (K8S), ECS,
FarGate (hosted ECS), AWS Lambda



Traffic Manager (global LB), Scale Set +
Launch Config + Resource Group + VM,
Azure K8S / AKS



Google Cloud Platform

Load Balancer, Managed Instance Group
(MIG) + Instance Template + Stateful
Configuration + Compute, GCP EKS / K8S



F5 / HAProxy / nginx Load Balancers, VMware
vRealize, VMware Pivotal Cloud Foundry (PKS,
PCS) / K8S

Filestore



S3, CloudFront (CDN)



Google Cloud Platform

Cloud Storage, Cloud CDN



Blob Storage, Content Delivery Network



SAN, NAS, GlusterFS, Minio / Ceph / Dell EMC
ECS S3-compatible, Akamai

SQL Databases



RDS (MySQL, Aurora, Postgresql, MSSQL, Oracle)



Google Cloud Platform

Cloud SQL (PostgreSQL, MySQL, SQL Server)



MSSQL, Oracle, MySQL, Postgres



MS SQL Server, Oracle DB, Sybase DB, DB2, MySQL, Postgresql

NoSQL Databases



ElasticSearch, MongoDB, DocumentDB,
Hadoop, DynamoDB



Google Cloud Platform

BigQuery, ElasticSearch, MongoDB Atlas,
BigTable



ElasticSearch, MongoDB, Azure
HDInsight Hadoop



ElasticSearch, MongoDB, Hadoop

In-Memory Databases



ElastiCache (Memcached, Redis)



Google Cloud Platform

GCP Memorystore (Redis, Memcached)

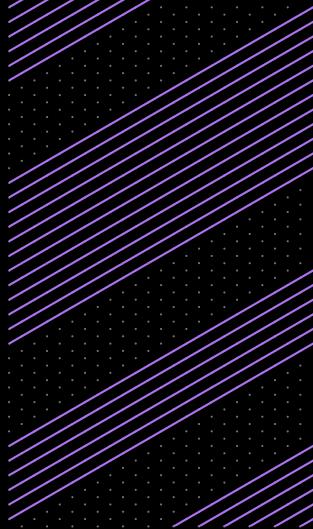


Azure Cache for Redis



Memcached, Redis

03



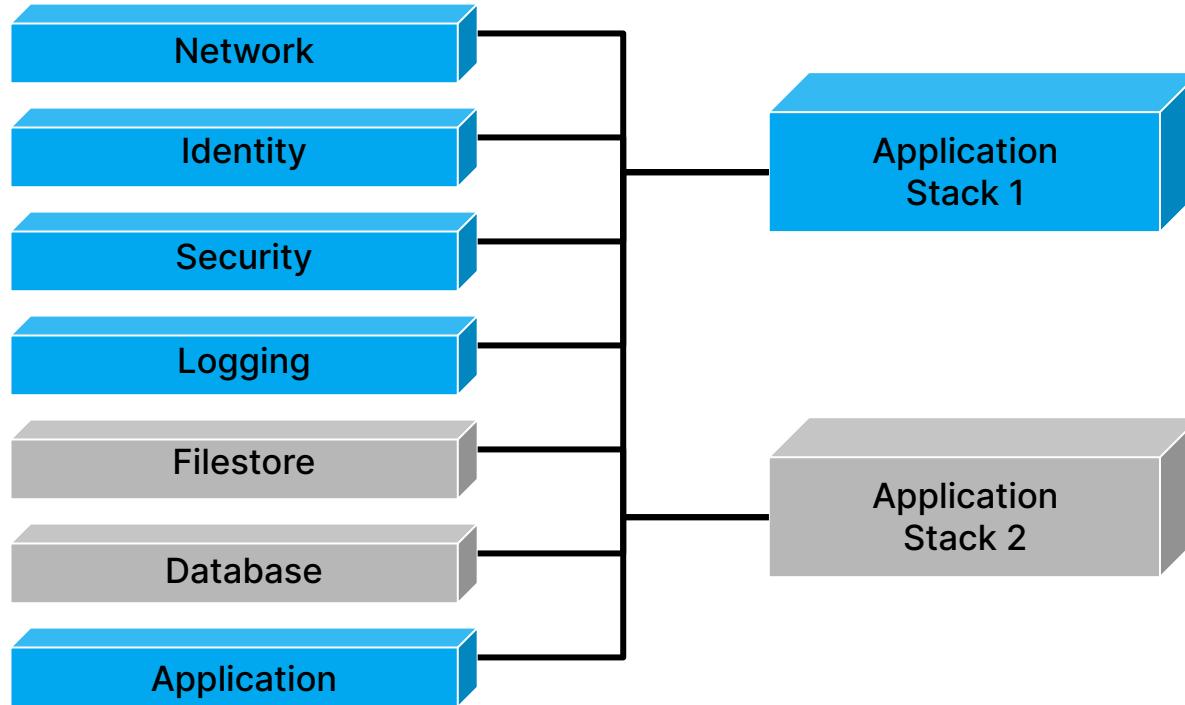
Private Module Registry

Private Module Registry (PMR)

- Hosts private providers and private modules which are only available to members of an organization unless explicitly shared to another organization
- Includes support for module versioning, a filterable list, and a configuration designer for rapid workspace build
- Uses configured VCS integrations and defers to the VCS provider for most management tasks like version releases
- Both public and private modules can be added to the registry and are only available to members of the organization
- Public modules are automatically synchronized from the Terraform Registry where they are hosted

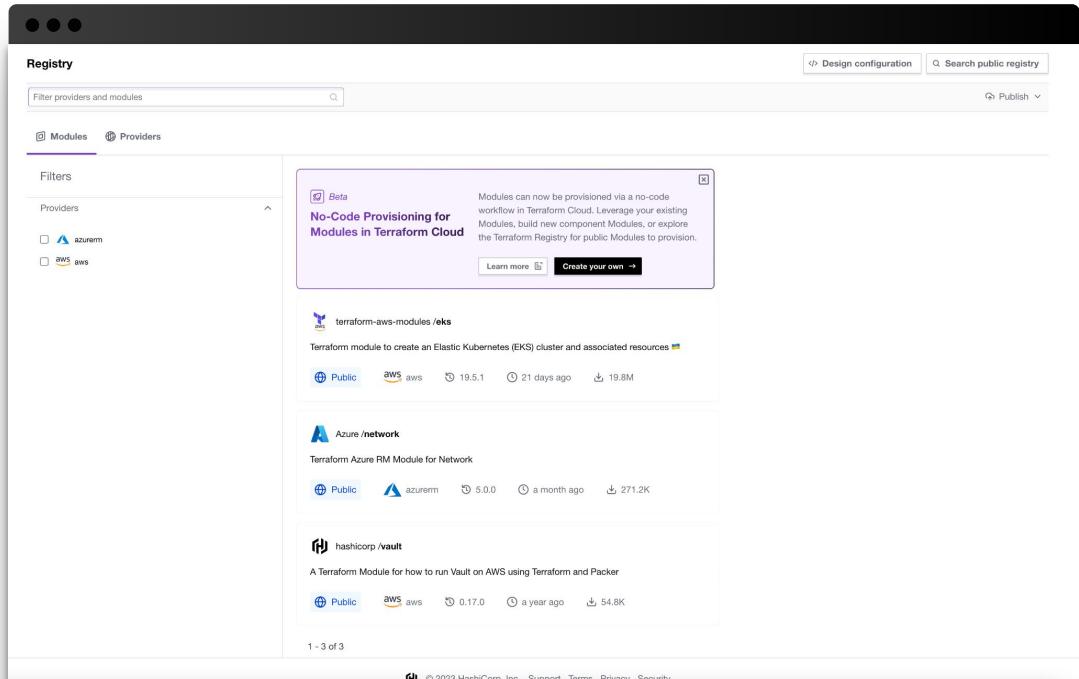


Module Sharing



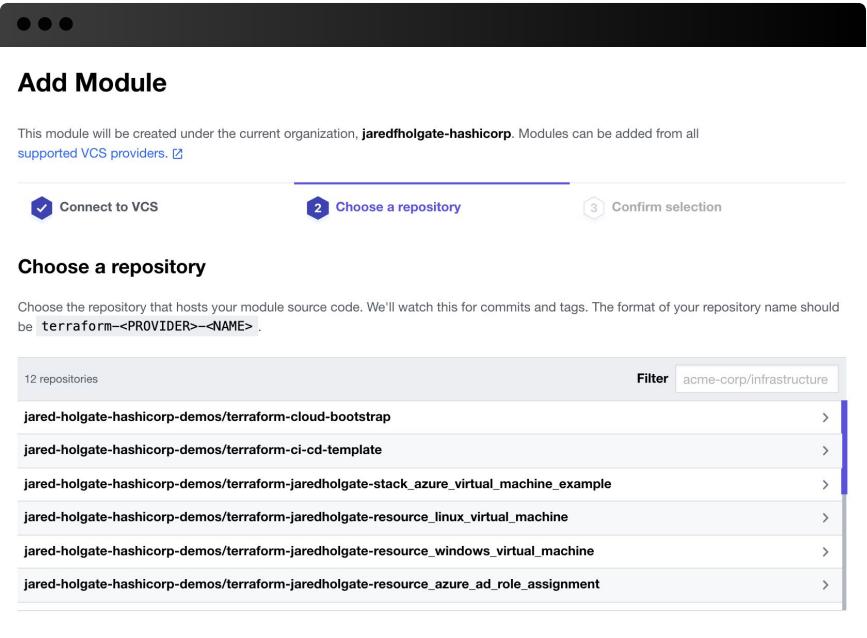
Private Module Registry

- VCS integration
- Versioning based on VCS tags
- Restrict which Modules and Module Versions are used, with Sentinel
- Must follow a specific convention



Private Module Registry

- Repo name must follow the convention: terraform-<provider>-<module name>
- e.g. terraform-myorganization-azure_network
- Must have a README.md
- Must have a main.tf file
- Must have a version tag in x.y.z format



Private Module Registry

- Specify which providers and modules are recommended
- Restrict using Sentinel

The screenshot displays two side-by-side views of the HashiCorp Terraform Registry interface.

Left View (Public Registry): Shows a list of public providers and modules. The providers listed are aws, azuread, azurerm, cloudflare, github, and google. Each provider entry includes its icon, name, maintainer, version (e.g., 3.69.0), last updated (e.g., 4 days ago), and popularity metrics (e.g., 2.0M).

Right View (Private Registry): Shows a search result for "azure". It lists one module: "Azure / network" by Terraform Azure RM Module for Network. The entry shows it's a Public module, created by azurerm, version 3.5.0, last updated 6 months ago, and has 130.0K popularity.

Filters: Both views include a sidebar with "Providers" and "Modules" tabs, and a "Filters" section. The "Providers" filter sidebar on the right shows "azurerm" is selected, while "jaredhoggate" is unselected. The "Registries" filter sidebar shows "Public" is selected, while "Private" is unselected.



Configuration Designer

- Helps to write HCL
- Still need to source control and have a workspace

The screenshot shows the Configuration Designer interface in three main stages:

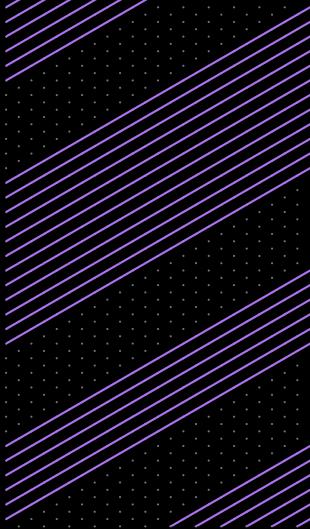
- Select Modules:** Shows two modules: `resource_windows_virtual_machine` (Configured, provider: jaredholgate, version: 2.0.2) and `resource_linux_virtual_machine` (Configured, provider: jaredholgate, version: 2.0.3). Both have a "Deferred" status note.
- Configure Variables:** For the `resource_linux_virtual_machine` module. Fields include:
 - `location`: REQUIRED, value: uksouth
 - `name`: REQUIRED, value: testing456vm
 - `resource_group.name`: REQUIRED, value: testing456rg
 - `source.image_gallery_name`: REQUIRED, value: Deferred: variable must be set at runtime
- Publish to VCS:** Step 1: Create a new repository in your version control system. Step 2: Download the configuration and push the contents to the newly created repository. Step 3: Create a new workspace in Terraform Cloud and configure the VCS source to the newly created repository. The code editor displays the generated HCL configuration:

```
variable "resource_windows_virtual_machine_source_image_publisher" {}
variable "resource_windows_virtual_machine_source_image_sku" {}
variable "resource_windows_virtual_machine_source_image_version" {}
variable "resource_windows_virtual_machine_subnet_id" {}
variable "resource_windows_virtual_machine_tags" {}

//-----
// Modules
module "resource_linux_virtual_machine" {
  source  = "app.terraform.io/jaredholgate/hashicorp/resource_linux_virtual_machine/jaredholgate"
  version = "2.0.3"

  admin_username = "${var.resource_linux_virtual_machine_admin_username}"
  cloud_init_script = "${var.resource_linux_virtual_machine_cloud_init_script}"
  has_managed_identity = "${var.resource_linux_virtual_machine_has_managed_identity}"
}
```

04



Workspaces

Considerations

- **Blast-Radius:** Do not put everything in one place
- **Least Privilege:** Divide cloud resources into multiple Workspaces so that a Team cannot change another Team's cloud resources
- **Rate of Change:** Common changes should not affect uncommonly changing resources

Example: In many environments the Networking layer will not change as often as the Compute layer

- **Ease of Maintenance:** Group similar resources to ensure maintenance changes don't affect other components

Example: upgrading all instances of Postgres / MySQL / MS-SQL should not affect change to Networking resources



1. Monolithic Workspace

<u>Production</u>	<u>Staging</u>	<u>QA</u>	<u>Dev</u>
network	network	network	network
security	security	security	security
identity	identity	identity	identity
compute	compute	compute	compute
filestore	filestore	filestore	filestore
sql	sql	sql	sql



2. Production vs. Non-production

<u>Production</u>	<u>Staging</u>	<u>QA</u>	<u>Dev</u>
network	network	network	network
security	security	security	security
identity	identity	identity	identity
compute	compute	compute	compute
filestore	filestore	filestore	filestore
sql	sql	sql	sql



3. Prod vs. Non-prod w/ Landing Zones

<u>Production</u>	<u>Staging</u>	<u>QA</u>	<u>Dev</u>
network	network	network	network
security	security	security	security
identity	identity	identity	identity
compute	compute	compute	compute
filestore	filestore	filestore	filestore
sql	sql	sql	sql



4. Divided by Environments (Envs)

Production

network
security
identity
compute
filestore
sql

Staging

network
security
identity
compute
filestore
sql

QA

network
security
identity
compute
filestore
sql

Dev

network
security
identity
compute
filestore
sql



5. Isolated Envs w/ Landing Zones (LZs)

<u>Production</u>	<u>Staging</u>	<u>QA</u>	<u>Dev</u>
network security identity	network security identity	network security identity	network security identity
compute filestore sql	compute filestore sql	compute filestore sql	compute filestore sql



6. Isolated Envs w/ Shared App Layers

<u>Production</u>	<u>Staging</u>	<u>QA</u>	<u>Dev</u>
network	network	network	network
security	security	security	security
identity	identity	identity	identity
compute	compute	compute	compute
filestore	filestore	filestore	filestore
sql	sql	sql	sql



7. Isolated Envs w/ Isolated Layers

<u>Production</u>	<u>Staging</u>	<u>QA</u>	<u>Dev</u>
network	network	network	network
security	security	security	security
identity	identity	identity	identity
compute	compute	compute	compute
filestore	filestore	filestore	filestore
sql	sql	sql	sql



Terraform tfe Provider

Automate Terraform Cloud Configuration

- [Terraform Cloud/Enterprise Provider](#)
- tfe = terraform enterprise
- Works with Terraform Cloud and Terraform Enterprise
- Requires a Token argument (i.e. API token)
- Comprehensive resource and data source coverage





Terraform Provider

[Terraform Cloud/Enterprise Provider - tfe](#)

```
...
# Configure a TF Workspace Variable called
# "tf_token" with the TFC API Token
terraform {
  required_providers {
    tfe = {
      source = "hashicorp/tfe"
      version = "~> 0.25.3"
    }
    null = {
      source = "hashicorp/null"
      version = "~> 3.1.0"
    }
  }
  #
  https://registry.terraform.io/providers/hashicorp/tfe/latest/docs
  provider "tfe" {
    hostname = var.tf_hostname
    token    = var.tf_token
  }
}
```

Workspace Creation Automation

```
...
variable "tf_organization" {
  type = string
  default = "Pyrocumulus"
}

variable "tf_workspaces" {
  type = set(string)
  default = ["workspaceA", "workspaceB",
    "workspaceC"]
}

resource "tfe_workspace" "test" {
  for_each = var.tf_workspaces
  name = each.key
  organization = var.tf_organization
}

output "tf_workspace_ids" {
  value = { for k, v in tfe_workspace.test :
    k => v.id }
}
```

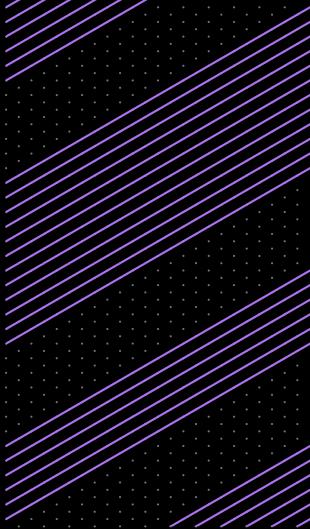
```
...
resource "tfe_variable" "test" {
  for_each = { for k, v in tfe_workspace.test:
    k => v.id }
  key = "test_key_name"
  value = "test_value_name"
  category = "terraform"
  workspace_id = each.value
}

resource "tfe_team" "test" {
  name = "test-team-name"
  organization = var.tf_organization
}

resource "tfe_team_access" "test" {
  for_each = { for k, v in tfe_workspace.test:
    k => v.id }
  access = "read"
  team_id = tfe_team.test.id
  workspace_id = each.value
}
```



05



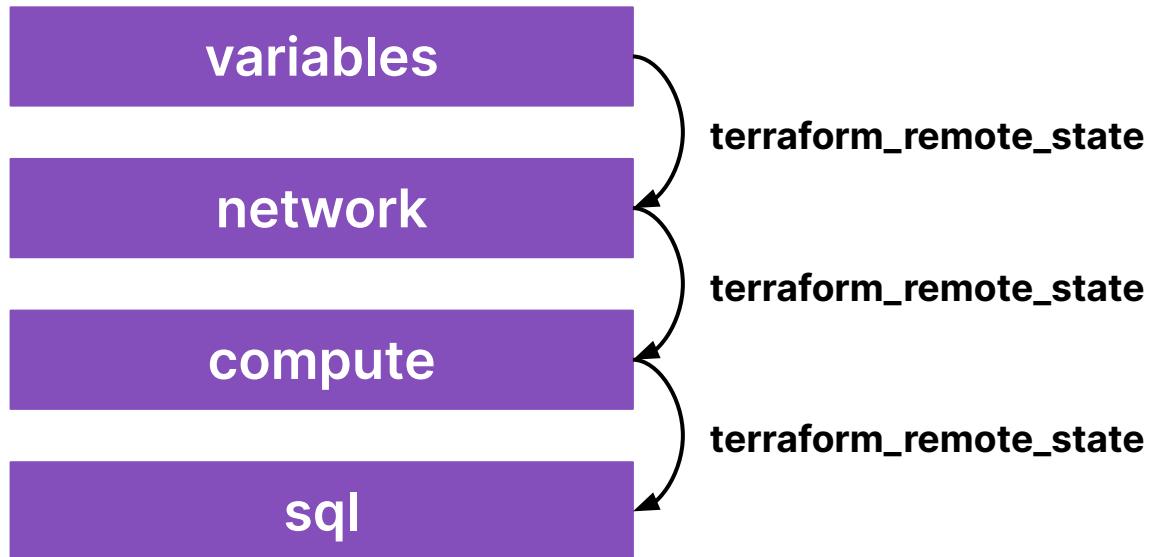
Workspace Variables

Workspaces, Secrets / Credentials

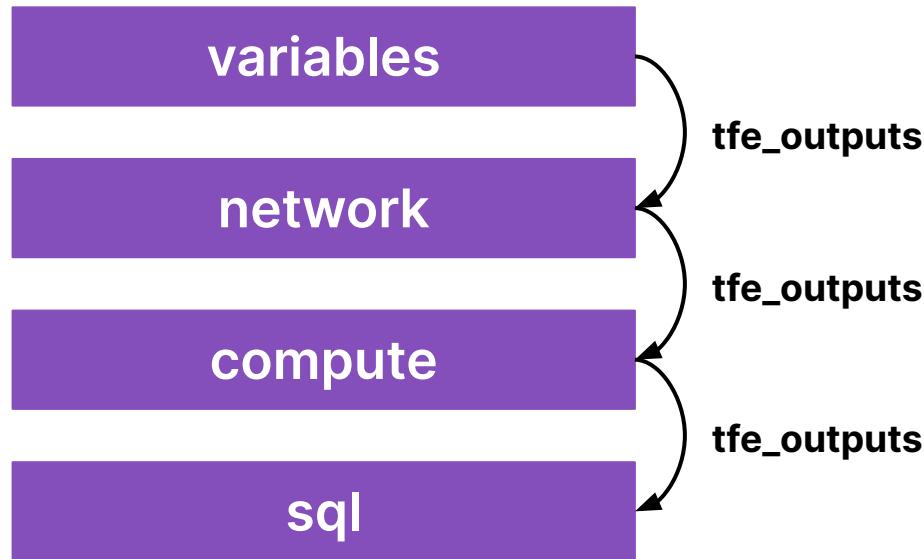
1. Vault Enterprise
2. Vault Open Source
3. Cloud Agents, with Cloud Identity Credentials (ex: AWS IAM Instance Profile)
4. Variable Sets
5. **tfe_outputs** data source, read between Workspaces
6. Workspace Variable, Sensitive
7. Workspace Environment Variable, Sensitive
8. CI/CD Inject Credentials at Run-time
9. Dynamic Credentials



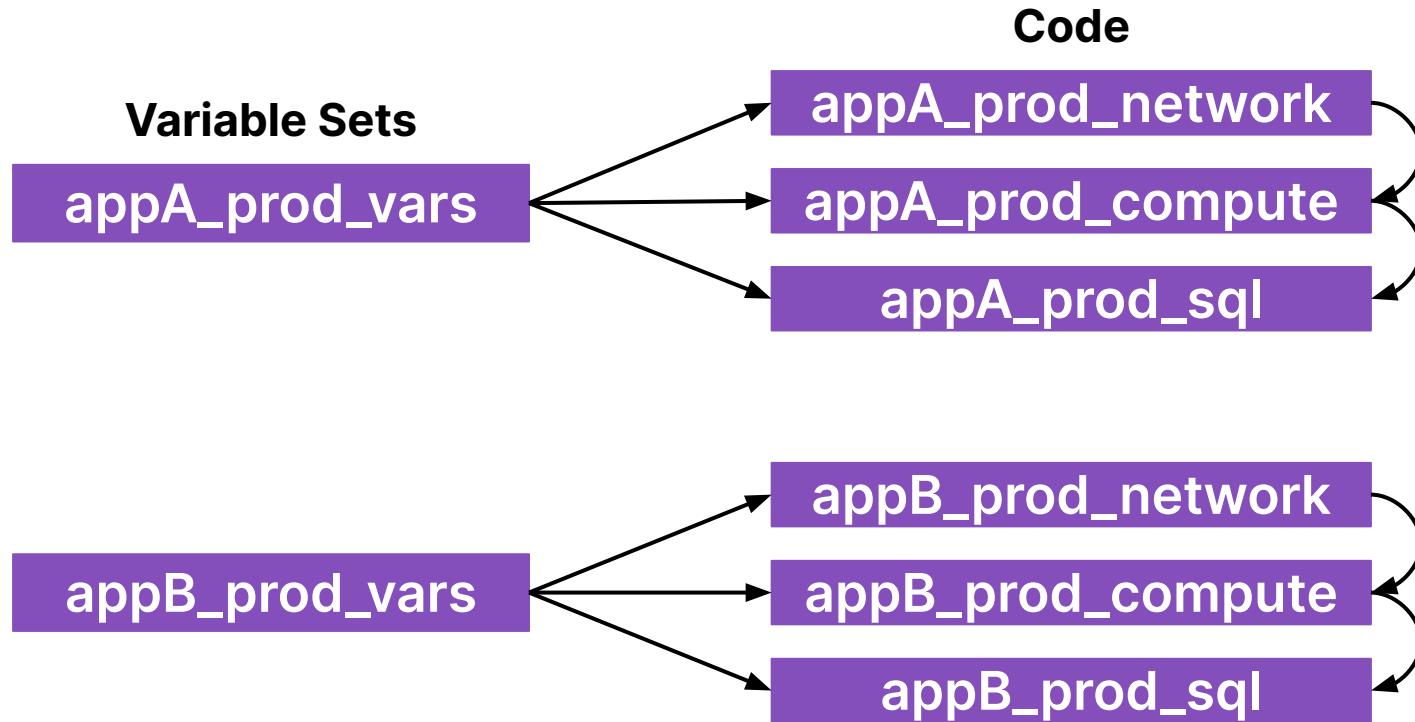
Changes Across Workspaces with Run Triggers



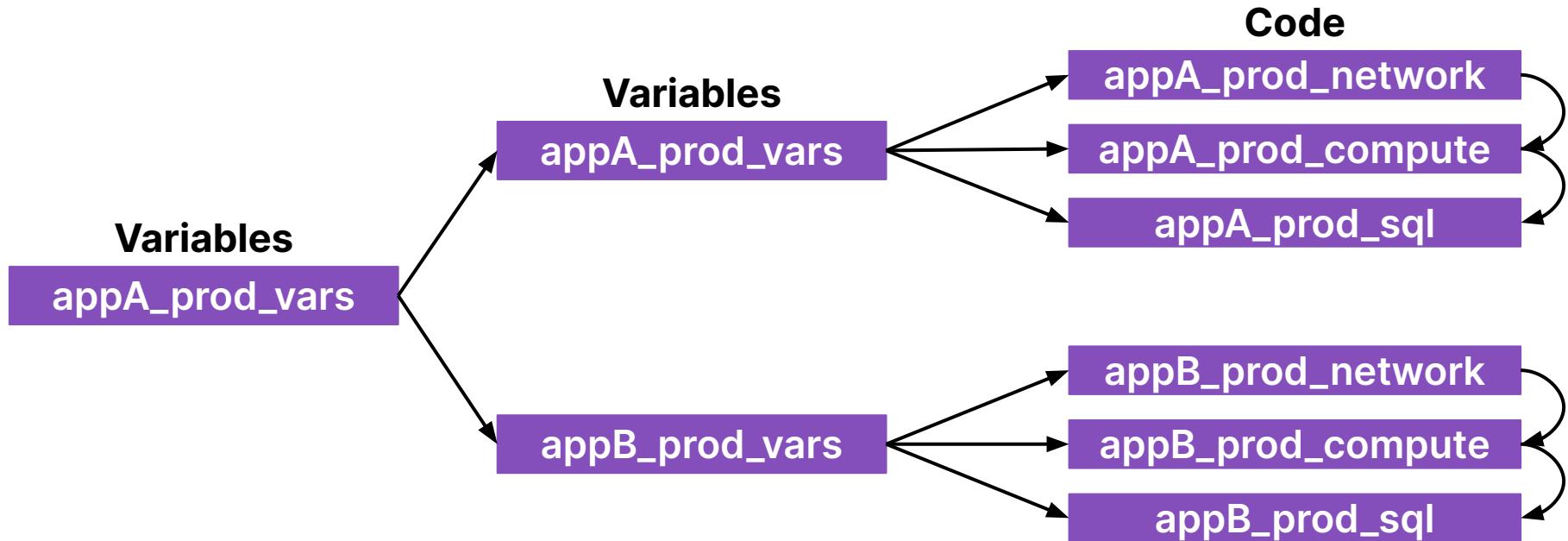
Changes Across Workspaces with Run Triggers



Changes Across Workspaces with Run Triggers



Changes Across Workspaces with Run Triggers



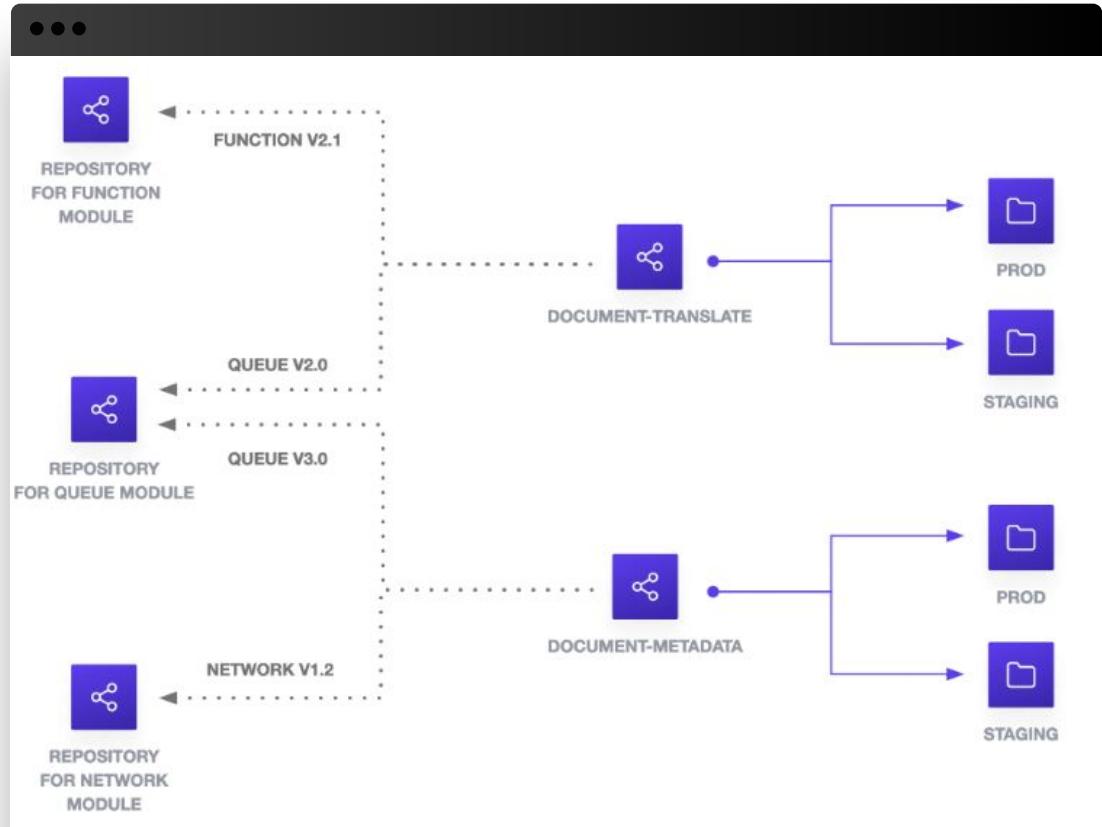
06

Git Repository Structure

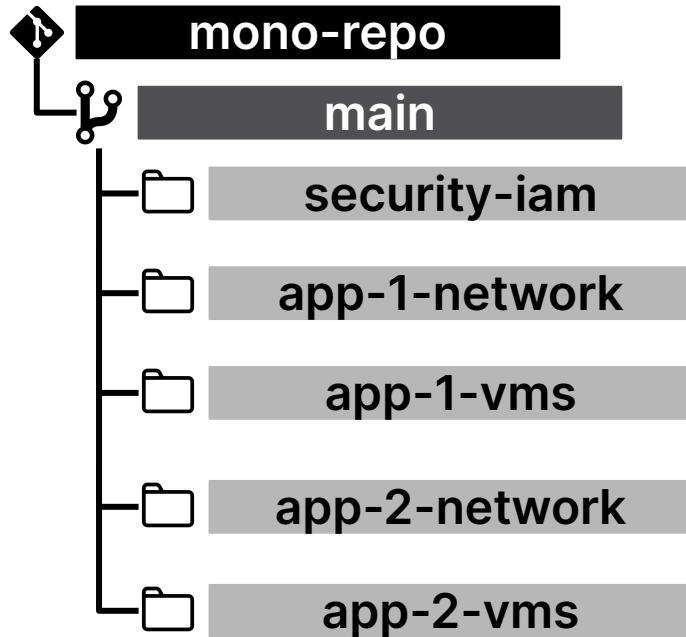


Mono Repo vs. Multi Repo

- TFE supports both models
- A **mono repo** is a single repo organized by folder
- A **multi repo** is dividing your repos per application / component
- For Private Registry Models you must use a repo per module when using VCS integration
- *CAUTION: Managing a large mono repo can be complex and may impact performance*

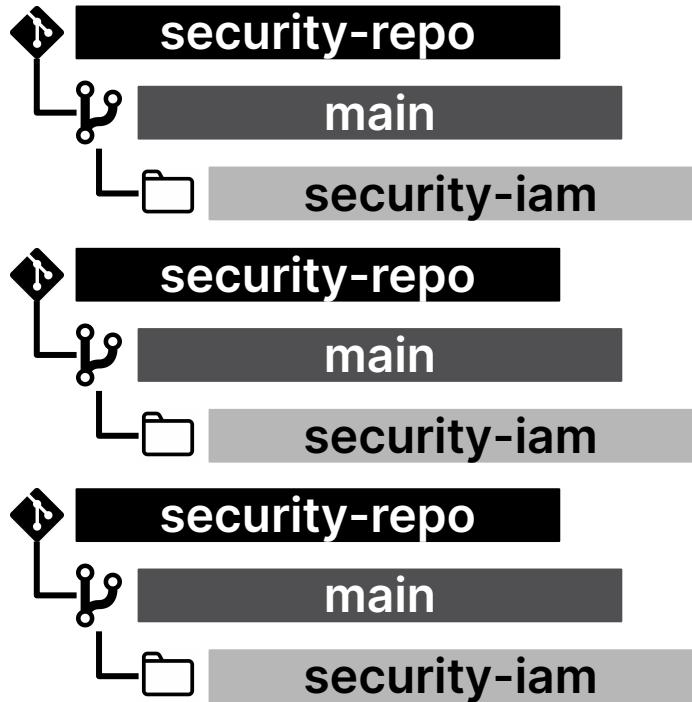


Repository Structure Mono Repo



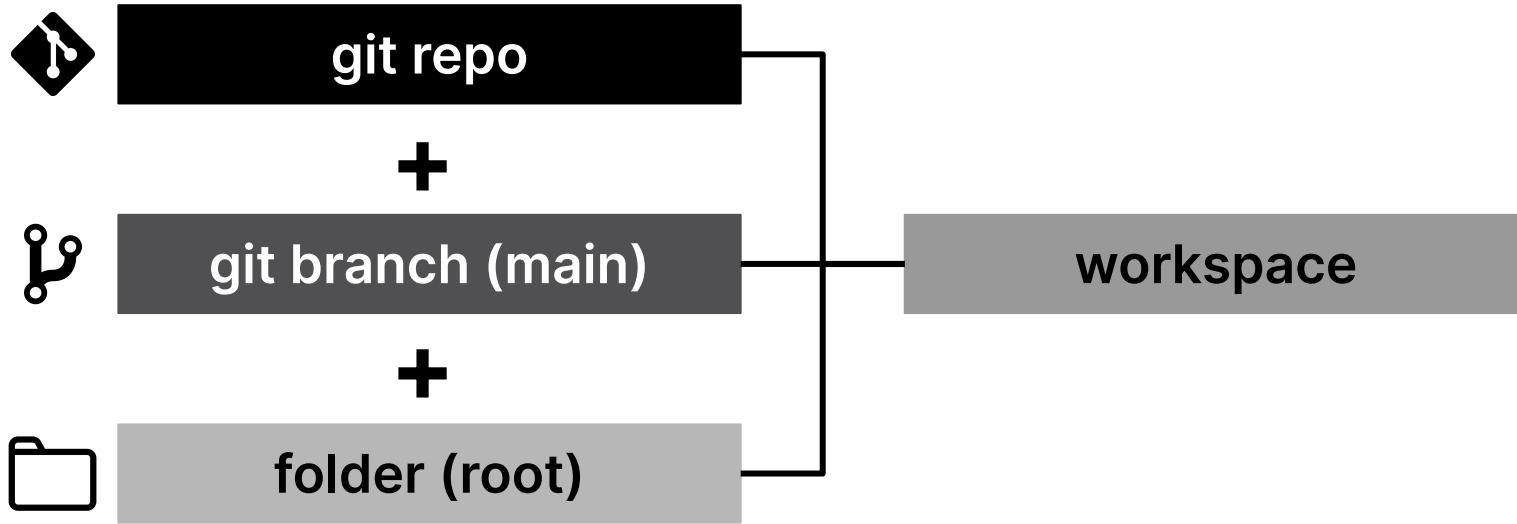
- Workspace per Env
- Single Git Repo
- Large git clones
- git tag / version is applied across the whole repo

Repository Structure Multi Repo



- Workspace per Env
- Multi Git Repo
- Small git clones
- git tag / version is per application

Components, for VCS-Driven Runs

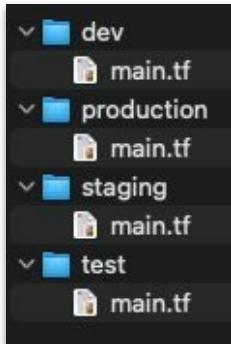


Code Structure for Environments

Three main options

1 Many Code copies with hard coded variables

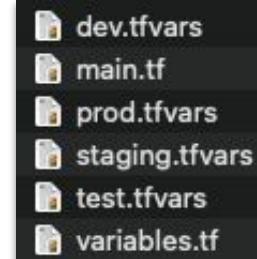
Do not use variables, simply hard code the differences between environments directly in the HCL files.



2 Single Code copy with variable files (API / CLI)

Use variables in the code and store any non-secret variables in source control per environment.

This pattern can be used for API / CLI driven workflows. You can specify the tfvars file on the command line or copy it to a *.auto.tfvars file.



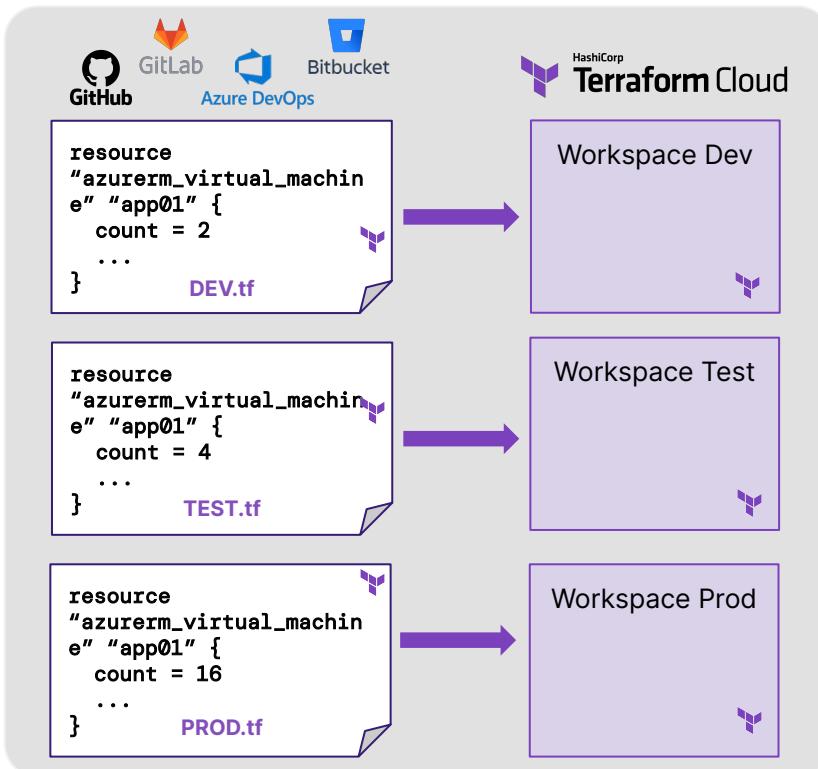
3 Single Code copy with deploy time variables

Use variables in the code and inject from the workspace or CD tool at deploy time.

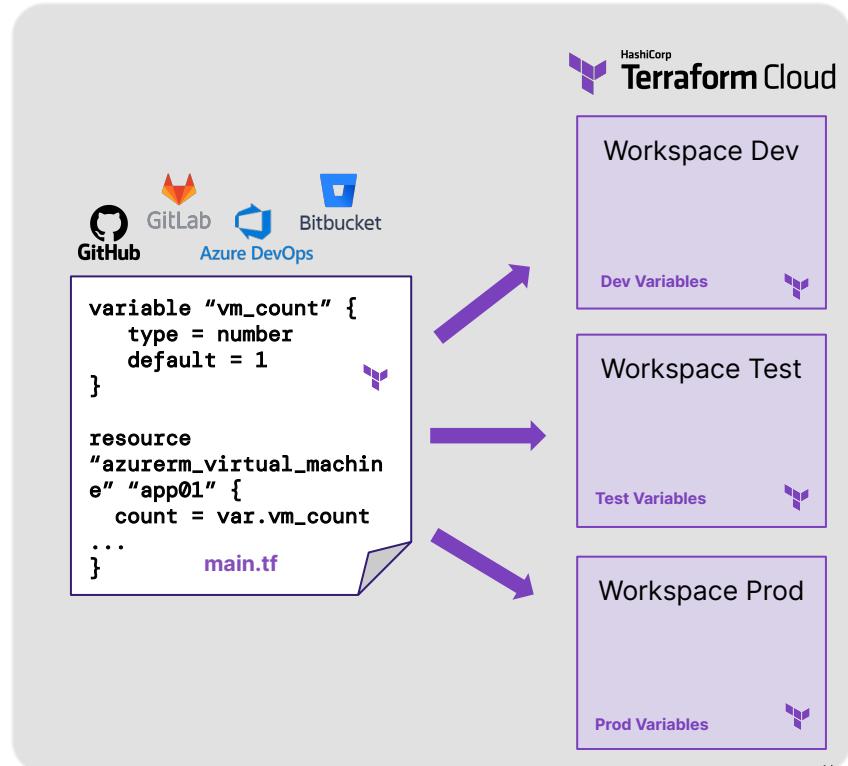


Code Structure for Environments

Many Copies vs Single Copy



VS



Branching Strategies

Main options for Branching with TFE

Trunk based

- Trunk based branching refers to having a single branch that you deploy from
- Changes get into the trunk branch via a Pull Request from a short lived feature branch
- Run Triggers or a CD tool are used to promote to environments

Git Flow or similar

- More complicated structure, you may have a dev branch that deploys to dev environment and release and hotfix branches that go to staging and production
- You would still likely use Run Triggers or a CD tool to get from staging to production

Branch per Environment

- Each environment has its own branch associated with a workspace
- To kick off runs to different environments, the branches are merged into each other going up the chain

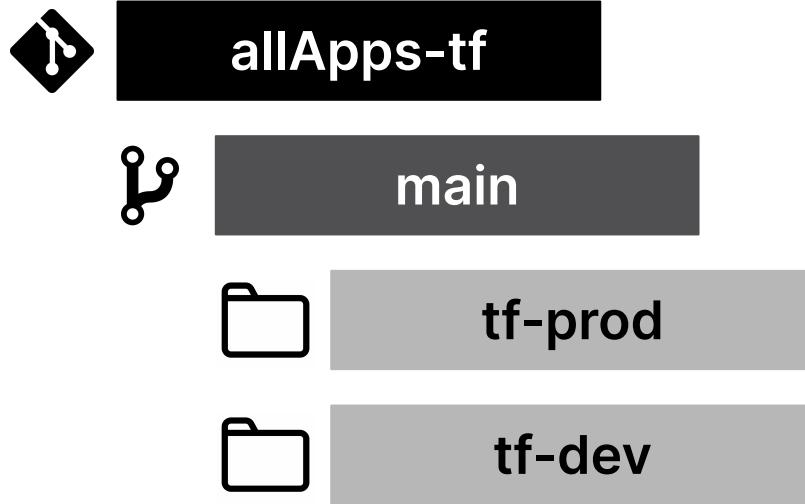


Monorepo, One Branch, One Folder



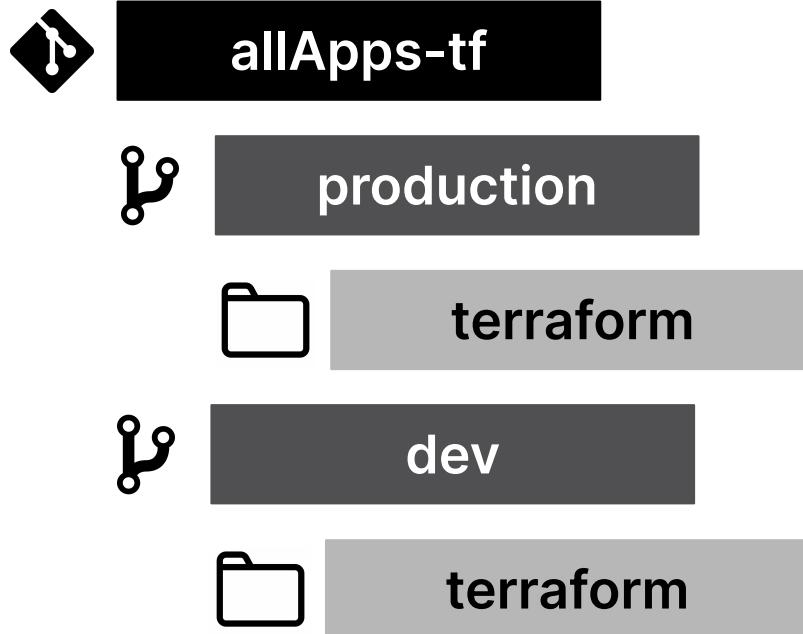
- Workspace per Env
- One folder for all Envs
- **impractical**
- **any errors would take down all infrastructure**

Monorepo, One Branch, Many Folders



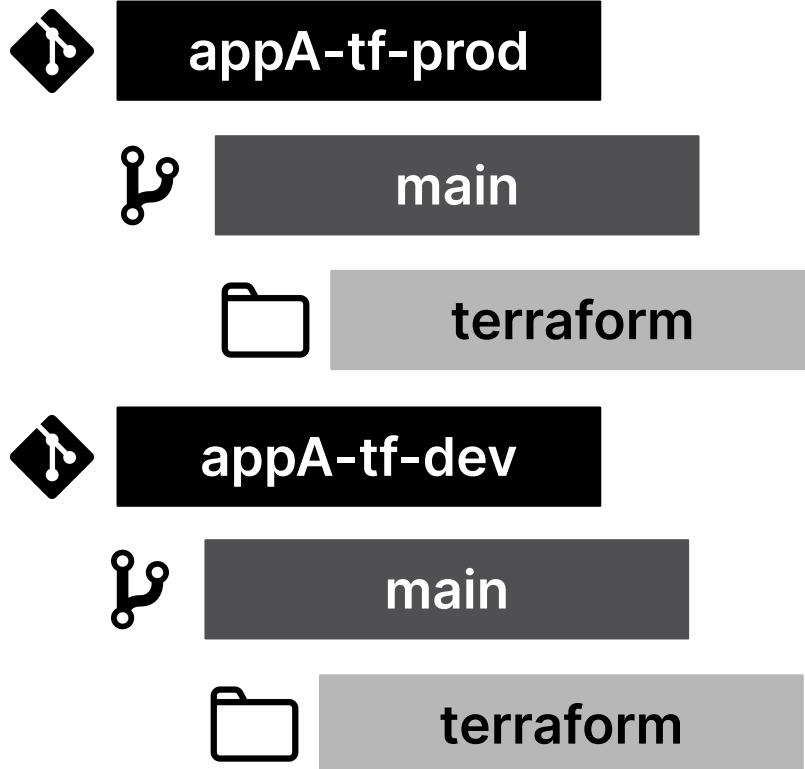
- Workspace per Env
- Folder per Env
- one git repo
- one git branch
- large git clones
- duplicate code
- difficult git PR merges
- cannot git tag / version

Monorepo, Many Branches, One Folder



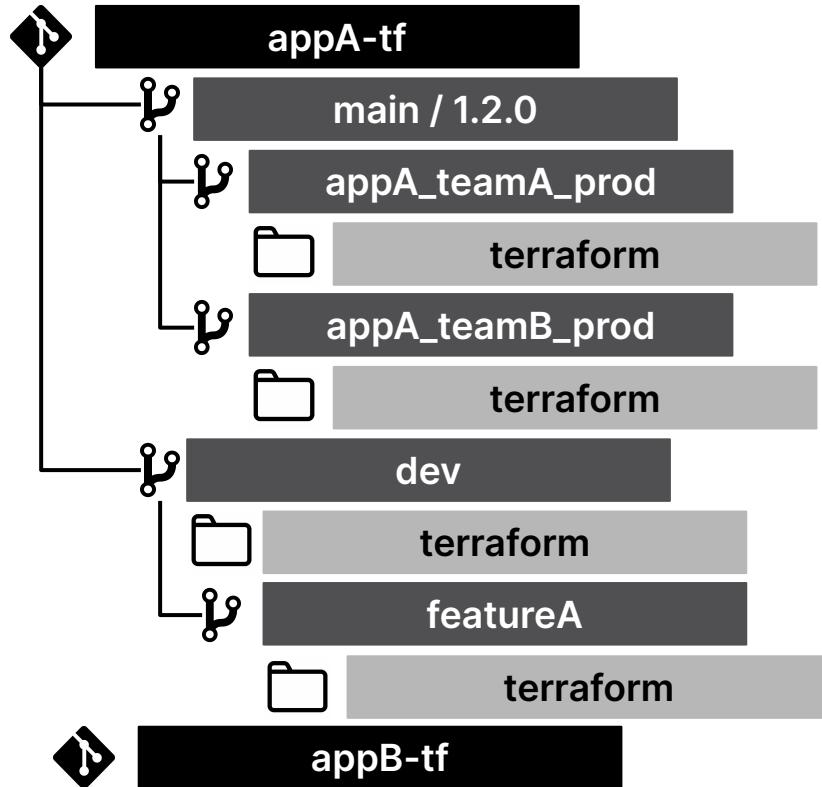
- Workspace per Env
- Git Branch per Env
- many git branches
- no duplicate code
- one git repo
- large git clones
- difficult git PR merges

Many Repos, One Branch, One Folder



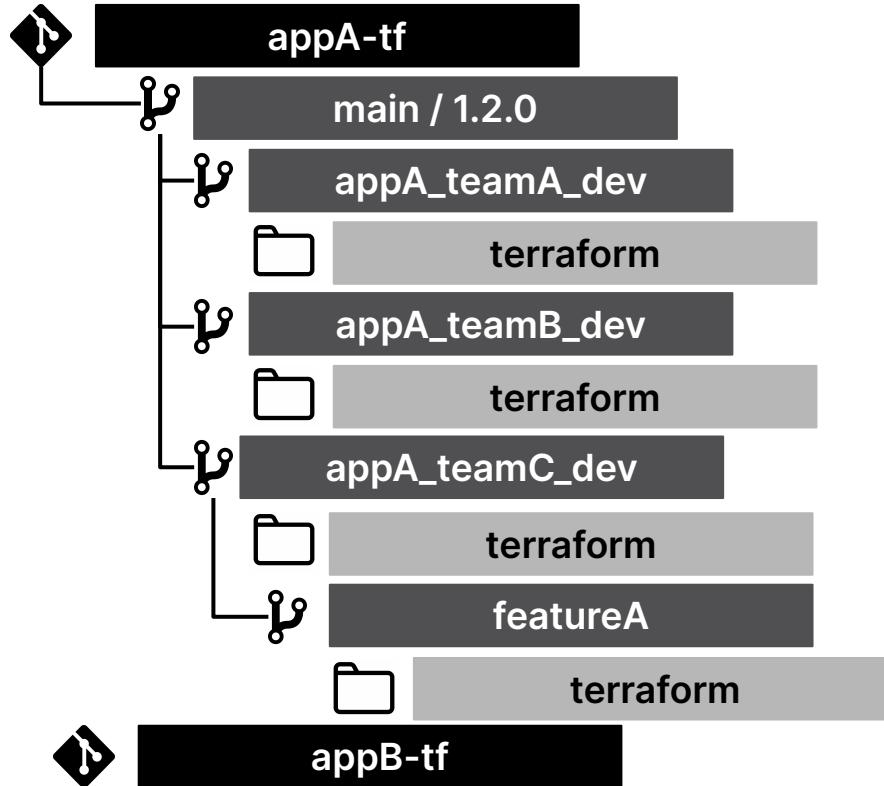
- Workspace per Env
- Git Repo per Env
- many git repos
- small git clones
- easy git PR merges
- one Folder per Env
- one git branch per repo
- duplicate code
- can't tag / version

Many Repos, Many Branches, One Folder



- Workspace per Env
- Git Branch per Env
- many Git Repos
- many Git Branches
- small git clones
- easy git PR merges
- no duplicate Code
- easily git tag / version

Many Repos, Many Branches, One Folder



- Workspace per Env
- Git Branch per Env
- many Git Repos
- many Git Branches
- small git clones
- easy git PR merges
- no duplicate Code
- easily git tag / version

Refactoring a Git Monorepo

1. Refactor to use Terraform Modules
2. Create Git Repos for each Terraform Module
3. Create Git Repos of Terraform Code for each App
4. Use Git Branches for each App Instance, and each Environment (production, dev, etc.)
5. Use folders for the TF Modules / “App Layers”
6. Use a GitOps Workflow, using Git Branches, and a Terraform “VCS-Driven Workflow” to update and modify code and configuration changes

[Terraform Repository Best Practices](#)



Next Steps



Tutorials

<https://developer.hashicorp.com/terraform/tutorials>

Step-by-step guides to accelerate deployment of Terraform Cloud

The screenshot shows the 'Tutorials' section of the Terraform Cloud developer documentation. The left sidebar lists various categories like 'Get Started' (AWS, Azure, Docker, GCP, OCI), 'Fundamentals' (CLI, Configuration Language, Modules, Provision, State), 'Use Cases' (Terraform Cloud, Applications, AWS Services), and 'Tutorials' (Overview, Terraform Cloud). The main content area displays a breadcrumb path: Developer / Terraform / Tutorials / Terraform Cloud. It features a 'Get Started - Terraform Cloud' section with a brief description and a 'Create an account' link. Below this, three tutorials are listed:

- What is Terraform Cloud - Intro and Sign Up** (5min) - Sign up for Terraform Cloud, which provides free remote state storage, a stable run environment, version control system (VCS) driven plans and applies, a collaborative web GUI, and more. Create your first organization.
- Log in to Terraform Cloud from the CLI** (3min) - Log into Terraform Cloud or Enterprise with the Terraform CLI to migrate state, trigger remote runs, and interact with Terraform Cloud.
- Create a Credentials Variable Set** (3min) - Create a variable set for your AWS IAM credentials that you can reuse across workspaces. Apply the variable set to a workspace.



Additional Resources

- [Module Creation - Recommended Pattern](#)
- [Add Public Providers & Modules to your Private Registry](#)
- [Share Modules in the Private Registry](#)
- [Terraform Registry Publishing](#)
- [Managing Credentials in Terraform Cloud & Enterprise](#)
- [Terraform Mono Repo vs. Multi Repo: The Great Debate](#)
- [Refactor Monolithic Terraform Configuration](#)

Need Additional Help?

Customer Success

Contact our Customer Success Management team with any questions. We will help coordinate the right resources for you to get your questions answered.

customer.success@hashicorp.com

Technical Support

Something not working quite right? Engage with HashiCorp Technical Support by opening a ticket for your issue at:

support.hashicorp.com

Discuss

Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers

discuss.hashicorp.com



Upcoming Webinars



Office Hours

Bring your questions to Office Hours!



Terraform Governance

Learn best practices and guidance for implementing key TFE features like Cloud Agents, RBAC, Sentinel, and run triggers and notifications



Operating your Terraform Instance

Learn best practices for operating and managing your Terraform Enterprise instances, content includes guidance on backup/restore, upgrades, monitoring, and auditing

Action Items

- If you haven't, share to customer.success@hashicorp.com
 - Authorized technical contacts for support
 - Stakeholders contact information (name and email addresses)
- Plan your workspace strategy & structure
- Begin creating some reusable modules
- Formulate your repository structure & migration plan (if required)



Q&A





Thank you

customer.success@hashicorp.com

www.hashicorp.com/customer-success