HashiCorp

# Terraform

# Scaling Infrastructure Automation

# Course Agenda

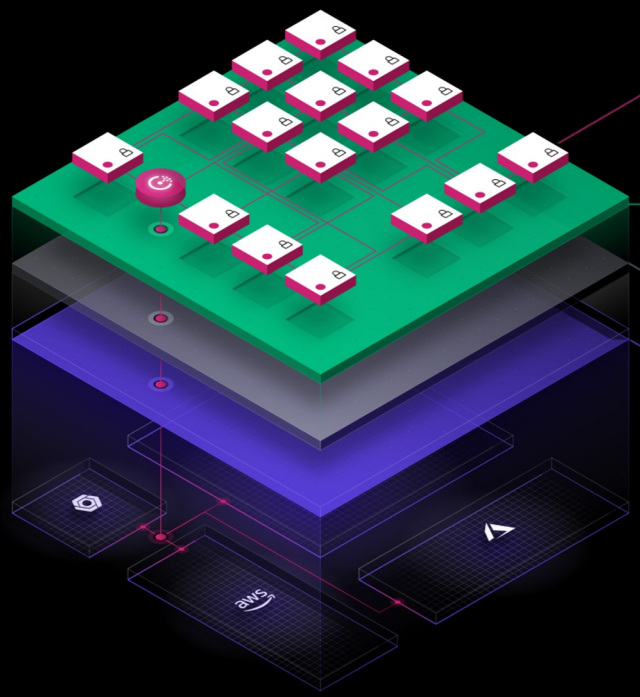
## Terraform 201 – Scaling Infrastructure Automation

Course Agenda:

- Terraform Cloud/Enterprise Overview
- Version Control
- Remote State Management
- Variable Management
- Data Blocks
- Conditional Logic
- Validation and Suppression
- Local Values
- For-Each Blocks
- Dynamic Blocks
- VCS Integration and Workflow
- Private Module Registry
- Null Resource and Remote State Data

# HashiCorp Ecosystem

Our open-source offerings for managing infrastructure.




## Connect

Consul - runtime service discovery and distributed key-value stores

## Run

Nomad - distributed application scheduler

## Secure

Vault - manages secrets across dev, ops, prod, etc.

## Provision

Terraform - provisions infrastructure resources

## Terraform's Goals

- Infrastructure as Code
- Modern data center support
- Safe, predictable operation
- Tech Agnostic
- Manage anything with an API


### Connect

Consul - runtime service discovery and distributed key-value stores

### Run

Nomad - distributed application scheduler

### Secure

Vault - manages secrets across dev, ops, prod, etc.

### Provision

Terraform - provisions infrastructure resources

# Introductions

- Background
- What would you like to get out of the class?
- AWS Experience? Azure? GCP?
- Coolest thing you have done with Terraform to date?
- What Time Zone are you joining from?

# Overview

## We will cover

- Tools for teams and scale
- Terraform Cloud
- Examples use AWS
- Q&A and your requests

## We will NOT cover

- Terraform Enterprise (OnPrem)
- Terraform Enterprise API
- Source code management tools
- Configuration management tools
- Every cloud

# What You'll Learn

- Use collaborative features of Terraform
- Organize Terraform projects for large scale infrastructure installations
- Use state
- Become confident with supporting tools

Terraform 201

# Review

# Key Terraform Syntax & Features

- Summary of providers, resources, variables, outputs
- Summary of modules

- Tips for upgrading commands and providers

# Terraform's Goals

Unify the view of resources using infrastructure as code

Support the modern data center (IaaS, PaaS, SaaS)

Expose a way for individuals and teams to safely and predictably change infrastructure

Provide a workflow that is technology agnostic

Manage anything with an API

# Terraform vs. Other Tools

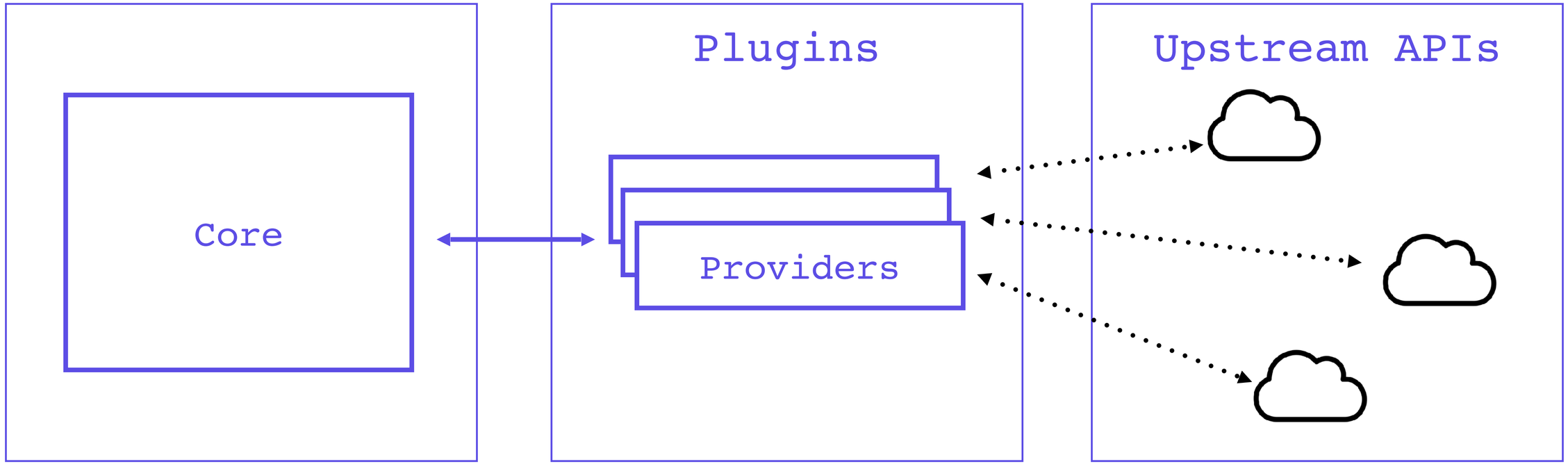
Provides a high-level abstraction of infrastructure (IaC)

Allows for composition and combination

Supports parallel management of resources (graph, fast)

Separates planning from execution (dry-run)

# Terraform's Internals: Structure



# Configuration Syntax

Configurations are written in HashiCorp Configuration Language (HCL)

HCL is designed to strike a balance between human-readable and machine-parsable

Terraform can also read configuration in JSON, but we recommend using HCL, except when the configuration is machine-generated

# Top Level Keywords

```
provider        module
variable        data
resource        terraform
output          locals
```

# Terraform's Internals: Core Concepts

**Config:** Target Reality

**State:** Current Reality

**Diff:** {Config - State}

**Plan:** Presents Diff

**Apply:** Resolves Diff

# Terraform Syntax

Providers - Cloud, VCS, Monitoring, etc.

Resources - EC2 Instances, IAM Users, etc.

Variables - Input for tagging or defining resources.

Outputs - Used for record keeping or as a variable to pass into other modules.

Modules - Segmented configurations for easy repeatability.

# Providers

Declaring plugins for Terraform to use in building infrastructure

CODE EDITOR

```
provider "aws" {}
```

# Secure Credentials with ENV Variables or Cloud Role

For flexibility and to nearly eliminate the possibility of accidentally committing cloud credentials to source code control systems, consider storing cloud credentials in environment variables. Or use cloud role.

If they are named correctly, the appropriate Terraform provider will discover and use them.

```
export AWS_ACCESS_KEY_ID="AAAAAA"
export AWS_SECRET_ACCESS_KEY="XXXXXXXX"
export AWS_DEFAULT_REGION="us-west-2"
```

# Securely Sourcing Credentials

With an AWS CLI profile, we can load credential profiles into our configuration

⚠ Important :

## Providers MUST be explicitly upgraded

We value stability and we respect your control over your environment.

By default, Terraform will lock on to the first version available when you first run init on a configuration.

To upgrade, you must explicitly state a version in your configuration, or run this command:

```
terraform init -upgrade
```

# Provider Versioning

You can explicitly declare the version number of the provider you want to use.

CODE EDITOR

```
provider "aws" {
  version = ">= 1.19.0"
}
```

# Resources

The resource stanza declares what infrastructure Terraform will build and the attributes of those resources

CODE EDITOR

```
provider "aws" {}

resource "aws_instance" "web" {
 ami           = "ami-e5d9439a"
 instance_type = "t2.nano"
}
```

# Variables

The variable stanza defines the parameters of data to be interpreted into our configuration later

CODE EDITOR

```
variable "ami" {
 description = "The Amazon Machine Image ID"
 default     = "ami-12345”
}

provider "aws" {}

resource "aws_instance" "web" {
 ami           = var.ami
 instance_type = "t2.nano"
}
```

# More variables!

Variables should be moved to separate “variables.tf” files.

Terraform can parse map and list types.

CODE EDITOR

```
variable "images" {
 type = "map"
 default = {
   us-east-1 = "image-1234"
   us-west-2 = "image-4567"
 }
}
variable "zones" {
 type = "list"
 default = ["us-east-1a", "us-east-1b"]
}
```

# Outputs

Outputs can render data from the state file for later use.

Should also be moved to a seperate “output.tf” file for maintainability

CODE EDITOR

```
provider "aws" {}

resource "aws_instance" "web" {
 ami           = var.images
 instance_type = "t2.nano"
}

output "public_ip" {
 value = aws_instance.web.public_ip
}
```

# Directory Layout

A common Terraform folder structure

TERMINAL

```
my-infrastructure
├── main.tf
├── output.tf
├── terraform.tfvars
├── terraform.tfstate
└── variables.tf
```

# Using modules

Modules are referenced with the local path.

CODE EDITOR

```
module "vpc" {
 source = "../vpc_module"
}
```

# Modules:

# File Structure

This is an example of a file structure for a module nested within our current directory.

TERMINAL

```
$ tree my-infrastructure/
.
├── main.tf
├── output.tf
├── terraform.tfvars
├── terraform.tfstate
├── variables.tf
└── vpc_module/
    ├── main.tf
    ├── output.tf
    └── variables.tf
```

# Built-in Functions

There are several functions for working with lists, files, and more

CODE EDITOR

```
# Useful built-in functions
basename(path)
cidrhost(prefix, hostnum)
concat(list1, list2, ...)
file(path)
join(separator, list)

my-list[7]
```

https://www.terraform.io/docs/configuration/functions.html

# Versioning

# Versioning

Terraform consults version constraints to determine whether it has acceptable versions of itself, any required provider plugins, and any required modules.

For plugins and modules, it will use the newest installed version that meets the applicable constraints.

If Terraform doesn't have an acceptable version of a required plugin or module, it will attempt to download the newest version that meets the applicable constraints.

# Versioning

If Terraform isn't able to obtain acceptable versions of external dependencies, or if it doesn't have an acceptable version of itself, it won't proceed with any plans, applies, or state manipulation actions.

Both the root module and any child module can constrain the acceptable versions of Terraform and any providers they use. Terraform considers these constraints equal, and will only proceed if all of them can be met.

# Version Syntax

A version constraint is a string literal containing one or more conditions, which are separated by commas.

Each condition consists of an operator and a version number.

# Version Syntax

Valid Operators:

= : Allows only one exact version number. Cannot be combined with other conditions.

!= : Excludes an exact version number.

>, >=, <, <=: Comparisons against a specified version, allowing versions for which the comparison is true.

~>: Allows only the rightmost version component to increment. For example, to allow new patch releases within a specific minor release, use the full version number: ~> 1.0.4 will allow installation of 1.0.5 and 1.0.10 but not 1.1.0.

```
terraform {
    required_version = ">= 1.1.0"
    required_providers {
      aws = {
       source  = “hashicorp/aws”
       version = “~3.0”
      }
    }
}
```

# Dependency Lock

Terraform configurations must declare which providers they require so that Terraform can install and use them.

Terraform will record the provider versions in a dependency lock file to ensure that others using this configuration will utilize the same Terraform and provider versions.

This file by default is saved as a *.terraform.lock.hcl* file

```
provider “registry.terraform.io/hashicorp/aws" {
  version     = “3.73.0"
  constraints = "~> 3.0"
  hashes = [
    "h1:fpZ14qQnn+uEOO2ZOlBFHgty48Ol8IOwd+ewxZ4z3zc=",
    "zh:0483ca802ddb0ae4f73144b4357ba72242c6e2641aeb460b1aa9a6f6965464b0",
    "zh:274712214ebeb0c1269cbc468e5705bb5741dc45b05c05e9793ca97f22a1baa1",
    "zh:3c6bd97a2ca809469ae38f6893348386c476cb3065b120b785353c1507401adf",
    "zh:53dd41a9aed9860adbbeeb71a23e4f8195c656fd15a02c90fa2d302a5f577d8c",
    "zh:65c639c547b97bc880fd83e65511c0f4bbfc91b63cada3b8c0d5776444221700",
    "zh:a2769e19137ff480c1dd3e4f248e832df90fb6930a22c66264d9793895161714",
    "zh:a5897a99332cc0071e46a71359b86a8e53ab09c1453e94cd7cf45a0b577ff590",
    "zh:bdc2353642d16d8e2437a9015cd4216a1772be9736645cc17d1a197480e2b5b7",
    "zh:cbeace1deae938f6c0aca3734e6088f3633ca09611aff701c15cb6d42f2b918a",
    "zh:d33ca19012aabd98cc03fdeccd0bd5ce56e28f61a1dfbb2eea88e89487de7fb3",
    "zh:d548b29a864b0687e85e8a993f208e25e3ecc40fcc5b671e1985754b32fdd658",
  ]
}
```

# Provider Upgrades

By default, if a .terraform.lock.hcl file exists within a Terraform working directory, the provider versions specified in this file will be used.

This lock file helps to ensure that runs accross teams will be consistent.

To upgrade provider versions we update our configuration definition to the desired provider version and running an upgrade.

TERMINAL

```
> terraform init -upgrade
```

```
terraform {
  required_version = "1.1.0"

  required_providers = {

    aws = {
      version = "~>3.0"
      source = "hashicorp/aws"
    }
  }
}
```

EXERCISE

# Lab 1: Getting Started

Duration: 15 minutes

Goal: Familiarize yourself with your workstation and deploy a basic configuration

EXERCISE

# Lab 2: Versions

Duration: 10 minutes

Goal: Control the version of Terraform and AWS provider the code is compatible with

# Terraform OSS and TFC/TFE

# Terraform Ecosystem

| Terraform CLI | Terraform Cloud | Terraform Enterprise |
|---|---|---|
| Command line tool | SaaS app | On premise/in your cloud |
| Infrastructure as Code | Free for individuals and teams | Workspace & team management |
| Open Source | Remote state | Governance and policy features |
| 150+ Providers | Version Control System Integration | Private module registry |
| Module Registry | | |

EXERCISE

# Lab 3: Getting Started with TFC

Duration: 20 minutes

Goal: Sign up and get familiar with Terraform Cloud by creating an organization and workspace

# Remote State Data and TFC

```
terraform {
  required_version = "1.1.0"

  required_providers = {

    aws = {
      version = "~>3.0"
      source = "hashicorp/aws"
    }
  }

  backend "s3" {

  }
}
```

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

TERMINAL

```
> terraform init –backend-config="secret_key=ASDFGHJJUH"

> export AWS_SECRET_ACCESS_KEY=ASDFGHJKIHN
```

TERMINAL

```
> terraform init —backend-config="secret_key=ASDFGHJJUH"

Initializing modules...

Initializing the backend...
Do you want to copy existing state to the new backend?

Pre-existing state was found while migrating the previous
"local" backend to the newly configured "remote" backend.
No existing state was found in the newly configured
"remote" backend. Do you want to copy this state to the new
"remote" backend? Enter "yes" to copy and "no" to start
with an empty state.

  Enter a value: yes
```

EXERCISE

# Lab 4: Remote State

Duration: 15 minutes

Goal: Migrate your current state to a Terraform Cloud workspace

# Variables in TFC

EXERCISE

# Lab 5: Secure Variables TFC

Duration: 10 minutes

Goal: Migrate your variable values to the TFC workspace and apply the configuration

# Data Sources and Conditional Logic

MAIN.TF

```
data "aws_ami" "ubuntu_20_04" {

  most_recent = true


  filter {

    name   = "name"

    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]

  }


  owners = ["099720109477"]

}
```

MAIN.TF

```
resource "aws_instance" "web" {

  ami                    = data.aws_ami.ubuntu_18_04.image_id

  instance_type          = "t2.micro"

  subnet_id              = var.subnet_id

  vpc_security_group_ids = var.vpc_security_group_ids
```

MAIN.TF

```
condition ? true_val : false_val
```

```
count = var.create_server ? 1 : 0
```

```
(var.server_os == "ubuntu") ? ami-1234 : ami-8765
```

EXERCISE

# Lab 6: Data Conditions

Duration: 15 minutes

Goal: Add data sources to your configuration and use conditional logic

# Local Values

MAIN.TF

```
locals {
  service_name = "Automation"
  owner        = "Cloud Team"
  common_tags = {
    Service = local.service_name
    Owner   = local.owner
  }
}
```

EXERCISE

# Lab 7: Local Variables

Duration: 15 minutes

Goal: Create local variables in your configuration and reference them

# Variable Validation and Suppression

MAIN.TF

```
variable "my_var" {

  type        = string


  validation {

    condition     = conditional_value

    error_message = "Full sentence describing error

  }

}
```

MAIN.TF

```
variable "my_var" {

  type        = string


  validation {

    condition     = conditional_value

    error_message = "Full sentence describing error

  }

}
```

MAIN.TF

```
variable "server_os" {

  type        = string

  description = "Server Operating System"

  default     = "ubuntu_20_04"


  validation {

    condition     = contains(["ubuntu_20_04", "ubuntu_18_04", "windows_2019"],
 lower(var.server_os))

    error_message = "You must use an approved operating system. Options are ub
untu_18_04, ubuntu_20_04, or windows_2019."

  }

}
```

```
variable "secret_key" {

  type = string

  sensitive = true

}

output "secret_key" {

  value = nonsensitive(var.secret_key)

}
```

EXERCISE

# Lab 8: Variable Validation and Suppression

Duration: 20 minutes

Goal: Add validation to a variable and explore sensitive variables

# For-Each and Dynamic Blocks

# Looping Expressions

# Looping Constructs

Terraform HCL is a declarative language with a few primitives that allow certain types of loops, if-statements and other logic.

Terraform offers several looping constructs:

- `count` - loop over resources
- `for_each` - loop over resources and inline blocks
- `for` - loop over lists and maps

# Meta-Arguments

## Count

The `count` argument allows for N number of identical resources to be created. This removes the need for iteration with "for" or "while" loops in many cases

```
# ...

resource "aws_instance" "web" {
  count           = 2

  ami             = "ami-07669fc90e6e6cc47"
  instance_type = "t2.micro"

# ...
}
```

```
# ...


output "public_ip" {
  value = aws_instance.web.*.public_ip
}

output "public_dns" {
  value = aws_instance.web.*.public_dns
}
```

```
module "server" {

  count                  = 2

  source                 = "./server"

  ami                    = ""

  server_os              = var.server_os

  subnet_id              = var.subnet_id

  vpc_security_group_ids = var.vpc_security_group_ids

  identity               = var.identity

}
```

TERMINAL

```
> terraform state list

module.server[0].aws_instance.web
module.server[1].aws_instance.web
```

# Meta-Arguments

## for_each

The `for_each` meta-argument accepts a map or a set of strings, and creates an instance for each item in that map or set.

Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

# The `for_each` Expression

```
resource "<PROVIDER>_<TYPE>" "<NAME>" {
  for_each = <COLLECTION>
  [CONFIG ...]
}
```

# The `each` Object

**each**

Allows you to modify the configuration of each instance. This object has two attributes:

`each.key` — The map key (or set member) corresponding to this instance.

`each.value` — The map value corresponding to this instance. (If a set was provided, this is the same as each.key.)

```
resource "aws_iam_user" "the-accounts" {
  for_each = toset( ["Todd", "James", "Alice", "Dottie"] )
  name     = each.key
}
```

```
resource "azurerm_resource_group" "rg" {
  for_each = {
    a_group = "eastus"
    another_group = "westus2"
  }
  name     = each.key
  location = each.value
}
```

```
module "server" {

  for_each                = toset(["windows_2019","ubuntu_20_04"])

  source                  = "./server"

  ami                     = ""

  server_os               = each.value

  subnet_id               = var.subnet_id

  vpc_security_group_ids  = var.vpc_security_group_ids

  identity                = var.identity

}
```

TERMINAL

```
> terraform state list

module.server["windows_2019"].aws_instance.web
module.server["ubuntu_20_04"].aws_instance.web
```

# The `for` Expression

**for**

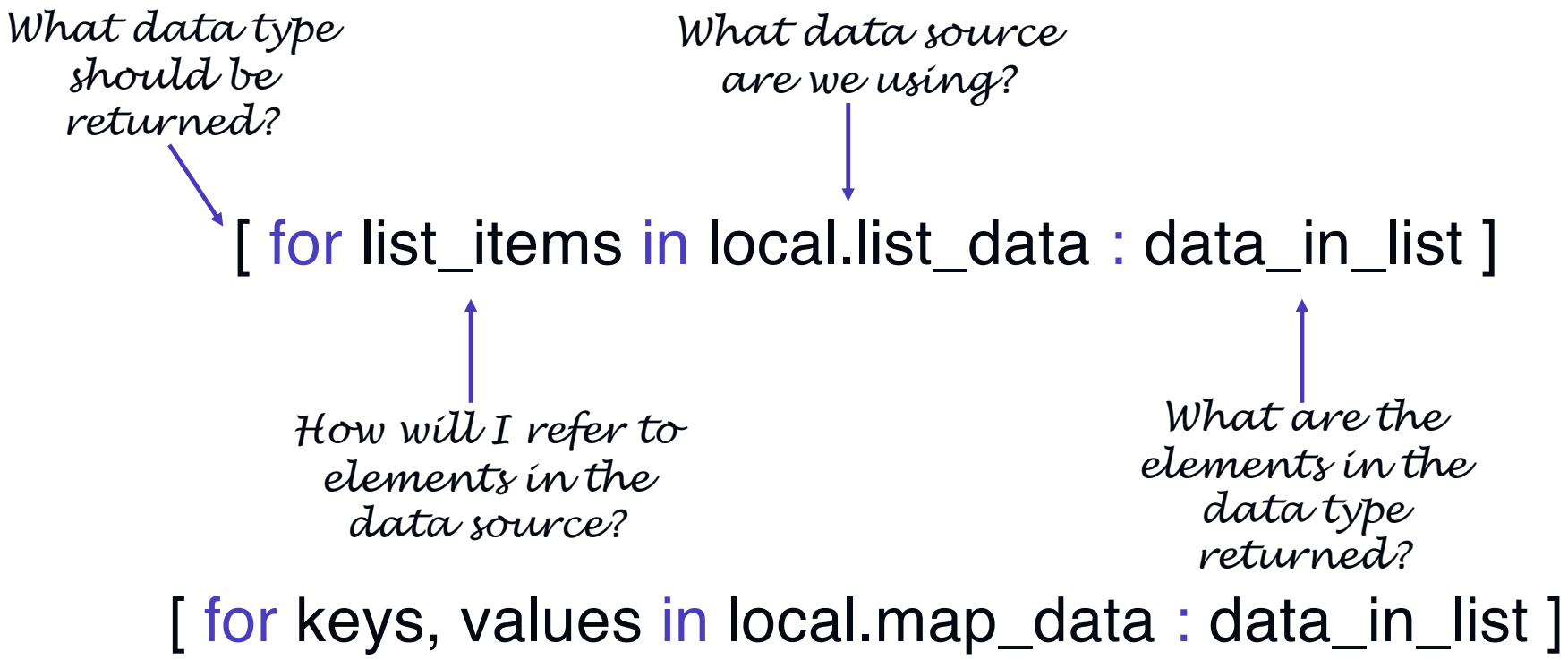
Terraform's `for` expressions also allow you to loop over a list or a map using the following syntax.

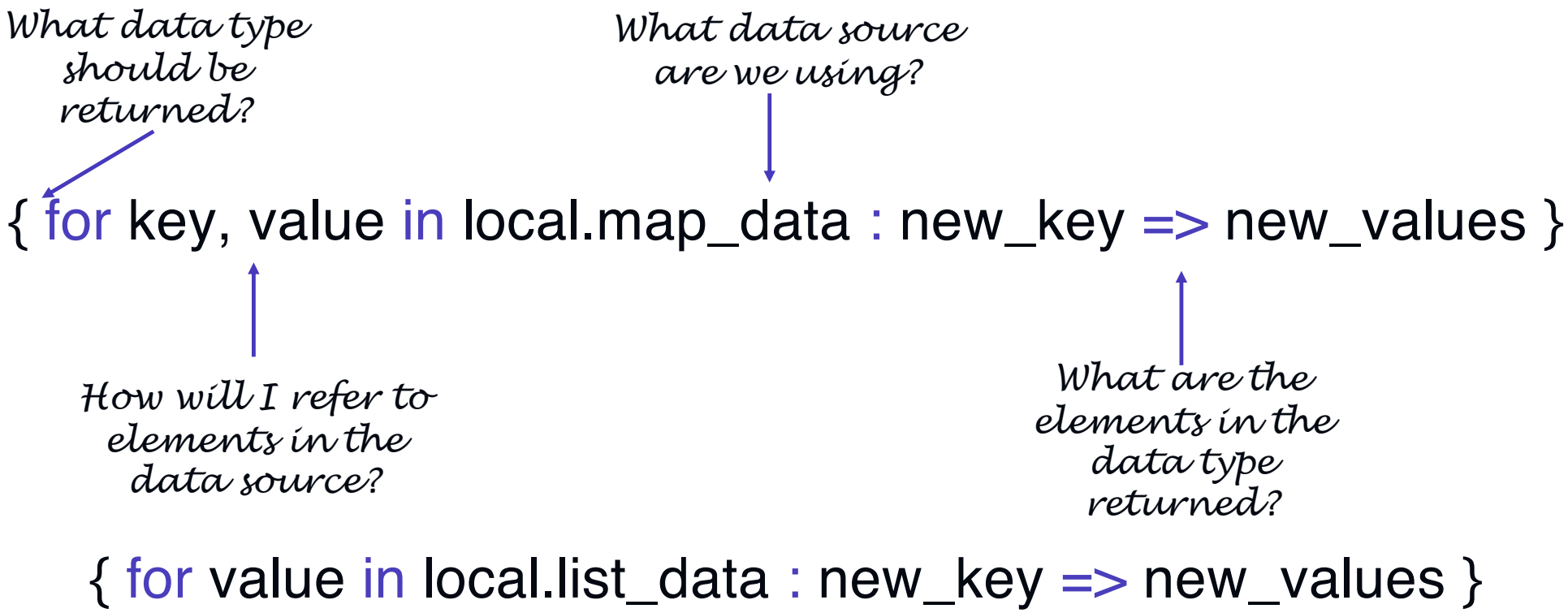
**Loop over a list / produce a list**

```
[for <ITEM> in <LIST> : <OUTPUT>]
```

**Loop over a map / produce a list**

```
[for <KEY>, <VALUE> in <MAP> : <OUTPUT>]
```


What data type should be returned?
What data source are we using?
[ for list_items in local.list_data : data_in_list ]
How will I refer to elements in the data source?
What are the elements in the data type returned?
[ for keys, values in local.map_data : data_in_list ]


What data type should be returned?
What data source are we using?
{ for key, value in local.map_data : new_key => new_values }
How will I refer to elements in the data source?
What are the elements in the data type returned?
{ for value in local.list_data : new_key => new_values }

# Looping

| | Use Case | Expression |
|---|---|---|
| `count` | Multiple copies of an entire resource | count = 2 |
| `for_each` | Multiple copies of an entire resource<br>Multiple copies of an inline block within a resource | `resource "<PROVIDER>_<TYPE>" “<NAME>" {`<br>`  for_each = <COLLECTION>`<br>`  [CONFIG ...]`<br>`}` |
| `for` | Generate values by looping over lists and/or maps | [for <ITEM> in <LIST> : <OUTPUT>]<br><br>[for <KEY>, <VALUE> in <MAP> : <OUTPUT>] |

EXERCISE

# Lab 9: For Each

Duration: 20 minutes

Goal: Observe the use of a count meta-argument and replace it with for-each

# Dynamic Blocks

# Looping

Sometimes you need to create multiple inline blocks within a resource.

You can dynamically construct repeatable nested blocks using Terraform dynamic blocks

A dynamic block acts much like a for expression, but produces nested blocks instead of a complex typed value. Utilized the `for_each` expression.

# Multiple Inline Blocks

```
resource "aws_security_group" "main" {
  name   = "core-sg"
  vpc_id = aws_vpc.vpc.id

  ingress {
    description = "Port 443"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "Port 80"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

# Dynamic Blocks

```
dynamic "<VAR_NAME>" {
  for_each = <COLLECTION>
  content {
    [CONFIG...]
  }
}
```

# List of Maps

```
locals {
  ingress_rules = [{
      port        = 443
      description = "Port 443"
    },
    {
      port        = 80
      description = "Port 80"
    }
  ]
}
```

# Dynamic Block

```
resource "aws_security_group" "main" {
  name   = "core-sg"
  vpc_id = aws_vpc.vpc.id

  dynamic "ingress" {
    for_each = local.ingress_rules

    content {
      description = ingress.value.description
      from_port   = ingress.value.port
      to_port     = ingress.value.port
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
}
```

```
dynamic "block_name" {

  for_each = list | map | set

  iterator = iterator_name

  content {

    key = iterator_name.value

    key = expression…

  }

}
```

```
resource "aws_security_group" "main" {

  name   = "core-sg"


  dynamic "ingress" {

    for_each = local.ingress_rules


    content {

      description = ingress.value.description

      from_port   = ingress.value.port

      to_port     = ingress.value.port

      protocol    = "tcp"

      cidr_blocks = ["0.0.0.0/0"]
```

EXERCISE

# Lab 10: Dynamic Block

Duration: 15 minutes

Goal: Create security group rules with a dynamic block

# Private Module Registry and Version Control

EXERCISE

# Lab 11: Private Module Registry

Duration: 10 minutes

Goal: Explore the private module registry in TFC and add a module from the public registry

EXERCISE

# Lab 12: Connect VCS

Duration: 10 minutes

Goal: Connect TFC to GitHub using Oauth for VCS integration

EXERCISE

# Lab 13: Publish Module to Registry

Duration: 20 minutes

Goal: Create a private module in GitHub and publish to the private registry

EXERCISE

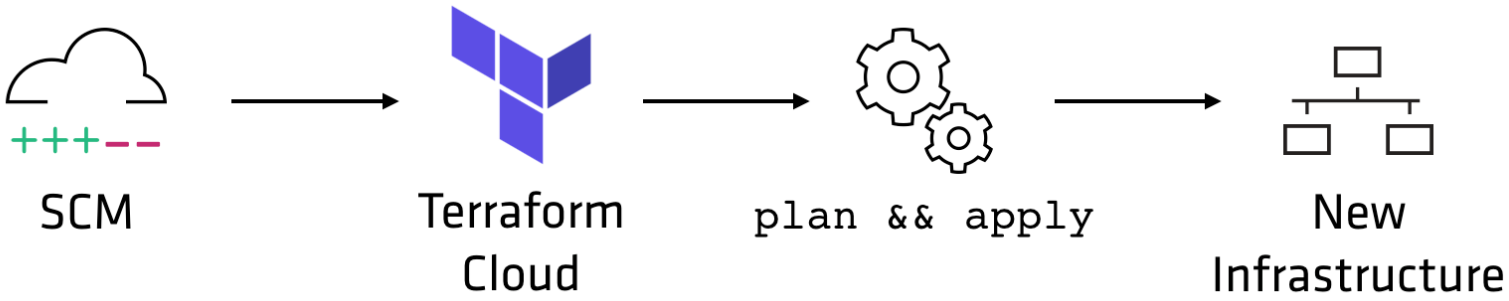
# Lab 14: Consume Registry

Duration: 10 minutes

Goal: Use the configuration designer with modules from the private registry

# TFC VCS Workflow

# VCS Workflow



EXERCISE

# Lab 15: VCS Workspace

Duration: 20 minutes

Goal: Link a TFC workspace to a VCS repository

# Null Resource and Remote State Data

```
resource "null_resource" "web_app" {

  # Triggers define what changes should trigger the null resource

  triggers = {  }


  # Run provisioners

  connection { }


  provisioner "file" {

  }


  provisioner "remote-exec" {

  }

}
```

```
data "terraform_remote_state" "state_name" {

  backend = "remote"


  config = {

    hostname = "app.terraform.io"

    organization = "org_name"


    workspaces = {

      name = "workspace_name"

    }

  }

}
```

EXERCISE

# Lab 16: Null Resources

Duration: 15 minutes

Goal: Use a null resource to execute scripts based on a trigger

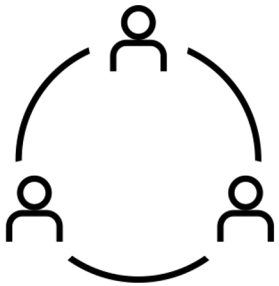
Terraform 201

# Code Organization for Teams

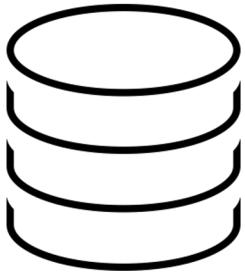
# What You'll Learn

- How Terraform projects evolve
- When to use modules vs a new project
- How to use shared state to manage complexity

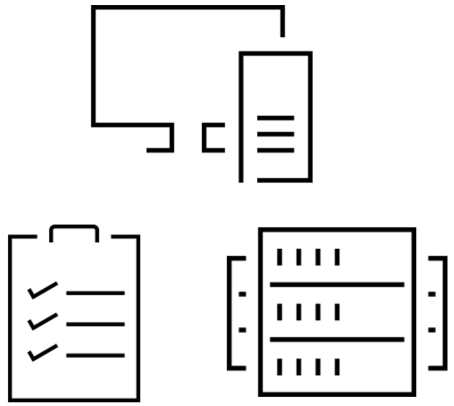
# Ways a Project Grows



Larger Team



More Resources

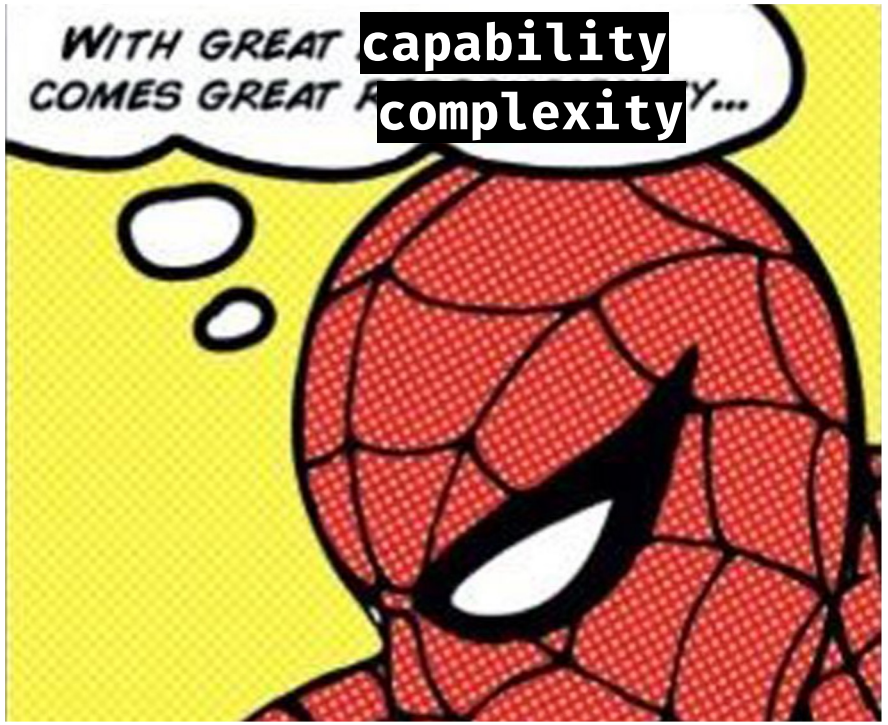


Dev, Test, Staging, Prod

# Discussion

- *Is this config owned by one team?*
- *Is this config used by more than one team?*
- *Does this config create resources?*

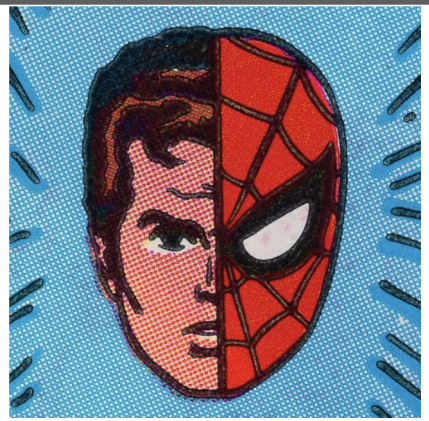
# Complexity, Capability




- A more expansive infrastructure will need a more complex Terraform project layout, but also requires more communication and additional layers of code management.

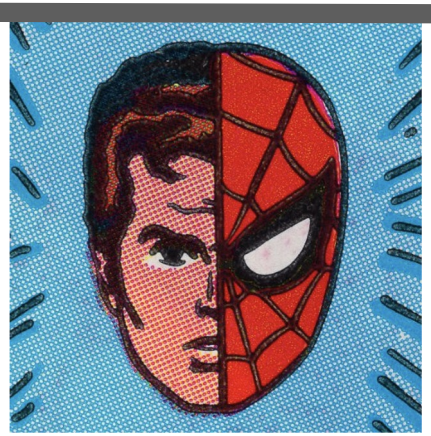
- Start simple. Refactor to manage technical or organizational complexity.

# The Growth Continuum



| Early Adoption | Mature Use |
| --- | --- |
| Dynamic | Stable |
| Explicit | DRY |
| Shared | Isolated |
| Customizable | Reusable |
| Creative | Consistent |
| Simple | Complex |

# The Growth Continuum



## Early Adoption

Single Local State

Root Module

Single Directory

Freeform Code

Full Control

## Mature Use

Many Remote State Files

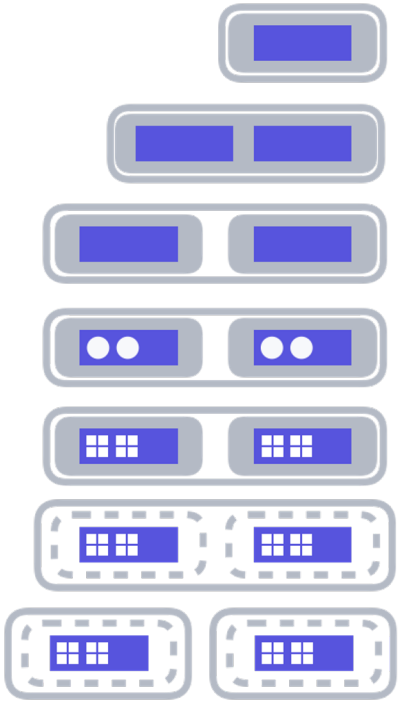
Versioned Modules

Many Repositories

Limited Configuration

Limited Access

# 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

TERMINAL

```
├── 1-single-file
    ├── main.tf
    ├── terraform.tfstate
    └── terraform.tfvars
```

# Multiple Environments

May contain test and prod (“web_test”, “web_prod”)

Possibly in different files within a single repo.

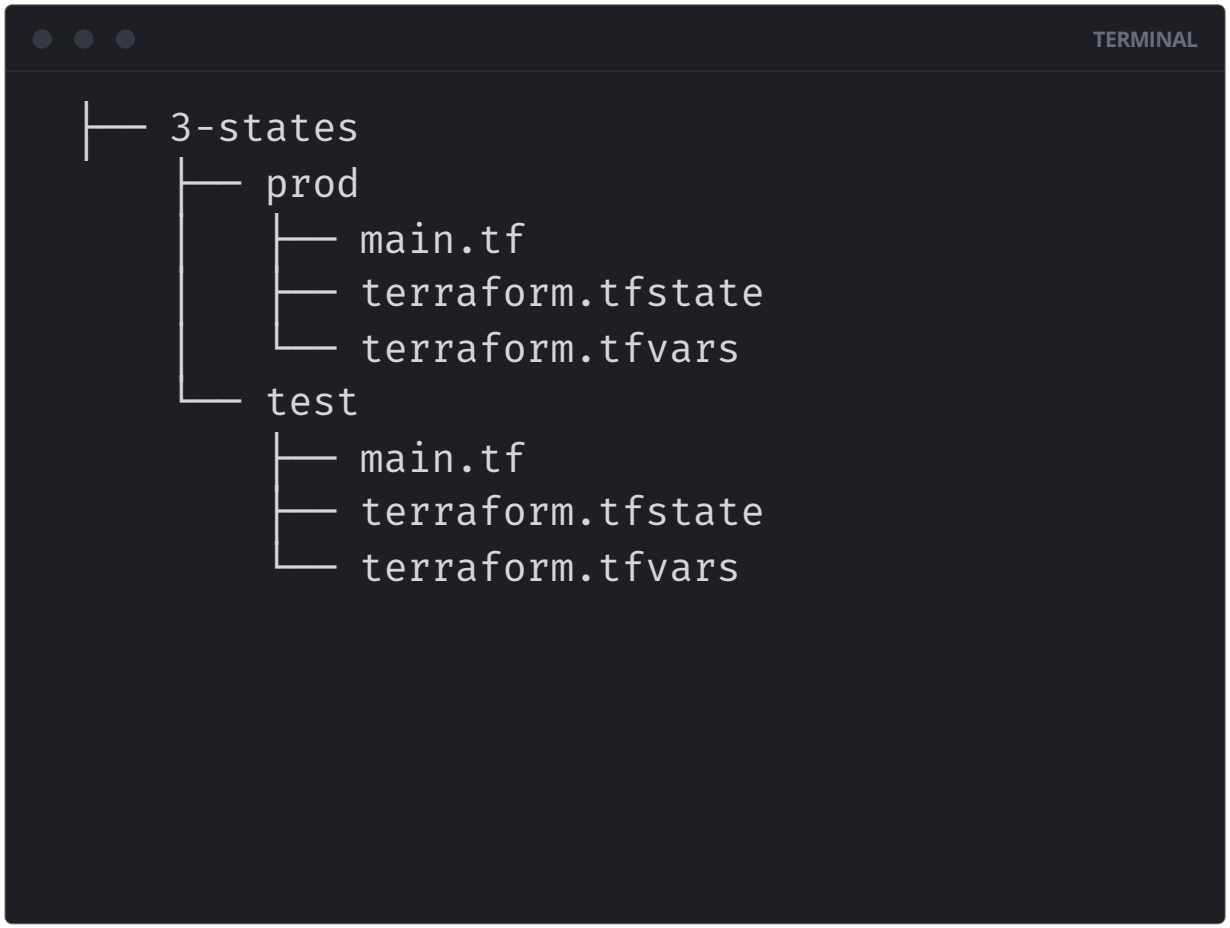
Lowest barrier to entry for testing changes.

# Separate States

Shrinks “blast radius” of changes between infrastructures.

Control timing and release schedule of provisioning

Provides ability to isolate resources and secrets.



# Resources in Modules

Improves long term maintenance

Eases reuse

Encourages consistency in design

TERMINAL

```
── 4-modules
    ├── envs
    │   ├── prod
    │   │   ├── main.tf
    │   │   ├── terraform.tfstate
    │   │   └── terraform.tfvars
    │   └── test
    │       ├── main.tf
    │       ├── terraform.tfstate
    │       └── terraform.tfvars
    └── modules
        ├── core
        │   ├── input.tf
        │   ├── main.tf
        │   └── output.tf
        ├── db
        ├── network
        └── webapp
```

# Nested Modules

Groups functionality of modules into logical slices.

Allows hierarchical organization of Terraform code.

Provides the ability to effectively namespace modules.

Allows using broad or concise pieces of larger modules.

TERMINAL

```
── 5-nested
    ├── envs
    │   ├── prod
    │   │   ├── main.tf
    │   │   ├── terraform.tfstate
    │   │   └── terraform.tfvars
    │   └── test
    │       ├── main.tf
    │       ├── terraform.tfstate
    │       └── terraform.tfvars
    └── modules
        ├── common
        │   └── aws
        │       ├── compute
        │       │   ├── db
        │       │   └── web
        │       └── network
        │           ├── priv_subnet
        │           ├── pub_subnet
        │           └── vpc
        └── project
            ├── core
            └── webapp
```

# State/Repo Per Component

Full isolation

Control team read/write permissions on each repo

Full control over access to state outputs

All apply actions are related to only the component.

TERMINAL

```
── 6-repo-isolation
    ├── envs
    │   ├── prod
    │   └── test
    │       ├── core
    │       │   └── terraform.tfstate
    │       ├── db
    │       │   └── terraform.tfstate
    │       ├── network
    │       │   └── terraform.tfstate
    │       └── webapp
    │           └── terraform.tfstate
    └── modules
        ├── common
        │   └── aws
        │       ├── compute
        │       │   ├── db
        │       │   └── web
        │       └── network
        │           ├── priv_subnet
        │           ├── pub_subnet
        │           └── vpc
        └── project
            ├── core
            └── webapp
```

# Discussion Conclusions

- *Is this config owned by one team?*
  - *The code should live in its own repository.*
- *Is this config used by more than one team?*
  - *Break code down into modules where possible.*
- *Does this config create resources?*
  - *Publish outputs to expose useful resource attributes.*

# Challenge Discussion

Challenge



Overview



Questions?

**How would you architect a project?**

Consider a larger project where you need to create compute instances, some networking, and some storage.

How would you divide the project into smaller configurations?

What values would need to be emitted by each configuration?

What values would need to be read by each?

# Challenge Discussion

Challenge

**How Terraform projects evolve**

**When to use modules vs a new project**

**Overview**

**How to use shared state to manage complexity**

Questions?

# Challenge Discussion

Challenge

Overview

**Questions?**

**How could you foresee these concerns impacting your environment?**

**Take a few minutes to discuss with your neighbor.**

Terraform 201

# Terraform Cloud Basics

# What You'll Learn

- Create a repository at GitHub
- Push your code to GitHub
- Create an organization at Terraform Cloud
- Connect to GitHub via OAuth
- Add a workspace that references a GitHub repo
- Commit changes and create infrastructure

# Terraform Cloud Features

Runs Terraform plan / apply

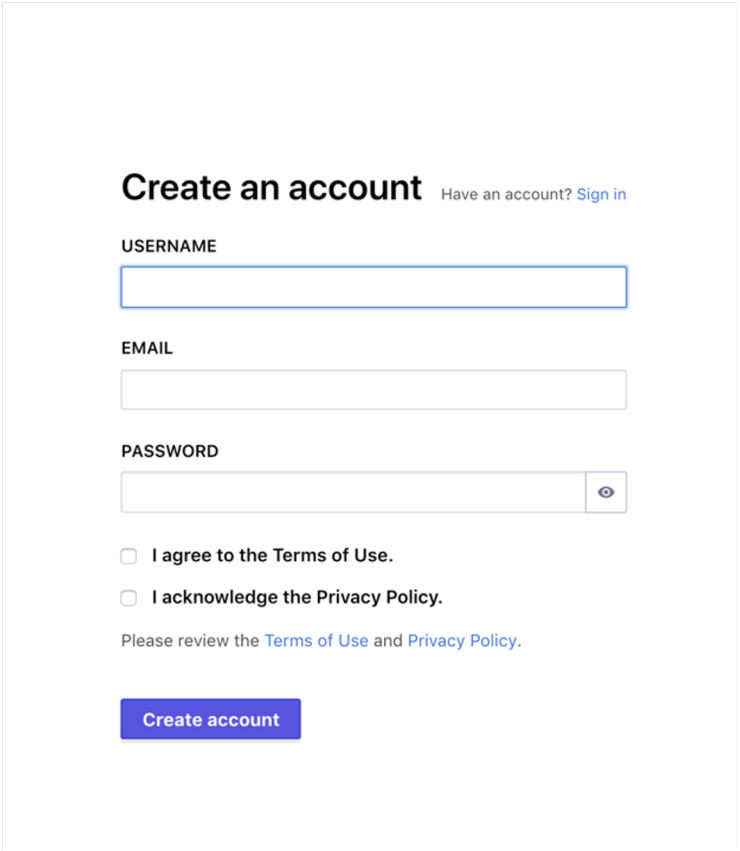
System of record for Terraform

Collaboration and Governance

VCS integration

Private install or hosted

# https://app.terraform.io/signup/account



# Create a new Workspace

Workspaces allow you to organize infrastructure and collaborate on Terraform runs.

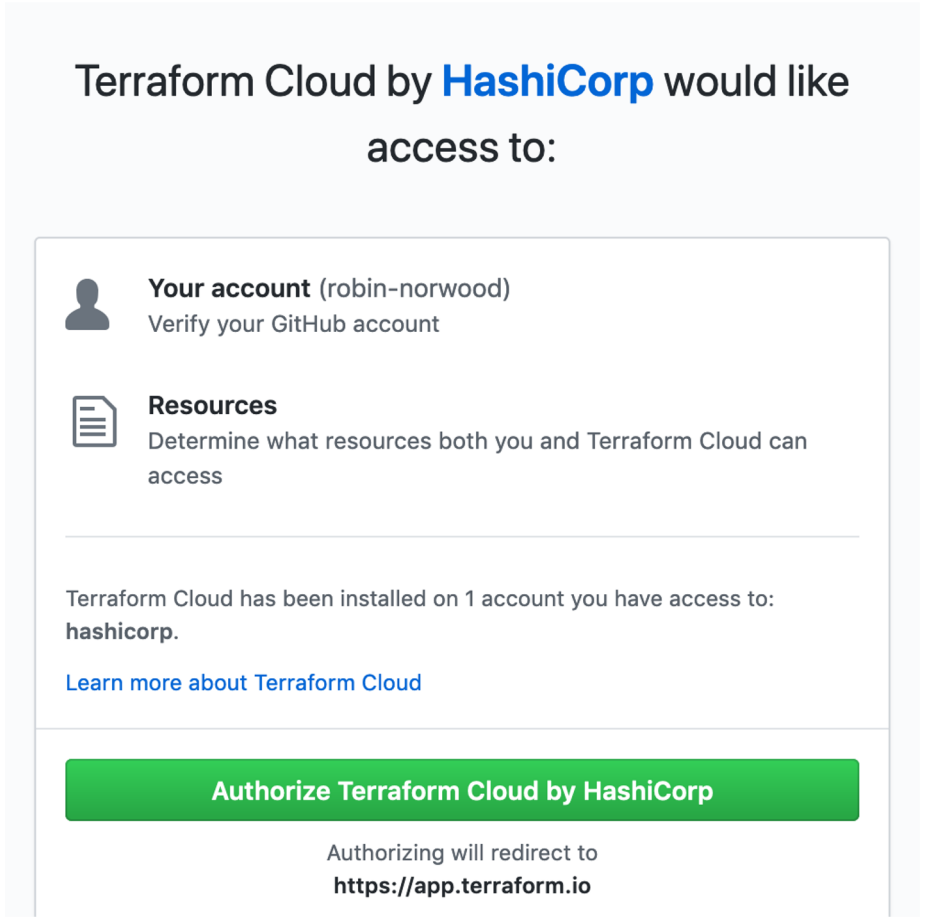
1 Connect to VCS  2 Choose a repository  3 Configure settings

## Connect to a version control provider

Choose the version control provider that hosts your Terraform source code. If you're planning to upload your configurations via the API, you can skip this step.

GitHub  GitLab  Bitbucket

New to Terraform? Take a tutorial


Terraform Cloud by HashiCorp would like access to:
Your account (robin-norwood)
Verify your GitHub account
Resources
Determine what resources both you and Terraform Cloud can access
Terraform Cloud has been installed on 1 account you have access to: hashicorp.
Learn more about Terraform Cloud
Authorize Terraform Cloud by HashiCorp
Authorizing will redirect to
https://app.terraform.io

# Create a new Workspace

Workspaces allow you to organize infrastructure and collaborate on Terraform runs.

Connect to VCS | 2 Choose a repository | 3 Configure settings

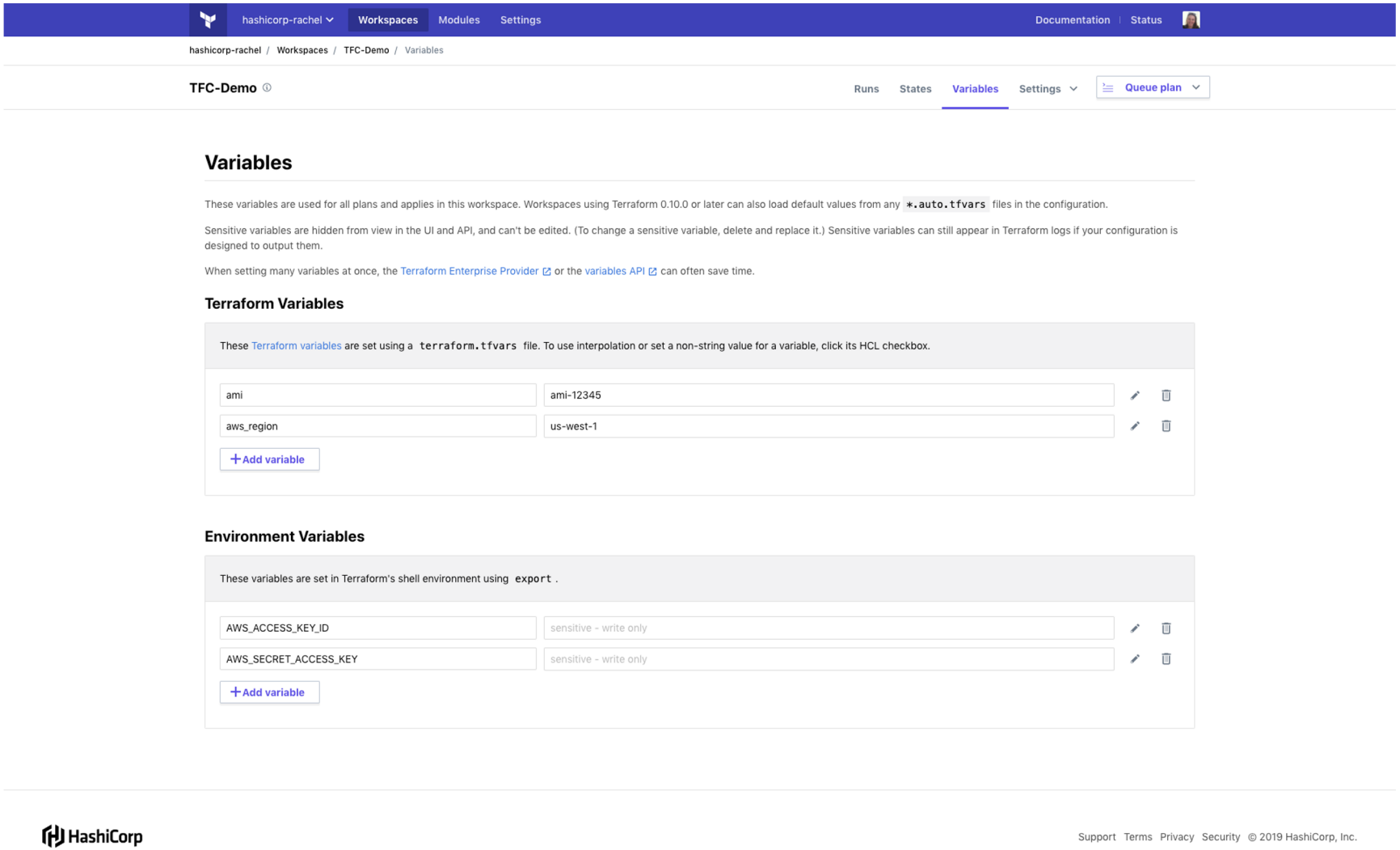
## Choose a repository

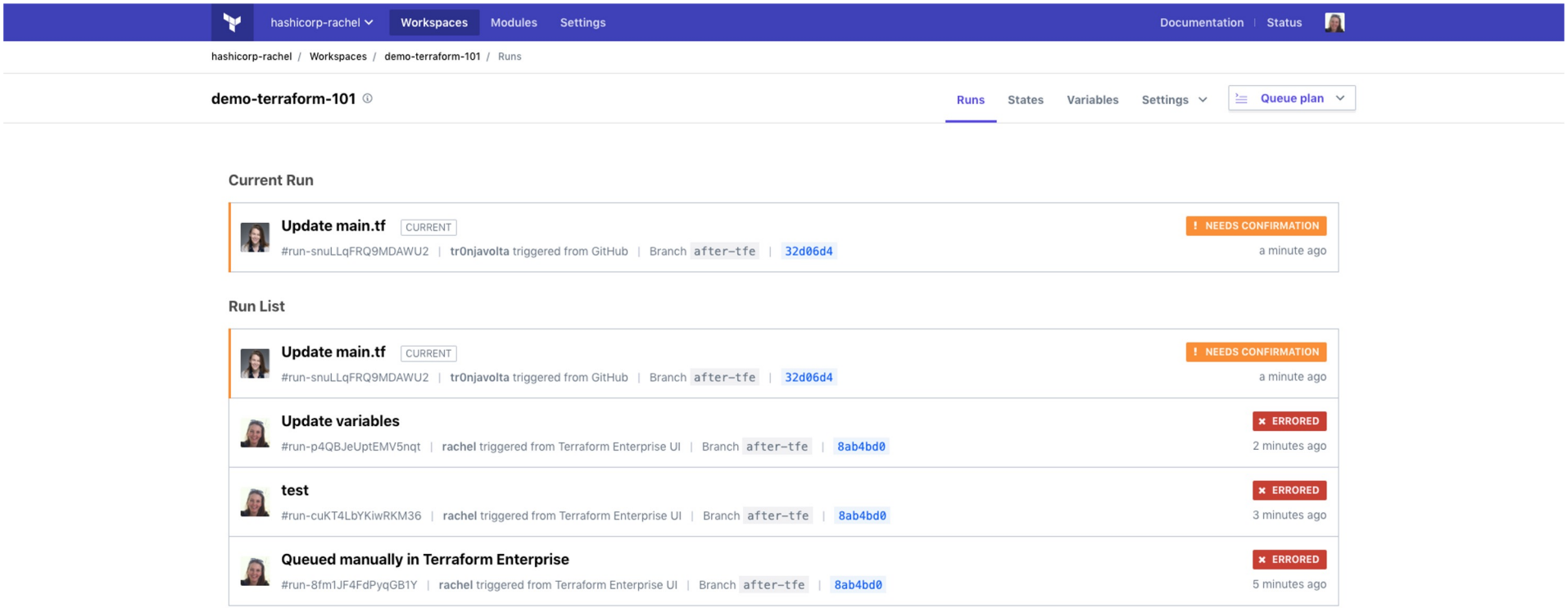
Choose the repository that hosts your Terraform source code. We'll watch this for commits and pull requests.

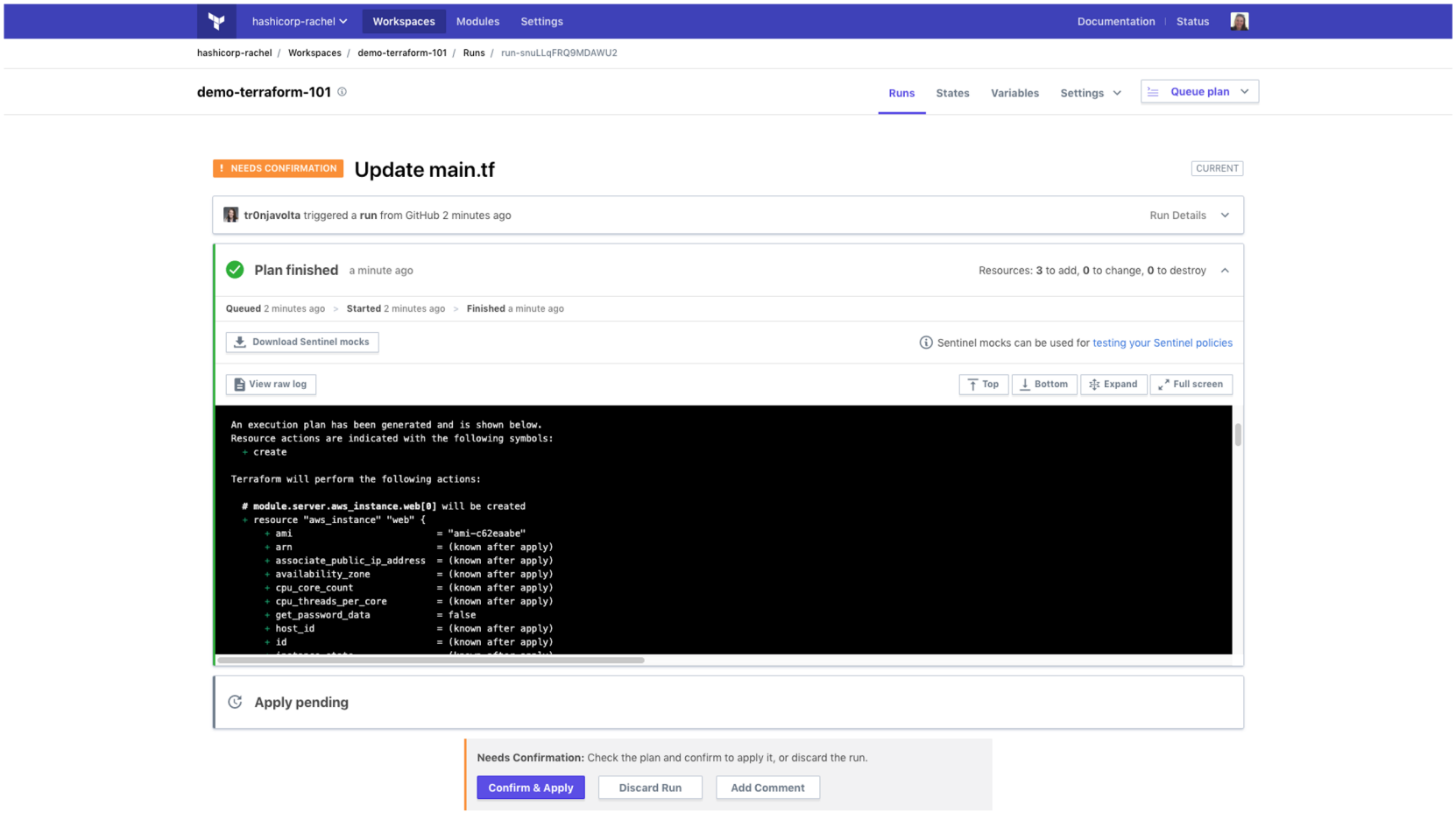
## Configuration uploaded successfully

Your configuration has been uploaded. Next, you probably want to configure variables (such as access keys or configuration values). If your configuration doesn't require variables, you can queue your first plan now.

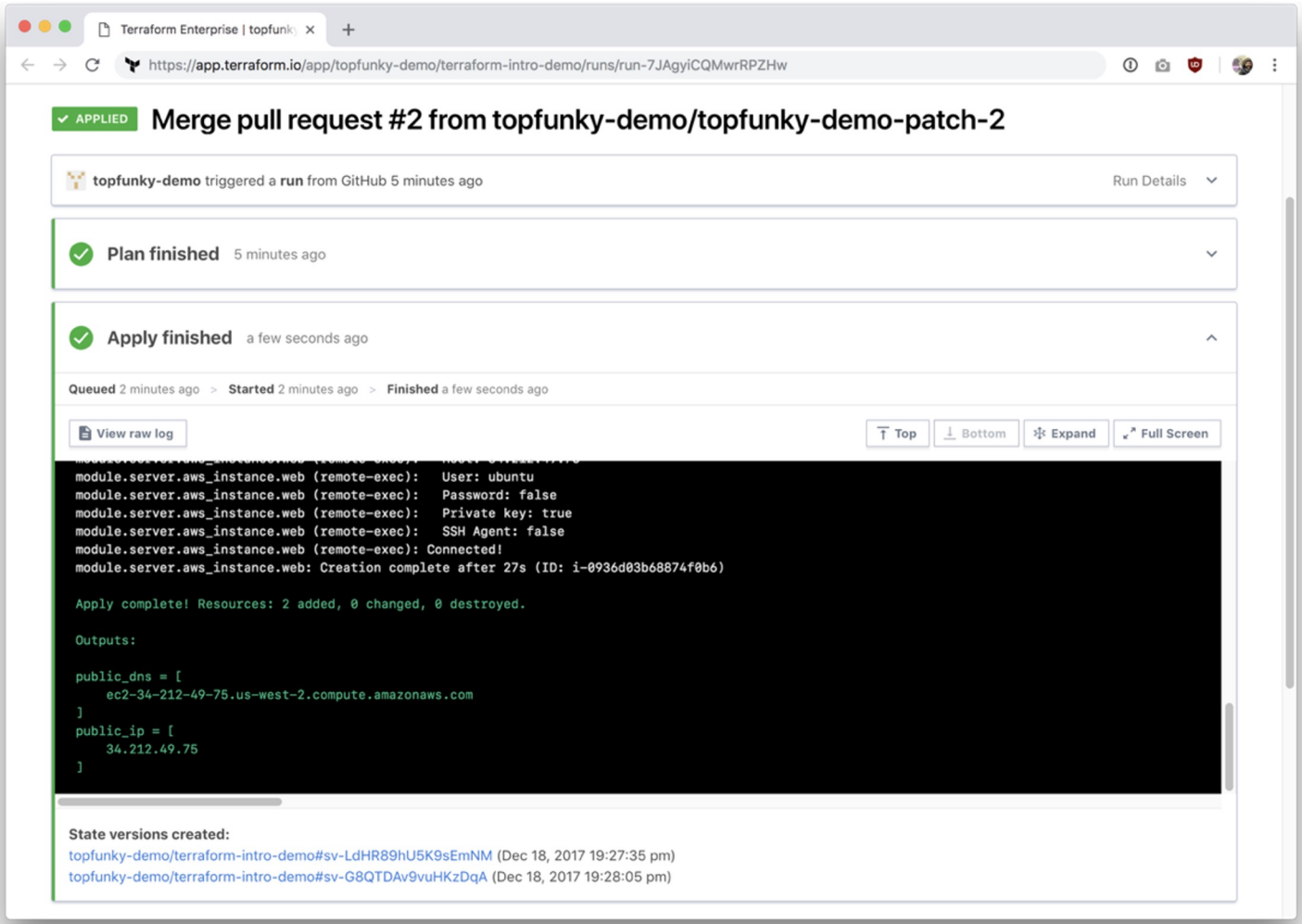
Configure variables | Queue plan


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Workspaces / TFC-Demo / Variables
TFC-Demo
Runs
States
Variables
Settings
Queue plan
Variables
These variables are used for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any *.auto.tfvars files in the configuration.
Sensitive variables are hidden from view in the UI and API, and can't be edited. (To change a sensitive variable, delete and replace it.) Sensitive variables can still appear in Terraform logs if your configuration is designed to output them.
When setting many variables at once, the Terraform Enterprise Provider or the variables API can often save time.
Terraform Variables
These Terraform variables are set using a terraform.tfvars file. To use interpolation or set a non-string value for a variable, click its HCL checkbox.
ami
ami-12345
aws_region
us-west-1
Add variable
Environment Variables
These variables are set in Terraform's shell environment using export .
AWS_ACCESS_KEY_ID
sensitive - write only
AWS_SECRET_ACCESS_KEY
sensitive - write only
Add variable
HashiCorp
Support Terms Privacy Security © 2019 HashiCorp, Inc.






Terraform Enterprise | topfunk
https://app.terraform.io/app/topfunky-demo/terraform-intro-demo/runs/run-7JAgyiCQMwrRPZHw
APPLIED
Merge pull request #2 from topfunky-demo/topfunky-demo-patch-2
topfunky-demo triggered a run from GitHub 5 minutes ago
Run Details
Plan finished 5 minutes ago
Apply finished a few seconds ago
Queued 2 minutes ago > Started 2 minutes ago > Finished a few seconds ago
View raw log
Top
Bottom
Expand
Full Screen
module.server.aws_instance.web (remote-exec): User: ubuntu
module.server.aws_instance.web (remote-exec): Password: false
module.server.aws_instance.web (remote-exec): Private key: true
module.server.aws_instance.web (remote-exec): SSH Agent: false
module.server.aws_instance.web (remote-exec): Connected!
module.server.aws_instance.web: Creation complete after 27s (ID: i-0936d03b68874f0b6)
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
Outputs:
public_dns = [
ec2-34-212-49-75.us-west-2.compute.amazonaws.com
]
public_ip = [
34.212.49.75
]
State versions created:
topfunky-demo/terraform-intro-demo#sv-LdHR89hU5K9sEmNM (Dec 18, 2017 19:27:35 pm)
topfunky-demo/terraform-intro-demo#sv-G8QTDAv9vuHKzDqA (Dec 18, 2017 19:28:05 pm)

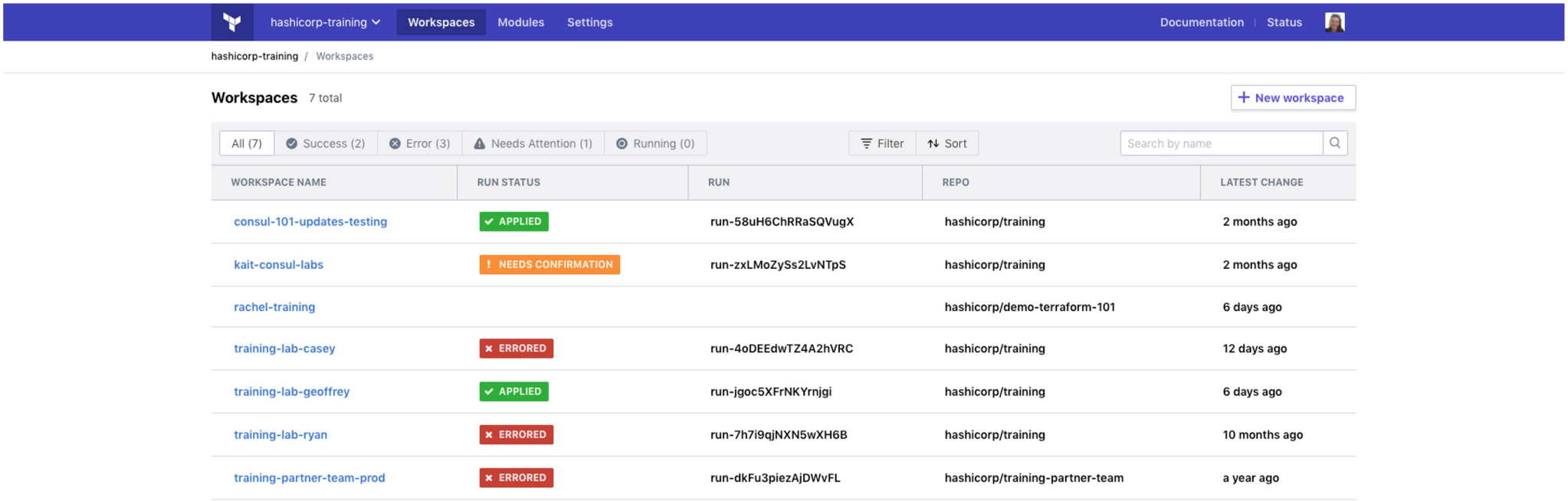
Terraform 201

# Collaborating with Terraform Cloud

# What You'll Learn

- Understand the role of Terraform Cloud in modern continuous provisioning workflows

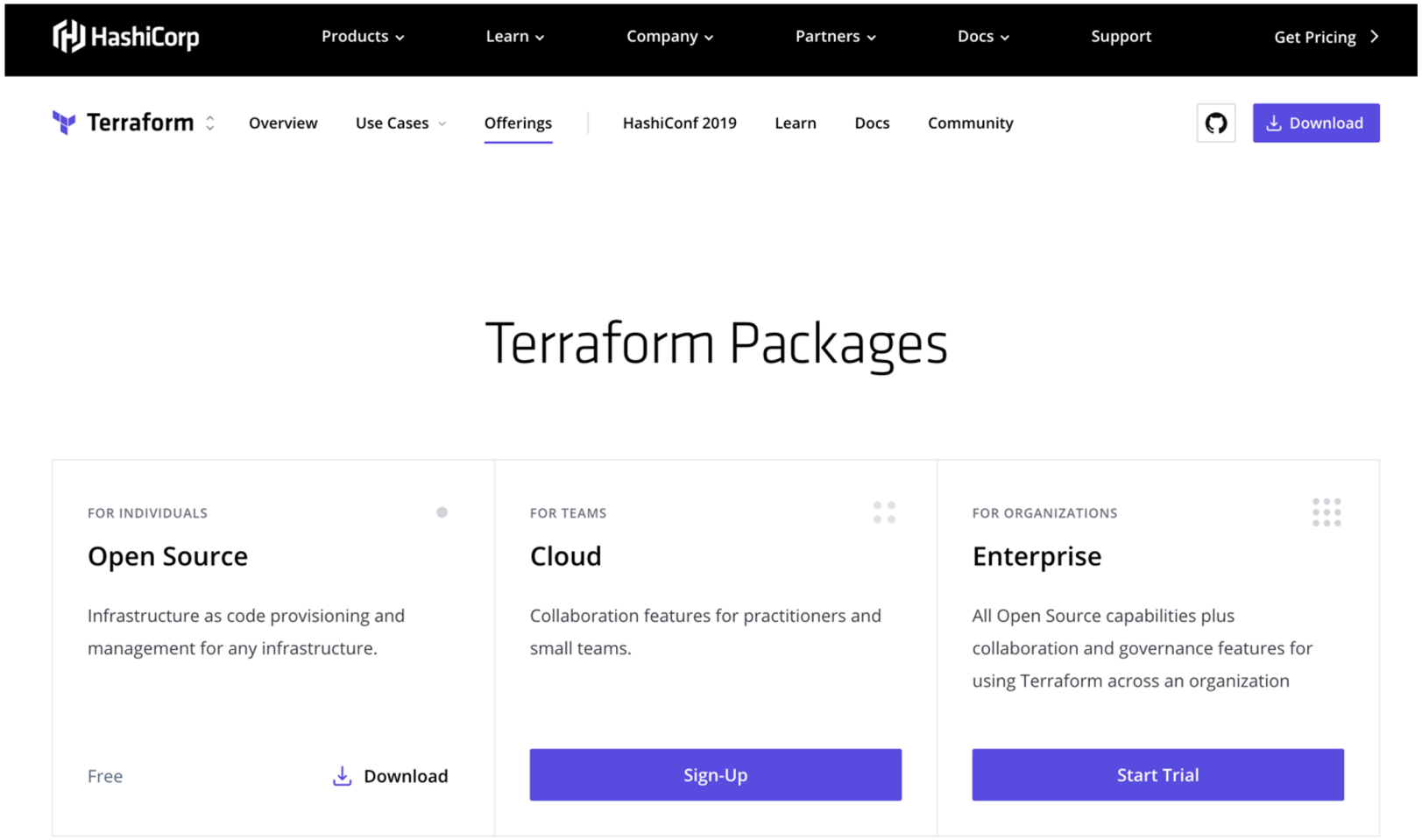
- Understand the components of Terraform Cloud: users, organizations, teams, workspaces



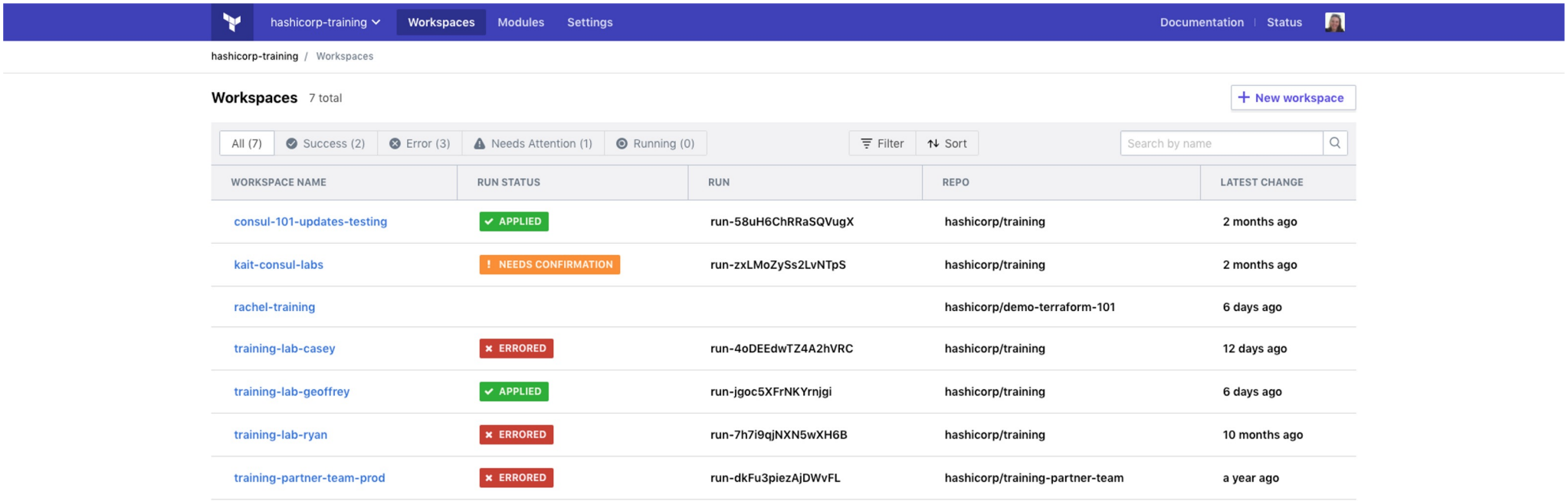
# What Is Terraform Cloud?

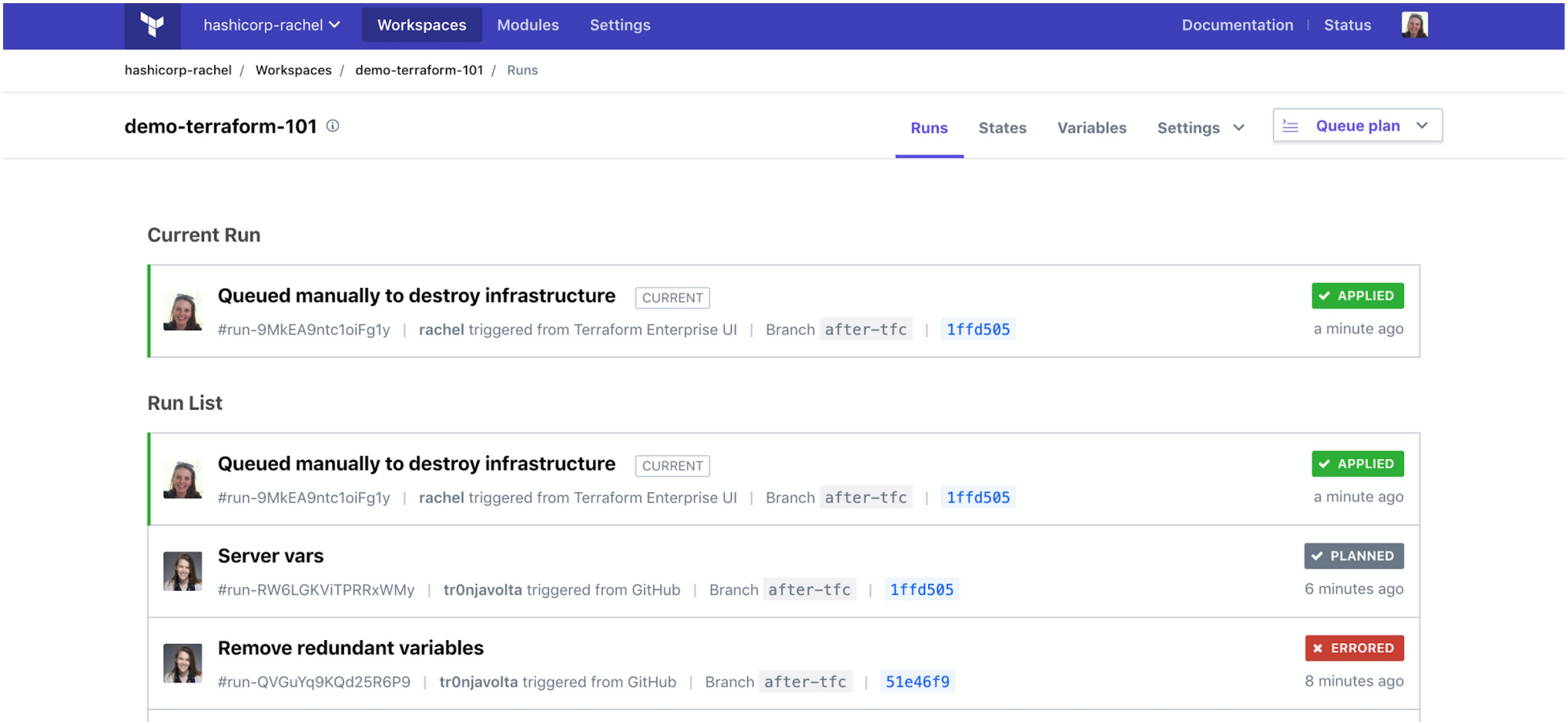
- Platform for collaboration and governance with Terraform
- Central module registry
- Connection to version control for Terraform configuration code
- Runs Terraform plan and apply

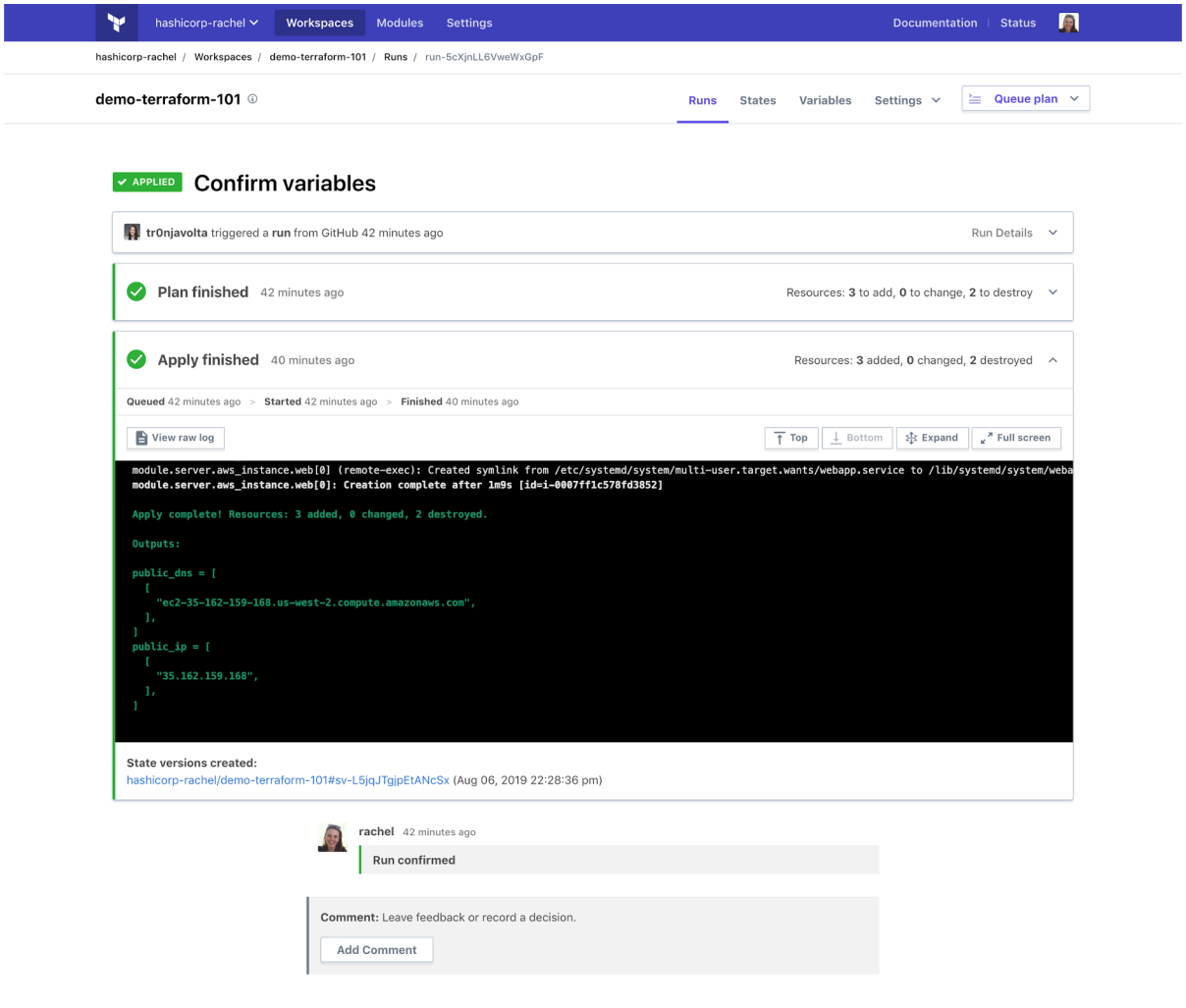
- Log of provisioning actions
- Stores state securely
- Shares state within organizations
- Configurable permissions for user and team access

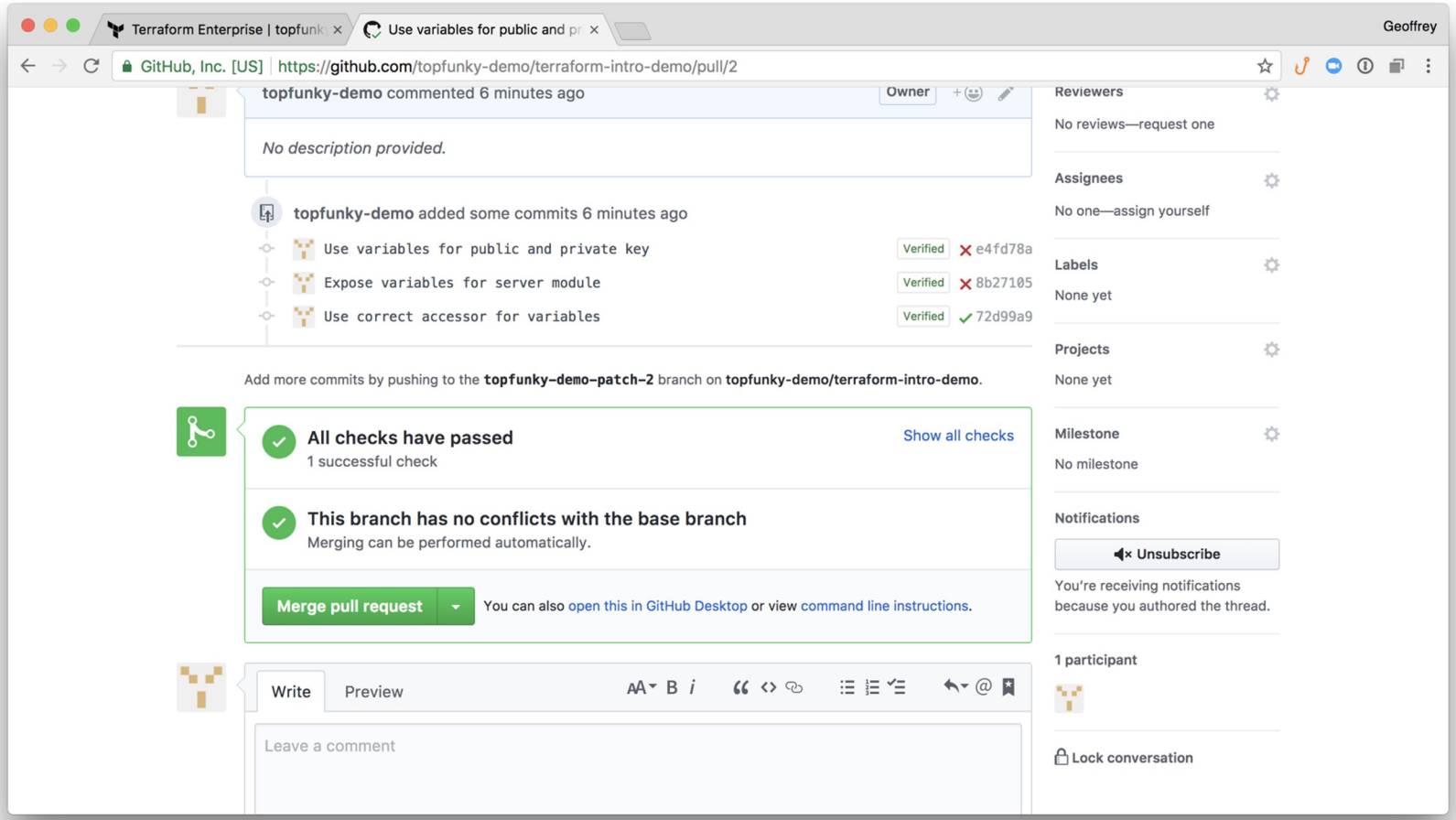



https://www.hashicorp.com/products/terraform/offerings




hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Workspaces / demo-terraform-101 / Runs
demo-terraform-101
Runs
States
Variables
Settings
Queue plan
Current Run
Queued manually to destroy infrastructure
CURRENT
#run-9MkEA9ntc1oiFg1y | rachel triggered from Terraform Enterprise UI | Branch after-tfc | 1ffd505
APPLIED
a minute ago
Run List
Queued manually to destroy infrastructure
CURRENT
#run-9MkEA9ntc1oiFg1y | rachel triggered from Terraform Enterprise UI | Branch after-tfc | 1ffd505
APPLIED
a minute ago
Server vars
#run-RW6LGKViTPRRxWMy | tr0njavolta triggered from GitHub | Branch after-tfc | 1ffd505
PLANNED
6 minutes ago
Remove redundant variables
#run-QVGuYq9KQd25R6P9 | tr0njavolta triggered from GitHub | Branch after-tfc | 51e46f9
ERRORED
8 minutes ago


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Workspaces / demo-terraform-101 / Runs / run-5cXjnLL6VweWxGpF
demo-terraform-101
Runs
States
Variables
Settings
Queue plan
APPLIED
Confirm variables
tr0njavolta triggered a run from GitHub 42 minutes ago
Run Details
Plan finished 42 minutes ago
Resources: 3 to add, 0 to change, 2 to destroy
Apply finished 40 minutes ago
Resources: 3 added, 0 changed, 2 destroyed
Queued 42 minutes ago > Started 42 minutes ago > Finished 40 minutes ago
View raw log
Top
Bottom
Expand
Full screen
module.server.aws_instance.web[0] (remote-exec): Created symlink from /etc/systemd/system/multi-user.target.wants/webapp.service to /lib/systemd/system/weba
module.server.aws_instance.web[0]: Creation complete after 1m9s [id=i-0007ff1c578fd3852]
Apply complete! Resources: 3 added, 0 changed, 2 destroyed.
Outputs:
public_dns = [
[
"ec2-35-162-159-168.us-west-2.compute.amazonaws.com",
],
]
public_ip = [
[
"35.162.159.168",
],
]
State versions created:
hashicorp-rachel/demo-terraform-101#sv-L5jqJTgjpEtANcSx (Aug 06, 2019 22:28:36 pm)
rachel 42 minutes ago
Run confirmed
Comment: Leave feedback or record a decision.
Add Comment
HashiCorp
Support Terms Privacy Security © 2019 HashiCorp, Inc.


Terraform Enterprise | topfunk
Use variables for public and pr
Geoffrey
GitHub, Inc. [US] https://github.com/topfunky-demo/terraform-intro-demo/pull/2
topfunky-demo commented 6 minutes ago
Owner
No description provided.
topfunky-demo added some commits 6 minutes ago
Use variables for public and private key
Verified
e4fd78a
Expose variables for server module
Verified
8b27105
Use correct accessor for variables
Verified
72d99a9
Add more commits by pushing to the topfunky-demo-patch-2 branch on topfunky-demo/terraform-intro-demo.
All checks have passed
1 successful check
Show all checks
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Reviewers
No reviews—request one
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Notifications
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant
Lock conversation

# User

Individual account on app.terraform.io

Can be a member of multiple teams and organizations

Signup is global at:

https://app.terraform.io/signup

Usernames are global within app.terraform.io.

# Create an account

Have an account? Sign in

USERNAME

EMAIL

PASSWORD

- [ ] I agree to the Terms of Use.
- [ ] I acknowledge the Privacy Policy.

Please review the Terms of Use and Privacy Policy.

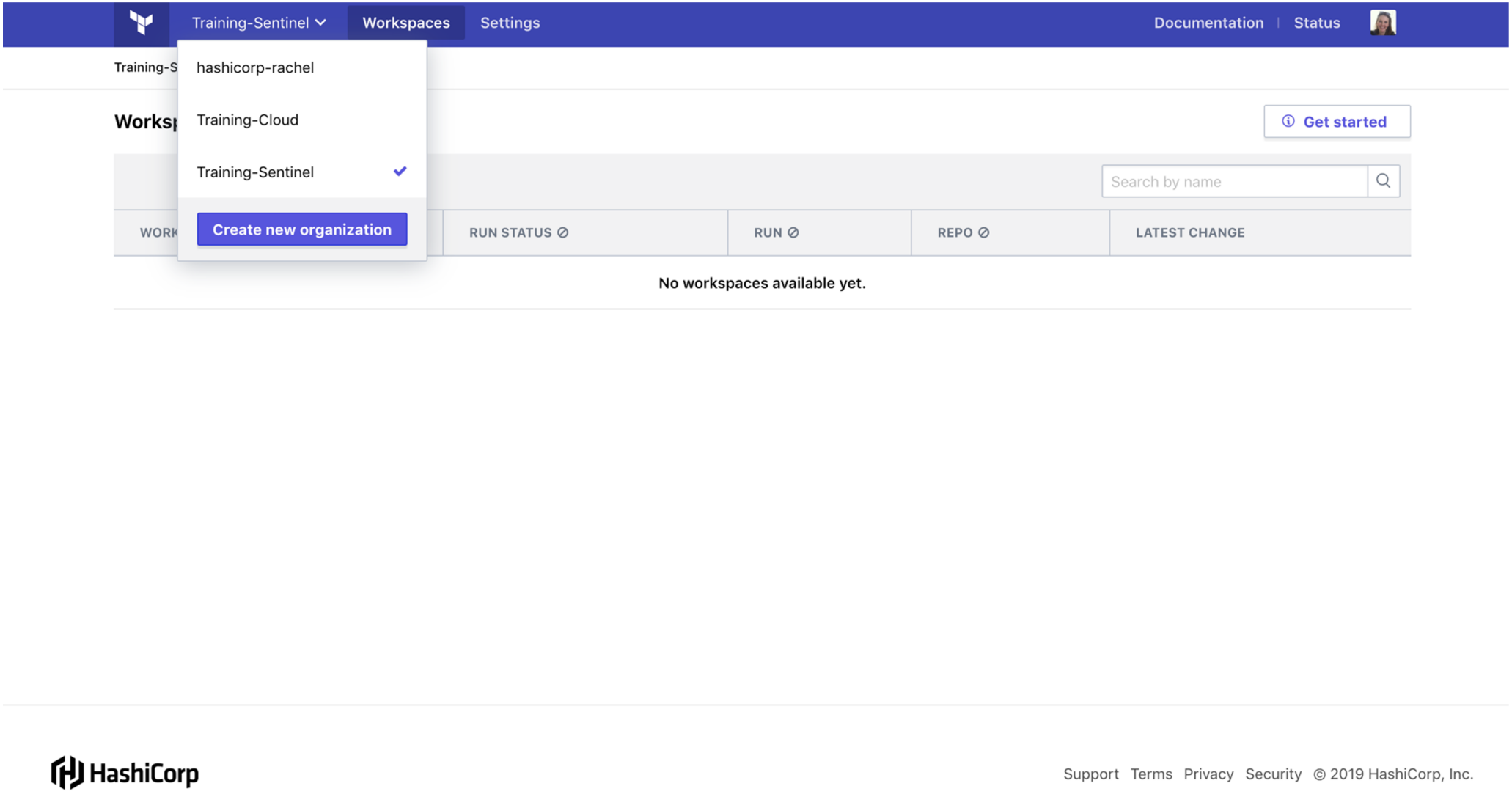
Create account

# Organization

Shared spaces for teams to collaborate on workspaces, policies, and Terraform modules.

Organizations can have many teams.

Sentinel policies are defined and enforced at the organization level across workspaces.



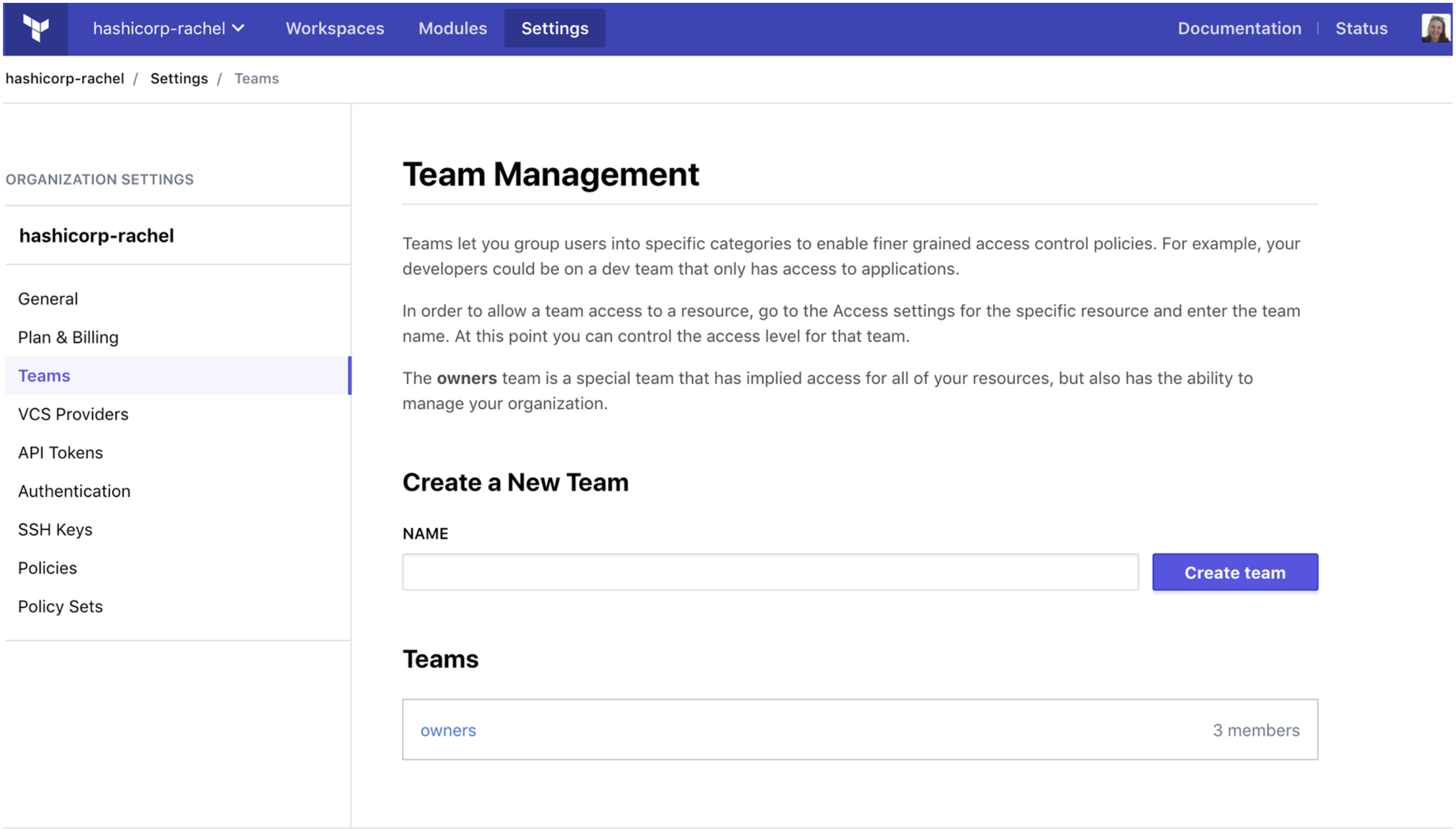
# Team

Groups of users that reflect your company's organization or project structure.

Teams can be granted read, plan, write, and admin permissions to workspaces.

Teams are created under the organization settings.

Permissions are configured under workspace access settings.


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Settings / Teams
ORGANIZATION SETTINGS
hashicorp-rachel
General
Plan & Billing
Teams
VCS Providers
API Tokens
Authentication
SSH Keys
Policies
Policy Sets
Team Management
Teams let you group users into specific categories to enable finer grained access control policies. For example, your developers could be on a dev team that only has access to applications.
In order to allow a team access to a resource, go to the Access settings for the specific resource and enter the team name. At this point you can control the access level for that team.
The owners team is a special team that has implied access for all of your resources, but also has the ability to manage your organization.
Create a New Team
NAME
Create team
Teams
owners
3 members

# Workspaces

Workspaces are how Terraform Cloud tracks infrastructure. Workspaces contain:

- Linked VCS repository with Terraform code.
- Variables and values used by the configuration.
- Current Terraform state file.
- Historical states and run logs.

# Workspace Approaches

One approach is to have one workspace per environment per config, named accordingly.

- billing-app-dev
- billing-app-stage
- billing-app-prod

- networking-dev
- networking-stage
- networking-prod

# Alternate Workspace Approaches

Or, you could create a separate organization for development, test, or staging environments.

Organizations cannot share state, so this keeps them isolated.

This also allows modifying Terraform modules without immediately impacting production workspaces.

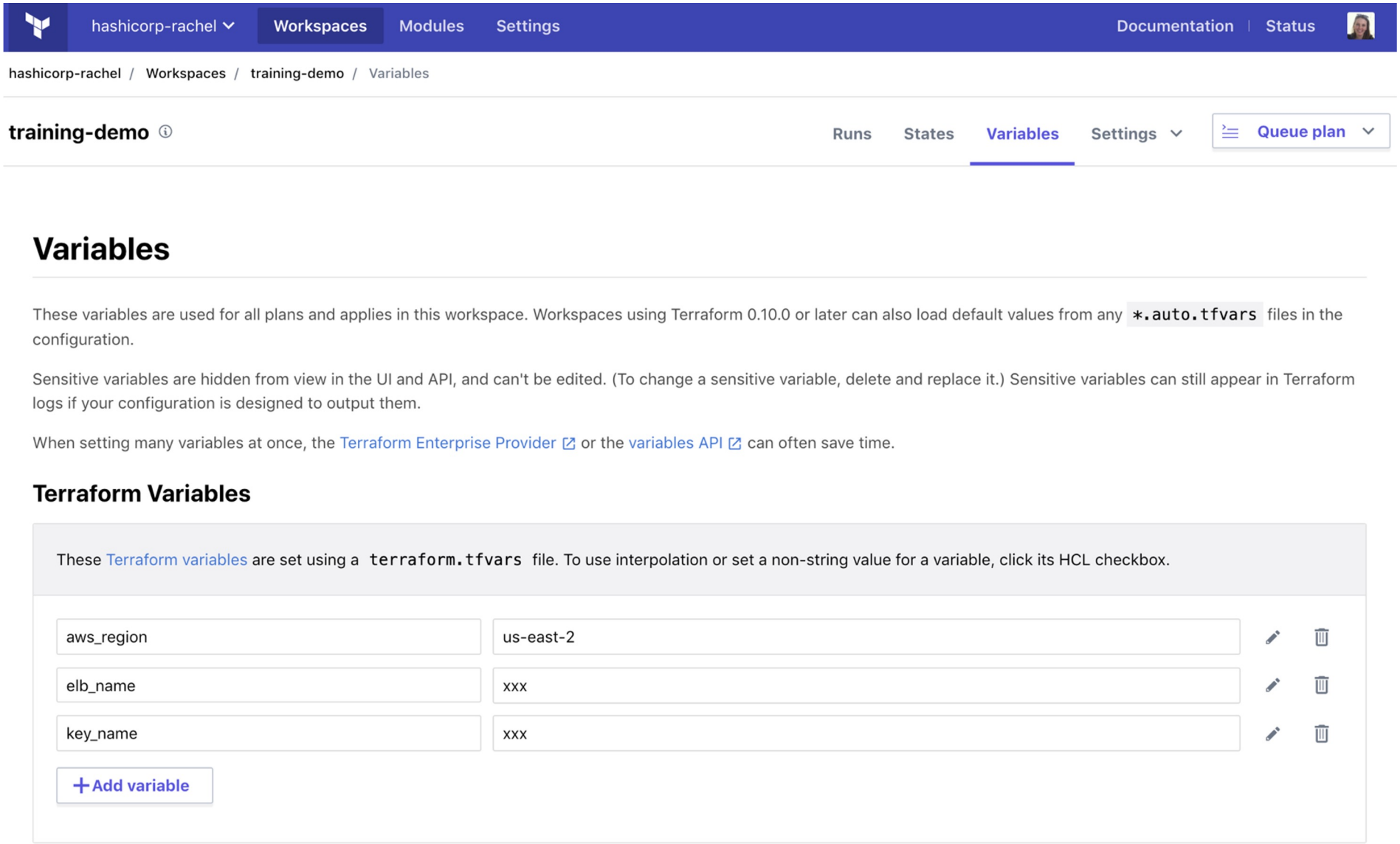
# Variables

Terraform Cloud allows configuring two types of variables within each workspace:

- Environment variables
- Terraform variables (contents of terraform.tfvars files)

Variables can be marked as *sensitive*, in which case they will not be displayed in the UI on subsequent visits.

Variables can be marked as HCL, allowing the user to define structured and typed values (maps, lists, etc.).


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Workspaces / training-demo / Variables
training-demo
Runs
States
Variables
Settings
Queue plan
Variables
These variables are used for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any *.auto.tfvars files in the configuration.
Sensitive variables are hidden from view in the UI and API, and can't be edited. (To change a sensitive variable, delete and replace it.) Sensitive variables can still appear in Terraform logs if your configuration is designed to output them.
When setting many variables at once, the Terraform Enterprise Provider or the variables API can often save time.
Terraform Variables
These Terraform variables are set using a terraform.tfvars file. To use interpolation or set a non-string value for a variable, click its HCL checkbox.
aws_region
us-east-2
elb_name
xxx
key_name
xxx
+ Add variable

# How your team works

Terraform Cloud can be used as a part of your team's existing workflows, or you can evolve your team workflows to take advantage of Terraform Cloud features.

- **Manual:** Queue runs on demand
- **Semi-auto:** plan runs on VCS commit
- **Continuous Infrastructure:** apply on VCS commit

# API

Terraform Cloud provides a comprehensive API. Many companies drive workflows entirely through the API.

Due to the uniqueness of applications and the depth of features offered, we won't discuss it here.

See the documentation for more details:

https://www.terraform.io/docs/cloud/api/index.html

# API Tokens

Three kinds of API tokens are offered: organization, team, or user.

Accessing state from outside Terraform Cloud requires a user token.

Team tokens are useful for triggering plans and applies from external CI/CD tools.

Organization tokens only modify teams and workspaces. They cannot trigger plan or apply.

# Backend Configuration

CODE EDITOR

```
data "terraform_remote_state" "vpc" {
 backend = "remote"
 config {
   organization = "company"

   workspaces = {
     name = "workspace"
   }
 }
}
# Elsewhere, use outputs from the state
data.terraform_remote_state.vpc.outputs.cidr_block
```

# Discussion

Overview

Questions?

**Understand the role of Terraform Cloud in modern continuous provisioning workflows**

**Understand the components of Terraform Cloud: users, organizations, teams, workspaces**

# Discussion

**What is the UI workflow?**



Overview

**Take a few minutes to discuss with your neighbor.**

**Questions?**

Terraform 201

# Sentinel

# What You'll Learn

Why Sentinel?

Syntax

Data structure

Debugging

Common use cases

Workflow + API

Testing/Mocking

Deployment

Architecture

# Why Write Policy as Code with Sentinel?

Codification

Version Control

Testing

Automation

# What Sentinel Does

Policy as code

Run between `plan` and `apply`

Analyzes current state plus values that can be calculated before `apply`

Analyzes state of configuration

Analyzes state of system at deploy time

Multi-cloud, multi-provider

Augments your deployment security strategy

# What Sentinel Does Not Do

Continuous security checks

Audit previously deployed systems

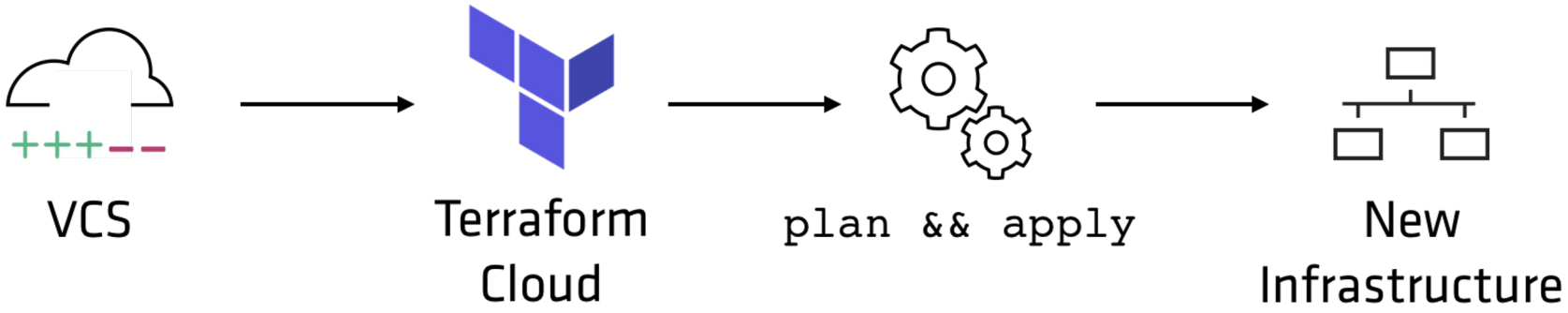
Analyze executable contents of a VM/container/etc.

Limit runtime actions of deployed applications

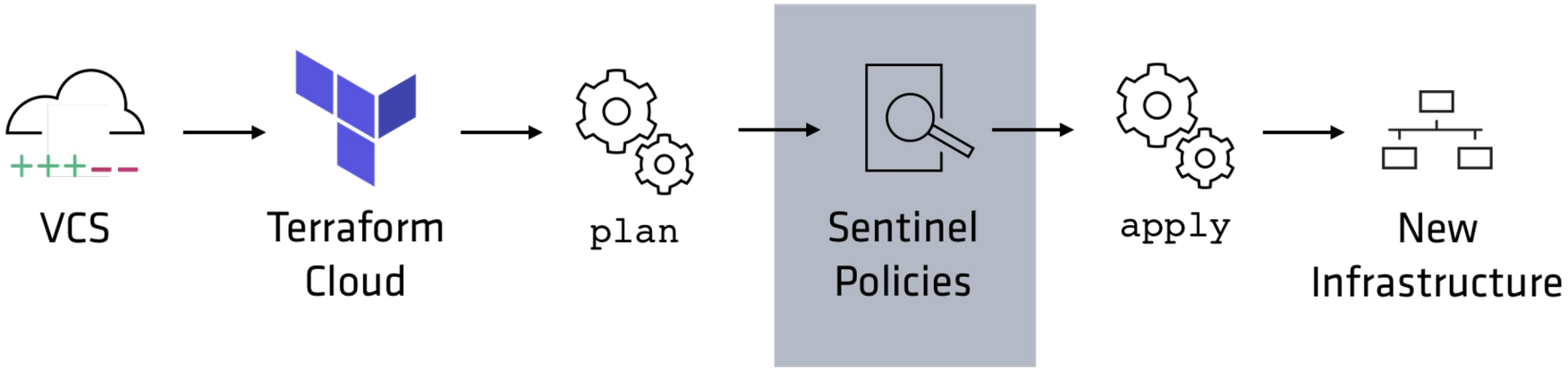
*In its current implementation, Sentinel is a tool for preventing mistakes by operators acting in good faith.*

*– Martin Atkins (Terraform Tech Lead)*

# Without Sentinel



# With Sentinel



# Become Familiar with all Sentinel Documentation

Main Sentinel docs:

https://docs.hashicorp.com/sentinel/

Terraform Enterprise docs:

https://www.terraform.io/docs/cloud/sentinel/index.html

Blog articles:

https://www.hashicorp.com/blog/category/sentinel

Learn Platform:

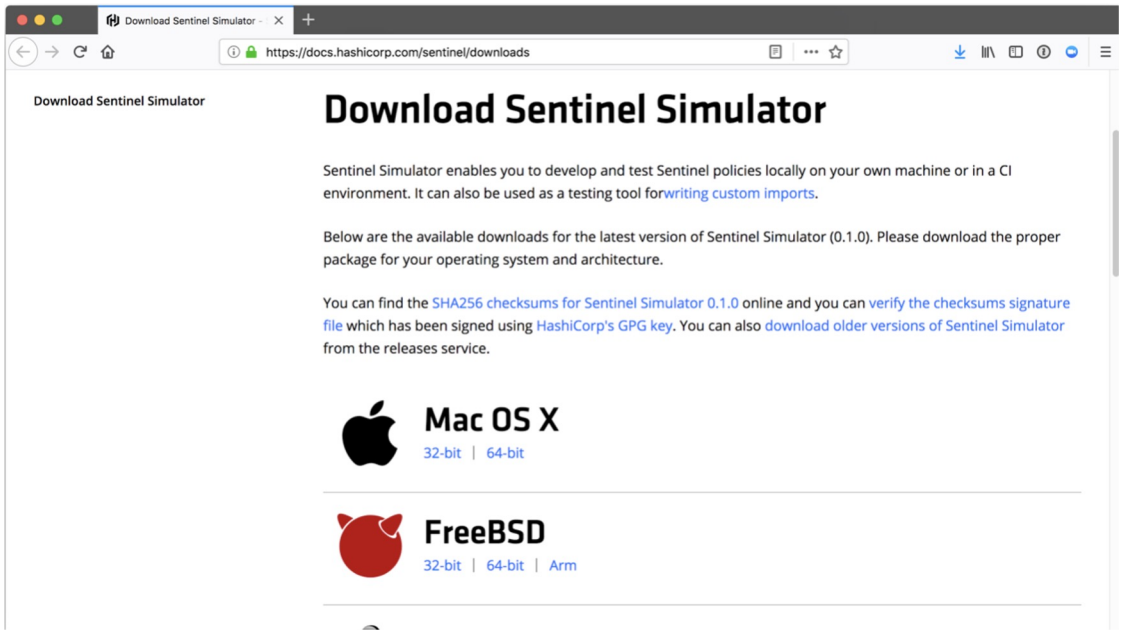
https://learn.hashicorp.com/terraform?track=sentinel#sentinel

# Your First Policy

**Run Task 1 in Sentinel Lab**

- Download Sentinel Simulator, copy to $PATH
- Try a simple policy named `simple.sentinel`:

```
main = rule { true }
```



Terraform 201 - Sentinel

# Syntax

# Syntax - Introduction

- Sentinel starts with a rule named “main".
- You can define many rules with any name, but “main” is the rule which Sentinel executes first.
- The boolean return value of this rule determines the pass/fail status of the policy.

CODE EDITOR

```
main = rule { true }
```

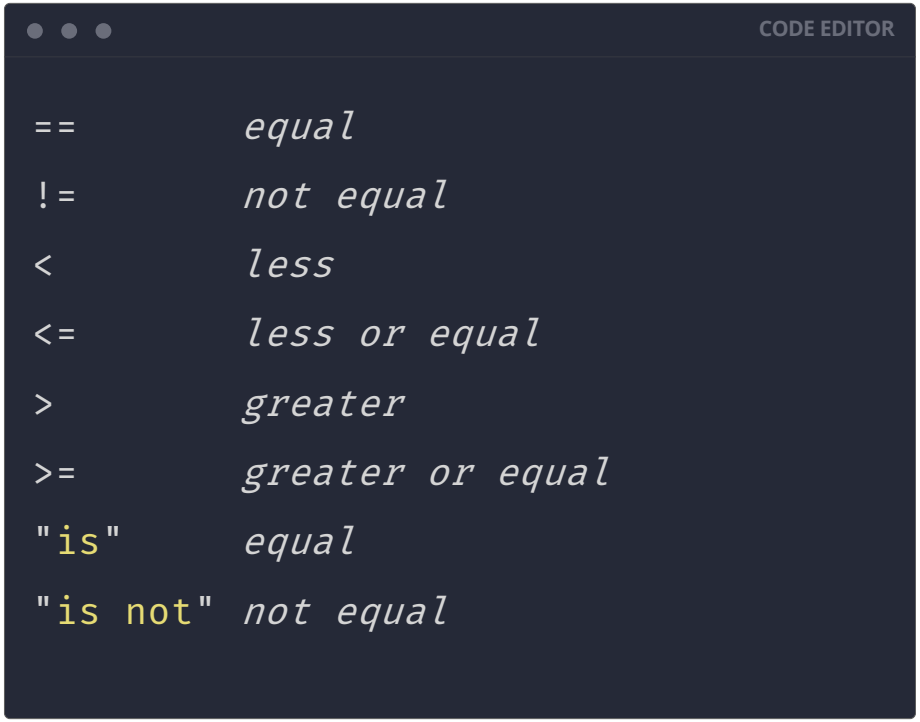
# Syntax - Conditionals

- Rules can also be guarded by conditionals, which is important when writing a policy against a diverse set of infrastructure resources.
- If a conditional returns false, the rule body is not evaluated, and the result is always “true”.

CODE EDITOR

```
# Conditional rule
# Passes if is_docker is false (rule doesn't apply)
# Otherwise evaluates has_exposed_ports
main = rule when is_docker { has_exposed_ports }
```

# Syntax - Comparators



# Syntax - Boolean Values

| Condition: | Evaluates to: |
|---|---|
| undefined or true | true |
| undefined or false | undefined |
| undefined or undefined | undefined |
| undefined and true | undefined |
| undefined and false | undefined |
| undefined and undefined | undefined |
| undefined xor true | undefined |
| undefined xor false | undefined |
| undefined xor undefined | undefined |

# Rules Are Aggressively Memoized

Rules are lazy and memoized (will not be recomputed unless absolutely necessary).

```
rule when Y { Z == true }
```

# Syntax Tip: Write small rules

It's slightly easier to debug policies if the rules are short and there are many of them.

CODE EDITOR

```
bacon_is_crispy = rule { … }
pancake_is_fluffy = rule { … }
oj_is_fresh = rule { … }

main = rule {
 bacon_is_crispy and
   pancake_is_fluffy and
   oj_is_fresh
}
```

# Syntax - Collections

```
all collection as index, item { } # Returns bool

any ... # Returns bool

for ... # NOTE: Does not return

contains

matches
```

# Tip: Be as Precise as You Can

Using precise comparators will determine how accurate your policies return.

For equality, use `is` rather than `contains`

# Syntax - Functions

```
a = func(x) { return x*2 }
```

Must return (or return undefined)

# Syntax - Imports

```
import “time”

main = rule { time.now.day != 1 }
```

Imports introduce additional data structures and functions for policies to leverage.

Import availability is application-specific.

Similar to imports in many programming languages.

Terraform 201 - Sentinel

# Testing

# Testing - Introduction

The policy itself is the test code.

JSON test file(s) may include both a fixture (mock data) and the expected result.

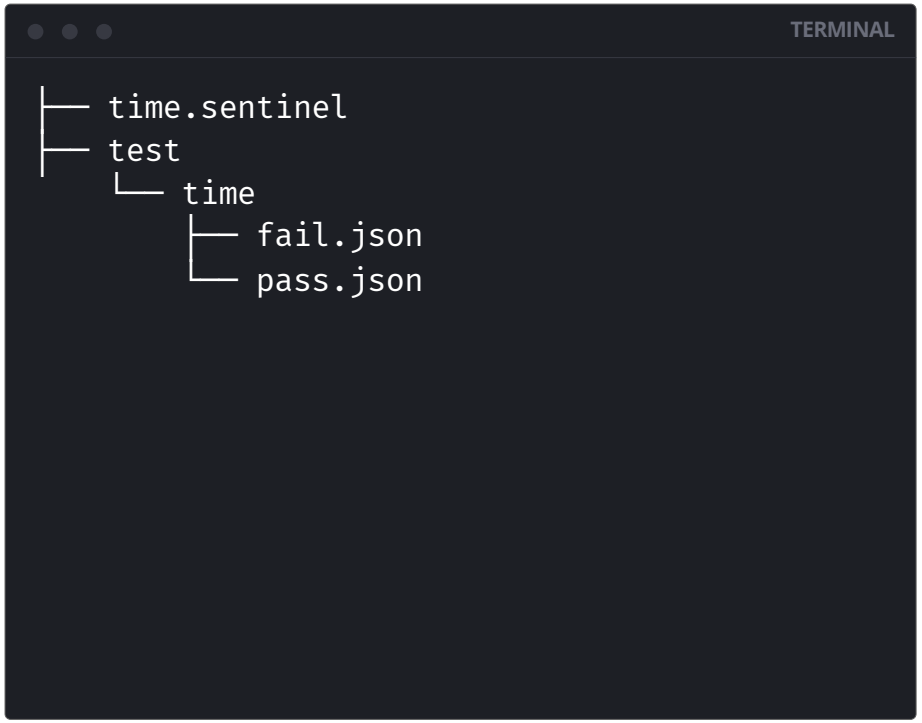
Multiple JSON test files may be present for each policy. Helps test policies against multiple fixtures with varying results.

CODE EDITOR

```
import "time"

is_business_hours = rule {
   time.now.hour >= 9 and
time.now.hour <= 17
}

main = rule { is_business_hours }
```

# Testing - Basic File Layout

- A sentinel policy file (time.sentinel)
- A test directory with a subdirectory matching the policy name (time)
- JSON files for each test case



## Simplest test case.

"`test`" defines which rules will be tested and the expected result of each.

In the previous slide, "`fail.json`" would expect main to return "`false`" here.

CODE EDITOR

```
{
 "test": {
   "main": true
 }
}
```

## Other testing data

- mock: stand in for values made available by "import".

CODE EDITOR

```
{
 "test": {
   "main": true
 },
 "mock": {
   "time": {
     "now": {
       "hour": 10
     }
   }
 }
}
```

## Other testing data

- global: values passed in to Sentinel from elsewhere

CODE EDITOR

```
{
 "test": {
   "main": true
 },
 "global": {
   "environment": "staging"
 }
}
```

# Sentinel Simulator

`sentinel test` with no arguments runs all test cases

TERMINAL

```
$ sentinel --help test

Test cases are expected to be in "test/<policy>/*.json"
files where "<policy>" is the name of the policy filename
without an extension.

$ sentinel test

PASS - time.sentinel
 PASS - test/time/pass.json  PASS - test/time/fail.json
```

# ⚠️ Important :

## Errors are minimal

You might see that a test fails but not have a clear reason, even with --verbose

For example, doing import "tfplan" in a policy will cause the test to fail unless mocked (because it can't get to the import).

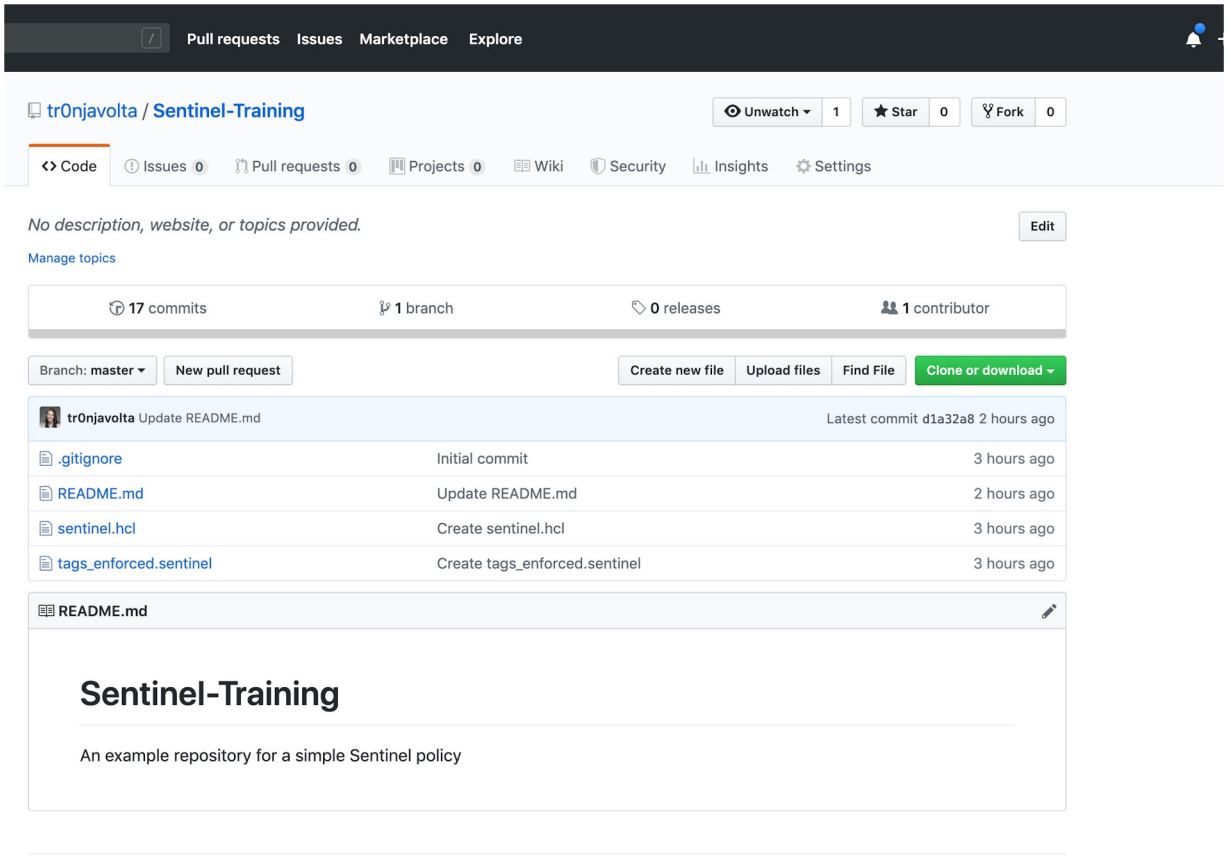
Terraform 201 - Sentinel

# Workflow in Terraform Cloud

# Sentinel Policy Sets & VCS

Managing individual policies in Sentinel is deprecated.

We now use VCS or push via the API to manage our policy sets.



# tags_enforced.sentinel

This policy in the previous repo evaluates if an AWS EC2 instance is tagged and fails if not.

CODE EDITOR

```
import "tfplan"

main = rule {
  all tfplan.resources.aws_instance as _,
instances {
    all instances as _, r {
      (length(r.applied.tags) else 0) > 0
    }
  }
}
```

# sentinel.hcl

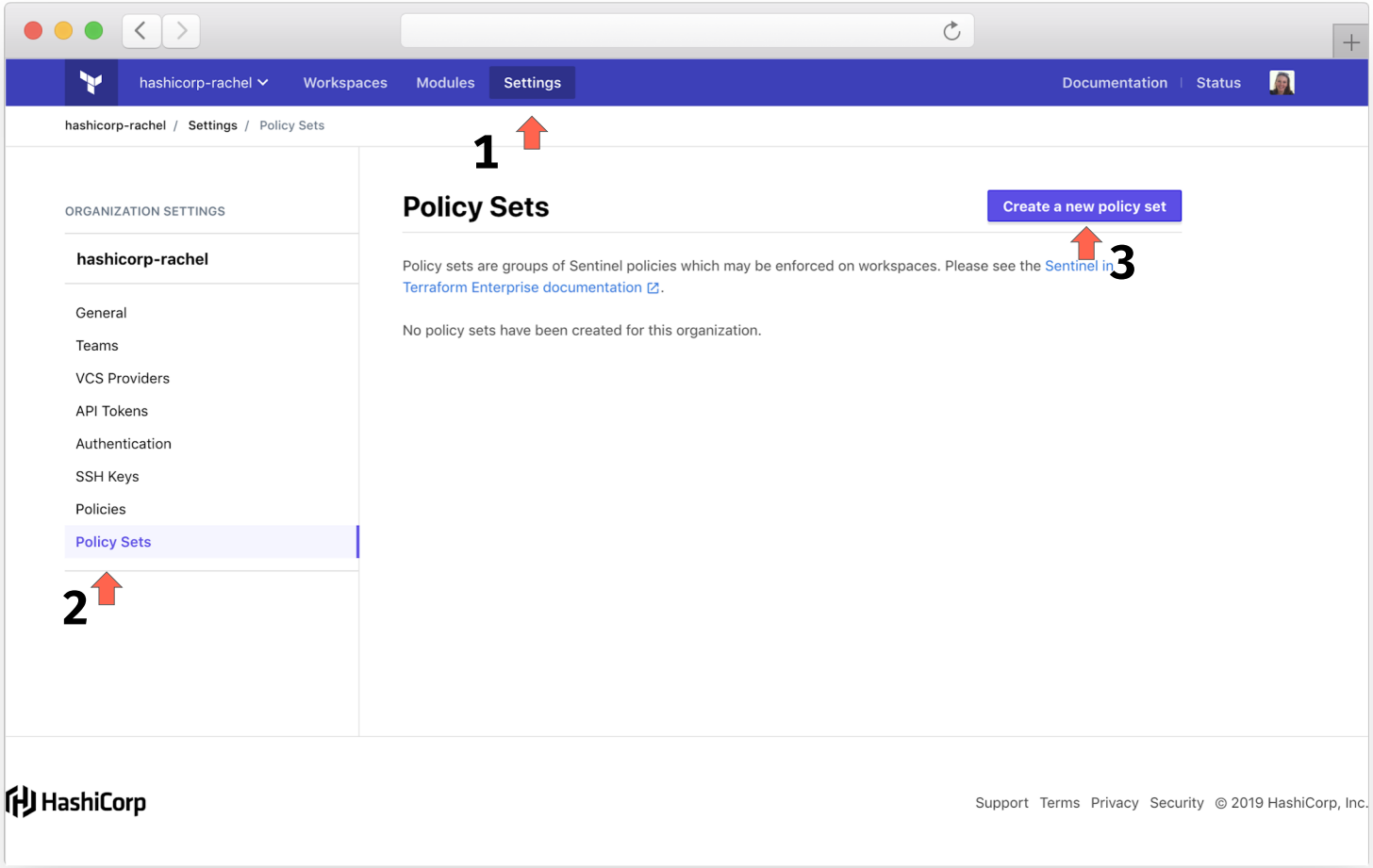
This hcl file in the same repo determines which policies to enforce and at what enforcement level.

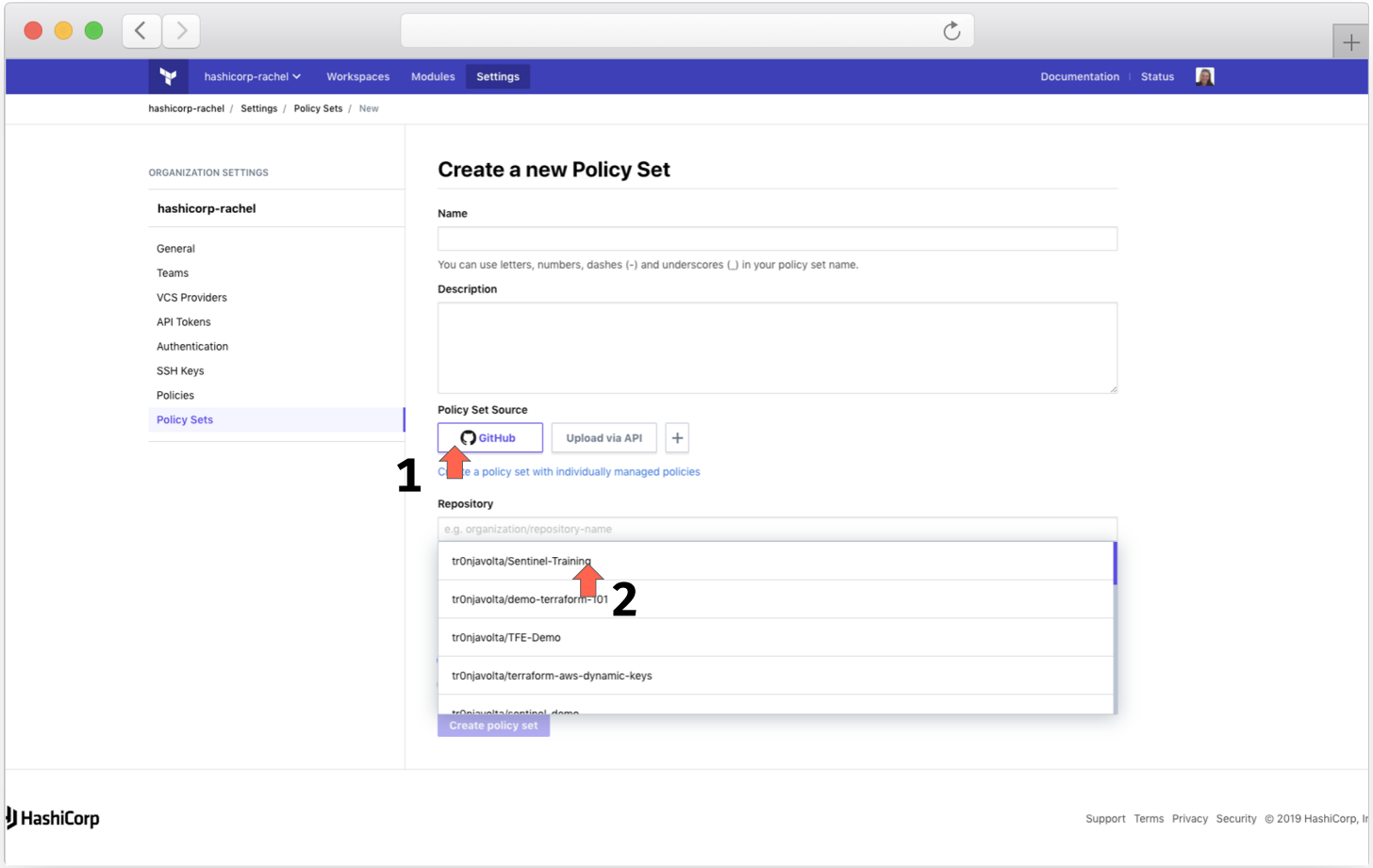
CODE EDITOR

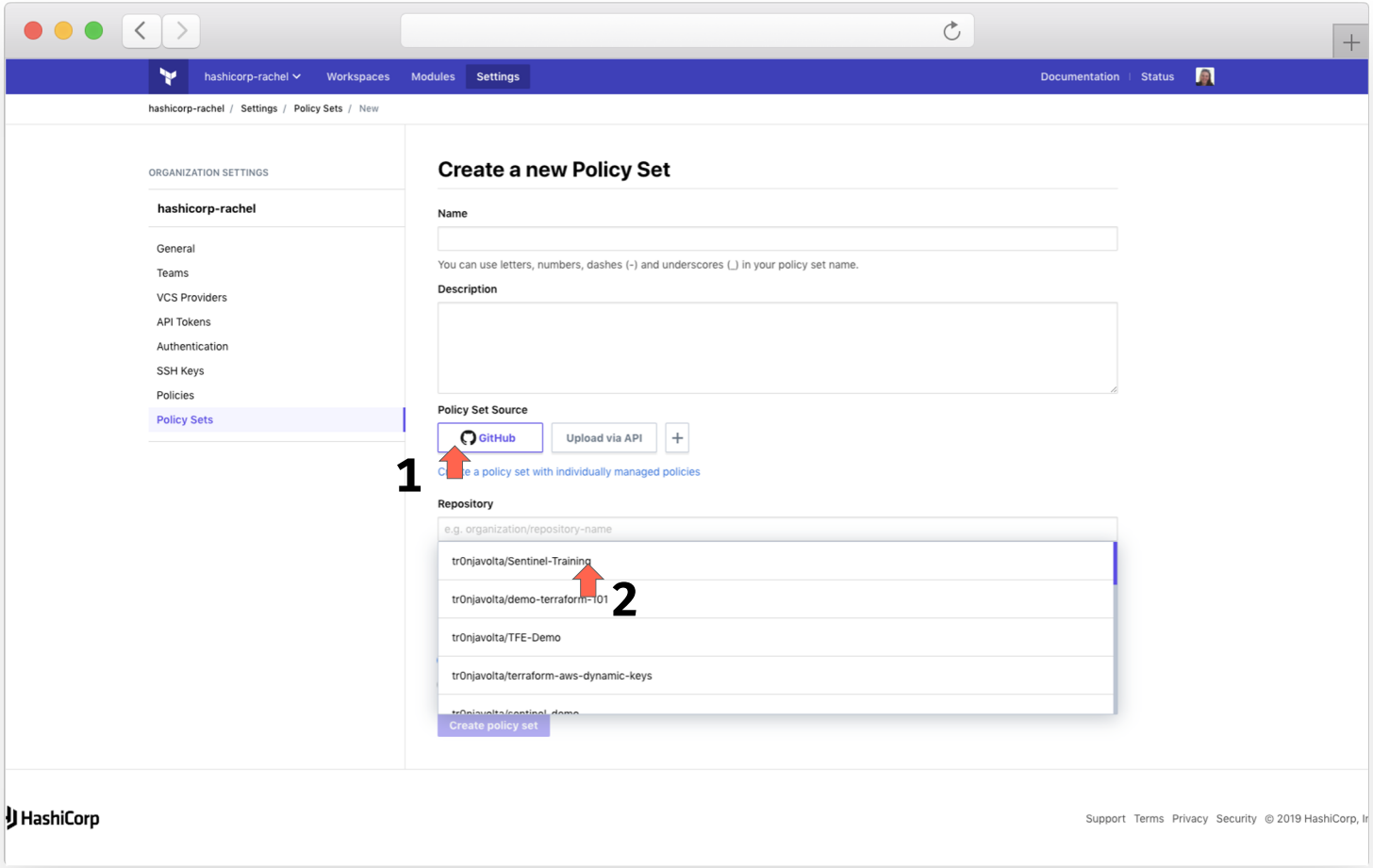
```
policy "tags_enforced" {
    enforcement_level = "hard-mandatory"
}
```

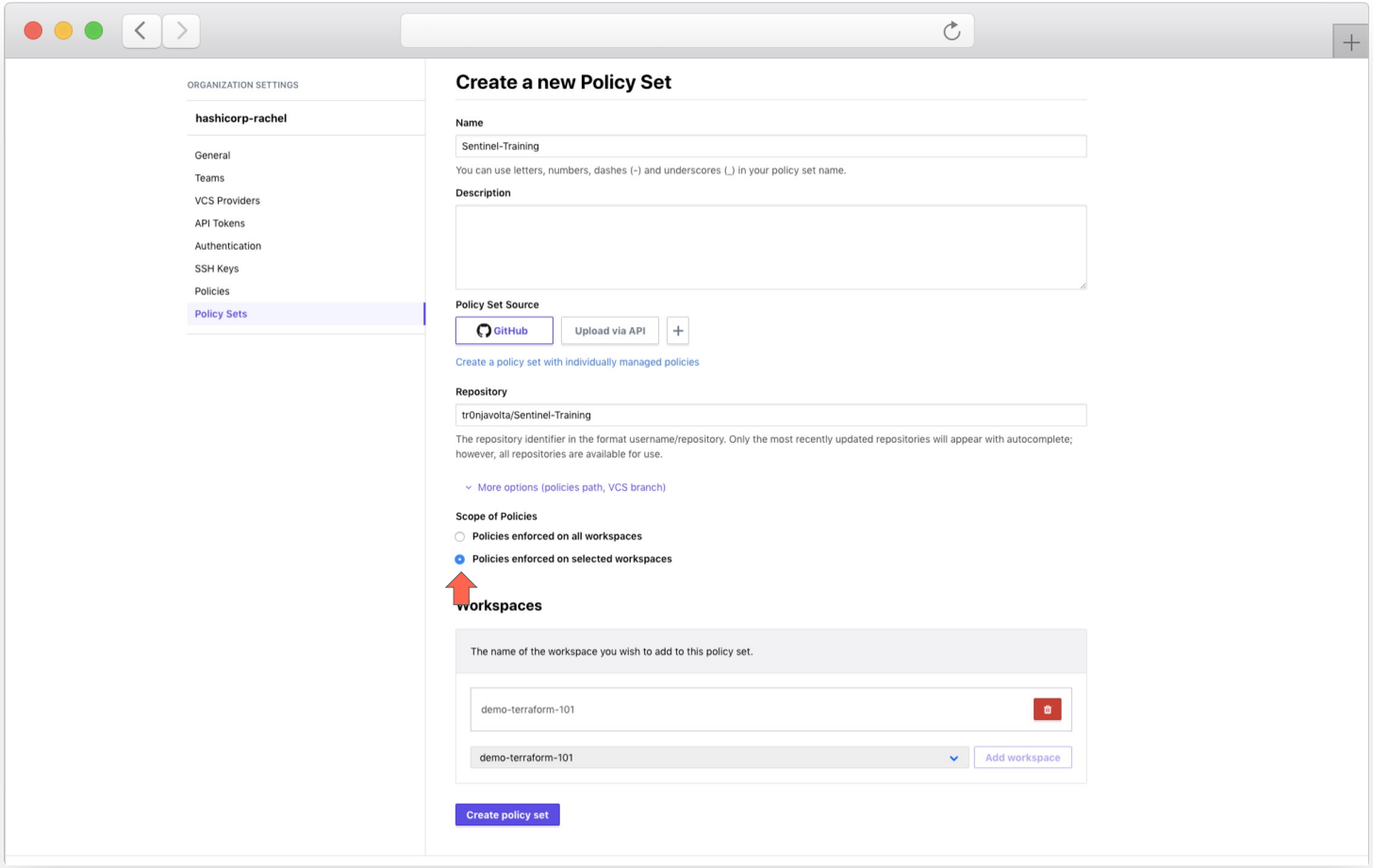
# Enforcement Levels

```
hard-mandatory (permanent)

soft-mandatory (overridable by org owners)

advisory (log only)
```


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Settings / Policy Sets
1
ORGANIZATION SETTINGS
hashicorp-rachel
General
Teams
VCS Providers
API Tokens
Authentication
SSH Keys
Policies
Policy Sets
2
Policy Sets
Create a new policy set
3
Policy sets are groups of Sentinel policies which may be enforced on workspaces. Please see the Sentinel in Terraform Enterprise documentation.
No policy sets have been created for this organization.
HashiCorp
Support Terms Privacy Security © 2019 HashiCorp, Inc.


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Settings / Policy Sets / New
ORGANIZATION SETTINGS
hashicorp-rachel
General
Teams
VCS Providers
API Tokens
Authentication
SSH Keys
Policies
Policy Sets
Create a new Policy Set
Name
You can use letters, numbers, dashes (-) and underscores (_) in your policy set name.
Description
Policy Set Source
GitHub
Upload via API
1
Repository
e.g. organization/repository-name
tr0njavolta/Sentinel-Training
tr0njavolta/demo-terraform-101
tr0njavolta/TFE-Demo
tr0njavolta/terraform-aws-dynamic-keys
2
Create policy set
HashiCorp
Support
Terms
Privacy
Security
© 2019 HashiCorp, Ir


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Settings / Policy Sets / New
ORGANIZATION SETTINGS
hashicorp-rachel
General
Teams
VCS Providers
API Tokens
Authentication
SSH Keys
Policies
Policy Sets
Create a new Policy Set
Name
You can use letters, numbers, dashes (-) and underscores (_) in your policy set name.
Description
Policy Set Source
GitHub
Upload via API
1
Repository
e.g. organization/repository-name
tr0njavolta/Sentinel-Training
tr0njavolta/demo-terraform-101
tr0njavolta/TFE-Demo
tr0njavolta/terraform-aws-dynamic-keys
2
Create policy set
HashiCorp
Support
Terms
Privacy
Security
© 2019 HashiCorp, Ir


ORGANIZATION SETTINGS
hashicorp-rachel
General
Teams
VCS Providers
API Tokens
Authentication
SSH Keys
Policies
Policy Sets
Create a new Policy Set
Name
Sentinel-Training
You can use letters, numbers, dashes (-) and underscores (_) in your policy set name.
Description
Policy Set Source
GitHub
Upload via API
Create a policy set with individually managed policies
Repository
tr0njavolta/Sentinel-Training
The repository identifier in the format username/repository. Only the most recently updated repositories will appear with autocomplete; however, all repositories are available for use.
More options (policies path, VCS branch)
Scope of Policies
Policies enforced on all workspaces
Policies enforced on selected workspaces
Workspaces
The name of the workspace you wish to add to this policy set.
demo-terraform-101
demo-terraform-101
Add workspace
Create policy set

## Mocked Data

Saving policies makes them immediately active for your organization's workspaces in Terraform Enterprise.

It is therefore important to test policy code locally prior to uploading.

A simple policy could be mocked with data that looks like this. This would allow the policy to be tested locally before saving it into Terraform Enterprise.

CODE EDITOR

```
    "tfplan": {
      "resources": {
        "animals": {
          "cow": [
            { "applied": { "id":
"animal-0" } }
          ]
        }
      }
    }
```

# Examples & Documentation

HashiCorp staff frequently build and publish sample policies.

Find them in places such as the `terraform-guides` repository:

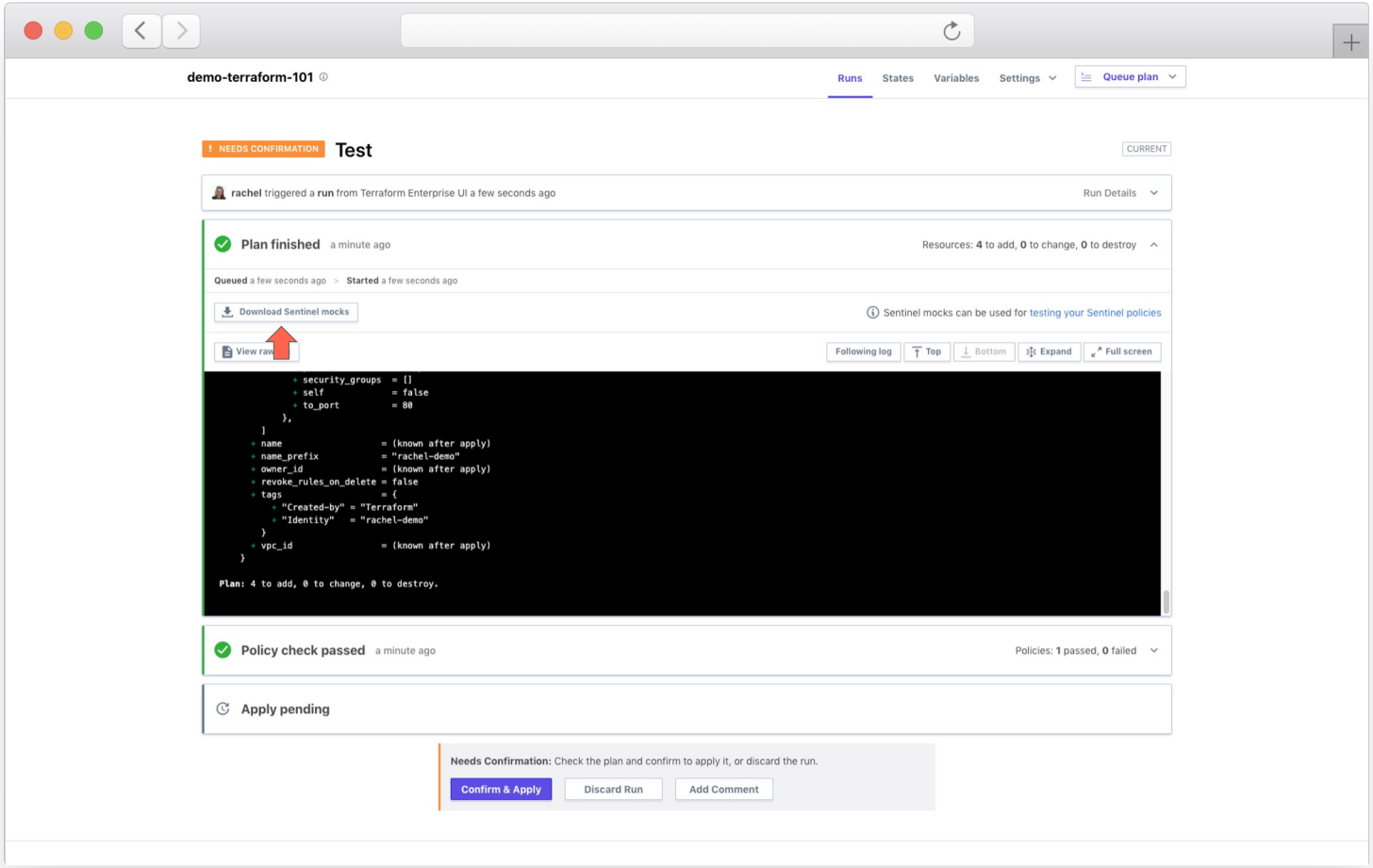
https://github.com/hashicorp/terraform-guides/tree/master/governance/second-generation

Read, understand, and refer to the documentation on imports (`tfplan, tfconfig, tfstate, tfrun`):

https://www.terraform.io/docs/cloud/sentinel/import/index.html

Terraform 201 - Sentinel

# Mock Data in Terraform Cloud


demo-terraform-101
Runs
States
Variables
Settings
Queue plan
NEEDS CONFIRMATION
Test
CURRENT
rachel triggered a run from Terraform Enterprise UI a few seconds ago
Run Details
Plan finished a minute ago
Resources: 4 to add, 0 to change, 0 to destroy
Queued a few seconds ago > Started a few seconds ago
Download Sentinel mocks
Sentinel mocks can be used for testing your Sentinel policies
View raw
Following log
Top
Bottom
Expand
Full screen
+ security_groups = []
+ self = false
+ to_port = 80
},
]
+ name = (known after apply)
+ name_prefix = "rachel-demo"
+ owner_id = (known after apply)
+ revoke_rules_on_delete = false
+ tags = {
+ "Created-by" = "Terraform"
+ "Identity" = "rachel-demo"
}
+ vpc_id = (known after apply)
}
Plan: 4 to add, 0 to change, 0 to destroy.
Policy check passed a minute ago
Policies: 1 passed, 0 failed
Apply pending
Needs Confirmation: Check the plan and confirm to apply it, or discard the run.
Confirm & Apply
Discard Run
Add Comment

# Mock data structure

TERMINAL

```
├── sentinel-demo
    ├── with-data.sentinel
    ├── sentinel.json
    ├── test
        ├── fail.json
        └── pass.json
    └── testdata
        ├── mock-tfconfig.sentinel
        ├── mock-tfplan.sentinel
        └── mock-tfstate.sentinel
```

# Mock data

TERMINAL

```
$ cat sentinel.json

{
 "mock": {
   "tfconfig": "testdata/mock-tfconfig.sentinel",
   "tfplan": "testdat/mock-tfplan.sentinel",
   "tfrun": "testdata/mock-tfrun.sentinel",
   "tfstate": "testdata/mock-tfstate.sentinel"
 }
}

$ sentinel apply -trace instance_type_is_medium.sentinel
-config=sentinel.json`
```

Terraform 201 - Sentinel

# Best Practices (and a few warnings)

# Local is Better

Work on the local machine, write tests. Work in small increments if possible.

Run a syntax check with `fmt`, leverage mock data where possible.

The cycle of write-execute-observe is long in Terraform Cloud, especially if your core Terraform plan is large.

# Warnings are minimal

You won't be warned for certain types of mistakes such as scoping errors.

Writing good tests and executing them prior to a production is imperative.

# Computed values can be loopholes

Certain values may not be known until apply time.

These values cannot be effectively used in policies.

Terraform config authors could exploit this to bypass policies that would otherwise fail.

The most defensive thing you can do is disallow computed values in the policy.

# Code defensively, favor denial as much as possible

When possible, use a whitelist instead of a blacklist.

Example: no cidr 0.0.0.0/0 vs 192.168.0.0/16

Whitelists are more secure than blacklists, in general.

Example: 000.000.000.000/0 is a valid IP that could evade checks for 0.0.0.0/0

# The `applied` data structure is most useful

When examining the `tfplan` data structure, look for `applied` to find the values that will be deployed

Terraform 201

# Authoring with the Registry

# What You'll Learn

- Use standard module structure to build a module
- Publish a module to the registry
- Use the published module from a Terraform configuration
- Understand how to version and update a module
- Use the module configuration designer

## Modules

Collaboration, Reuse, Consistency

EXAMPLES

AWS Networking
App Server
DB Cluster


## VCS

Access Control, Pull Request Workflow, CI

EXAMPLES

GitHub
Bitbucket (private)

## Terraform Registry

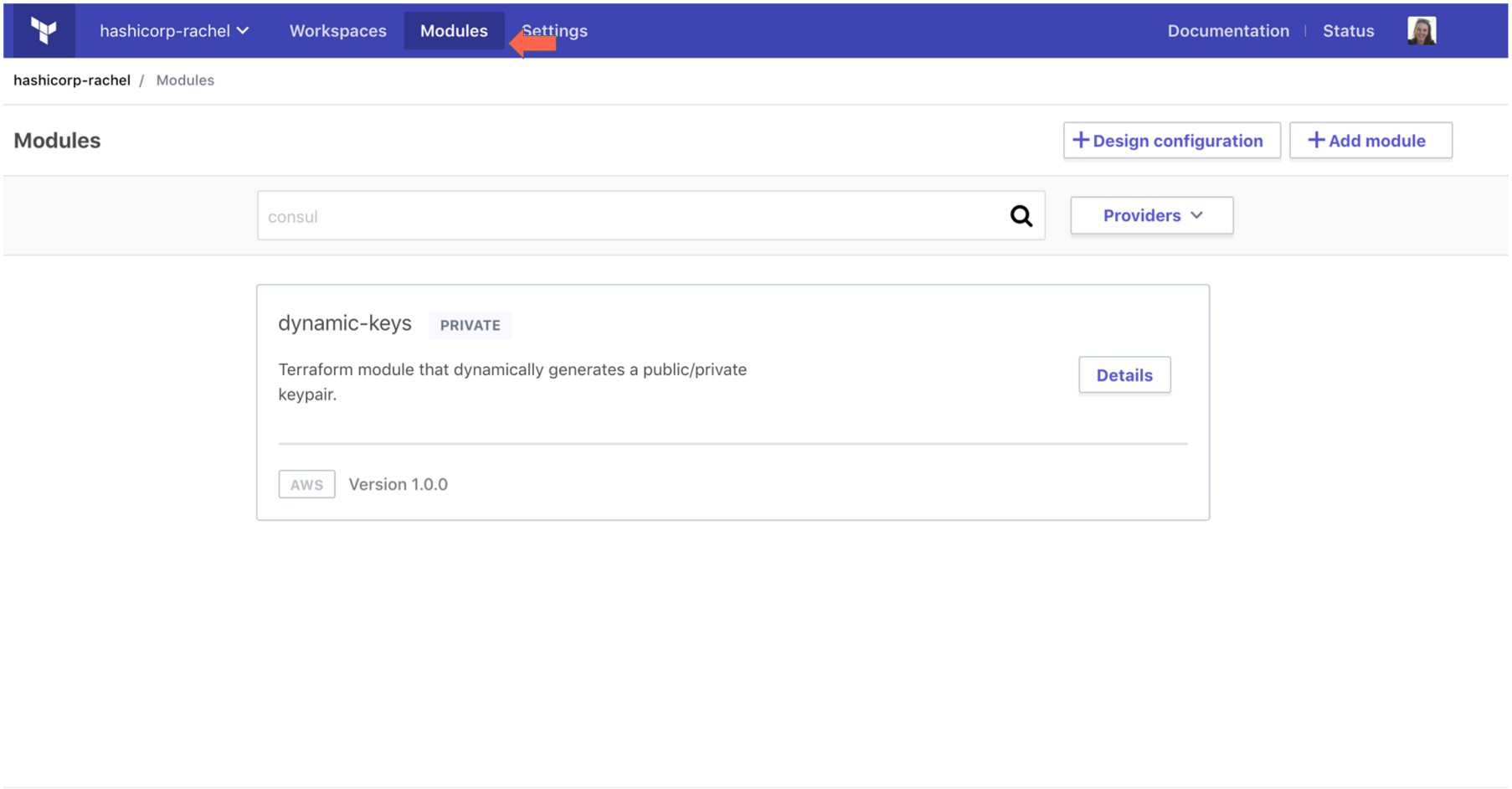
Versioning, Documentation, Search

EXAMPLES

terraform-aws-networking
terraform-go-server
terraform-pg-cluster

# Private Module Registry

Both the hosted and private instances of Terraform include your own private module registry which can be used to securely and privately publish modules within your company.


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Modules
Modules
+ Design configuration
+ Add module
consul
Providers
dynamic-keys
PRIVATE
Terraform module that dynamically generates a public/private keypair.
Details
AWS
Version 1.0.0

# Module Registry Workflow: Git

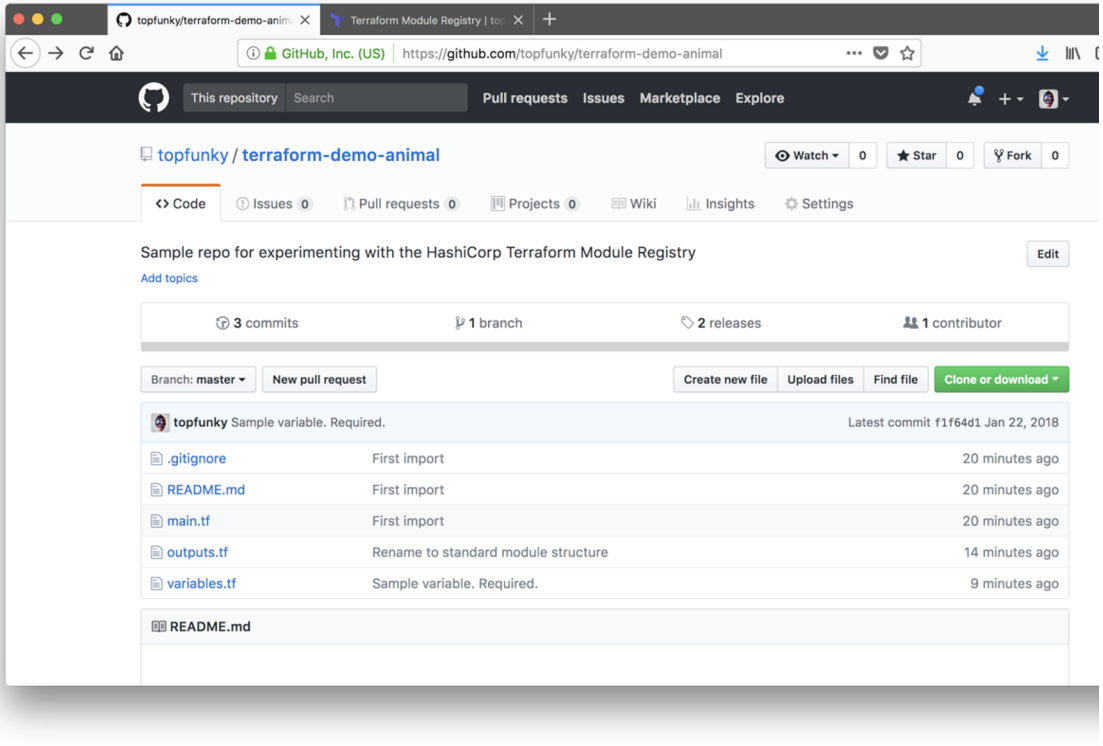
Push code to GitHub or Bitbucket. The repository must be named in the format:

terraform-PROVIDER-NAME

Examples:

terraform-aws-peering

terraform-azure-autoscaling

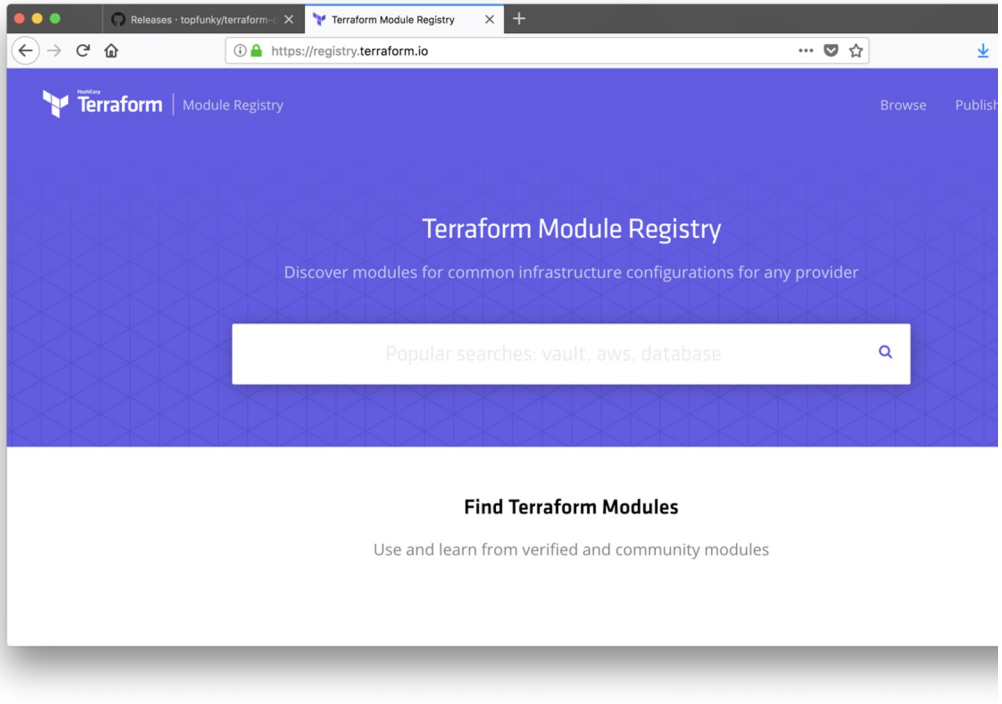


# Module Registry Workflow: Registry

Connect your SCM account to the Terraform Registry. Publish a module.

Tagged commits will be synced automatically.

v1.0.6



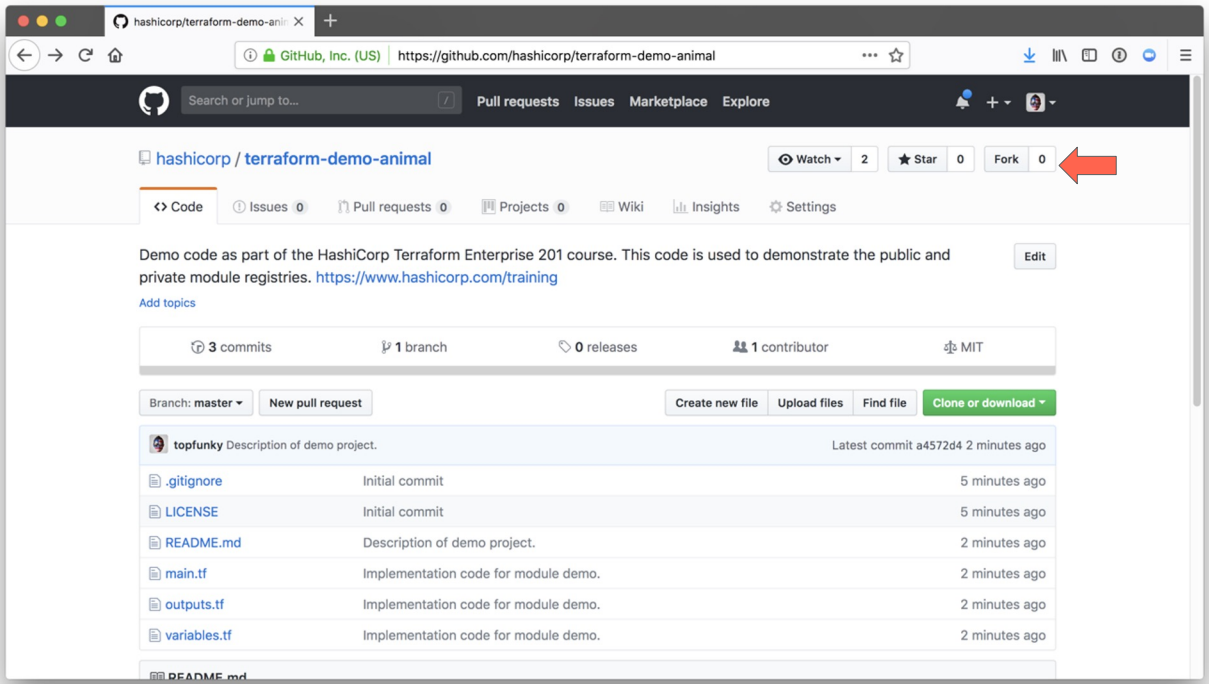
# Module Registry Workflow: Config

Use the code snippet from the registry to reference the module from your code. Provide all necessary variables.

Run `init` to pull in the correct version of the module.

```
terraform init
```

# Example of Modules for Registry



hashicorp/terraform-demo-ani
GitHub, Inc. (US)
https://github.com/hashicorp/terraform-demo-animal
Search or jump to...
Pull requests
Issues
Marketplace
Explore
hashicorp / terraform-demo-animal
Watch 2
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Demo code as part of the HashiCorp Terraform Enterprise 201 course. This code is used to demonstrate the public and private module registries. https://www.hashicorp.com/training
Edit
Add topics
3 commits
1 branch
0 releases
1 contributor
MIT
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
topfunky Description of demo project.
Latest commit a4572d4 2 minutes ago
.gitignore Initial commit 5 minutes ago
LICENSE Initial commit 5 minutes ago
README.md Description of demo project. 2 minutes ago
main.tf Implementation code for module demo. 2 minutes ago
outputs.tf Implementation code for module demo. 2 minutes ago
variables.tf Implementation code for module demo. 2 minutes ago

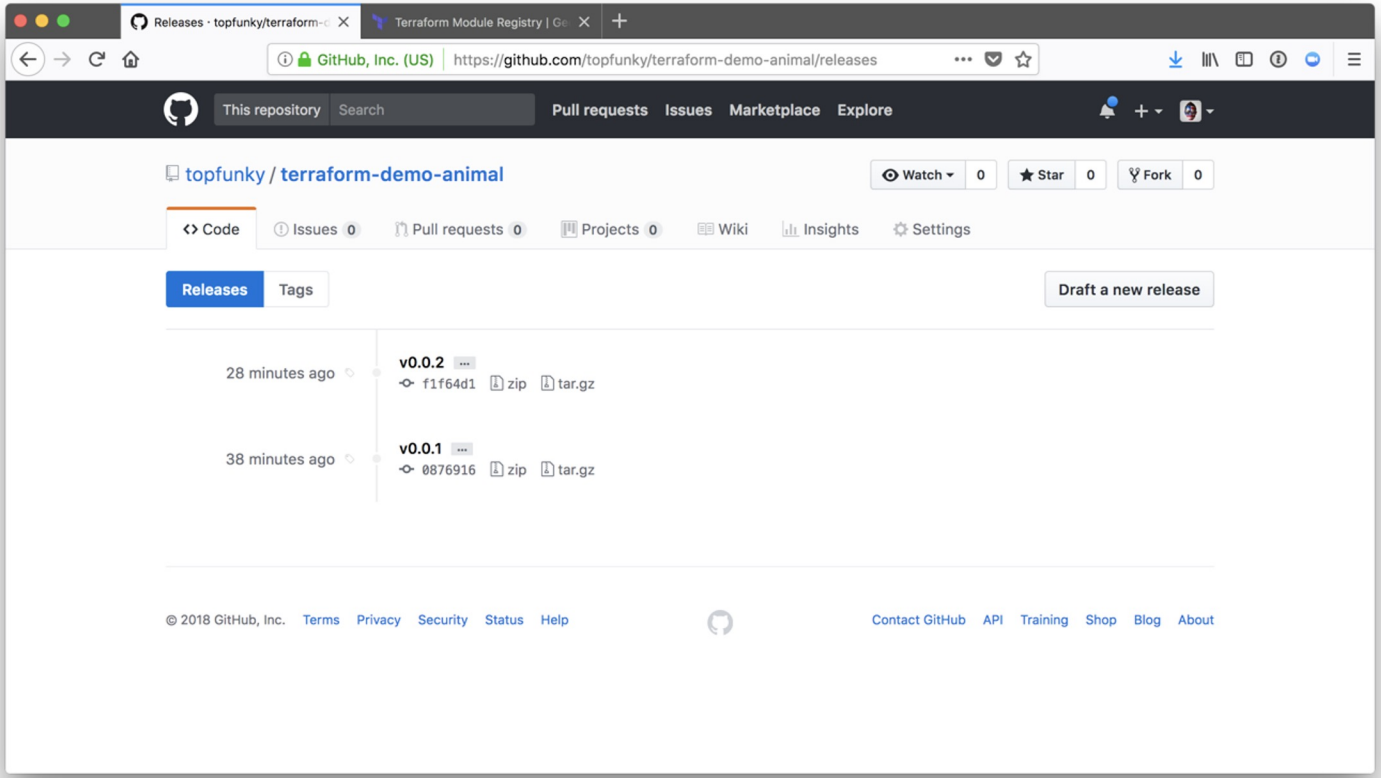
# Push tags from your Fork

# Publish the Code on Your Private Module Registry

Go to https://app.terraform.io/ and go to the workspaces dashboard.

Click the "Modules" button in the menu bar.

Use the "+ Add Module" button to select your repository and add it to the module registry.


Releases · topfunky/terraform-
Terraform Module Registry | Ge
GitHub, Inc. (US)
https://github.com/topfunky/terraform-demo-animal/releases
This repository
Search
Pull requests
Issues
Marketplace
Explore
topfunky / terraform-demo-animal
Watch 0
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Releases
Tags
Draft a new release
28 minutes ago
v0.0.2
f1f64d1 zip tar.gz
38 minutes ago
v0.0.1
0876916 zip tar.gz
© 2018 GitHub, Inc.
Terms
Privacy
Security
Status
Help
Contact GitHub
API
Training
Shop
Blog
About

Terraform 201 - Module Registry

# Best Practices

# Module Versioning

In your published module, don't forget to set a version for the resources and providers that you rely on.

Be sure to provide as much version flexibility as possible to make your module easy to use among various Terraform provider versions. I.e., prefer ">= 1.0" to "1.0".

CODE EDITOR

```
# main.tf
provider "random" {
 version = "~> 1.1"
}
```

## Module Storage Structure

Look at the hidden ".terraform" directory and you'll see that all versions of the module are stored locally.

Metadata is in .terraform/modules/modules.json

TERMINAL

```
$ tree .terraform

.terraform
├── modules
│   ├── 495f1ee96941ccf0a6c619930ea1160d
│   │   └── topfunky-terraform-demo-animal-f1f64d1
│   │       ├── README.md
│   │       ├── main.tf
│   │       ├── outputs.tf
│   │       └── variables.tf
│   ├── 753525cdd62578ec39462ba136cf9f7e
│   │   └── topfunky-terraform-demo-animal-0876916
│   │       ├── README.md
│   │       ├── main.tf
│   │       └── output.tf
│   └── modules.json
└── plugins
    └── darwin_amd64
        ├── lock.json
        └── terraform-provider-random_v1.1.0_x4
```

## Versioning & Dependency Metadata

The contents of .terraform/modules/modules.json show how Terraform tracks version numbers and maps them to local directories.

If you need to reset all dependencies, you can delete this directory and run "terraform init" again.

CODE EDITOR

```
{
   "Modules" : [
      {
         "Root" : "topfunky-terraform-demo-animal-0876916",
         "Version" : "0.0.1",
         "Source" : "topfunky/animal/demo",
         "Key" : "1.animal;topfunky/animal/demo.0.0.1",
         "Dir" :
".terraform/modules/753525cdd62578ec39462ba136cf9f7e"
      },
      {
         "Dir" :
".terraform/modules/495f1ee96941ccf0a6c619930ea1160d",
         "Source" : "topfunky/animal/demo",
         "Key" : "1.animal;topfunky/animal/demo.0.0.2",
         "Version" : "0.0.2",
         "Root" : "topfunky-terraform-demo-animal-f1f64d1"
      }
   ]
}
```

# By Default, Paths are Relative to the Terraform CLI

If you want to reference resources in a published module's own directory, use

```
${path.module}
```

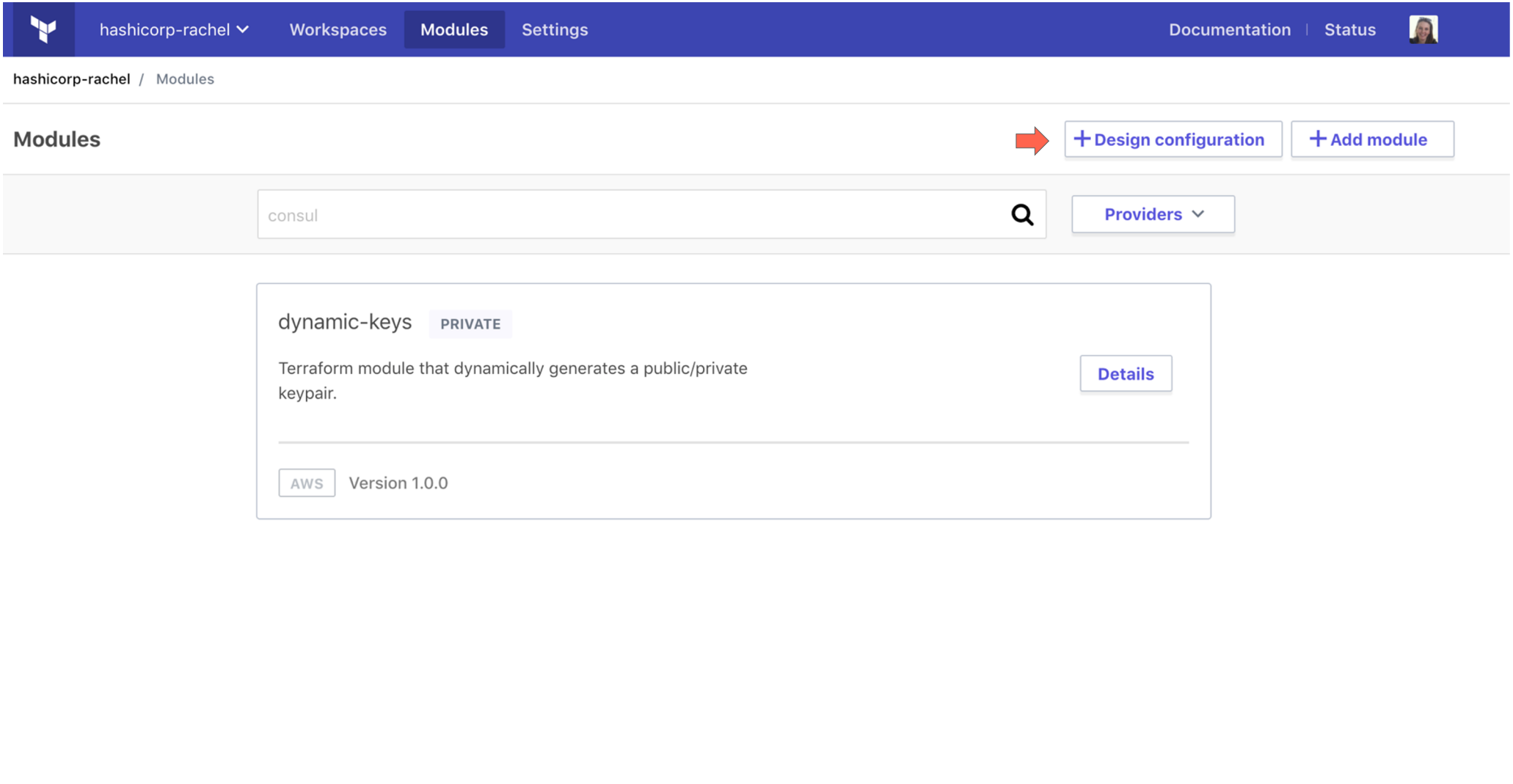
Terraform 201 - Module Registry

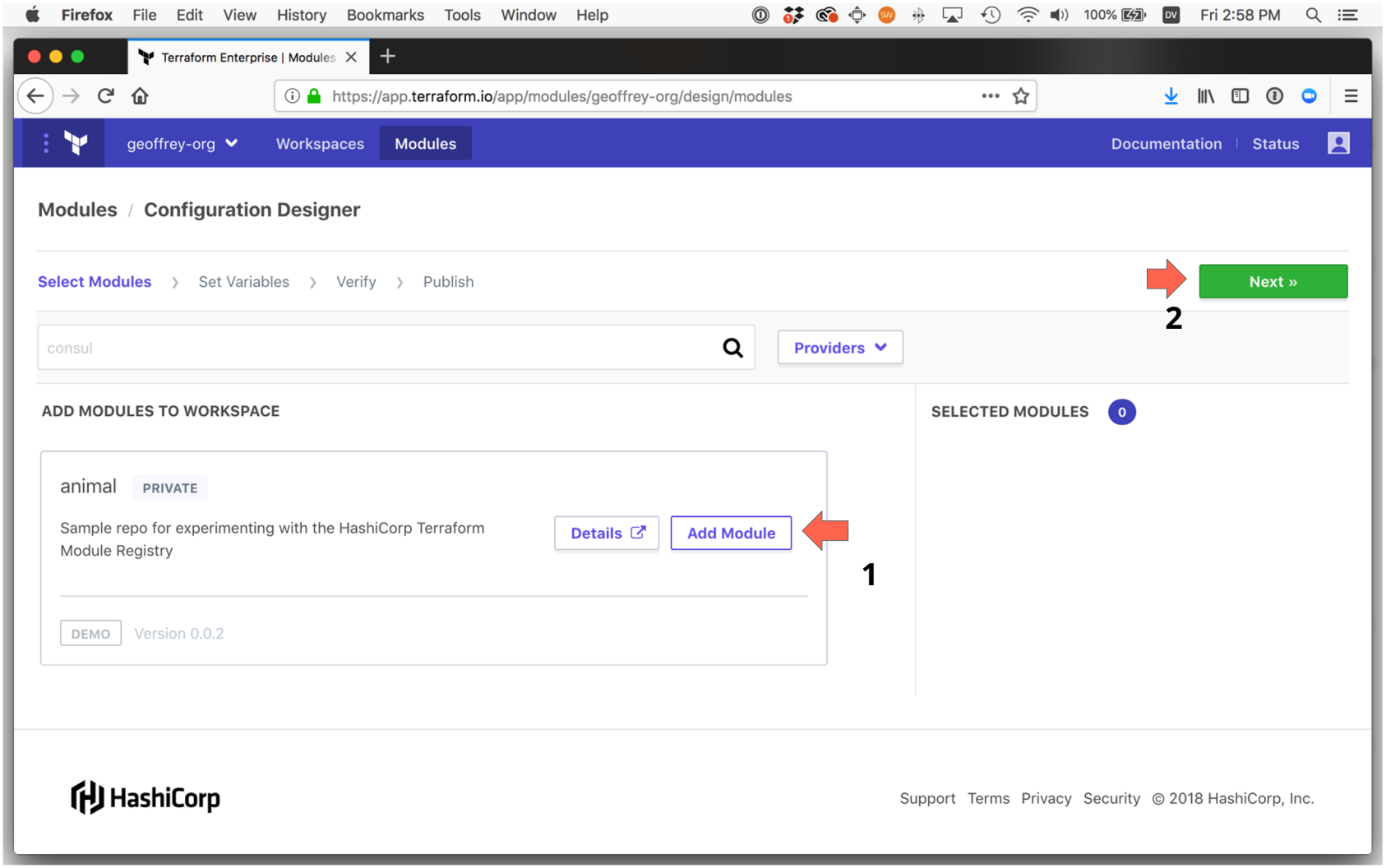
# Configuration Designer

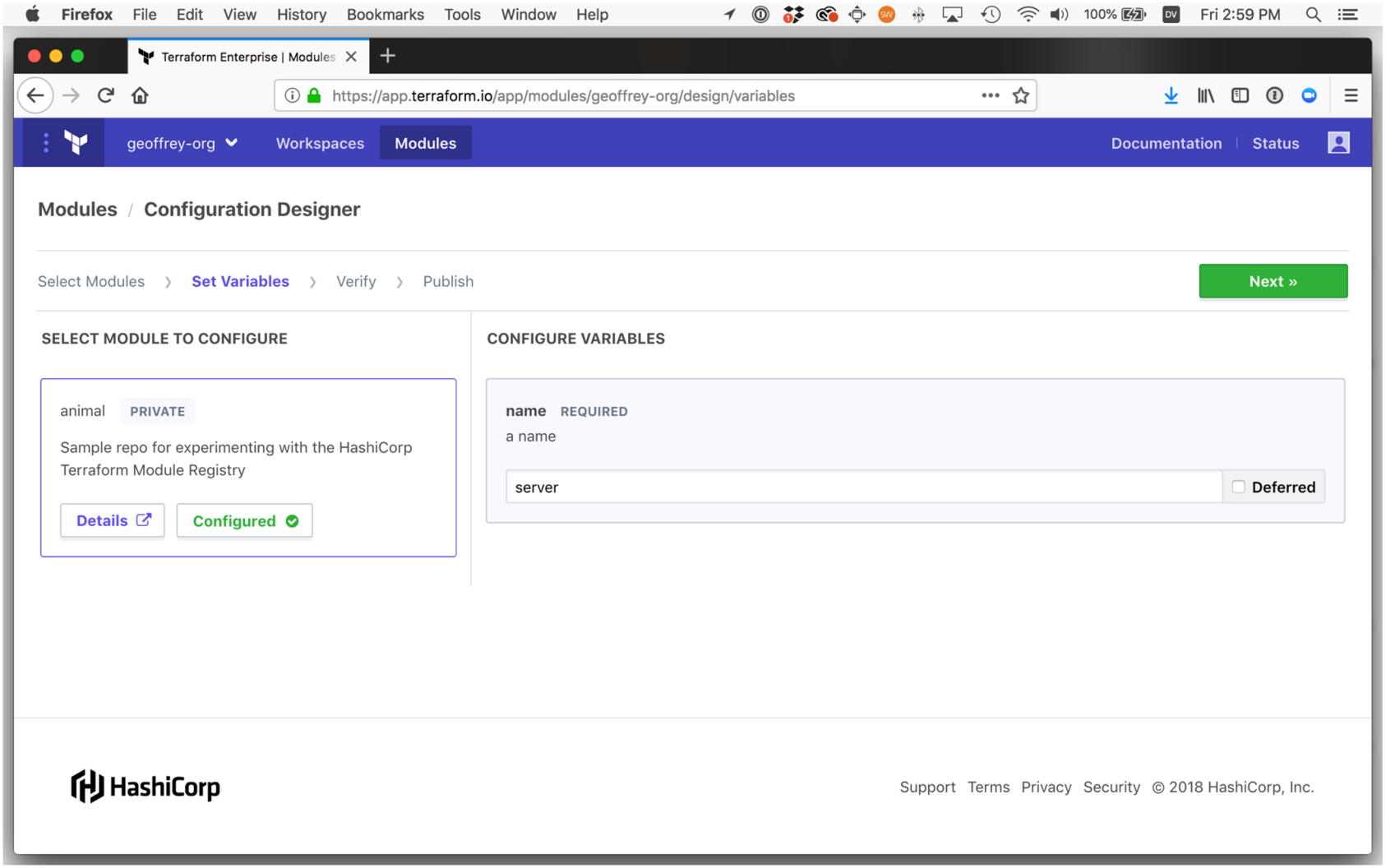
# Configuration Designer

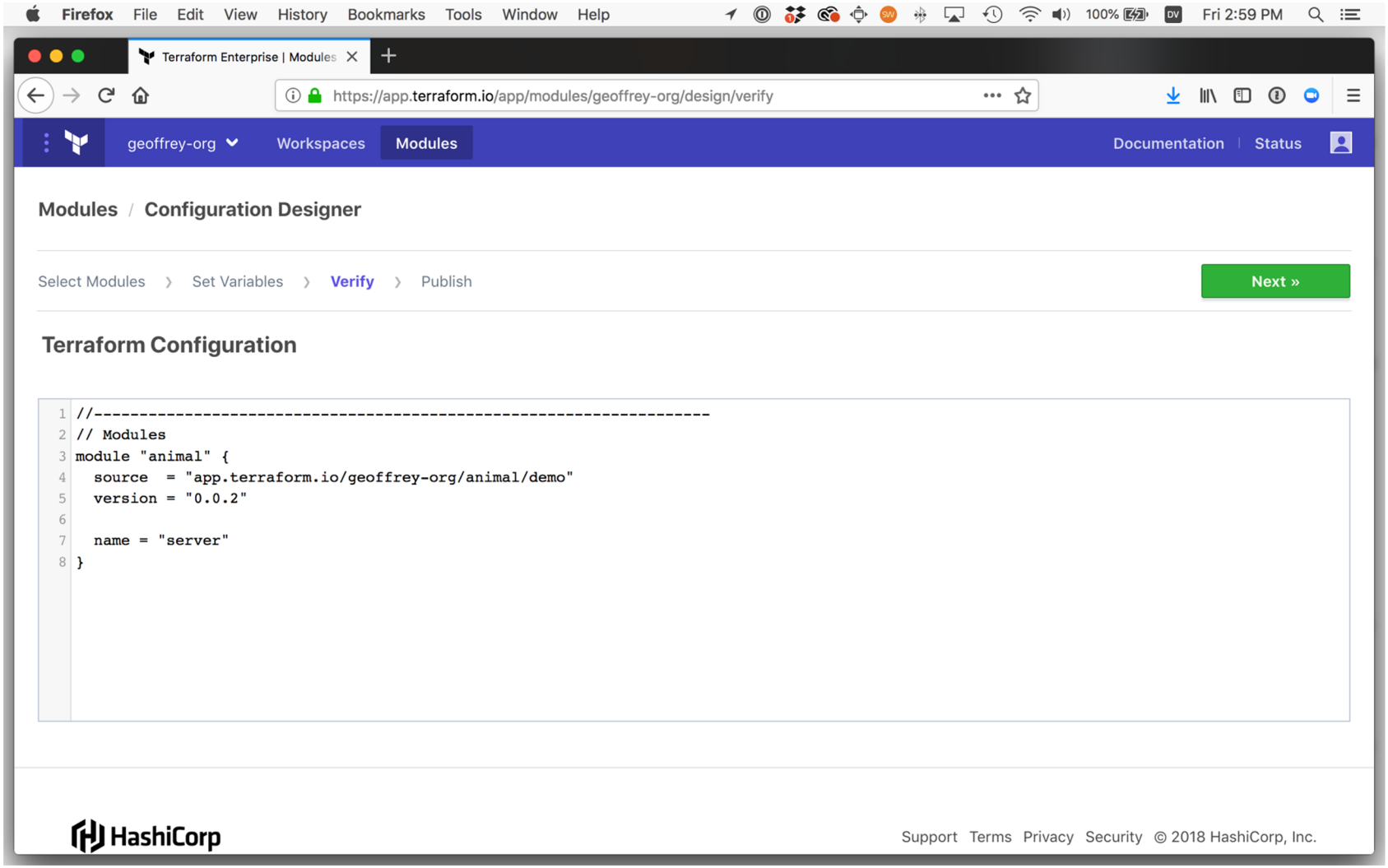
User interface for selecting modules and specifying their versions and required values.

Generates boilerplate Terraform configuration code to import the desired modules and pass in the appropriate values.


hashicorp-rachel
Workspaces
Modules
Settings
Documentation
Status
hashicorp-rachel / Modules
Modules
+ Design configuration
+ Add module
consul
Providers
dynamic-keys
PRIVATE
Terraform module that dynamically generates a public/private keypair.
Details
AWS
Version 1.0.0


Firefox File Edit View History Bookmarks Tools Window Help
Fri 2:58 PM
Terraform Enterprise | Modules
https://app.terraform.io/app/modules/geoffrey-org/design/modules
geoffrey-org
Workspaces
Modules
Documentation
Status
Modules / Configuration Designer
Select Modules
Set Variables
Verify
Publish
Next »
2
consul
Providers
ADD MODULES TO WORKSPACE
SELECTED MODULES
0
animal
PRIVATE
Sample repo for experimenting with the HashiCorp Terraform Module Registry
Details
Add Module
1
DEMO
Version 0.0.2
HashiCorp
Support Terms Privacy Security © 2018 HashiCorp, Inc.


Firefox File Edit View History Bookmarks Tools Window Help
100% Fri 2:59 PM
Terraform Enterprise | Modules
https://app.terraform.io/app/modules/geoffrey-org/design/variables
geoffrey-org
Workspaces
Modules
Documentation
Status
Modules / Configuration Designer
Select Modules › Set Variables › Verify › Publish
Next »
SELECT MODULE TO CONFIGURE
animal PRIVATE
Sample repo for experimenting with the HashiCorp Terraform Module Registry
Details
Configured
CONFIGURE VARIABLES
name REQUIRED
a name
server
Deferred
HashiCorp
Support Terms Privacy Security © 2018 HashiCorp, Inc.

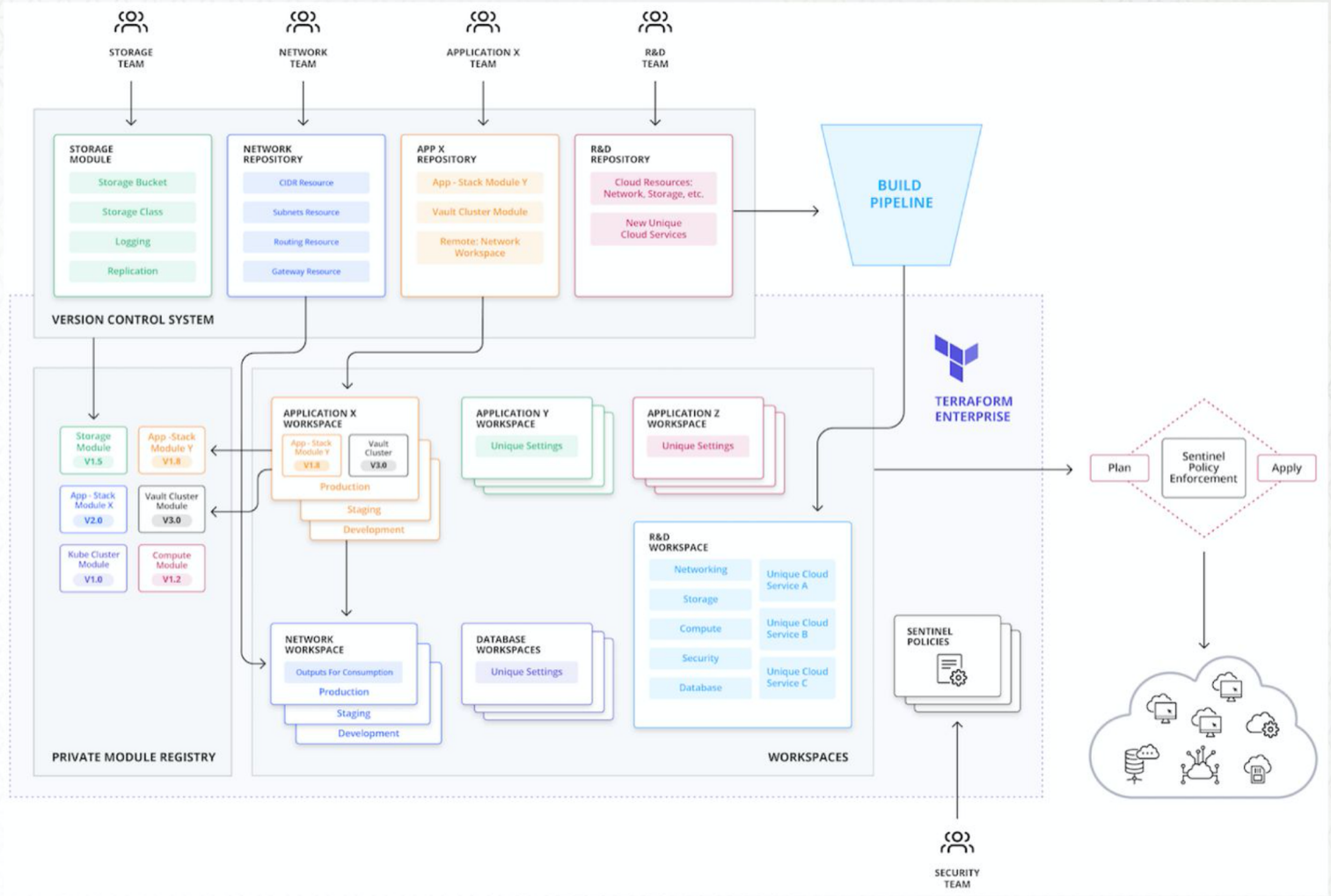

Firefox File Edit View History Bookmarks Tools Window Help
100% Fri 2:59 PM
Terraform Enterprise | Modules
https://app.terraform.io/app/modules/geoffrey-org/design/verify
geoffrey-org
Workspaces
Modules
Documentation
Status
Modules / Configuration Designer
Select Modules › Set Variables › Verify › Publish
Next »
Terraform Configuration
1 //-------------------------------------------------------------------
2 // Modules
3 module "animal" {
4 source = "app.terraform.io/geoffrey-org/animal/demo"
5 version = "0.0.2"
6
7 name = "server"
8 }
HashiCorp
Support Terms Privacy Security © 2018 HashiCorp, Inc.

# Configuration Designer

Supports the producer/consumer pattern.

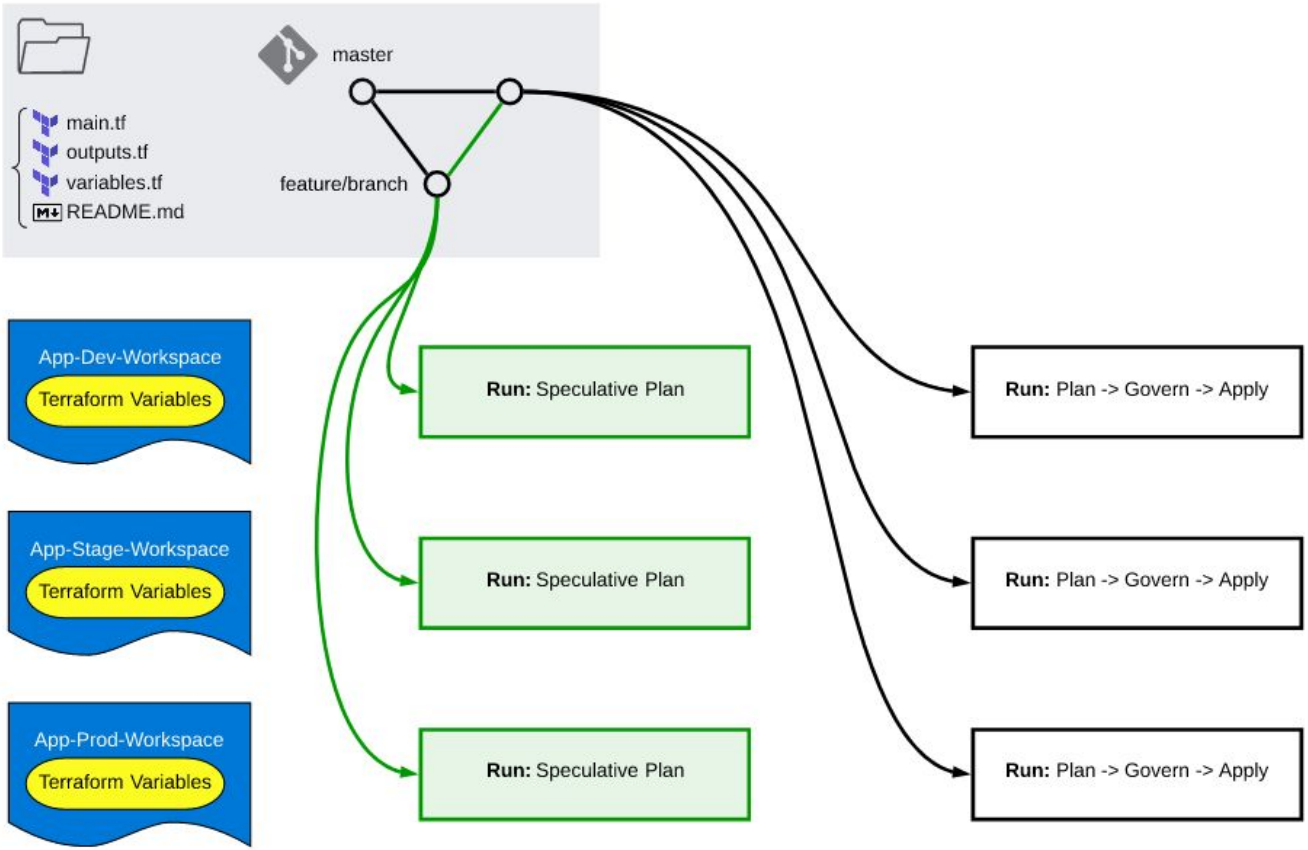
Teams can create well-documented modules with clear arguments that can be configured and used by other teams. Only minimal coding ability is required.

Producer teams can define best practices in code. Consuming teams don't need to understand the details, but can comply with infrastructure provisioning and security policies.

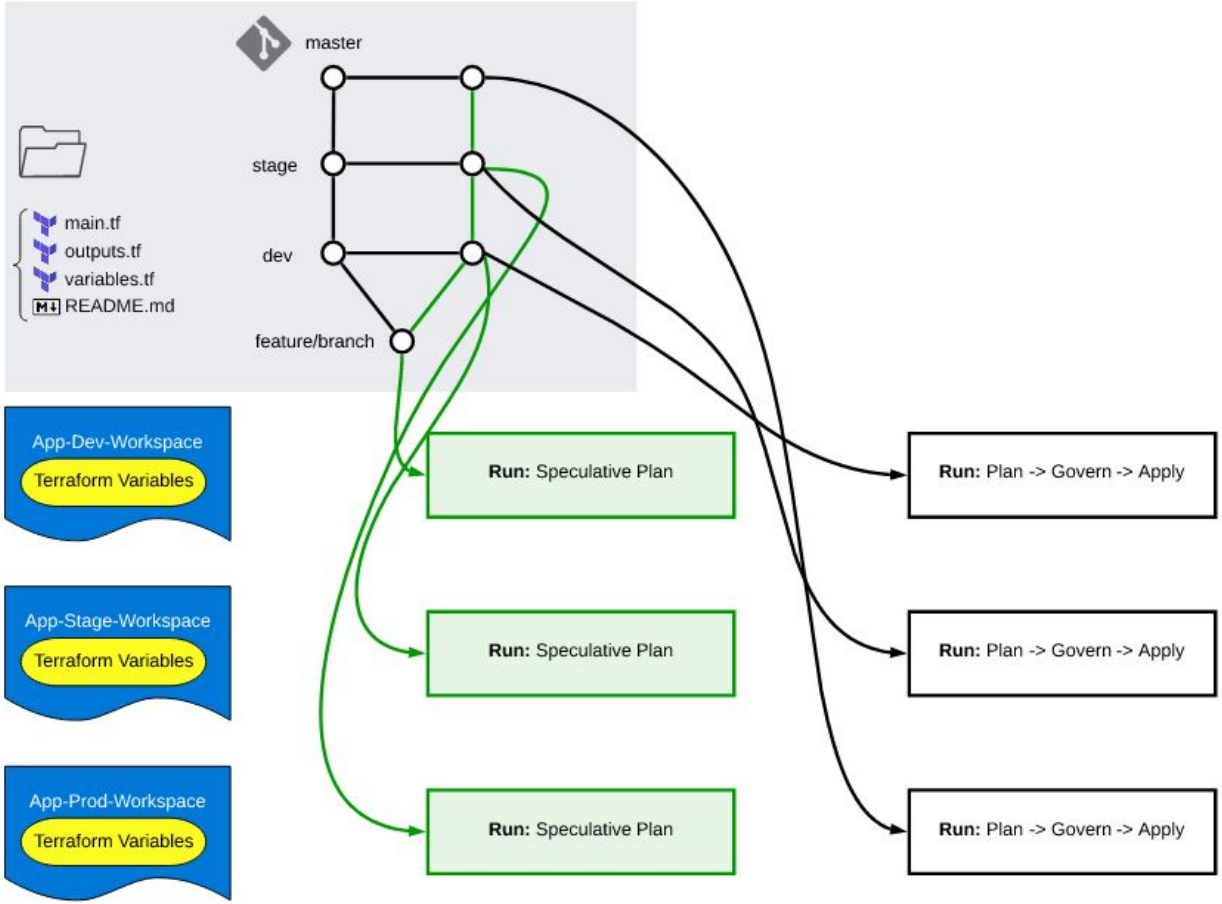

STORAGE TEAM
NETWORK TEAM
APPLICATION X TEAM
R&D TEAM
STORAGE MODULE
Storage Bucket
Storage Class
Logging
Replication
NETWORK REPOSITORY
CIDR Resource
Subnets Resource
Routing Resource
Gateway Resource
APP X REPOSITORY
App - Stack Module Y
Vault Cluster Module
Remote: Network Workspace
R&D REPOSITORY
Cloud Resources: Network, Storage, etc.
New Unique Cloud Services
VERSION CONTROL SYSTEM
BUILD PIPELINE
TERRAFORM ENTERPRISE
Storage Module V1.5
App -Stack Module Y V1.8
App - Stack Module X V2.0
Vault Cluster Module V3.0
Kube Cluster Module V1.0
Compute Module V1.2
PRIVATE MODULE REGISTRY
APPLICATION X WORKSPACE
App - Stack Module Y V1.8
Vault Cluster V3.0
Production
Staging
Development
APPLICATION Y WORKSPACE
Unique Settings
APPLICATION Z WORKSPACE
Unique Settings
R&D WORKSPACE
Networking
Storage
Compute
Security
Database
Unique Cloud Service A
Unique Cloud Service B
Unique Cloud Service C
NETWORK WORKSPACE
Outputs For Consumption
Production
Staging
Development
DATABASE WORKSPACES
Unique Settings
WORKSPACES
SENTINEL POLICIES
Plan
Sentinel Policy Enforcement
Apply
SECURITY TEAM

# Workspace Repo Structure Options

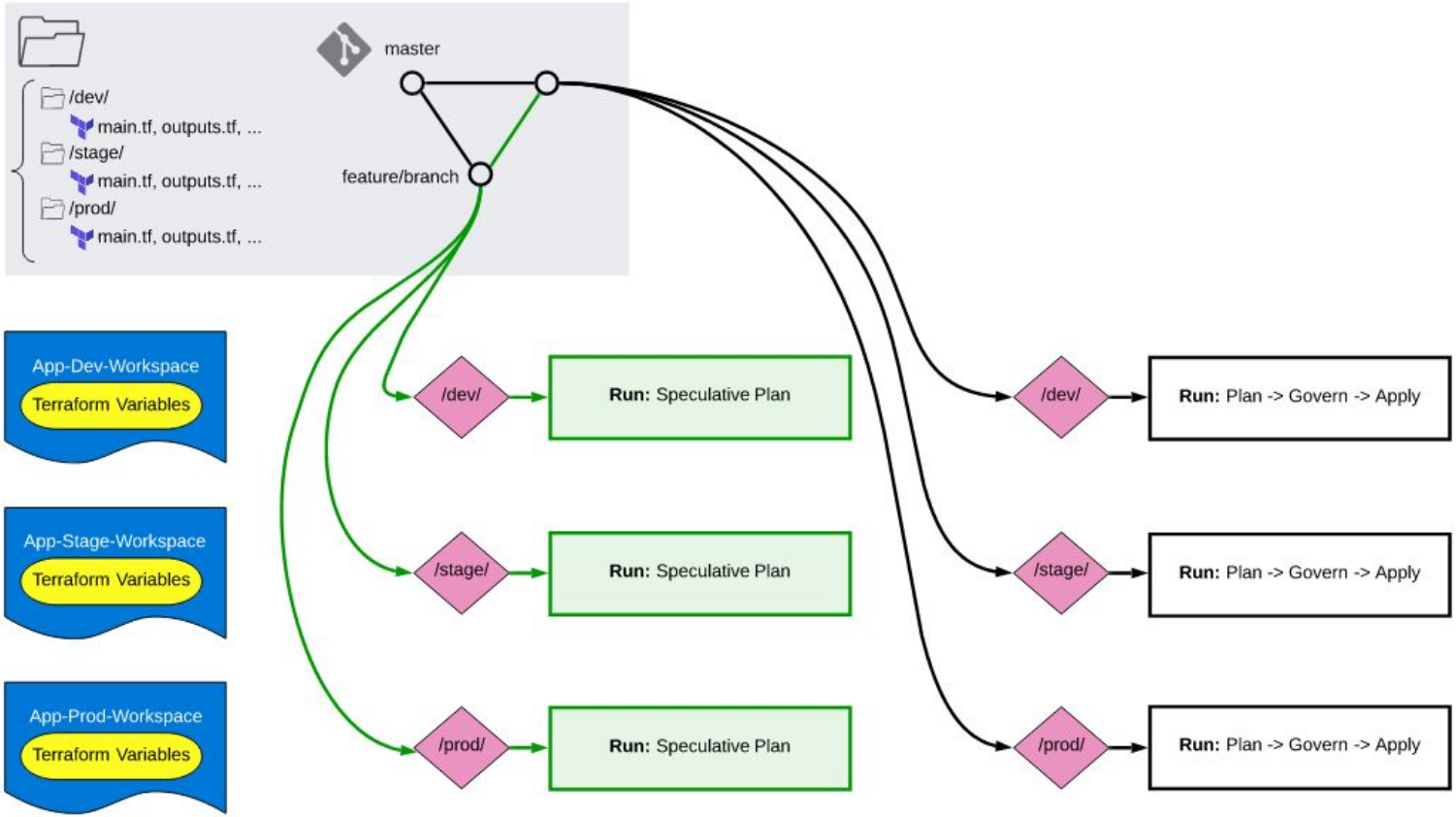
HashiCorp Terraform Enterprise



Terraform Enterprise



HashiCorp Terraform Enterprise



# 1 Terraform Coding Best Practices

## 1.1 Purpose

This document serves as a guidepost for effective development with Terraform across multiple team members and workstreams. Our goal is to provide a unifed and consistent approach to managing code by outlining some rules of engagement. These guidelines should be referenced during code reviews in order to align new development with existing work and to update old code. As a disclaimer, the guidelines found here are not law but rather best practices that we have encountered in our numerous infrastructure-as-code experiences. In cases where new needs require deviation from this guide, we should adjust the guide.

## 1.2 Intended Audience

These standards are targeted towards developers and administrators who will be writing Terraform code on a regular basis.

## 1.3 Terraform Modules

The following guidelines apply to reusable Terraform modules and root configs.

### 1.3.1 Module Structure

1. Terraform modules must follow the standard module structure.
2. Every module must start with a `main.tf` file, where resources are located by default.
    - Among other resources, all locals should be defined here in a single locals block.
3. All modules must have a `README.md` (with basic documentation in Markdown format).
4. Examples should be located in `examples/`, each with its own subdirectory and a `README.md` file
5. Logical groupings of resources can be grouped into their own files and given descriptive names, like `network.tf`, `instances.tf`, or `loadbalancer.tf`.
    - Avoid giving every resource its own file. Resources should be grouped by shared purpose. For example, “google_dns_managed_zone” and“google_dns_record_set” would likely be combined in `dns.tf`.
6. Only Terraform (`*.tf`) and repo metadata files (like `README.md`, `CHANGELOG.md` or `kitchen.yml`) should exist at the root directory of a module.

7. Additional documentation should be stored in a `docs/` subdirectory.

### 1.3.2 Naming Convention

All configuration objects should be named using underscores to delimit multiple words. This practice ensures consistency with the naming convention for resource types, data source types, and other predefined values. Note that this convention does not apply to name arguments.

```
# Good
resource "google_compute_instance" "web_server" {
  name = “web-server”
# ... }

# Bad
resource “google_compute_instance” “web-server” {
  name = “web-server”
#... }
```

### 1.3.3 Variables

1. All variables shall be declared in `variables.tf`.
2. Variables shall have descriptive names relevant to their usage or purpose.
    - Inputs, local variables, and outputs representing numeric values such as disk sizes or RAM size SHOULD be named with units (like ram_size_gb). APIs typically do not have standard units, so naming variables with units makes the expected input unit clear for configuration maintainers.
    - For units of storage, the unit prefix (kilo, mega, giga) SHOULD be binary (powers of 1024). For all other units of measurement, the unit prefix SHOULD be decimal (powers of 1000).
    - Boolean variables SHOULD be named with positive values (like enable_external_access) to simplify conditional logic.
3. Variables must have descriptions. These are automatically included in any published modules' auto-generated documentation through terraform-docs. Descriptions add additional context for new developers that descriptive names cannot.
4. Variables should have defined types.
5. Variables with non-environment-specific values (like disk size) should be given default values.
6. Variables for environment-specific values (like project_id) should not be given defaults. This forces the calling module to provide meaningful values.

7. Variables should only have empty defaults (like empty strings or lists) where leaving the variable empty is a valid preference which will not be rejected by the underlying API(s).
8. Be thoughtful in your use of static literals (hardcoded strings, etc.) and parameterize anything which must vary per instance or environment. Local values can be used in cases where a literal is reused in multiple places without exposing it as a variable.
9. When deciding whether to expose a variable, ensure that you have a concrete use case for changing that variable. Don't expose variables on the off chance that it's needed.
    - Adding a variable with a default value is backwards compatible, and thus "cheap."
    - Removing a variable is backwards incompatible, and thus "expensive."
10. Boolean variables should be explicitly declared as strings until Terraform adds further support for boolean input variables.

#### 1.3.4 Outputs

1. All outputs shall be organized into `outputs.tf`.
2. Outputs should have meaningful descriptions.
3. Output descriptions should be documented in the README. Descriptions should also be auto-generated on commit with terraform-docs.
4. Make an effort to output all the useful values root modules would want to reference or share with modules. Particularly for open source or heavily used modules, expose all outputs that have potential for consumption.

#### 1.3.5 Data Sources

1. Data sources are located adjacent to the resources which reference them.

- If you are fetching an image to be used in launching an instance, you can place it alongside the instance instead of collecting data resources in their own file.

2. If the number of data sources grows considerably, it's a reasonable practice to move these to a dedicated `data.tf` file.
3. Data sources should use variable or resource interpolation, where appropriate, to fetch data relative to your current environment.

#### 1.3.6 Scripts (Called by Terraform)

1. Bespoke scripts can be called by Terraform through provisioners, including the local-exec provisioner.

2. Custom scripts should be avoided, if possible, and constrained to instances where native Terraform resources do not support the desired behavior. Any custom scripts used must have a clearly documented reasoning and ideally a deprecation plan. Additionally, custom scripts should not replace configuration management tools in cases where they are more appropriate.
3. Bespoke scripts called by Terraform must be organized into `scripts/`.
4. Use scripts only when absolutely necessary, as the state of resources created through scripts is not accounted for or managed by Terraform. You'll likely want to add a policy of `ignore_changes = [*]` on such resources.

#### 1.3.7 Helper Scripts (not called by Terraform)

1. Helper scripts should be organized in a `./helpers` directory.
2. Helper scripts should be documented in the README with an explanation and example invocations.
3. Helper scripts accepting arguments should provide argument-checking and `--help` output.

#### 1.3.8 Static Files

1. Static files which are referenced by Terraform (like Startup scripts loaded onto instances) but not executed must be organized into `files/`.
2. Lengthy heredocs should be externalized from their HCL and into external files. These should be referenced with the `file()` function.

#### 1.3.9 Templates

1. Files which are injected with the Terraform `template_file` resource should be given the file extension `.tpl`.
2. Templates must be placed in `templates/`.

#### 1.3.10 Resources

1. Resources that are the only one of their type (i.e., a single load balancer for an entire module) should be named 'main' to simplify references to that resource. It takes extra mental work to remember: `some_google_resource.my_unique_name.id` vs. `some_google_resource.main.id`
2. Resources that share the same type as others in the same module should be given meaningful names to differentiate them.

3. Resources must be named in snake-case (like db_instance).
4. Resource names should be singular.
5. Resource names shouldn't repeat the resource type within the name. For example: Do this: `resource "google_compute_global_address "main" {...}` Not this: `resource "google_compute_global_address" "main_global_address" { ...}`
6. Ensure that deletion protection is enabled for stateful resources like databases. For example:

```
resource "google_sql_database_instance" "main" { name = "master-
instance"
settings {
tier = "D0" }
  lifecycle {
    prevent_destroy = true
}
}
```

#### 1.3.11 Formatting

1. All Terraform files must conform to the standards of `terraform fmt`.

#### 1.3.12 Expressions

1. Limit the complexity of any individual interpolated expressions. If many functions are needed in a single expression, consider splitting it out into multiple expressions using locals.
2. Never have more than one ternary operation in a single line. Instead, use multiple local values to build up the logic.
3. Be sparing when using user-specified variables to set the `count` variable for resources. If a resource attribute is provided for such a variable (like `project_id`) and that resource does not yet exist, Terraform will not be able to generate a plan and will report the error "value of count cannot be computed."
4. Use count to instantiate a resource conditionally. For example:

```
variable "readers" {
    description = "..."
    type = "list"
    default = []
}
```

```
resource "foo" "bar" {
    // Do not create this resource if the list of readers is empty.
    count = "${length(var.readers) == 0 ? 0 : 1}"
...
}
```

## 1.4 Common Modules

Modules that are meant for reuse should follow the following standards, as well as the normal Terraform guidelines.

### 1.4.1 Structure

1. All common modules should have an OWNERSfile (or CODEOWNERS on GitHub) documenting who is responsible for the module.
2. Common modules should follow SemVer v2.0.0 when new versions are tagged/released.
3. Modules must not declare providers or backends. Leave that to the root modules.
    - Working examples should codify if a specific provider version is needed for a given module.

### 1.4.2 Variables

It's a good practice to allow flexibility in the labelling of resources through the module's interface. Consider providing a `labels` variable with a default value of an empty map to apply throughout labelable resources:

```
variable "labels" {
description = "A map of labels to apply to contained
resources."
default = {} type = "map"
}
```

### 1.4.3 Outputs

Outputs are required for common modules that define resources.

- Variables and outputs are used to infer dependencies between modules and resources. Without any outputs, users cannot properly order your module in relation to their Terraform configurations.

- Every resource defined in a common module should have at least one output which references that resource.

### 1.4.4 Inline modules

1. Inline modules may be used to organize complex Terraform modules into smaller units, or de-duplicate common resources.
2. Inline modules shall be placed in `modules/$modulename`.
3. Inline modules should be treated as private and should not be used by outside modules, unless the common module specifically documents them otherwise.
4. Be aware that Terraform doesn't track refactored resources; if you start out with a number of resources in the top level module and then push them into submodules,Terraform will try to recreate all refactored resources.
5. Outputs defined by internal modules are not automatically exposed; if you want to share outputs from internal modules you'll need to re-output them.

## 1.5 Root Configs

Root configs, or root modules, are the working directories from which you run the Terraform CLI. They should follow the following standards, as well as the normal Terraform guidelines where applicable. Explicit recommendations for root modules supersede the general guidelines. Resources for different applications and projects should be separated into their own Terraform directories that can be managed independently of each other. A service might represent a particular application or a common service like shared networking. Importantly, all the Terraform code for a particular service should be nested under one directory (including subdirectories).

### 1.5.1 Directory Structure

There are multiple ways to organize Terraform root configurations, especially when it comes to managing multiple environments. When it comes to managing the Terraform config for a particular service, the recommended structure is to use environment directories.

***Directories per environment*** In this style, each service must split its Terraform config into multiple directories. In this structure, the directory layout must be as follows:

```
-- SERVICE-DIRECTORY/
   -- OWNERS
```

```
   -- modules/
      -- service/
         -- main.tf
         -- variables.tf
         -- outputs.tf
         -- README
-- ...other... -- environments/
      -- dev/
         -- backend.tf
         -- main.tf
         -- provider.tf
      -- qa/
         -- backend.tf
         -- main.tf
         -- provider.tf
      -- prod/
         -- backend.tf
         -- main.tf
         -- provider.tf
```

Environment directories Each environment directory within corresponds to a Terraform Workspace and deploys a version of the service to that environment. This config should reference modules to share code across environments, including typically a service module which includes the base shared Terraform config for the service.

This environment directory must contain the following files: * A `backend.tf` file declaring the Terraform backend state location (typically GCS). * A `main.tf` file which instantiates the service module. * A `provider.tf` file which declares provider configuration.

***Workspaces per environment*** Alternatively, a single Terraform directory can be used per service and shared across environments. Each environment would have its own workspace. When using workspaces, all environments share the same modules, and the configuration is driven by a tfvars file and a workspace. Workspaces are helpful in that they limit the amount of code that must be copy-pasted between environment directories, which can help enforce parity between environments while maintaining their own state files. By default a single workspace named “default” exists. It is recommended to create and use a workspace for each environment and use the “default” workspace only when working with resources that may be used across multiple environments like some service accounts.

### 1.5.2 Outputs

1. Information from a root module which other root modules may depend on must be exported as outputs.
2. Root module outputs can be referenced using remote state.

***Publishing outputs with remote states***

1. Make sure to re-output nested module outputs that are useful as remote state. Only root module-level outputs can be referenced from other Terraform environments/applications.
2. Information related to a service's endpoints should be exported to remote state to allow use by other dependent apps for configuration.

## 1.6 Versioning

### 1.6.1 Terraform

Terraform v0.12 is a significant release that will include some backwards incompatibilities. Pin the Terraform version (example given below) to a known safe version until v0.12 has been released and stabilized.

```
terraform {
required_version = "~> 0.11.10"
}
```

### 1.6.2 Providers

Pin any providers to a known good version, and make updating the version pin a regular practice.

```
provider "google" { version = "~> 1.19.1"
}
```

### 1.6.3 Modules

References to shared modules must be constrained to a release tag. Targeting a specific commit hash or branch is dangerous as it gives no context to the version of the underlying module. Updating modules should involve as little guesswork as possible for both authors and reviewers.

### 1.6.4 Constrain by git reference

References to shared modules may be constrained to any arbitrary git reference (commit, branch, or tag). For reasons outlined above, we only recommend using this to reference tags:

```
module "vpc" {
  source = “git::https://github.com/terraform-google-modules/terraform-
google-network?ref=v0.4.0”
... }
```

### 1.6.5 Constrain by version

When a git tag is released to the Terraform Module Registry, it creates a numbered version of that module (note that this does not apply to Github repositories, only to modules released to registries). An invocation can be constrained to said version:

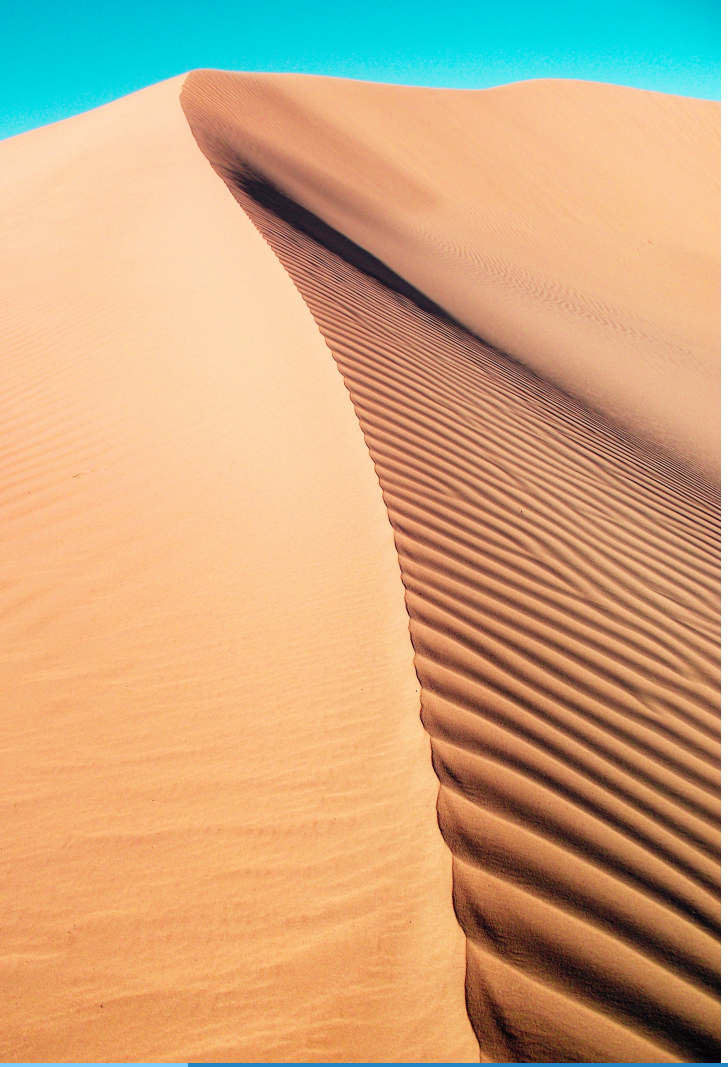
```
module "nat_gateway" {
  source = “GoogleCloudPlatform/nat-gateway/google”
  version = “1.2.2”
  ...
}
```

# Exploring Modules and Testing

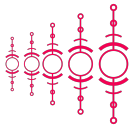
# Hello!

## Exploring Modules and Testing

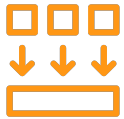
Don't Repeat Yourself (DRY) is a key aspect of programming for applications and infrastructure. Just as important is testing your code works as intended.



# DRY (Don't Repeat Yourself)



Reduce redundancy

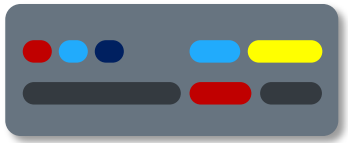


Create abstractions

Increase reusability

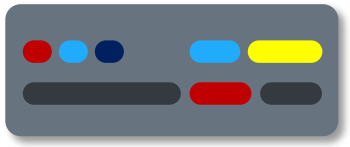
| Fruit |
| --- |
| Grapes |
| Apples |
| Oranges |
| Bananas |
| Pears |

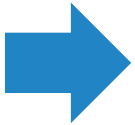
sorting_script.sh



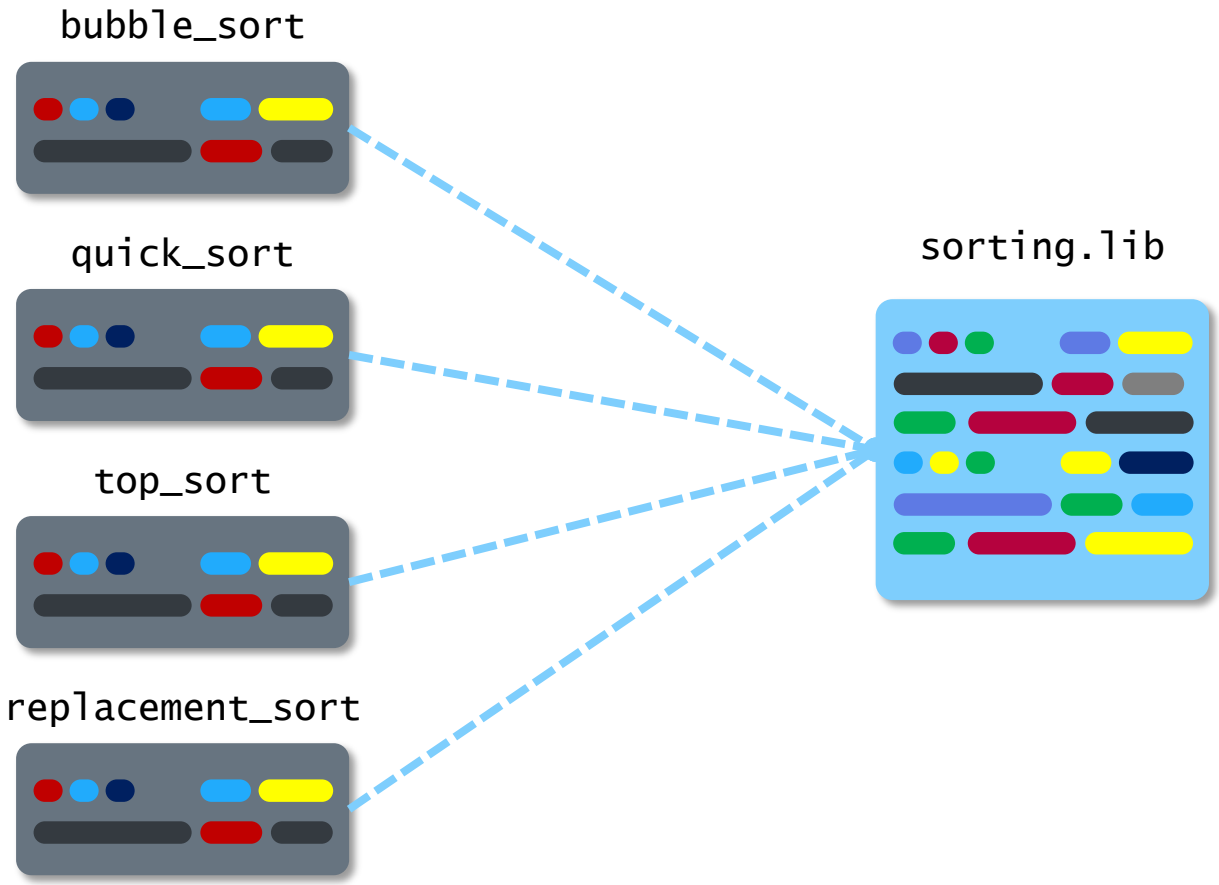
| Fruit |
| --- |
| Apples |
| Bananas |
| Grapes |
| Oranges |
| Pears |

sorting_script.sh

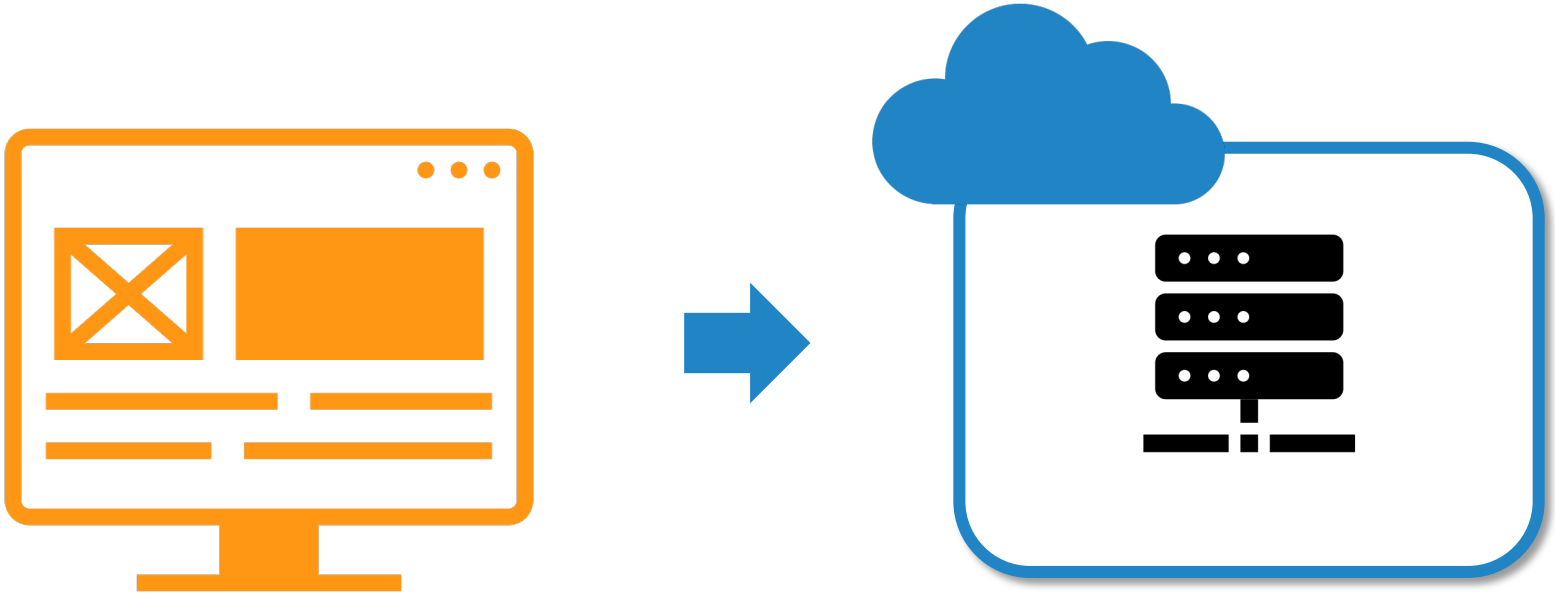


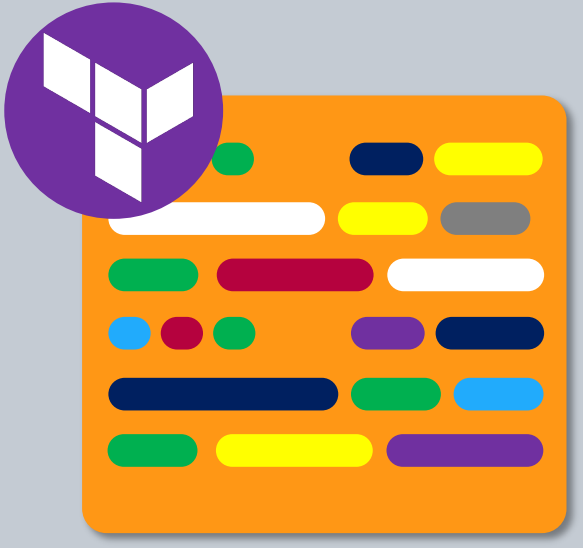


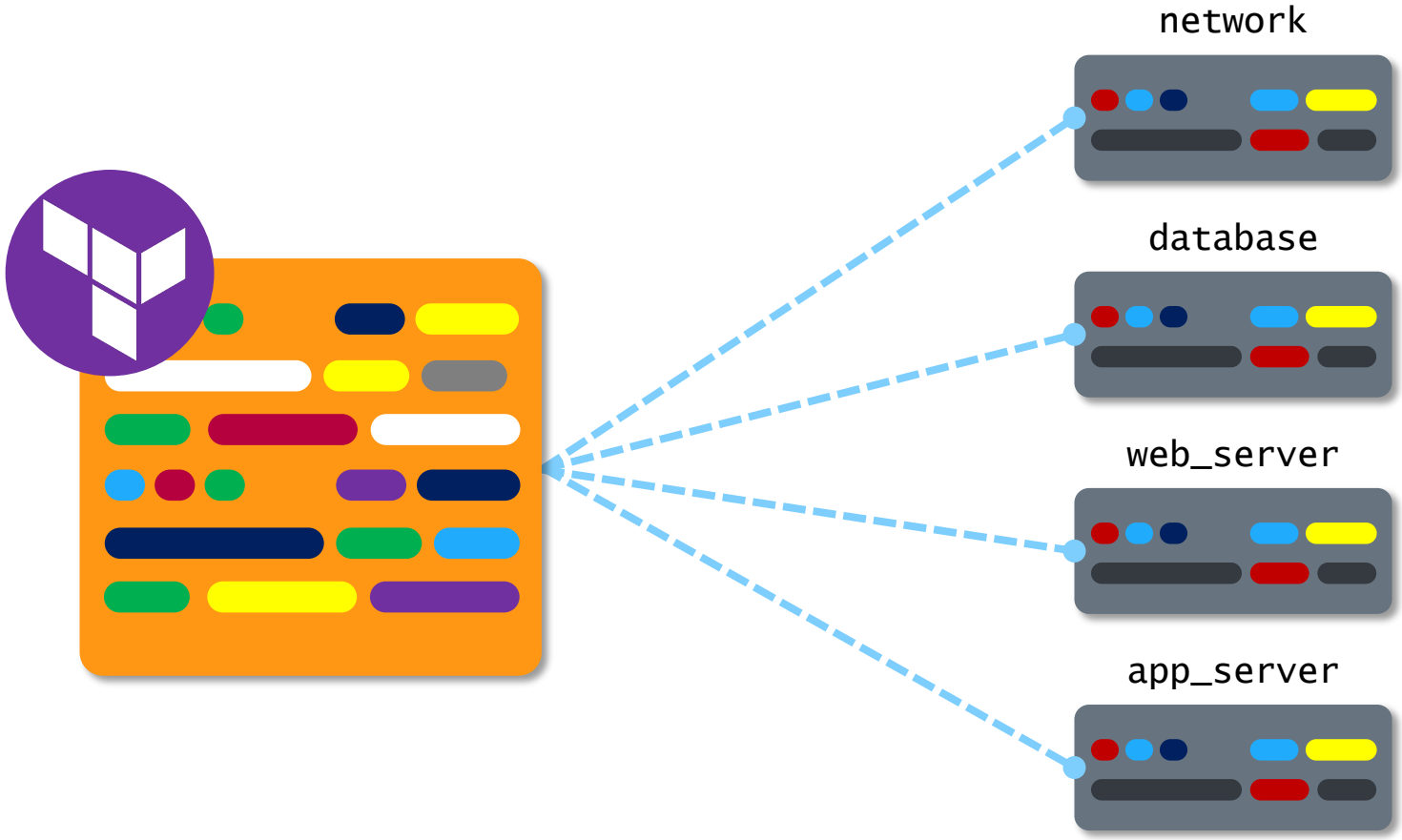
```
function sort(items)
{
  # Code to sort items

  ...
  ...

  return sorted_items;
}
```

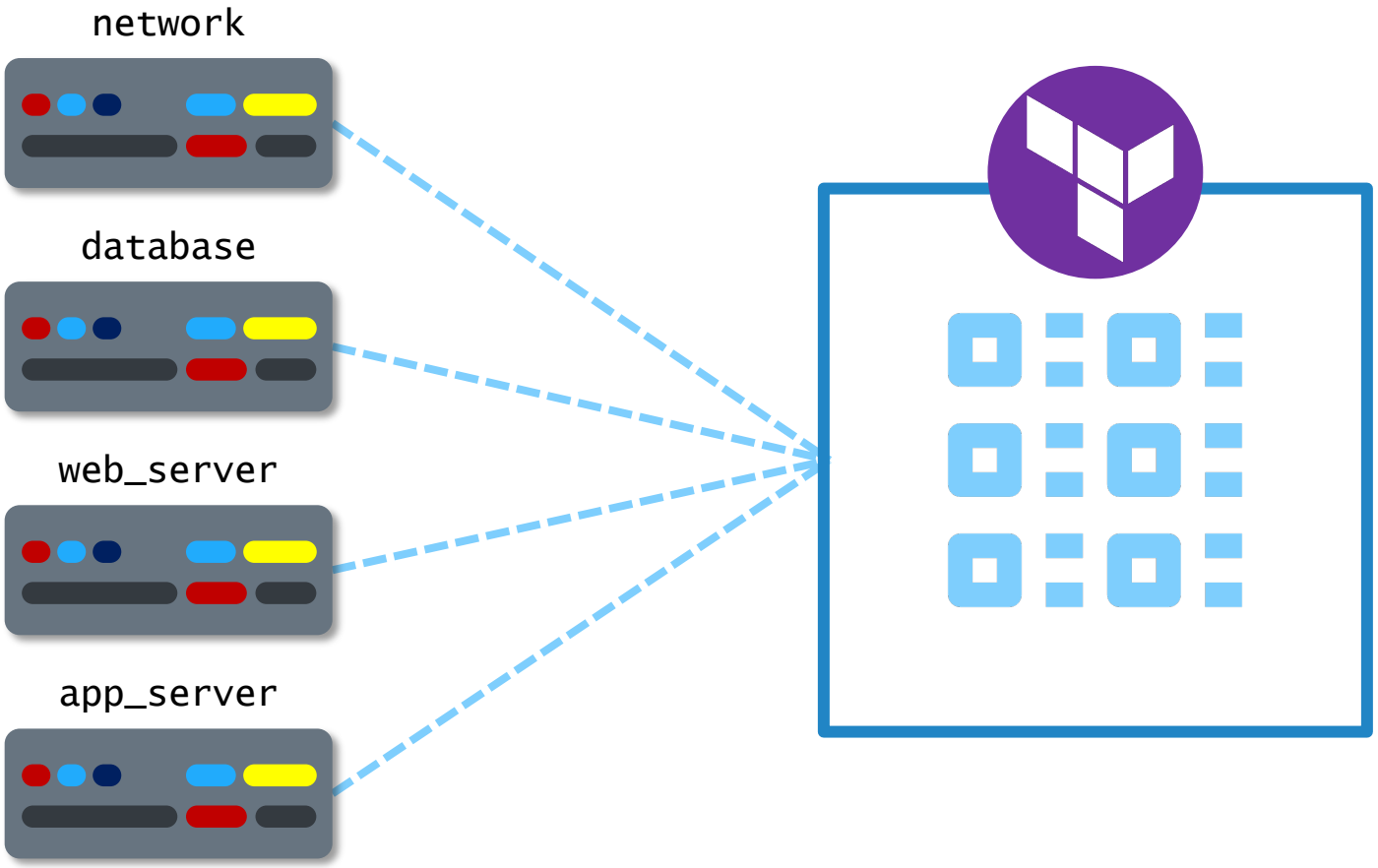

bubble_sort
quick_sort
top_sort
replacement_sort
sorting.lib

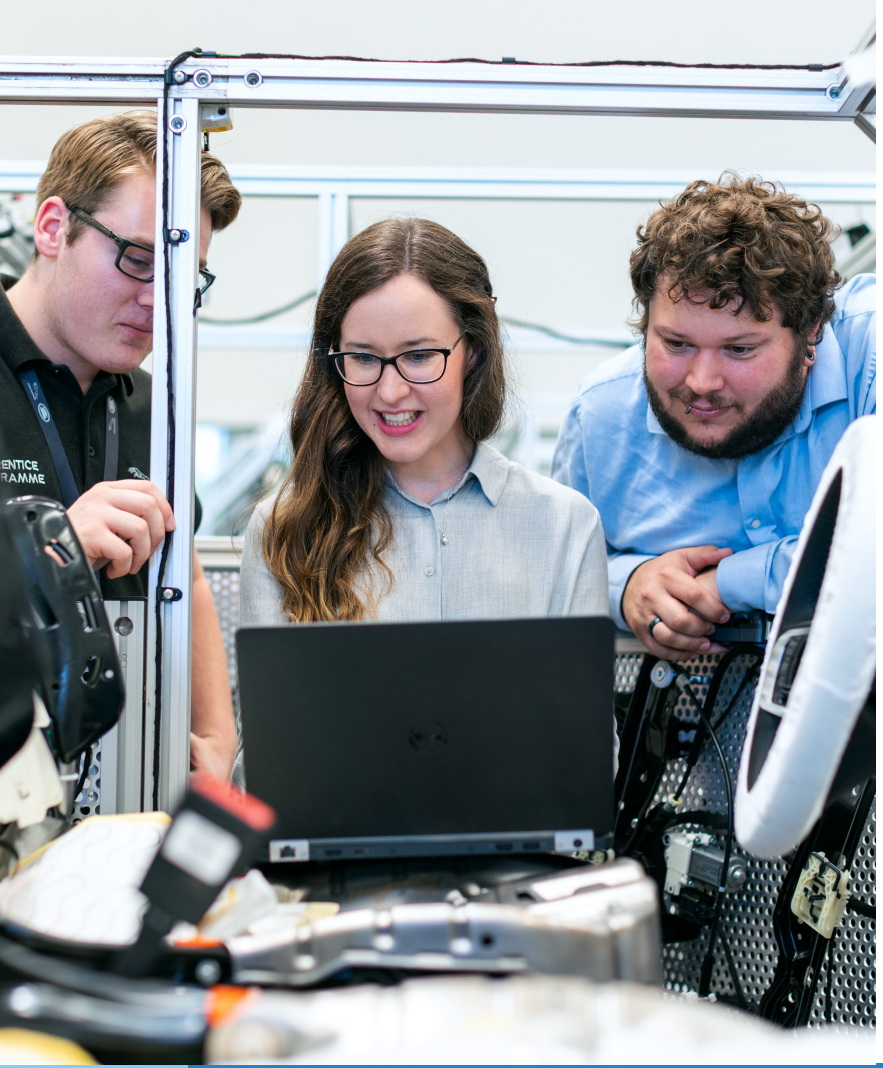
# Infrastructure Deployment






network
database
web_server
app_server

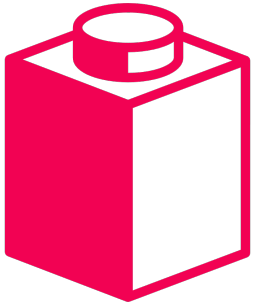

network
database
web_server
app_server



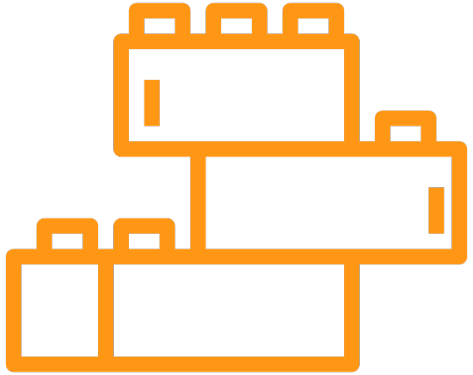
# Testing

- Maintenance of code
- Breaking changes
- Use by others

# Test Types

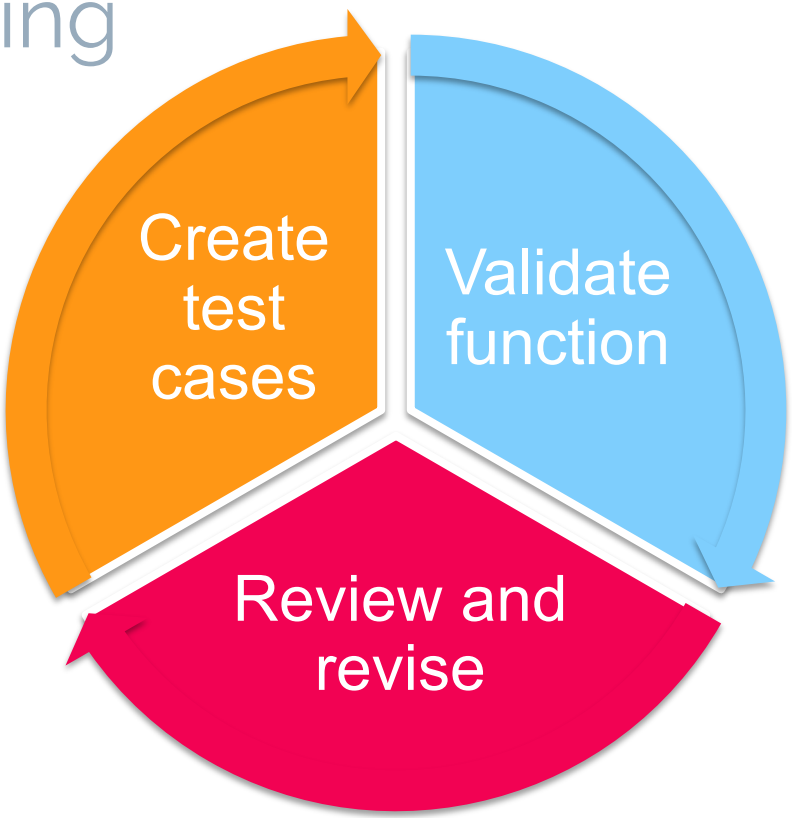


Unit testing



Integration testing

# Unit Testing



# Integration Testing

Units work together

Longer than unit test

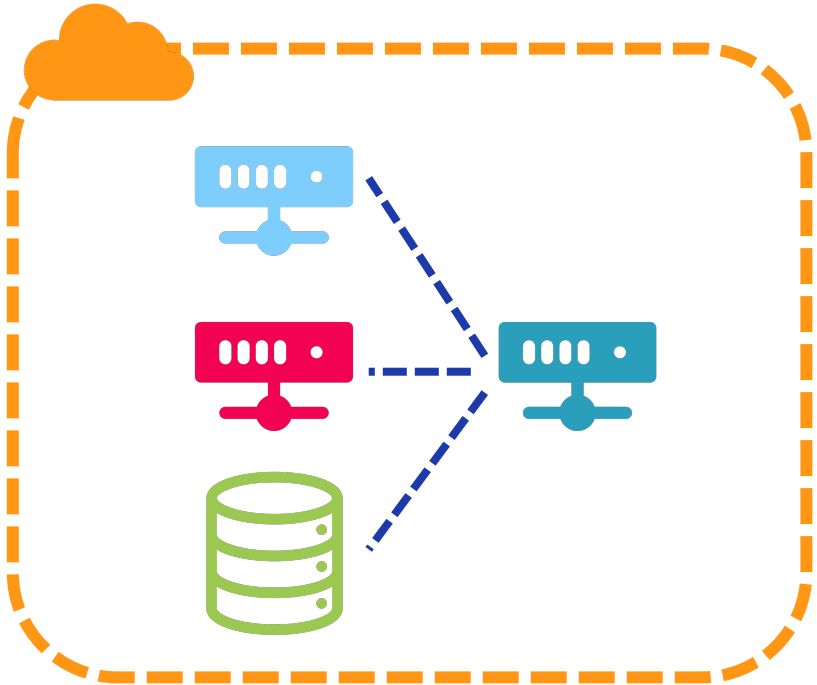
Test in parallel

Retry on flakes

# Infrastructure as Code

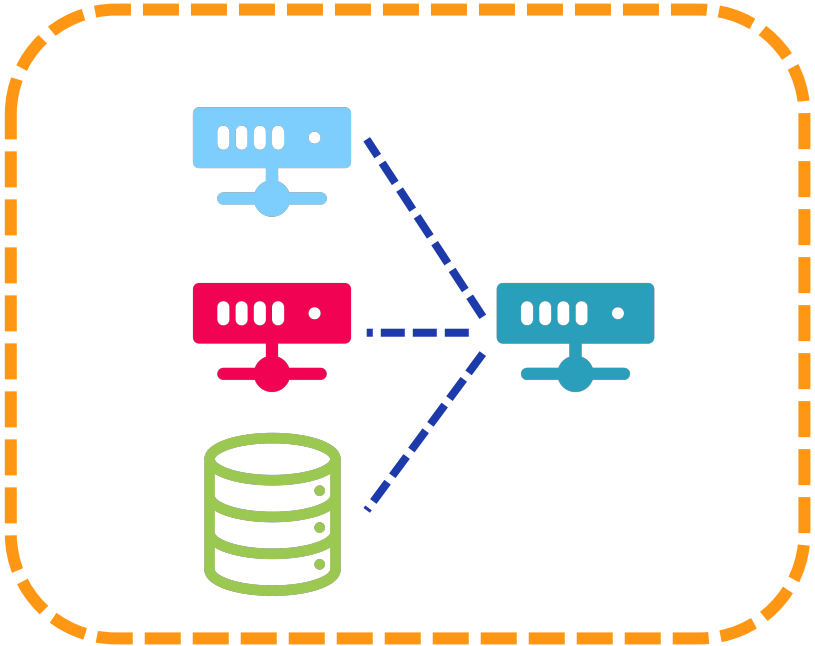
- Actual resources deployed for testing
- Time consuming for large configurations
- Unit testing handled by providers
- Test desired functionality of infrastructure

# Module Testing



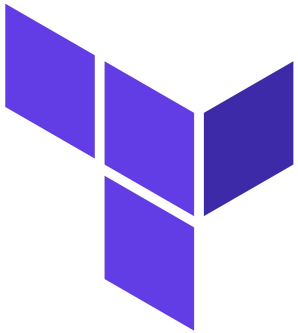
```
.
├── main.tf
├── app_server
│   ├── main.tf
│   └── test
│       └── test.go
├── db_instance
│   ├── main.tf
│   └── test
│       └── test.go
└── web_server
    ├── main.tf
    └── test
        └── test.go
```

# Module Testing



```
web_server:
  - port: 443
  - status: 200

application_server:
  - port: 443

database_instance:
  - port: 3306
  - query: master_db
```

# Testing Tools

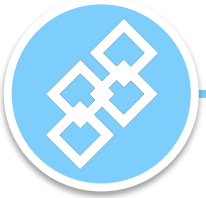


Terragrunt

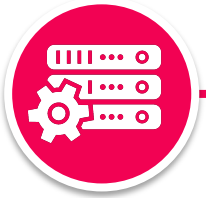
Molecule


Checkov

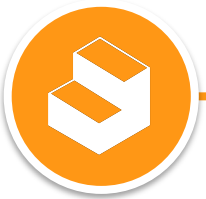
# End-to-End Testing



Test everything together



Dedicated environment



Incremental changes

Validate changes only

# Summary

- Practice DRY when writing code for apps or infrastructure
- Infrastructure can be abstracted with modules and playbooks
- Testing is critical to dependable code
- IaC testing requires more resources and time

# API Driven Workflows

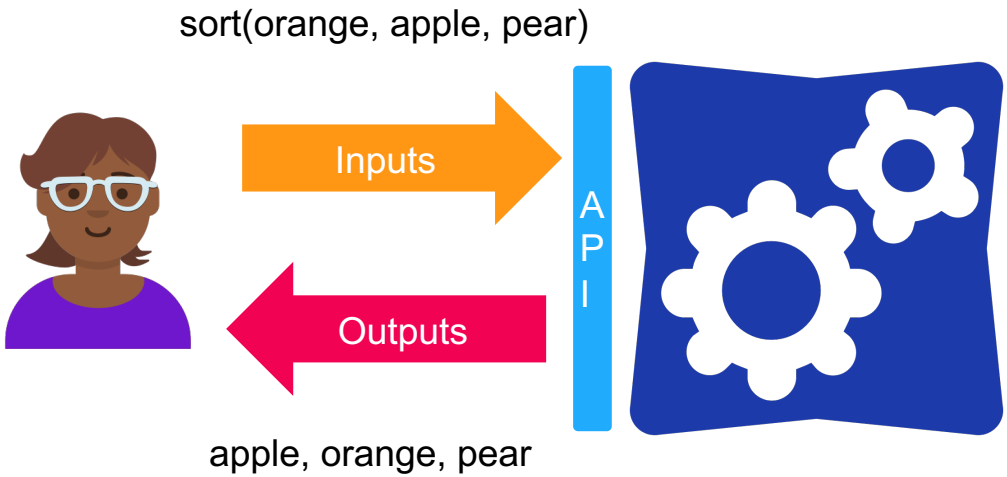
# Hello!

## API Driven Workflows

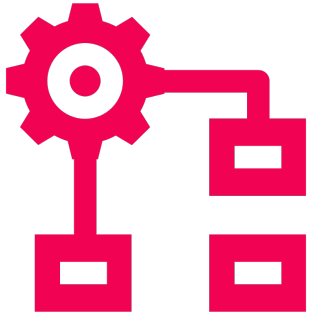
Orchestrating the process of creating  and maintaining infrastructure by leveraging APIs to perform actions.

An Application Programming Interface is a way to interact with software using well-defined instructions. Using APIs enables the automation of manual tasks.

# Application Programming Interfaces (APIs)



# Types of APIs



RPC

REST

GraphQL

# API Interaction

- Direct access
- Graphical Interface
- Precompiled tools
- Software development kits

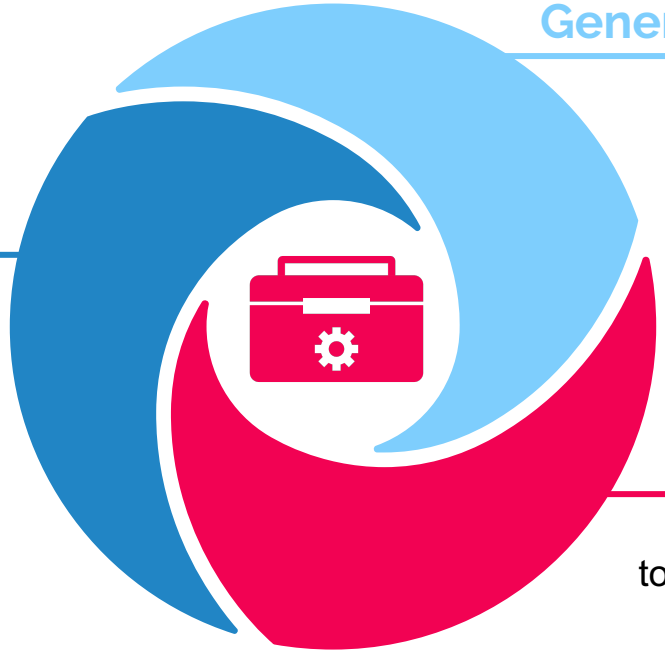
# Software Development Kits

## General Purpose Language

SDKs provide libraries for general purpose programming languages to use.

## Use Common Tools

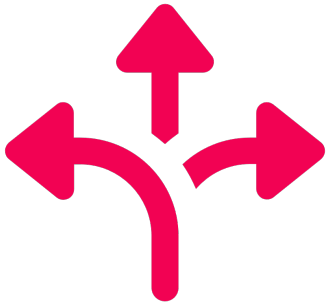
Leverage all the common programming tools like testing, linting, and scanning.



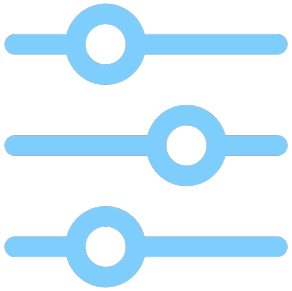
## Multiple APIs

A single SDK can stitch together multiple APIs into a unified library.

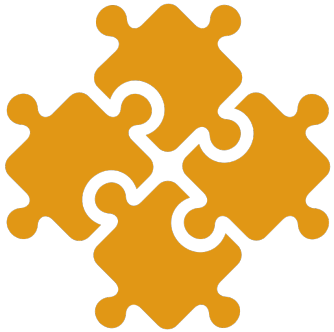
# API Driven Workflow Benefits



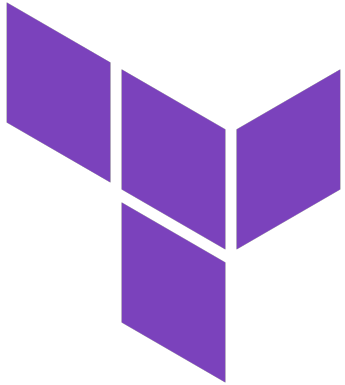
Flexibility



Customization



Integration



HashiCorp

# Terraform

# APIs in Terraform

Terraform Providers

Written in Go

Target SDK or API

Terraform Cloud

Public API

Provider in Terraform

Web Interface

Integrations

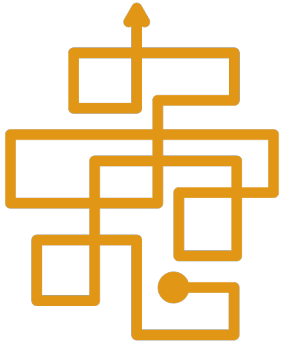
# Terraform Cloud Workflows

Command line interface

Version control system

Application programming interface

# API Workflow

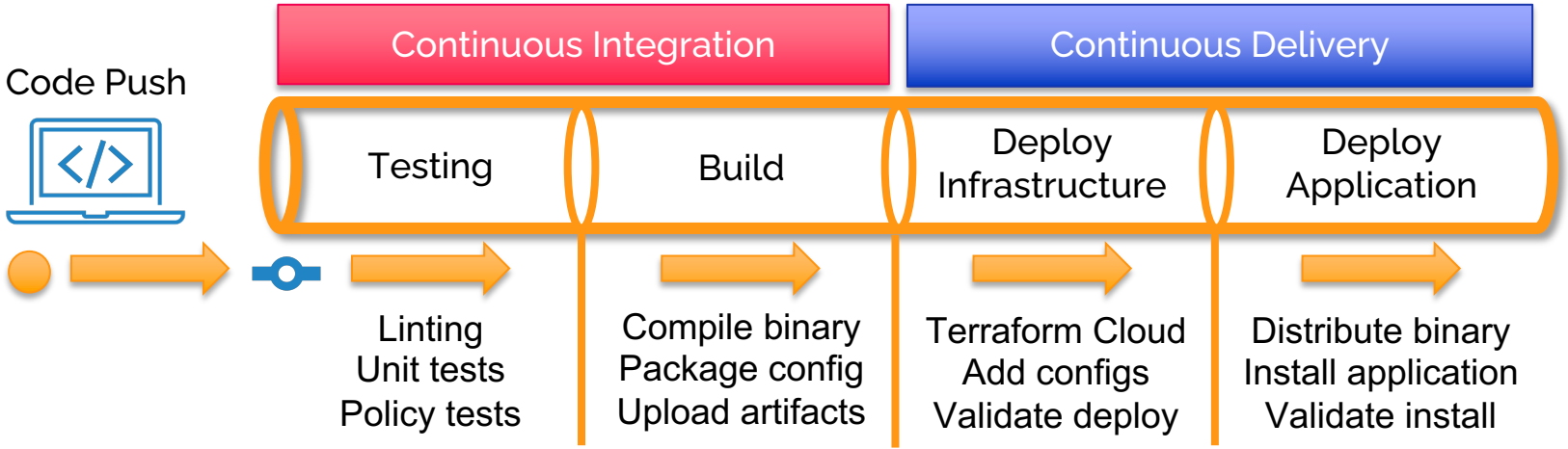


Most complicated and flexible


Bundle configurations in tarball

Relies on external orchestration

# API Workflow



# Summary

- Application programming interfaces are a contract between the client and the app
- APIs can be used in an automation workflow to interact with other services and applications
- Terraform Cloud and Enterprise have APIs that can integrate with your existing workflow