Terraform 201

Resource Lifecycles

What You'll Learn

- How Terraform creates and destroys resources
- How to diverge from the default behavior
- What code organization techniques go with which lifecycle behavior

Default Behavior

MODIFY

When possible

EXAMPLES

Add a tag
Change SSH user (GCP)

DESTROY

Freely

EXAMPLES

Rename an instance
Change machine type
Decrease count
Refactor to modules
Change startup script

Do Nothing

Occasionally

EXAMPLES

Change provisioner
Add an output
Refactor to variables

terraform plan

TERMINAL

```
$ terraform plan

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
 ~ update in-place
-/+ destroy and then create replacement

Terraform will perform the following actions:

 ~ aws_instance.example
     vpc_security_group_ids.#:"" => <computed>

-/+ aws_security_group.training (new resource required)
     id: "sg-6d36fc1c" => <computed> (forces new resource)
```

⚠️ Important :

Destroy, Then Create

Existing resources are first destroyed before new ones are created.

If the destruction succeeded cleanly, then and only then are replacement resources created.

Dependency Errors

TERMINAL

```
$ terraform apply

aws_security_group: Still destroying... (14m00s elapsed)
aws_security_group: Still destroying... (14m10s elapsed)

Error: Error applying plan:

1 error(s) occurred:

* aws_security_group.training (destroy): 1 error(s) occurred:

* aws_security_group.training: DependencyViolation: resource sg-6d36fc1c has a
dependent object
  status code: 400, request id: 4665247f-165b-46fc-b8ea-9c01a23dd4e9
```

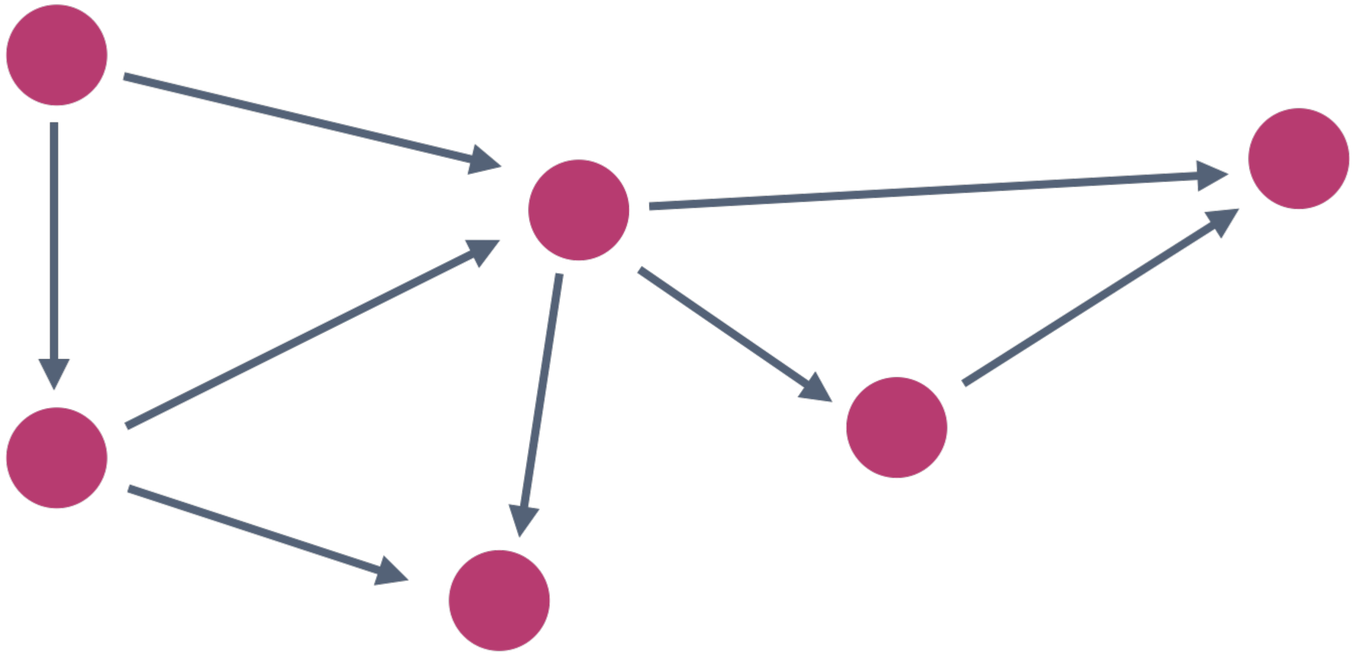
The lifecycle **block**

To control dependency errors

CODE EDITOR

```
lifecycle {
  create_before_destroy = true
  prevent_destroy = true
}
```

Improved Application of Graph Theory



Partial Applies Will Clean Up On The Next Apply

When using create_before_destroy, it’s possible to encounter a runtime error where Terraform cannot complete the apply.

This could leave both the old and the new instances intact.

The old instance is stored in terraform.tfstate under a deposed key. It will be destroyed after the next successful apply.

Prevent Destroy

Warns if any change would result in destroying a resource

All resources that this resource depends on must also be set to prevent_destroy

CODE EDITOR

```
lifecycle {
  prevent_destroy = true
}
```

prevent_destroy error

Here's an example of the error message you'll see if you try to destroy a resource protected by `prevent_destroy`.

TERMINAL

```
Error: Error running plan: 1 error(s) occurred:

* google_compute_instance.test: 1 error(s) occurred:

* google_compute_instance.test: google_compute_instance.test: the plan would destroy this resource, but it currently has lifecycle.prevent_destroy set to true. To avoid this error and continue with the plan, either disable lifecycle.prevent_destroy or adjust the scope of the plan using the -target flag.
```

Lab 14: Lifecycles

In this lab, we will demonstrate how to apply both of the lifecycle directives that we've discussed so far

We will be creating a few new resources and then protecting them.

You should see a few different scenarios to help hammer home the concepts that we have discussed so far

Terraform 201

Code Organization for Teams

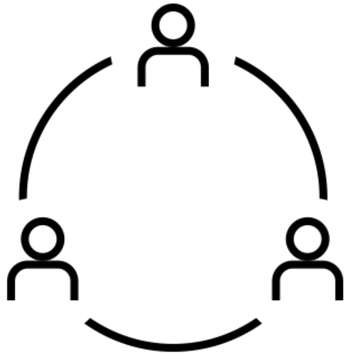
What You'll Learn

How Terraform projects evolve

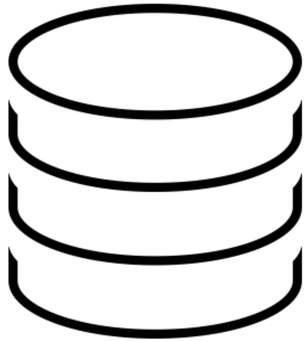
When to use modules vs a new project

How to use shared state to manage complexity

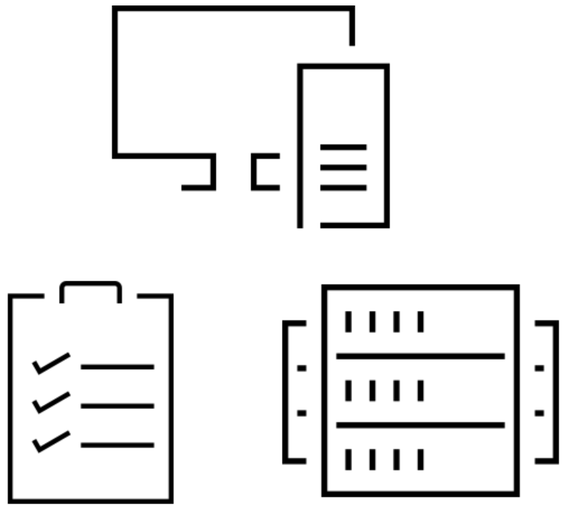
Ways a Project Grows



Larger Team



More Resources



Dev, Test, Staging, Prod

Discussion

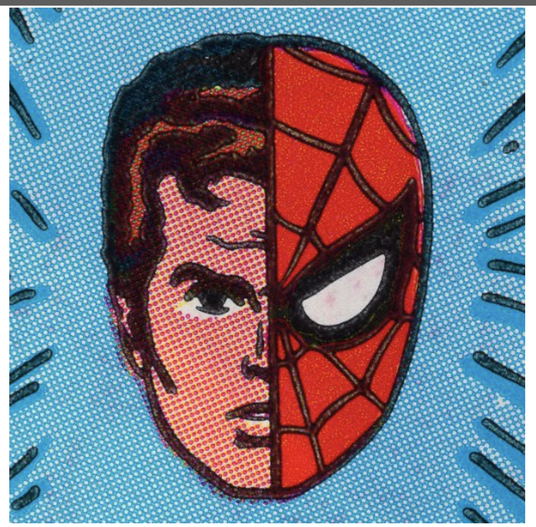
- *Is this config owned by one team?*
- *Is this config used by more than one team?*
- *Does this config create resources?*

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.

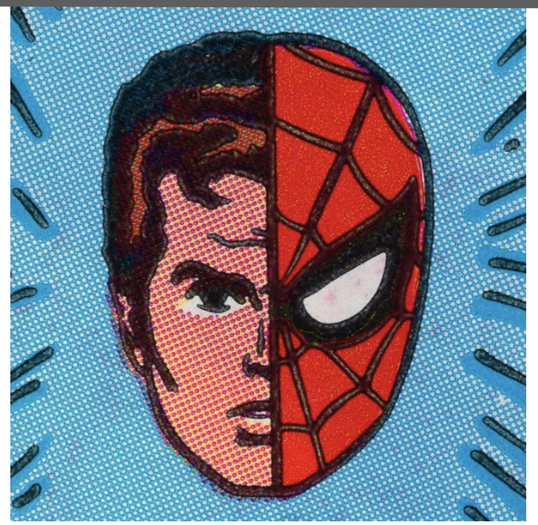
- Start simple. Refactor to manage technical or organizational complexity.

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

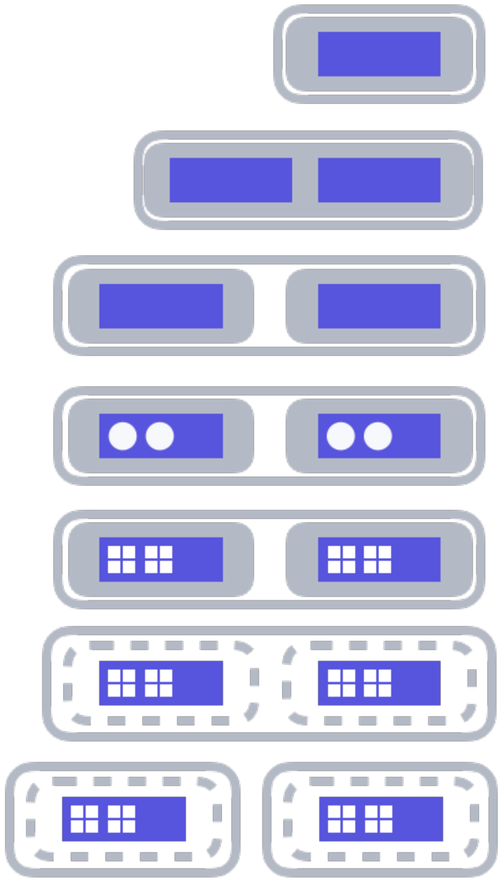
Versioned Modules

Many Repositories

Limited Configuration

Limited Access

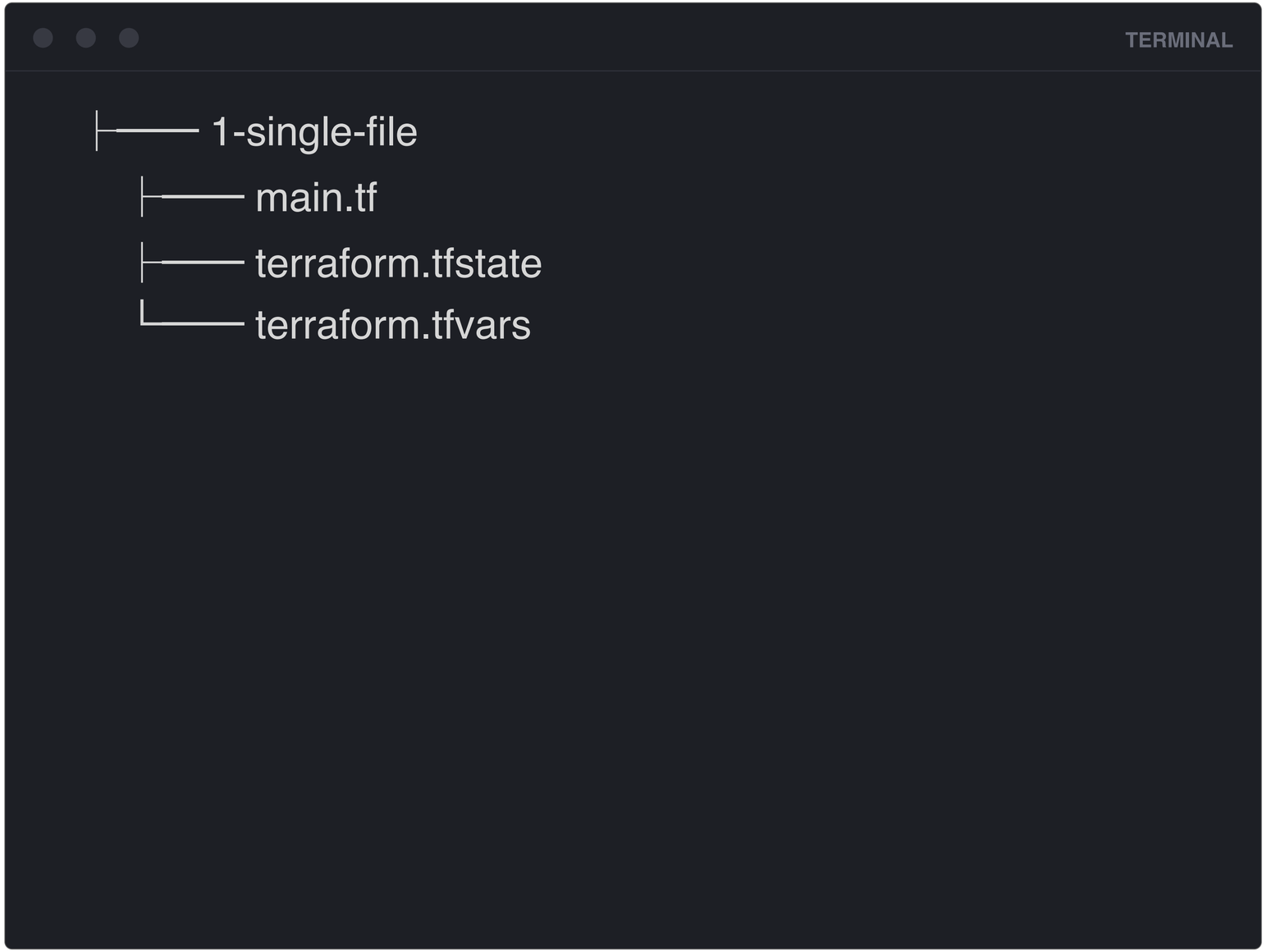
Evolution of a Terraform Project



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



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

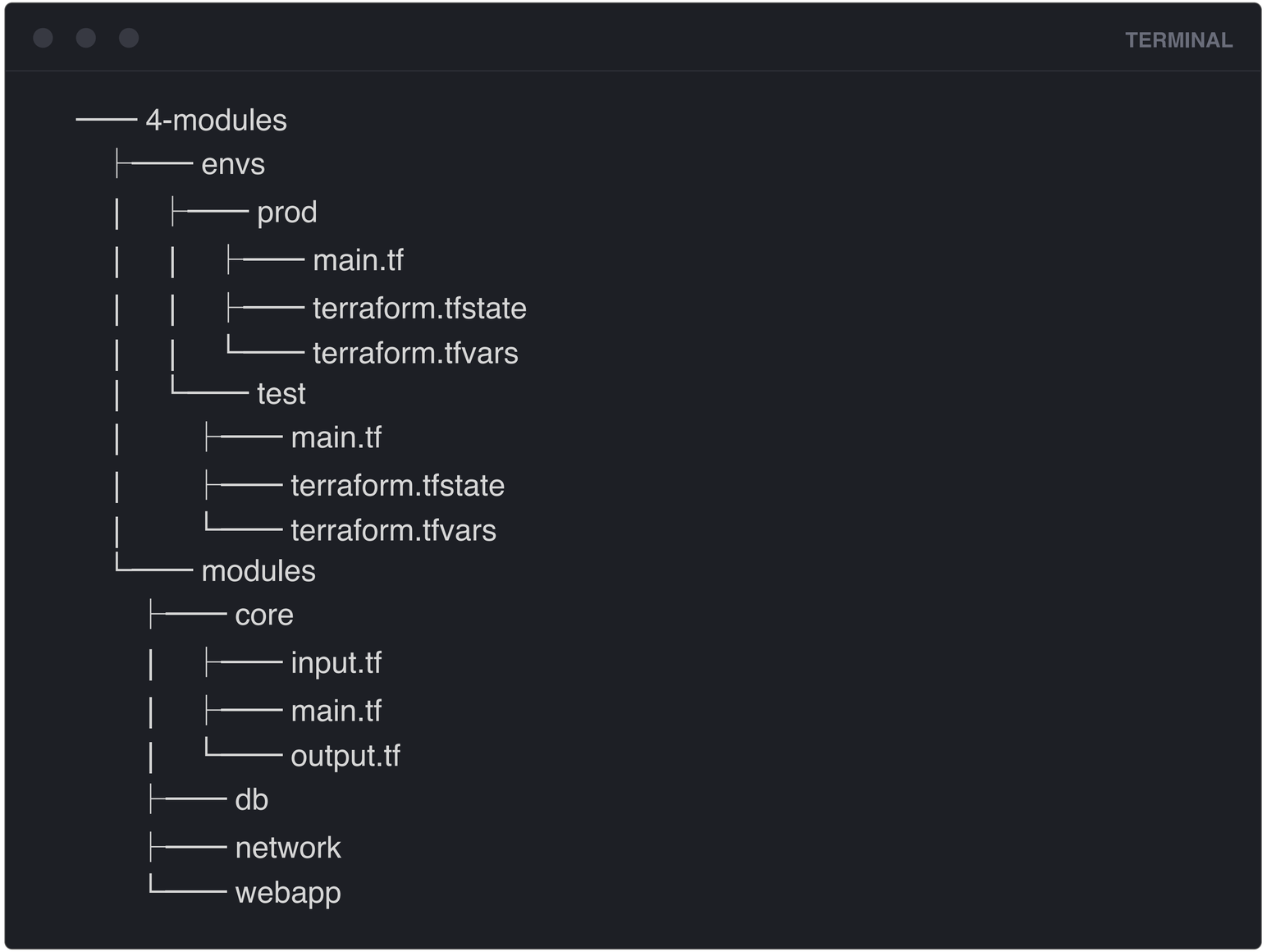
Provides ability to isolate resources and secrets.

Resources in Modules

Improves long term maintenance

Eases reuse

Encourages consistency in design



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.

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.

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

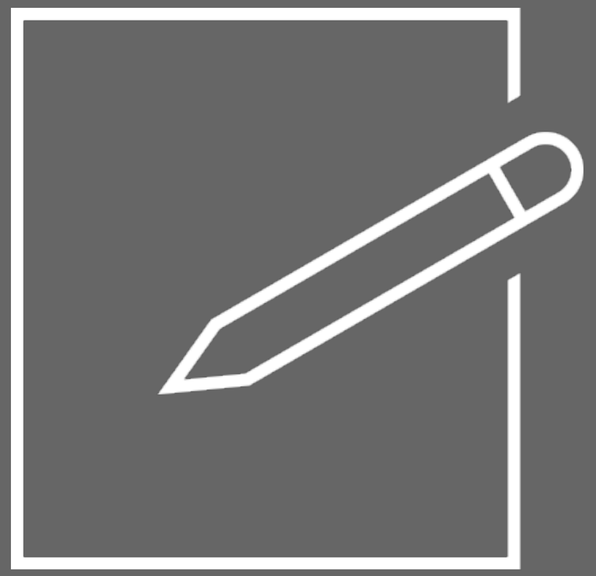
Template Files

What You'll Learn

- Use the templatefile function
- Render an IAM policy from a template
- Understand real world scenarios for using templates

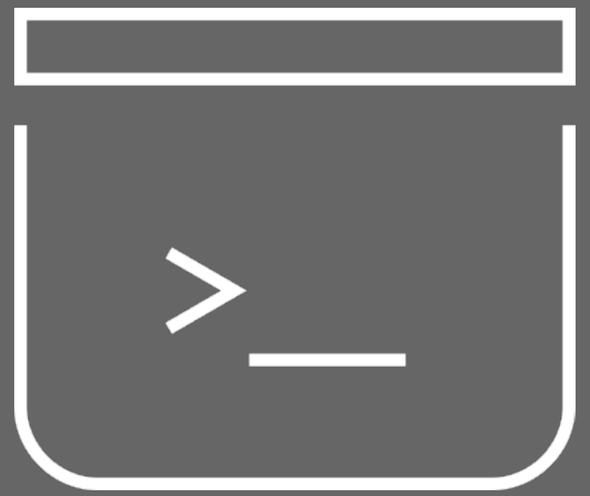
What is a Template File in Terraform?

- Reads a file at a given path and renders its content as a template using a supplied set of variables
- Invoked using the keyword templatefile
- Uses the same interpolation syntax as strings



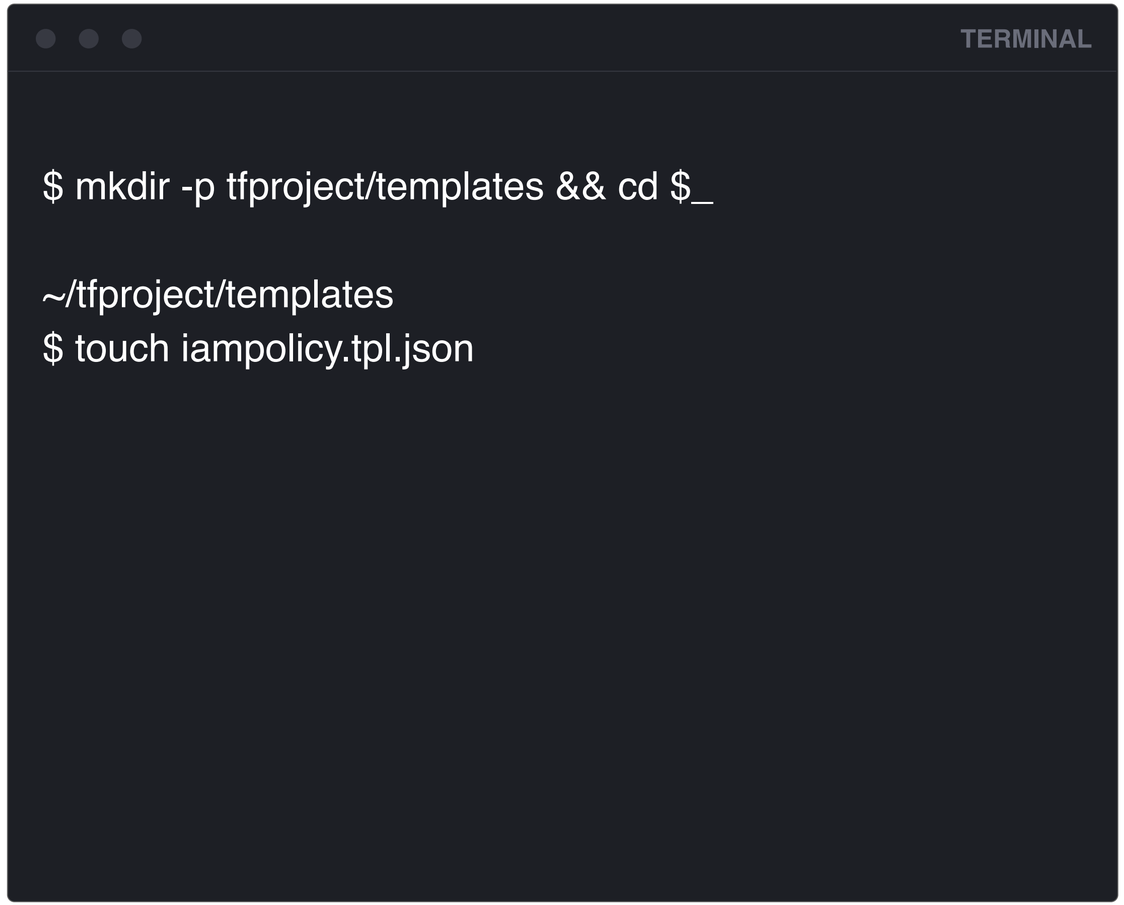
Plain Text Template

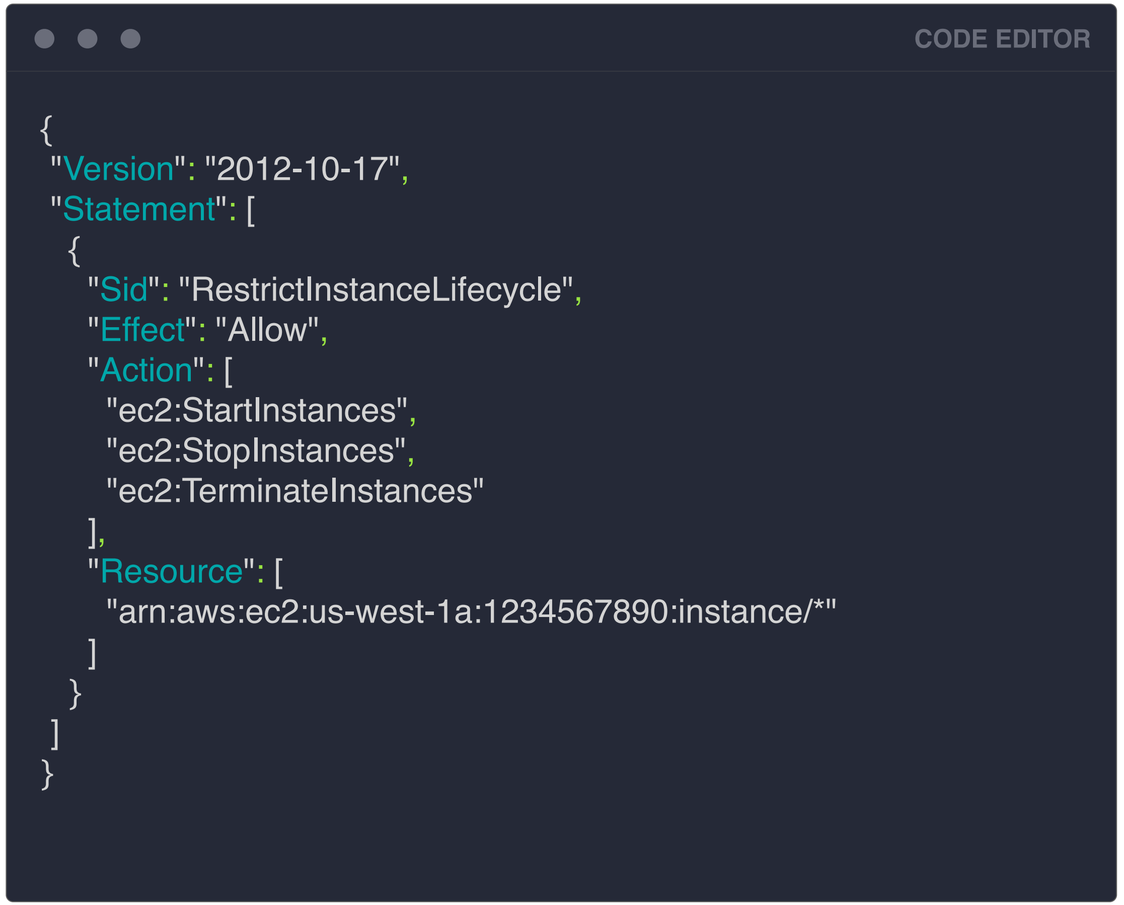
Variables from Cloud or Other Resources



Render with Terraform

Creating a Template File





Interpolation in Template Files

We can replace the region & owner IDs for more dynamic templates.

CODE EDITOR

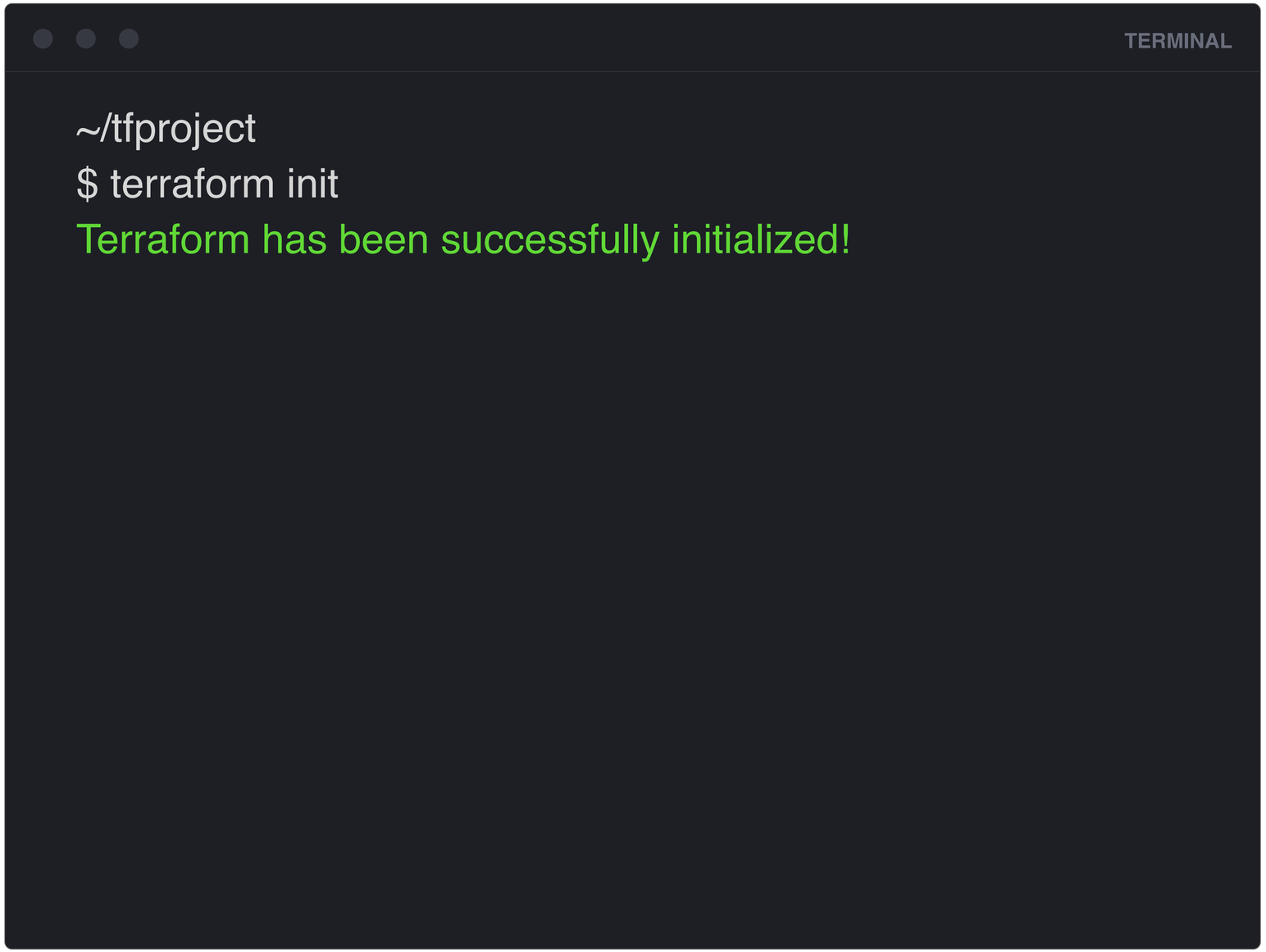
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RestrictInstanceLifecycle",
      "Effect": "Allow",
      "Action": [
        "ec2:StartInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": [
    "arn:aws:ec2:${region}:${owner_id}:instance/*"
      ]
    }
  ]
}
```

CODE EDITOR

```
variable "region" {
  default = "us-west-2"
}

variable "owner_id" {
  default = "1234567890"
}

output "iam_policy" {
  value = templatefile("terraform/templates/iam_policy.tpl.json", {
    region = var.region,
    owner_id = var.owner_id })
}
```

terraform init

To update the template function in your config



terraform apply

Rendered file will be reported as an output

TERMINAL

```
$ terraform apply
data.template_file.iam_policy: Refreshing state...

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

Outputs:

iam_policy = {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RestrictInstanceLifecycle",
      "Effect": "Allow",
      "Action": [
        "ec2:StartInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": [
        "arn:aws:ec2:us-west-2:123456789:instance/*"
      ]
    }
  ]
}
```

Real world scenarios

Pass custom scripts to an instance's user_data

Create a new IAM policy with the policy argument

Customize a configuration file, such as a startup script or a Consul config

Build Absolute Paths With path.module

When rendering templates from within modules, it's possible to end up in the wrong directory.

To fix this, use path.module when supplying the path to a template. This builds an absolute path to the current module.

template = templatefile("${path.module}/templates/init.tpl", {input1 = var.input1,

input2 = var.input2 })

Example IAM Policy

CODE EDITOR

```
resource "aws_iam_policy" "policy" {
  name = "policy"
  policy = templatefile("${path.module}/templates/iam_policy.tpl.json", {
    region = var.region,
    owner_id = var.owner_id })
}
```

Example User Data startup script

CODE EDITOR

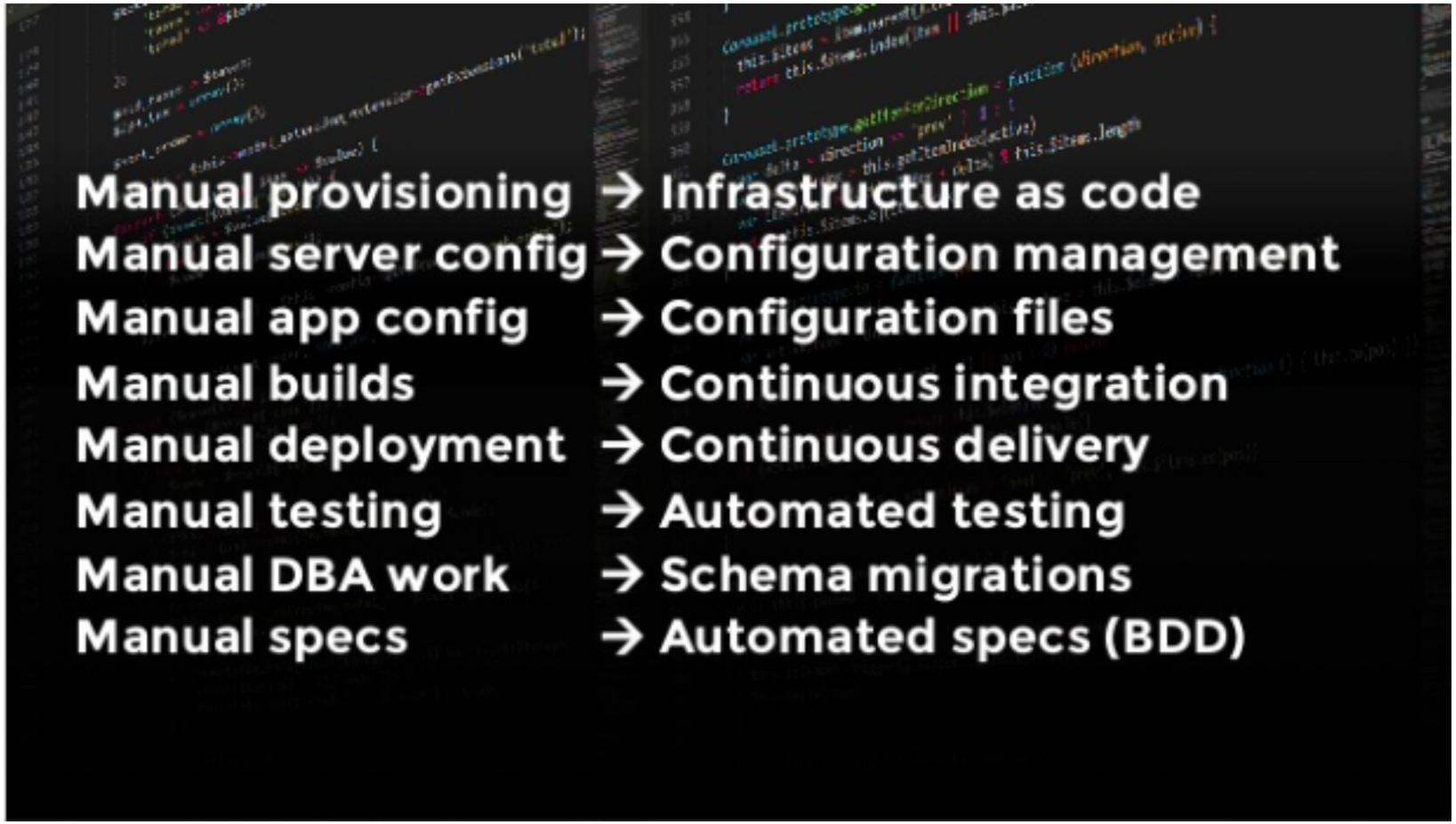
```
resource "aws_instance" "web" {
  ami = "ubuntu"
  instance_type = "t2.micro"

  user_data =
  templatefile("${path.module}/templates/user-data.tpl.json", {username =
  var.username})
}
```

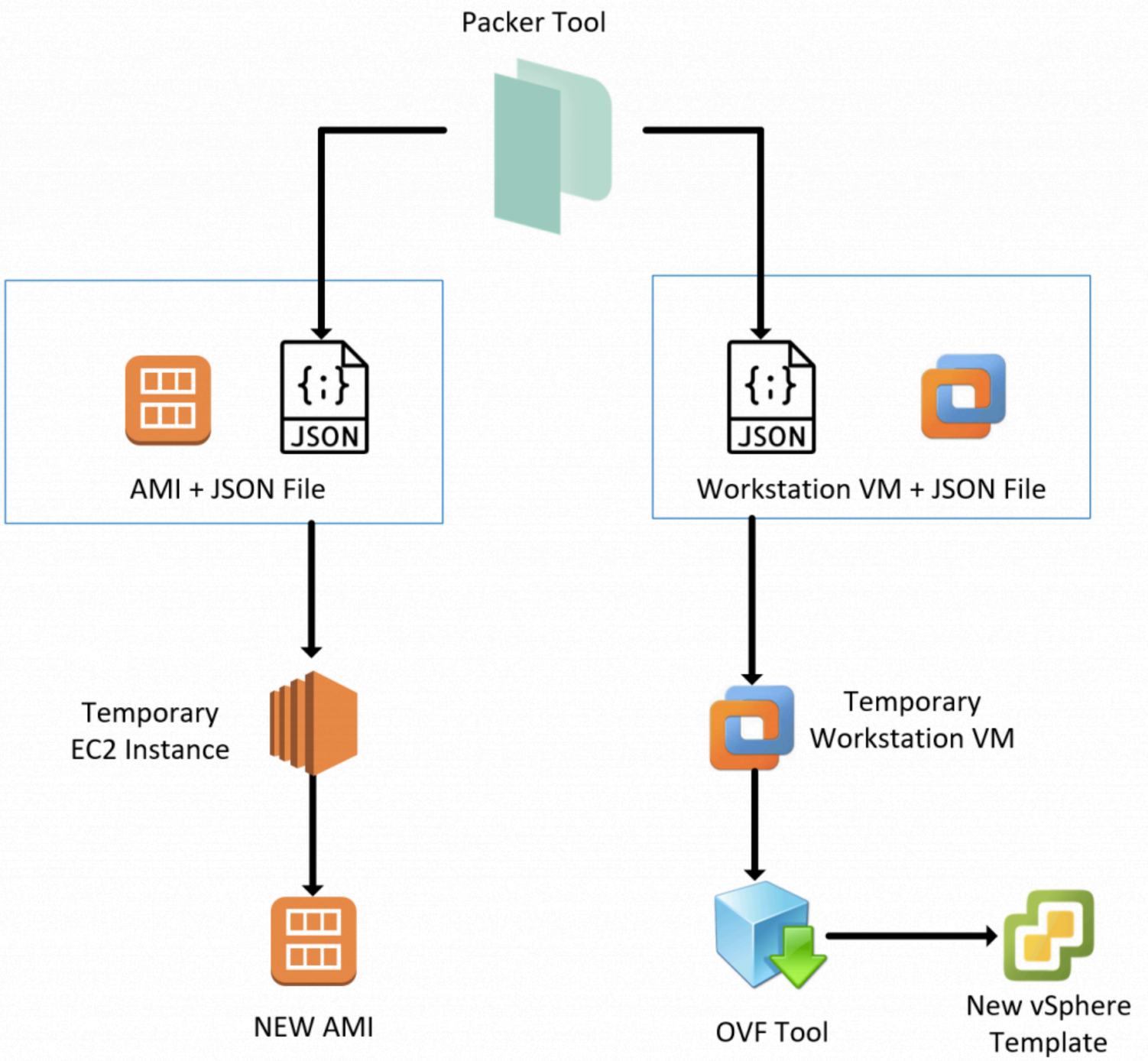
Terraform 201

Creating Images with Packer

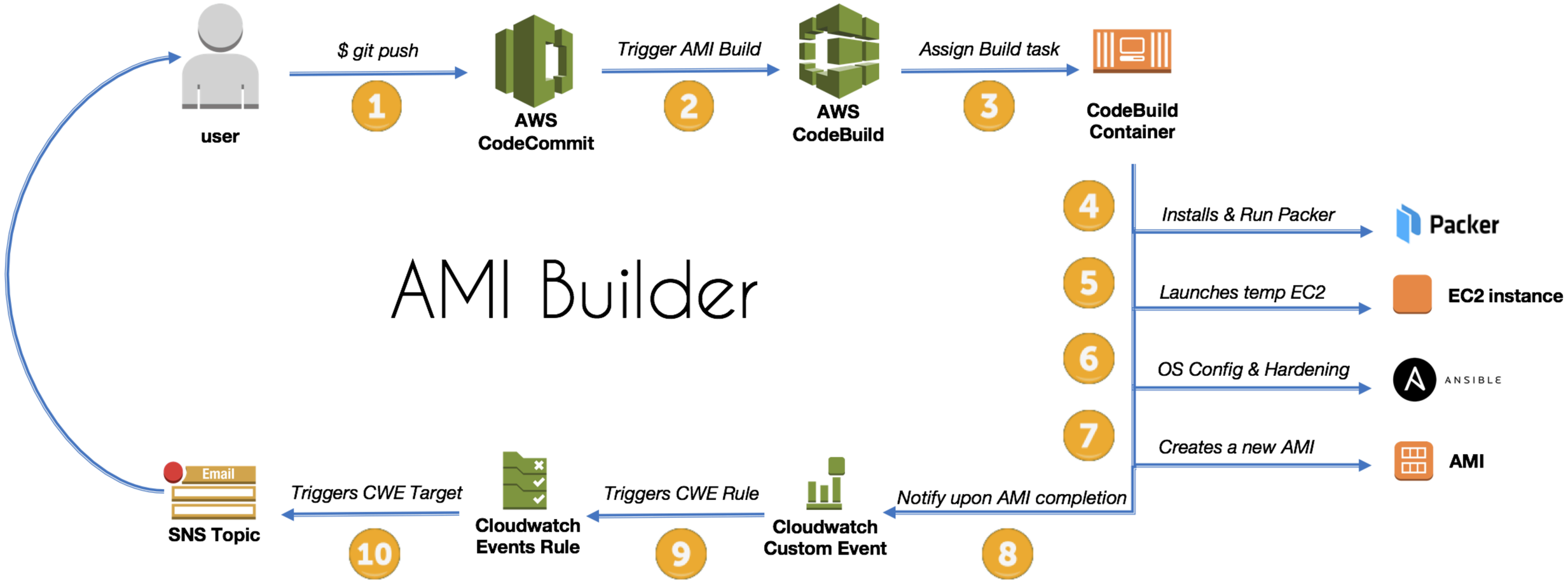
Packer

- Automate building machine images and it supports VMware, Amazon EC2, Docker, Google Compute, and others
- Building for a heterogenous cloud environment


Manual provisioning → Infrastructure as code
Manual server config → Configuration management
Manual app config → Configuration files
Manual builds → Continuous integration
Manual deployment → Continuous delivery
Manual testing → Automated testing
Manual DBA work → Schema migrations
Manual specs → Automated specs (BDD)


Packer Tool
JSON
AMI + JSON File
JSON
Workstation VM + JSON File
Temporary
EC2 Instance
Temporary
Workstation VM
NEW AMI
OVF Tool
New vSphere
Template

Automation the Build of AMIs in AWS



Examples

- Tons of Examples of Packer Templates

- https://github.com/chef/bento

Terraform 201

Multi-Provider

What You'll Learn

- How to configure multiple providers

- How to use attributes of one provider as arguments in another

Providers

- ACME
- Akamai
- Alibaba Cloud
- Archive
- Arukas
- Avi Vantage
- AWS
- Azure
- Azure Active Directory
- Azure Stack
- Bitbucket
- Brightbox
- CenturyLinkCloud
- Chef
- Circonus
- Cisco ASA
- Cisco ACI
- Cloudflare
- CloudScale.ch
- Google Cloud Platform
- Grafana
- Gridscale
- Hedvig
- Helm
- Heroku
- Hetzner Cloud
- HTTP
- HuaweiCloud
- Icinga2
- Ignition
- InfluxDB
- JDCloud
- Kubernetes
- Librato
- Linode
- Local
- Logentries
- LogicMonitor
- Packet
- PagerDuty
- Palo Alto Networks
- PostgreSQL
- PowerDNS
- ProfitBricks
- RabbitMQ
- Rancher
- Rancher2
- Random
- RightScale
- Rundeck
- RunScope
- Scaleway
- Selectel
- SignalFx
- Skytap
- SoftLayer
- Spotinst
- CloudStack
- Cobbler
- Consul
- Datadog
- DigitalOcean
- DNS
- DNSimple
- DNSMadeEasy
- Docker
- Dyn
- External
- F5 BIG-IP
- Fastly
- FlexibleEngine
- FortiOS
- GitHub
- GitLab
- Mailgun
- MySQL
- Naver Cloud
- Netlify
- New Relic
- Nomad
- NS1
- Null
- Nutanix
- 1&1
- OpenStack
- OpenTelekomCloud
- OpsGenie
- Oracle Cloud Infrastructure
- Oracle Cloud Platform
- Oracle Public Cloud
- OVH
- StatusCake
- TelefonicaOpenCloud
- Template
- TencentCloud
- Terraform
- Terraform Enterprise
- TLS
- Triton
- UCloud
- UltraDNS
- Vault
- VMware NSX-T
- VMware vCloud Director
- VMware vRA7
- VMware vSphere
- Yandex

More providers can be found on our Community Providers page.

Defining multiple providers

CODE EDITOR

```
provider "github" {
 token        = var.github_token
 organization = "hashicorp"
}

provider "aws" {
 version = ">= 2.26.0"
}

data "github_ip_ranges" "test" {}

resource "aws_security_group" "test" {
...
egress {
 from_port   = 0
 to_port     = 0
 protocol    = "-1"
 cidr_blocks = data.github_ip_ranges.test.pages
      }
}
```

Lab Exercise 5

Challenge

Overview

Questions?

Use GitHub IP Addresses to Constrain an AWS Security Group

Configure a GitHub provider and an AWS provider.

Define a data source for GitHub IP address ranges.

Configure an AWS security group so that egress traffic is only allowed to access GitHub IP address ranges.

Verify by sending a ping to a GitHub IP address as well as to hashicorp.com

Lab Exercise 5

Challenge

How to configure multiple providers

Overview

How to use attributes of one provider as arguments in another

Questions?

Lab Exercise 5

Challenge

Overview

Questions?

What other uses could be found for multiple provider configurations?

Take a few minutes to discuss with your neighbor.

Recap

Glossary

Provider

A provider is an abstraction of the API/service provider such as vSphere, AWS, GCP, DNSimple, or Fastly. Providers typically require some sort of configuration data such as an API key or credential file.

Glossary

Resource

A resource represents a component of a provider such as an "VMware VM", "DNSimple Record", or "Fastly service". Resources have both *arguments* (inputs) and *attributes* (outputs) which are specific to the resource. Resources also have meta-parameters such as `count` and `lifecycle`.

Glossary

(Resource) Argument

An argument is an input or configuration option to a resource. An VMware VM instance accepts `memory` as an input parameter. This makes `memory` an argument to the `vsphere_virtual_machine` resource.

Glossary

(Resource) Attribute

An attribute is an output or computed value available only after resource creation. A VMware VM provides `default_ip_address` as an output parameter. This makes `default_ip_address` an attribute to the `vsphere_virtual_machine` resource. This makes sense, because an instance's IP address is assigned during creation.

Glossary

Graph

The graph is the internal structure for Terraform's resource dependencies and order. The graph implements a directed acyclic graph (DAG) which allows Terraform to optimize for parallelism while adhering to dependency ordering. It is possible to generate the graph as a DOT file for human viewing.

Glossary

(Remote/Local) State

Terraform stores the last-known arguments and attributes for all resources. These contents known as "state" can be stored locally as a JSON file (local state) or stored in a remote shared location like Terraform Enterprise (remote state).

Glossary

Module

A module is a blackbox, self-contained package of Terraform configurations. Modules are like abstract classes that are imported into other Terraform configurations.

Parallels: Chef Cookbook, Puppet Module, Ruby gem

Glossary

Variable

A variable is a user or machine-supplied input in Terraform configurations. Variables can be supplied via environment variables, CLI flags, or variable files. Combined with modules, variables help make Terraform flexible, sharable, and extensible.

Glossary

Output

An output is a configurable piece of information that is highlighted at the end of a Terraform run.

Glossary

Interpolation

Terraform includes a built-in syntax for referencing attributes of other resources. This technique is called interpolation. Terraform also provides built-in functions for performing string manipulations, evaluating math operations, and doing list comprehensions.

Glossary

HashiCorp Configuration Language (HCL)

Terraform's syntax and interpolation are part of an open source language and specification called HCL.

Looks a bit like nginx configuration.

Used in Terraform, Nomad, and (soon) Consul.

State Resolutions

Configuration	State	Reality	**Operation**
aws_instance.web			
aws_instance.web	aws_instance.web		
aws_instance.web	aws_instance.web	aws_instance.web	
	aws_instance.web	aws_instance.web	
		aws_instance.web	
aws_instance.web		aws_instance.web	
	aws_instance.web		

State Resolutions

Configuration	State	Reality	Operation
aws_instance.web			**create**
aws_instance.web	aws_instance.web		**create**
aws_instance.web	aws_instance.web	aws_instance.web	**noop**
	aws_instance.web	aws_instance.web	**delete**
		aws_instance.web	**noop**
aws_instance.web		aws_instance.web	**re-create***
	aws_instance.web		**update state**

HashiCorp Terraform Quiz

Question 1: When multiple arguments with single-line values appear on consecutive lines at the same nesting level, HashiCorp recommends that you:

A place all arguments using a variable at the top

```
ami = var.aws_ami
instance_type = var.instance_size
subnet_id = "subnet-0bb1c79de3EXAMPLE"
tags = {
  Name = "HelloWorld"
}
```

B place a space in between each line

```
type = "A"

ttl = "300"

zone_id = aws_route53_zone.primary.zone_id
```

C put arguments in alphabetical order

```
name = "www.example.com"
records = [aws_eip.lb.public_ip]
type = "A"
ttl = "300"
zone_id = aws_route53_zone.primary.zone_id
```

D align their equals signs

```
ami           = "abc123"
instance_type = "t2.micro"
```

Question 1: When multiple arguments with single-line values appear on consecutive lines at the same nesting level, HashiCorp recommends that you:

A place all arguments using a variable at the top

```
ami = var.aws_ami
instance_type = var.instance_size
subnet_id = "subnet-0bb1c79de3EXAMPLE"
tags = {
  Name = "HelloWorld"
}
```

B place a space in between each line

```
type = "A"

ttl = "300"

zone_id = aws_route53_zone.primary.zone_id
```

C put arguments in alphabetical order

```
name = "www.example.com"
records = [aws_eip.lb.public_ip]
type = "A"
ttl = "300"
zone_id = aws_route53_zone.primary.zone_id
```

D align their equals signs

```
ami           = "abc123"
instance_type = "t2.micro"
```

Explanation

HashiCorp style conventions suggest you that align the equals sign for consecutive arguments for easing readability for configurations

ami = "abc123"

instance_type = "t2.micro"

Question 2: When using providers that require the retrieval of data, such as the HashiCorp Vault provider, in what phase does Terraform actually retrieve the data required?:

A **terraform delete**

B **terraform init**

C **terraform apply**

D **terraform plan**

Question 2: When using providers that require the retrieval of data, such as the HashiCorp Vault provider, in what phase does Terraform actually retrieve the data required?:

A	**terraform delete**
B	**terraform init**
C	**terraform apply**
D	**terraform plan**

Explanation

It is important to consider that Terraform reads from data sources during the plan phase and writes the result into the plan. For something like a Vault token which has an explicit TTL, the apply must be run before the data, or token, in this case, expires, otherwise, Terraform will fail during the apply phase.

Question 3: Which Terraform command will check and report errors within modules, attribute names, and value types to make sure they are syntactically valid and internally consistent?

A terraform validate

B terraform format

C terraform show

D terraform fmt

Question 3: Which Terraform command will check and report errors within modules, attribute names, and value types to make sure they are syntactically valid and internally consistent?

A. `terraform validate`

B. `terraform format`

C. `terraform show`

D. `terraform fmt`

Explanation

The `terraform validate` command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

Validate runs checks that verify whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state. It is thus primarily useful for general verification of reusable modules, including the correctness of attribute names and value types

Question 4: True or False: When using the Terraform provider for Vault, the tight integration between these HashiCorp tools provides the ability to mask secrets in the `terraform plan` and state files.

A **False**

B **True**

Question 4: True or False: When using the Terraform provider for Vault, the tight integration between these HashiCorp tools provides the ability to mask secrets in the `terraform plan` and state files.

A **False**

B **True**

Explanation

Currently, Terraform has no mechanism to redact or protect secrets that are returned via data sources, so secrets read via this provider will be persisted into the Terraform state, into any plan files, and in some cases in the console output produced while planning and applying. These artifacts must, therefore, all be protected accordingly.

Question 5: Terraform Enterprise (also referred to as pTFE) requires what type of backend database for a clustered deployment?

A **PostgreSQL**

B **MySQL**

C **Cassandra**

D **MSSQL**

Question 5: Terraform Enterprise (also referred to as pTFE) requires what type of backend database for a clustered deployment?

A **PostgreSQL**

B **MySQL**

C **Cassandra**

D **MSSQL**

Explanation

Production - External Services mode stores the majority of the stateful data used by the instance in an external PostgreSQL database and an external S3-compatible endpoint or Azure blob storage. There is still critical data stored on the instance that must be managed with snapshots. Be sure to check the PostgreSQL Requirements for information that needs to be present for Terraform Enterprise to work. This option is best for users with expertise managing PostgreSQL or users that have access to managed PostgreSQL offerings like AWS RDS.

Question 6: The following is a snippet from a terraform configuration file:-

```
provider "aws" {
  region  = "us-east-1"
}

provider "aws" {
  region = "us-west-1"
}
```

which, when validated, results in the following error:-

```
Error: Duplicate provider configuration

  on main.tf line 5:
   5: provider "aws" {

A default provider configuration for "aws" was already given at
main.tf:1,1-15. If multiple configurations are required, set the "_____"
argument for alternative configurations.
```

Fill in the blank in the error message with the correct string from the list below.

A label

B version

C multi

D alias

Question 6: The following is a snippet from a terraform configuration file:-

```
provider "aws" {
  region  = "us-east-1"
}

provider "aws" {
  region = "us-west-1"
}
```

which, when validated, results in the following error:-

```
Error: Duplicate provider configuration

  on main.tf line 5:
   5: provider "aws" {

A default provider configuration for "aws" was already given at
main.tf:1,1-15. If multiple configurations are required, set the "_____"
argument for alternative configurations.
```

Explanation

An `alias` meta-arguement is used when using the same provider with different configurations for different resources.

https://www.terraform.io/docs/configuration/providers.html#alias-multiple-provider-instances

Fill in the blank in the error message with the correct string from the list below.

A label

B version

C multi

D alias

Question 7: A user has created three workspaces from the command line - prod, dev and test. The user wants to create a fourth, stage. Which command will the user execute to accomplish this?

A terraform workspace – create stage

B terraform workspace – new stage

C terraform workspace create stage

D terraform workspace new stage

Question 7: A user has created three workspaces from the command line - prod, dev and test. The user wants to create a fourth, stage. Which command will the user execute to accomplish this?

A **terraform workspace – create stage**

B **terraform workspace – new stage**

C **terraform workspace create stage**

D **terraform workspace new stage**

Explanation

The `terraform workspace` new command is used to create a new workspace.

https://www.terraform.io/docs/commands/workspace/new.html

Question 8: In terraform, most resource dependencies are handled automatically. Which of the following statements describes best how terraform resource dependencies are handled?

A

Resource dependencies are handled automatically by the `depends_on` meta_argument which is set to true by default

B

The terraform binary contain a built-in reference map of all the defined terraform resource dependencies. Updates to this dependency map are reflected in terraform versions. To ensure you are work with the latest resource dependency map you must be running the latest version of terraform.

C

Resource dependencies are identified and maintained in a file called `resource.dependencies` .Each terraform provider is required to maintain a list of all resource dependencies for the provider and its included with the plugin during initialization when terraform init is executed. The file is located in the `terraform.d` folder

D

Terraform analyses any expression within a resource block to find references to other objects and treats those references as implicit ordering requirement when creating, uploading or destroying resources.

Question 8: In terraform, most resource dependencies are handled automatically. Which of the following statements describes best how terraform resource dependencies are handled?

A

Resource dependencies are handled automatically by the `depends_on` meta_argument which is set to true by default

B

The terraform binary contain a built-in reference map of all the defined terraform resource dependencies. Updates to this dependency map are reflected in terraform versions. To ensure you are work with the latest resource dependency map you must be running the latest version of terraform.

C

Resource dependencies are identified and maintained in a file called `resource.dependencies` .Each terraform provider is required to maintain a list of all resource dependencies for the provider and its included with the plugin during initialization when terraform init is executed. The file is located in the `terraform.d` folder

D

Terraform analyses any expression within a resource block to find references to other objects and treats those references as implicit ordering requirement when creating, uploading or destroying resources.

Explanation: https://www.terraform.io/docs/configuration/resources.html

Question 9: Why might a user opt to include the following snippet in their configuration file?

```
terraform {
  required_version = ">= 0.12"
}
```

A — Terraform 0.12 introduced substantial changes to the syntax used to write Terraform configuration

B — Versions before Terraform 0.12 were not approved by HashiCorp to be used in production

C — The user wants to ensure that the application being deployed is a minimum version of 0.12

D — This ensures that all Terraform providers are above a certain version to match the application being deployed

Question 9: Why might a user opt to include the following snippet in their configuration file?

```
terraform {
  required_version = ">= 0.12"
}
```

A. Terraform 0.12 introduced substantial changes to the syntax used to write Terraform configuration

B. Versions before Terraform 0.12 were not approved by HashiCorp to be used in production

C. The user wants to ensure that the application being deployed is a minimum version of 0.12

D. This ensures that all Terraform providers are above a certain version to match the application being deployed

Explanation:

You can use the required_version to ensure that the user deploying infrastructure is using Terraform 0.12 or great, due to the vast number of changes that were introduced. As a result, many previously written configurations had to be converted or rewritten.

Question 10: **In order to reduce the time it takes to provision resources, Terraform uses parallelism. By default, how many resources will Terraform provision concurrently?**

A **5**

B **10**

C **20**

D **50**

Question 10: In order to reduce the time it takes to provision resources, Terraform uses parallelism. By default, how many resources will Terraform provision concurrently?

A **5**

B **10**

C **20**

D **50**

Explanation

Terraform can limit the number of concurrent operations as Terraform walks the graph using the `-parallelism=n` argument. The default value for this setting is 10. This setting might be helpful if you're running into API rate limits.

Question 11: Which of the following commands will launch the Interactive console for Terraform interpolations?

A **terraform**

B **terraform console**

C **terraform cli**

D **terraform cmdline**

Question 11: Which of the following commands will launch the Interactive console for Terraform interpolations?

A **terraform**

B **terraform console**

C **terraform cli**

D **terraform cmdline**

Explanation

The `terraform console` command provides an interactive console for evaluating expressions.

https://www.terraform.io/docs/commands/console.html

Question 12: Which of the following is not a valid Terraform string function?

A **format**

B **tostring**

C **join**

D **replace**

Question 12: Which of the following is not a valid Terraform string function?

A **format**

B **tostring**

C **join**

D **replace**

Explanation
tostring is not a string function, it is a type conversion function. `tostring` converts its argument to a string value.
https://www.terraform.io/docs/configuration/functions/tostring.html

Question 13:Anyone can publish and share modules on the Terraform Public Module Registry and meeting the requirements for publishing a module is extremely easy. Select from the following list all valid requirements.

A

The registry uses tags to identify module versions. Release tag names must be for the format x.y.z, and can optionally be prefixed with a v

B

Module repositories must use this three-part name format, terraform-<PROVIDER>-<Name>.

C

The module must be PCI/HIPPA compliant

D

The module must be on Github and must be a public repo.

Question 13:Anyone can publish and share modules on the Terraform Public Module Registry and meeting the requirements for publishing a module is extremely easy. Select from the following list all valid requirements.

A

The registry uses tags to identify module versions. Release tag names must be for the format x.y.z, and can optionally be prefixed with a v

B

Module repositories must use this three-part name format, terraform- <PROVIDER>-<Name>.

C

The module must be PCI/HIPPA compliant

D

The module must be on Github and must be a public repo.

Explanation

The list below contains all the requirements for publishing a module. Meeting the requirements for publishing a module is extremely easy. The list may appear long only to ensure we're detailed but adhering to the requirements should happen naturally.

GitHub. The module must be on GitHub and must be a public repo. This is only a requirement for the public registry. If you're using a private registry, you may ignore this requirement.

Named terraform-<PROVIDER>-<NAME>. Module repositories must use this three-part name format, where <NAME> reflects the type of infrastructure the module manages and <PROVIDER> is the main provider where it creates that infrastructure. The <NAME> segment can contain additional hyphens. Examples: terraform-google-vault or terraform-aws-ec2-instance.

Repository description. The GitHub repository description is used to populate the short description of the module. This should be a simple one sentence description of the module.

Standard module structure. The module must adhere to the standard module structure. This allows the registry to inspect your module and generate documentation, track resource usage, parse submodules and examples, and more.

x.y.z tags for releases. The registry uses tags to identify module versions. Release tag names must be a semantic version, which can optionally be prefixed with a v. For example, v1.0.4 and 0.9.2. To publish a module initially, at least one release tag must be present. Tags that don't look like version numbers are ignored.

https://www.terraform.io/docs/registry/modules/publish.html#requirements

Question 14: A user has created a module called "my_test_module" and committed it to GitHub. Over time, several commits have been made with updates to the module, each tagged in GitHub with an incremental version number. Which of the following lines would be required in a module configuration block in terraform to select tagged version v1.0.4?

A source = “git::https://example.com/my_test_module.git#tag=v1.0.4”

B source = “git::https://example.com/my_test_module.git@tag=v1.0.4”

C source = “git::https://example.com/my_test_module.git&ref=v1.0.4”

D source = “git::https://example.com/my_test_module.git?ref=v1.0.4”

Question 14: A user has created a module called "my_test_module" and committed it to GitHub. Over time, several commits have been made with updates to the module, each tagged in GitHub with an incremental version number. Which of the following lines would be required in a module configuration block in terraform to select tagged version v1.0.4?

A source = “git::https://example.com/my_test_module.git#tag=v1.0.4”

B source = “git::https://example.com/my_test_module.git@tag=v1.0.4”

C source = “git::https://example.com/my_test_module.git&ref=v1.0.4”

D source = “git::https://example.com/my_test_module.git?ref=v1.0.4”

Explanation

By default, Terraform will clone and use the default branch (referenced by HEAD) in the selected repository. You can override this using the ref argument:

```
module "vpc" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}
```

The value of the ref argument can be any reference that would be accepted by the git checkout command, including branch and tag names.

https://www.terraform.io/docs/modules/sources.html#selecting-a-revision

Question 15:Select all Operating Systems that Terraform is available for.

A **macOS**

B **Solaris**

C **FreeBSD**

D **Windows**

E **Linux**

Question 15:Select all Operating Systems that Terraform is available for.

A **macOS**

B **Solaris**

C **FreeBSD**

D **Windows**

E **Linux**

Explanation

Terraform is available for macOS, FreeBSD, OpenBSD, Linux, Solaris, Windows

https://www.terraform.io/downloads.html

Question 16: Select the most accurate statement to describe the Terraform language from the following list.

A

Terraform is an immutable, procedural, Infrastructure as Code configuration management language based on HashiCorp Configuration Language or optionally JSON.

B

Terraform is an immutable, declarative, Infrastructure as Code provisioning language based on HashiCorp Configuration Language or optionally JSON.

C

Terraform is an mutable, declarative, Infrastructure as Code configuration management language based on HashiCorp Configuration Language or optionally JSON.

D

Terraform is an mutable, procedural, Infrastructure as Code provisioning language based on HashiCorp Configuration Language or optionally YAML.

Question 16: Select the most accurate statement to describe the Terraform language from the following list.

A

Terraform is an immutable, procedural, Infrastructure as Code configuration management language based on HashiCorp Configuration Language or optionally JSON.

B

Terraform is an immutable, declarative, Infrastructure as Code provisioning language based on HashiCorp Configuration Language or optionally JSON.

C

Terraform is an mutable, declarative, Infrastructure as Code configuration management language based on HashiCorp Configuration Language or optionally JSON.

D

Terraform is an mutable, procedural, Infrastructure as Code provisioning language based on HashiCorp Configuration Language or optionally YAML.

Explanation

Terraform is not a configuration management tool - https://www.terraform.io/intro/vs/chef-puppet.html

Terraform is a declarative language - https://www.terraform.io/docs/configuration/index.html

Terraform supports a syntax that is JSON compatible - https://www.terraform.io/docs/configuration/syntax-json.html

Terraform is primarily designed on immutable infrastructure principles - https://www.hashicorp.com/resources/what-is-mutable-vs-immutable-infrastructure

Question 17: True or False: Each Terraform workspace uses its own state file to manage the infrastructure associated with that particular workspace.

A **False**

B **True**

Question 17: True or False: Each Terraform workspace uses its own state file to manage the infrastructure associated with that particular workspace.

A **False**

B **True**

Explanation

The persistent data stored in the backend belongs to a workspace. Initially, the backend has only one workspace, called "default", and thus there is only one Terraform state associated with that configuration.

Question 18: Terraform Cloud is more powerful when you integrate it with your version control system (VCS) provider. List all supported VCS providers.

A **Bitbucket Cloud**

B **Azure DevOps Server**

C **CVS Version Control**

D **Github**

E **GitHub Enterprise**

Question 18: Terraform Cloud is more powerful when you integrate it with your version control system (VCS) provider. List all supported VCS providers.

A **Bitbucket Cloud**

B **Azure DevOps Server**

C **CVS Version Control**

D **Github**

E **GitHub Enterprise**

Explanation

Terraform Cloud supports the following VCS providers:

GitHub
GitHub Enterprise
GitLab.com
GitLab EE and CE
Bitbucket Cloud
Bitbucket Server
Azure DevOps Server
Azure DevOps Services

https://www.terraform.io/docs/cloud/vcs/index.html#supported-vcs-providers

Question 19: True or False? By default, Terraform `destroy` will prompt for confirmation before proceeding.

A **False**

B **True**

Question 19: True or False? By default, Terraform `destroy` will prompt for confirmation before proceeding.

A **False**

B **True**

Explanation

Terraform destroy will always prompt for confirmation before executing unless passed the `-auto-approve` flag.

```
$ terraform destroy
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value:
```

Question 20: When using constraint expressions to signify a version of a provider, which of the following are valid provider versions that satisfy the expression found in the following code snippet:

```
terraform {
  required_providers {
    aws = "~> 1.2.0"
  }
}
```

A **1.2.9**

B **1.3.1**

C **1.3.0**

D **1.2.3**

Question 20: When using constraint expressions to signify a version of a provider, which of the following are valid provider versions that satisfy the expression found in the following code snippet:

```
terraform {
  required_providers {
    aws = "~> 1.2.0"
  }
}
```

A **1.2.9**

B **1.3.1**

C **1.3.0**

D **1.2.3**

Explanation

`~>` will match any non-beta version of the provider between `>=` and `<`.For example, `1.2.X` `1.2.0` `1.3.0`

1.2.0

Question 21: A user creates three workspaces from the command line - prod, dev and test. Which of the following commands will the user run to switch to the dev workspace?

A **terraform workspace select dev**

B **terraform workspace switch dev**

C **terraform workspace dev**

D **terraform workspace -switch dev**

Question 21: A user creates three workspaces from the command line - prod, dev and test. Which of the following commands will the user run to switch to the dev workspace?

A **terraform workspace select dev**

B **terraform workspace switch dev**

C **terraform workspace dev**

D **terraform workspace -switch dev**

Explanation

The terraform workspace select command is used to choose a different workspace to use for further operations.

https://www.terraform.io/docs/commands/workspace/select.html

Question 22: The terraform language supports a number of different syntaxes for comments. Select all that are supported.

A	/* and */
B	<* and *>
C	#
D	//

Question 22: The terraform language supports a number of different syntaxes for comments. Select all that are supported.

A /* and */

B <* and *>

C #

D //

Explanation

https://www.terraform.io/docs/configuration/syntax.html#comments

Question 23: Which of the following terraform subcommands could be used to remove the lock on the state for the current configuration?

A **state-unlock**

B **unlock**

C **Removing the lock on a state file is not possible**

D **force unlock**

Question 23: Which of the following terraform subcommands could be used to remove the lock on the state for the current configuration?

A	**state-unlock**
B	**unlock**
C	**Removing the lock on a state file is not possible**
D	**force unlock**

Explanation

terraform force-unlock removes the lock on the state for the current configuration.

https://www.terraform.io/docs/commands/force-unlock.html

Question 24: When Terraform needs to be installed in a location where it does not have internet access to download the installer and upgrades, the installation is generally known as to be _____.

A **a private install**

B **disconnected**

C **air-gapped**

D **non-traditional**

Question 24: When Terraform needs to be installed in a location where it does not have internet access to download the installer and upgrades, the installation is generally known as to be _____.

A **a private install**

B **disconnected**

C **air-gapped**

D **non-traditional**

Explanation

A Terraform Enterprise install that is provisioned on a network that does not have Internet access is generally known as an air-gapped install. These types of installs require you to pull updates, providers, etc. from external sources vs. being able to download them directly.

Question 25: The terraform refresh command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. If drift is detected between the real-world infrastructure and the last know-state, it will modify the infrastructure to correct the drift. True or False.

A **False**

B **True**

Question 25: The `terraform refresh` command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. If drift is detected between the real-world infrastructure and the last know-state, it will modify the infrastructure to correct the drift. True or False.

A **False**

B **True**

Explanation

The `terraform refresh` command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. This can be used to detect any drift from the last-known state, and to update the state file.

This does not modify infrastructure, but does modify the state file. If the state is changed, this may cause changes to occur during the next plan or apply.

https://www.terraform.io/docs/commands/refresh.htm
l

Question 26: While Terraform is generally written using the Terraform language, what other syntax can Terraform be expressed in?

A **XML**

B **JSON**

C **YAML**

D **TypeScript**

Question 26: While Terraform is generally written using the Terraform language, what other syntax can Terraform be expressed in?

A **XML**

B **JSON**

C **YAML**

D **TypeScript**

Explanation

The constructs in the Terraform language can also be expressed in JSON syntax, which is harder for humans to read and edit but easier to generate and parse programmatically.

Question 27: During a `terraform plan` a resource is successfully created but eventually fails during provisioning. What happens to the resource?

A. it is automatically deleted

B. Terraform attempts to provision the resource up to three times before exiting with an error

C. The terraform plan is rolled back and all provisioned resources are removed

D. The resources is marked as tainted

Question 27: During a `terraform plan` a resource is successfully created but eventually fails during provisioning. What happens to the resource?

A. it is automatically deleted

B. Terraform attempts to provision the resource up to three times before exiting with an error

C. The terraform plan is rolled back, and all provisioned resources are removed

D. The resources is marked as tainted

Explanation

If a resource successfully creates but fails during provisioning, Terraform will error and mark the resource as "tainted". A resource that is tainted has been physically created but can't be considered safe to use since provisioning failed.

Terraform also does not automatically roll back and destroy the resource during the apply when the failure happens, because that would go against the execution plan: the execution plan would've said a resource will be created but does not say it will ever be deleted.

Question 28: The terraform `state` command can be used to..

A. view state

B. there is no such command

C. modify state

D. refresh state

Question 28: The terraform `state` command can be used to..

A **view state**

B **there is no such command**

C **modify state**

D **refresh state**

Explanation

The `terraform state` command is used for advanced state management. Rather than modify the state directly, the `terraform state` commands can be used in many cases instead.

https://www.terraform.io/docs/commands/state/index.html

Question 29: Select the operating systems which are supported for a clustered Terraform Enterprise:

A **CentOS**

B **Amazon Linux**

C **Red Hat**

D **Ubuntu**

Question 29: Select the operating systems which are supported for a clustered Terraform Enterprise:

A **CentOS**

B **Amazon Linux**

C **Red Hat**

D **Ubuntu**

Explanation

Terraform Enterprise currently supports running under the following operating systems for a Clustered deployment:

- Ubuntu 14.04 / 16.04 / 18.04
- Red Hat Enterprise Linux 7.2 through 7.6

Clusters currently don't support other Linux variants. In particular, note that RHEL 7.7 is not currently supported.

Question 30: Select all features which are exclusive to Terraform Enterprise.

A **Cost Estimation**

B **Sentinel**

C **Clustering**

D **Audit Logs**

E **SAML/SSO**

Question 30: Select all features which are exclusive to Terraform Enterprise.

A	**Cost Estimation**
B	**Sentinel**
C	**Clustering**
D	**Audit Logs**
E	**SAML/SSO**

Explanation

Sentinel and Cost Estimation are both available in Terraform Cloud, though not at the free tier level.

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

Question 31:What is the result of the follow terraform function call:-

A **0**

B **2**

C **true**

D **1**

Question 31:What is the result of the follow terraform function call:-

A **0**

B **2**

C **true**

D **1**

Explanation

`index` finds the element index for a given value in a list starting with index 0.

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

Question 32:From the code below, identify the implicit dependency.

```
resource "aws_eip" "public_ip" {
    vpc = true
    instance = aws_instance.web_server.id
}

resource "aws_instance" "web_server" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
  depends_on = [aws_s3_bucket.company_data]
}
```

A **The AMI used for the EC2 instance**

B **The EIP with an id of ami-2757f631**

C **The S3 bucket labeled company_data**

D **The EC2 instance labeled web_server**

Question 32:From the code below, identify the implicit dependency.

```
resource "aws_eip" "public_ip" {
    vpc = true
    instance = aws_instance.web_server.id
}

resource "aws_instance" "web_server" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
  depends_on = [aws_s3_bucket.company_data]
}
```

A **The AMI used for the EC2 instance**

B **The EIP with an id of ami-2757f631**

C **The S3 bucket labeled company_data**

D **The EC2 instance labeled web_server**

Explanation

The EC2 instance labeled web_server is the implicit dependency as the `aws_eip` cannot be created until the `aws_instance` labeled `web_server` has been provisioned and the id is available.

Note that I`aws_s3_bucket.example` is an explicit dependency.

Question 33:What feature of Terraform Cloud and/or Terraform Enterprise can you publish and maintain a set of custom modules which can be used within your organization.

A	**Custom VCS integration**
B	**remote runs**
C	**private module registry**
D	**Terraform registry**

Question 33:What feature of Terraform Cloud and/or Terraform Enterprise can you publish and maintain a set of custom modules which can be used within your organization.

A **Custom VCS integration**

B **remote runs**

C **private module registry**

D **Terraform registry**

Explanation

You can use modules from a private registry, like the one provided by Terraform Cloud. Private registry modules have source strings of the form

<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER

This is the same format as the public registry, but with an added hostname prefix.

Question 34:Which of the following is an invalid variable name?

A **count**

B **web**

C **Instance_name**

D **var1**

Question 34:Which of the following is an invalid variable name?

A **count**

B **web**

C **Instance_name**

D **var1**

Explanation

`count` is a reserved word. The `count` parameter on resources can simplify configurations and let you scale resources by simply incrementing a number.

https://www.terraform.io/intro/examples/count.html

Question 35:Provider dependencies are created in several different ways. Select the 3 valid provider dependencies from the following list:

A — Use of any resource belonging to a particular provider in a resource or data block in configuration

B — Explicit use of a provider block in configuration, optionally including a version constraint

C — Existence of any provider plugins found locally in the working directory

D — Existence of any resource instance belonging to a particular provider in the current state

Question 35:Provider dependencies are created in several different ways. Select the 3 valid provider dependencies from the following list:

A — Use of any resource belonging to a particular provider in a resource or data block in configuration

B — Explicit use of a provider block in configuration, optionally including a version constraint

C — Existence of any provider plugins found locally in the working directory

D — Existence of any resource instance belonging to a particular provider in the current state

Explanation

The existence of a provider plugin found locally in the working directory does not itself create a provider dependency. The plugin can exist without any reference to it in the terraform configuration.

https://www.terraform.io/docs/commands/providers.html

Question 36:Complete the following sentence:-

For local state, the workspaces are stored directly in a...

A **Directory called terraform.tfstate.d**

B **a file called terraform.tfstate**

C **a file called terraform.tfstate.backup**

D **directory called terraform.workspaces.tfstate**

Question 36:Complete the following sentence:-

For local state, the workspaces are stored directly in a...

A **Directory called terraform.tfstate.d**

B **a file called terraform.tfstate**

C **a file called terraform.tfstate.backup**

D **directory called terraform.workspaces.tfstate**

Explanation

For local state, Terraform stores the workspace states in a directory called Terraform.tfstate.d.

https://www.terraform.io/docs/state/workspaces.html#workspace-internals

Question 37:Which of the following statements best describes the Terraform list.... type:-

A **A sequence of values identified by consecutive whole number starting with zero**

B **A collection of named attributed that each have their own type**

C **A collection of values where each is identified by a string label**

D **A collection of unique values that do not have any secondary identifiers or ordering**

Question 37:Which of the following statements best describes the Terraform list.... type:-

A. A sequence of values identified by consecutive whole number starting with zero

B. A collection of named attributed that each have their own type

C. A collection of values where each is identified by a string label

D. A collection of unique values that do not have any secondary identifiers or ordering

Explanation

A terraform list is a sequence of values identified by consecutive whole numbers starting with zero.

https://www.terraform.io/docs/configuration/types.html#structural-types

Question 38:What is the result of the follow terraform function call:-

A **c**

B **hello**

C **what?**

D **goodbye**

Question 38:What is the result of the follow terraform function call:-

A **c**

B **hello**

C **what?**

D **goodbye**

Explanation

`lookup` retrieves the value of a single element from a map, given its key. If the given key does not exist, the given default value is returned instead. In this case, the function call is searching for the key "c". Because there there is no key "c", the default vault "what?" is returned.

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

Question 39:Terraform has detailed logs which can be enabled by setting the _____ environmental variable.

A **TF_INFO**

B **TF_LOG**

C **TF_TRACE**

D **TF_DEBUG**

Question 39:Terraform has detailed logs which can be enabled by setting the _____ environmental variable.

A **TF_INFO**

B **TF_LOG**

C **TF_TRACE**

D **TF_DEBUG**

Explanation

Terraform has detailed logs which can be enabled by setting the `TF_LOG` environment variable to any value. This will cause detailed logs to appear on stderr.

You can set `TF_LOG` to one of the log levels `TRACE` or `DEBUG` `INFO` `WARN` `ERROR` to change the verbosity of the logs. `TRACE` is the most verbose and it is the default if `TF_LOG` is set to something other than a log level name.

https://www.terraform.io/docs/internals/debugging.html

Question 40:A user runs `terraform init` on their RHEL based server and per the output, two provider plugins are downloaded:-

Where are these plugins downloaded to?

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.44.0...
- Downloading plugin for provider "random" (hashicorp/random) 2.2.1...

                :

Terraform has been successfully initialized!
```

A **The .terraform directory in the directory terraform init was executed in**

B **The .terraform.d directory in the directory terraform init was executed in**

C **The .terraform.plugins directory in the directory terraform init was executed in**

D **/etc/terraform/plugins**

Question 40:A user runs `terraform init` on their RHEL based server and per the output, two provider plugins are downloaded:-

Where are these plugins downloaded to?

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.44.0...
- Downloading plugin for provider "random" (hashicorp/random) 2.2.1...

                :

Terraform has been successfully initialized!
```

A **The .terraform directory in the directory terraform init was executed in**

B **The .terraform.d directory in the directory terraform init was executed in**

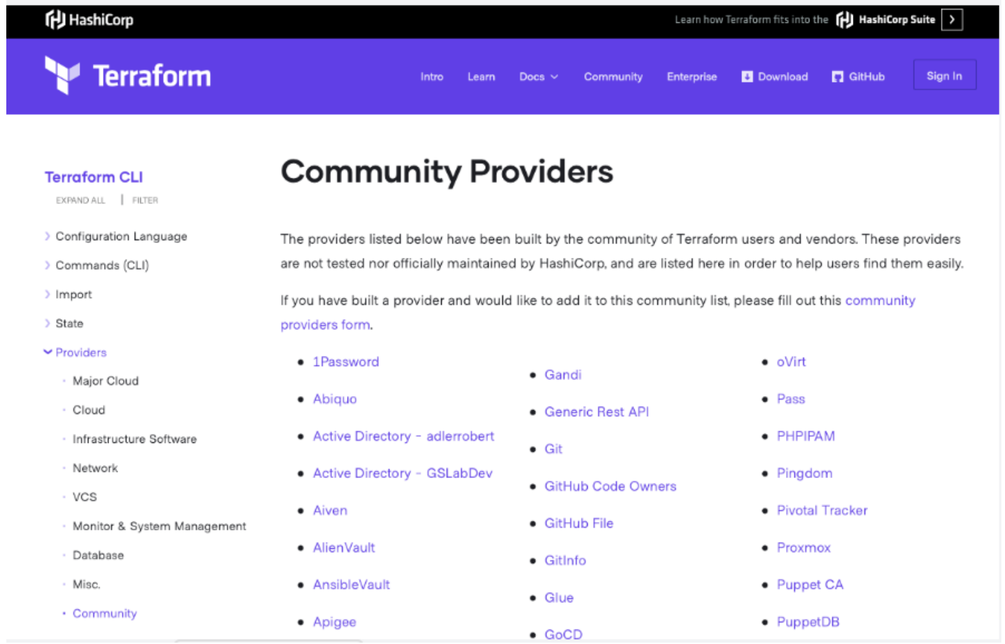
C **The .terraform.plugins directory in the directory terraform init was executed in**

D **/etc/terraform/plugins**

Explanation

By default, `terraform init` downloads plugins into a subdirectory of the working directory, .terraform, so that each working directory is self-contained.

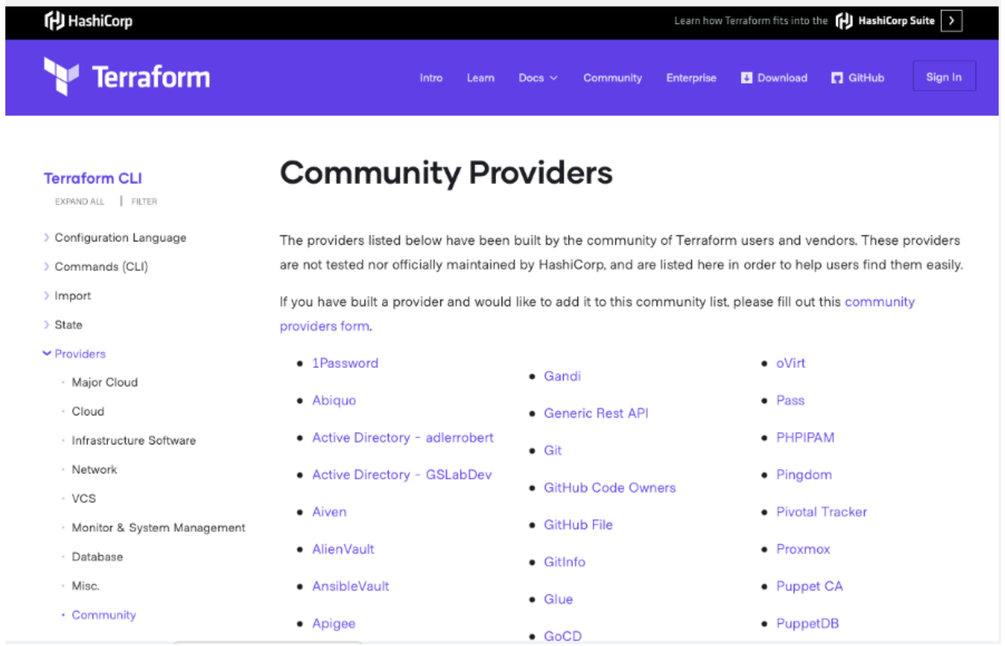
Question 41: True of False, `terraform init` cannot automatically download Community providers?



A **False**

B **True**

Question 41: True of False, `terraform init` cannot automatically download Community providers?



A **False**

B **True**

Explanation

Anyone can develop and distribute their own Terraform providers. (See Writing Custom Providers for more about provider development.) These third-party providers must be manually installed, since `terraform init` cannot automatically download them.

https://www.terraform.io/docs/configuration/providers.html#third-party-plugins

Question 42: In the example below, the `depends_on` argument creates what type of dependency?

```
resource "aws_instance" "example" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
  depends_on = [aws_s3_bucket.company_data]
}
```

A **implicit dependency**

B **internal dependency**

C **explicit dependency**

D **non-dependency resource**

Question 42: In the example below, the `depends_on` argument creates what type of dependency?

```
resource "aws_instance" "example" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
  depends_on = [aws_s3_bucket.company_data]
}
```

A **implicit dependency**

B **internal dependency**

C **explicit dependency**

D **non-dependency resource**

Explanation

Sometimes there are dependencies between resources that are not visible to Terraform. The `depends_on` argument is accepted by any resource and accepts a list of resources to create explicit dependencies for.

Question 43: Which statements best describes what the local variable assignment is doing in the following code snippet

```
variable "subnet_details" {
  type = list(object({
    cidr             = string
    subnet_name      = string
    route_table_name = string
    aznum            = number
  }))
}

locals {
  route_tables_all = distinct([for s in var.subnet_details : s.route_table_name ])
}
```

A **Create a map of route table names from a list of subnet names**

B **Create a list of route table names eliminating duplicates**

C **Create a map of route table names to subnet names**

D **Create a distinct list of route table names objects**

Question 43: Which statements best describes what the local variable assignment is doing in the following code snippet

```
variable "subnet_details" {
  type = list(object({
    cidr             = string
    subnet_name      = string
    route_table_name = string
    aznum            = number
  }))
}

locals {
  route_tables_all = distinct([for s in var.subnet_details : s.route_table_name ])
}
```

A — Create a map of route table names from a list of subnet names

B — Create a list of route table names eliminating duplicates

C — Create a map of route table names to subnet names

D — Create a distinct list of route table names objects

Explanation

route_tables_all is assigned a list of unique route table names filtered from a list of objects describing subnet details, one of those object attributes being route_table_name.

Question 44: When writing Terraform code, HashiCorp recommends that you use how many spaces between each nesting level?

A	**2**
B	**4**
C	**1**
D	**5**

Question 44: When writing Terraform code, HashiCorp recommends that you use how many spaces between each nesting level?

A **2**

B **4**

C **1**

D **5**

Explanation

HashiCorp style conventions state that you should use 2 spaces between each nesting level to improve the readability of Terraform configurations.

Check this link for more information

Question 45: In the following code snippet, the block type is identified by which string:-

```
resource "aws_instance" "db" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
}
```

A **instance_type**

B **resource**

C **“db”**

D **“aws_instance”**

Question 45: In the following code snippet, the block type is identified by which string:-

```
resource "aws_instance" "db" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
}
```

A **instance_type**

B **resource**

C **“db”**

D **“aws_instance”**

Explanation

The format of resource block configurations is as follows:-

<block type> "<resource type>" "<local name/label>"

Question 46: A "backend" in Terraform determines how state is loaded and how an operation such as `apply` is executed. Which of the following is not a supported backend type:-

A **Terraform enterprise**

B **consul**

C **github**

D **artifactory**

E **S3**

Question 46: A "backend" in Terraform determines how state is loaded and how an operation such as `apply` is executed. Which of the following is not a supported backend type:-

A **Terraform enterprise**

B **consul**

C **github**

D **artifactory**

E **S3**

Explanation

github is not a supported backend type.

https://www.terraform.io/docs/backends/types/index.html

Question 47: Environment variables can be used to set variables. The environment variables must be in the format "____"_<variablename>. Select the correct prefix string from the following list.

A **TF_ENV_VAR**

B **TF_VAR_NAME**

C **TF_VAR**

D **TF_ENV**

Question 47: Environment variables can be used to set variables. The environment variables must be in the format "____"_<variablename>. Select the correct prefix string from the following list.

A **TF_ENV_VAR**

B **TF_VAR_NAME**

C **TF_VAR**

D **TF_ENV**

Environment variables can be used to set variables. The environment variables must be in the format `TF_VAR_name` and this will be checked last for a value. For example:

```
export TF_VAR_region=us-west-1
export TF_VAR_ami=ami-049d8641
export TF_VAR_alist='[1,2,3]'
export TF_VAR_amap='{ foo = "bar", baz = "qux" }'
```

https://www.terraform.io/docs/commands/environment-variables.html

Terraform Cloud & Terraform Enterprise

Overview

Terraform Cloud or Terraform Enterprise?

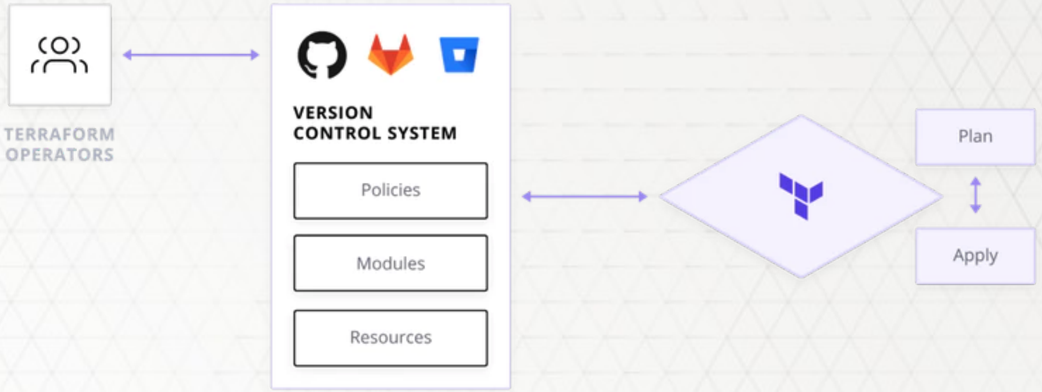
Terraform Cloud is a hosted application that provides features like remote state management, API driven runs, policy management and more. Many users prefer a cloud based SaaS solution because they don't want to maintain the infrastructure to run it.

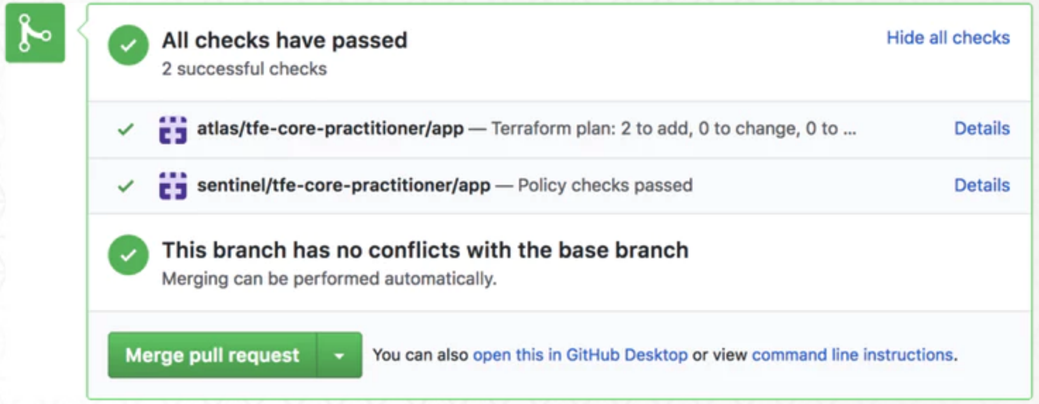
Terraform Enterprise is the same application, but it runs in your cloud environment or data center. Some users require more control over the Terraform Enterprise application, or wish to run it in restricted networks behind corporate firewalls.

Terraform Cloud Features

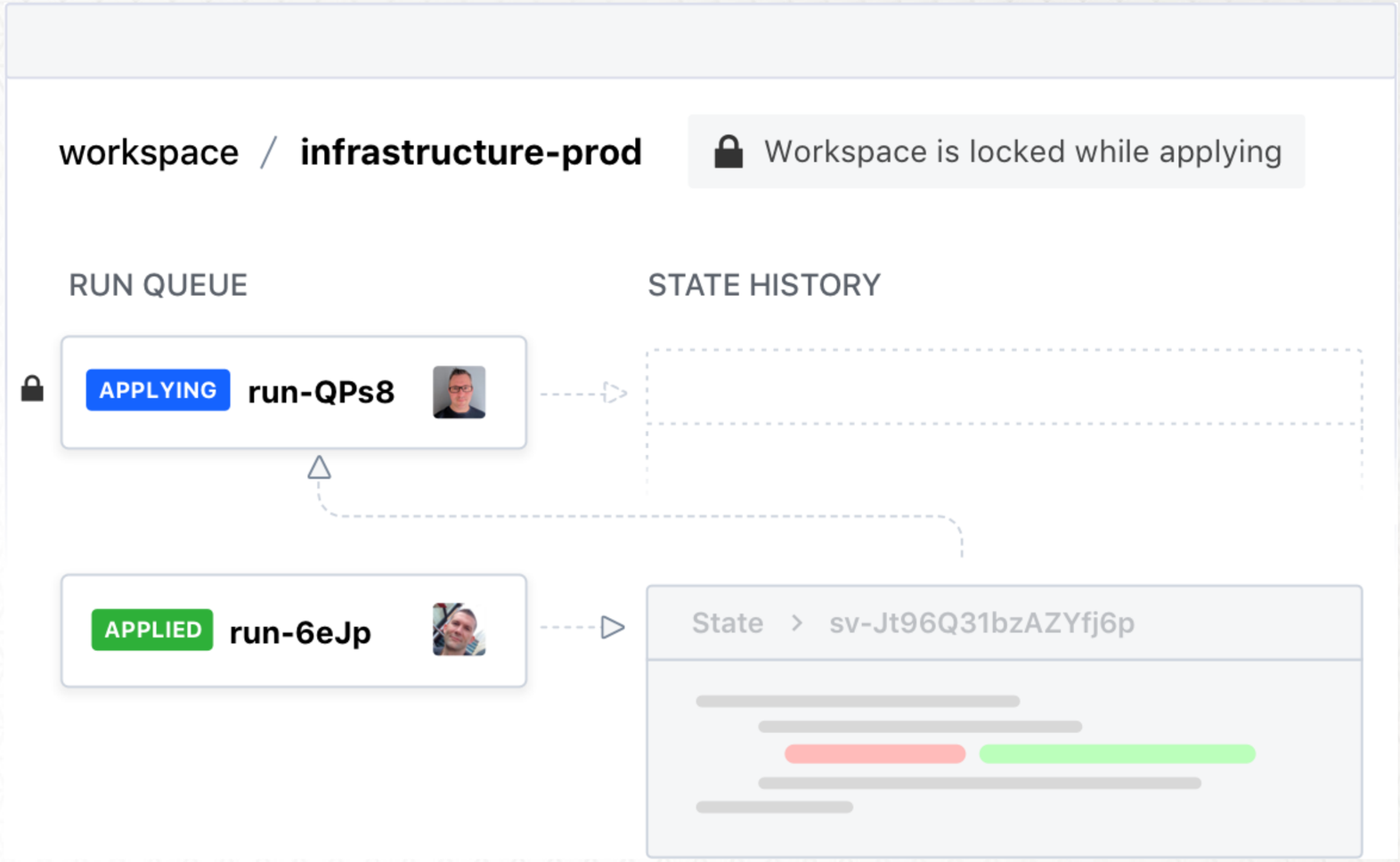
- SaaS (no infrastructure to manage)
- VCS Integration
- Workspace Management
- Statefile Management
- Private Module Registry
- Remote Runs
- Team Management
- Sentinel (Policy-as-Code)
- Cost Estimation (AWS, Azure, GCP)

VCS Flow

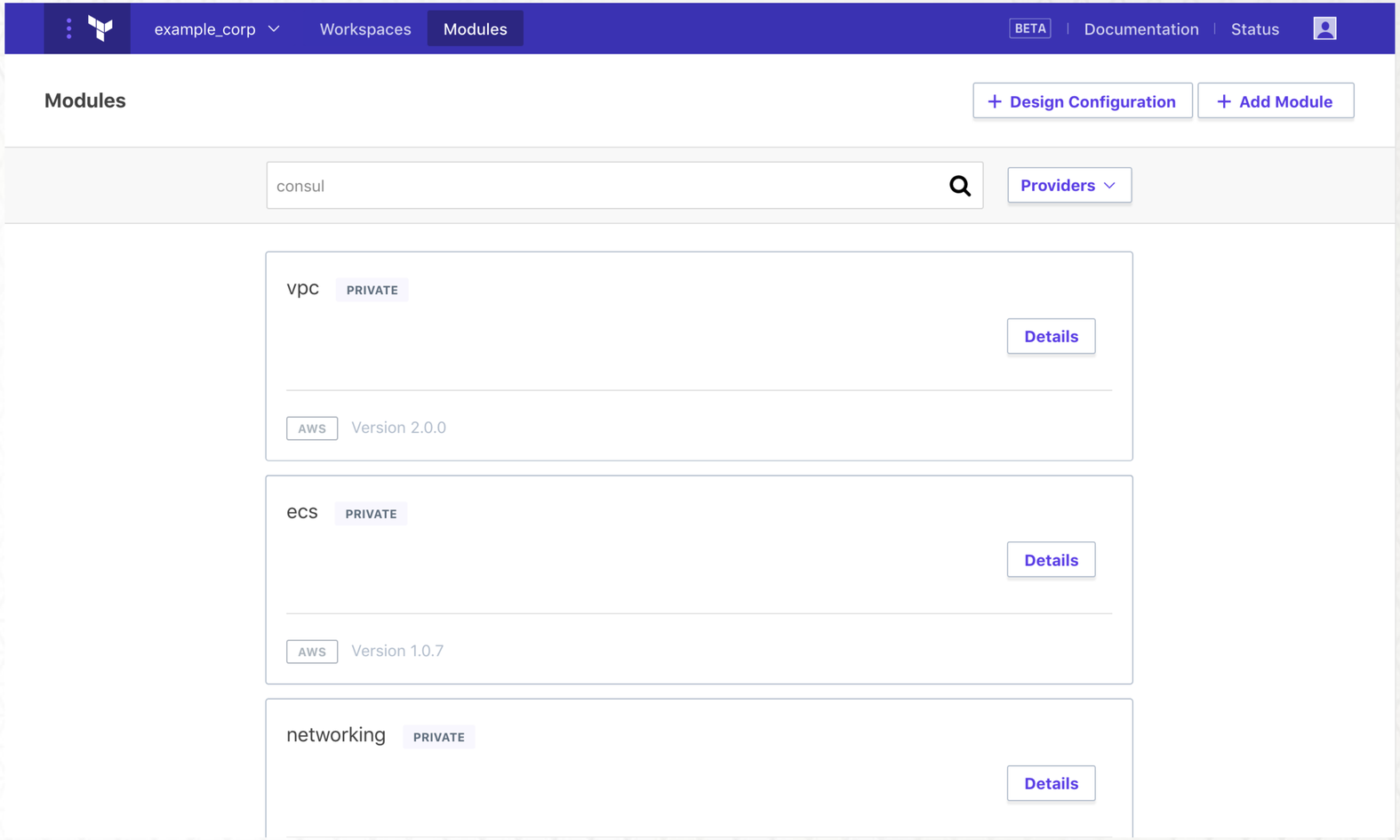

TERRAFORM
OPERATORS
VERSION
CONTROL SYSTEM
Policies
Modules
Resources
Plan
Apply


All checks have passed
2 successful checks
Hide all checks
atlas/tfe-core-practitioner/app — Terraform plan: 2 to add, 0 to change, 0 to ...
Details
sentinel/tfe-core-practitioner/app — Policy checks passed
Details
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.

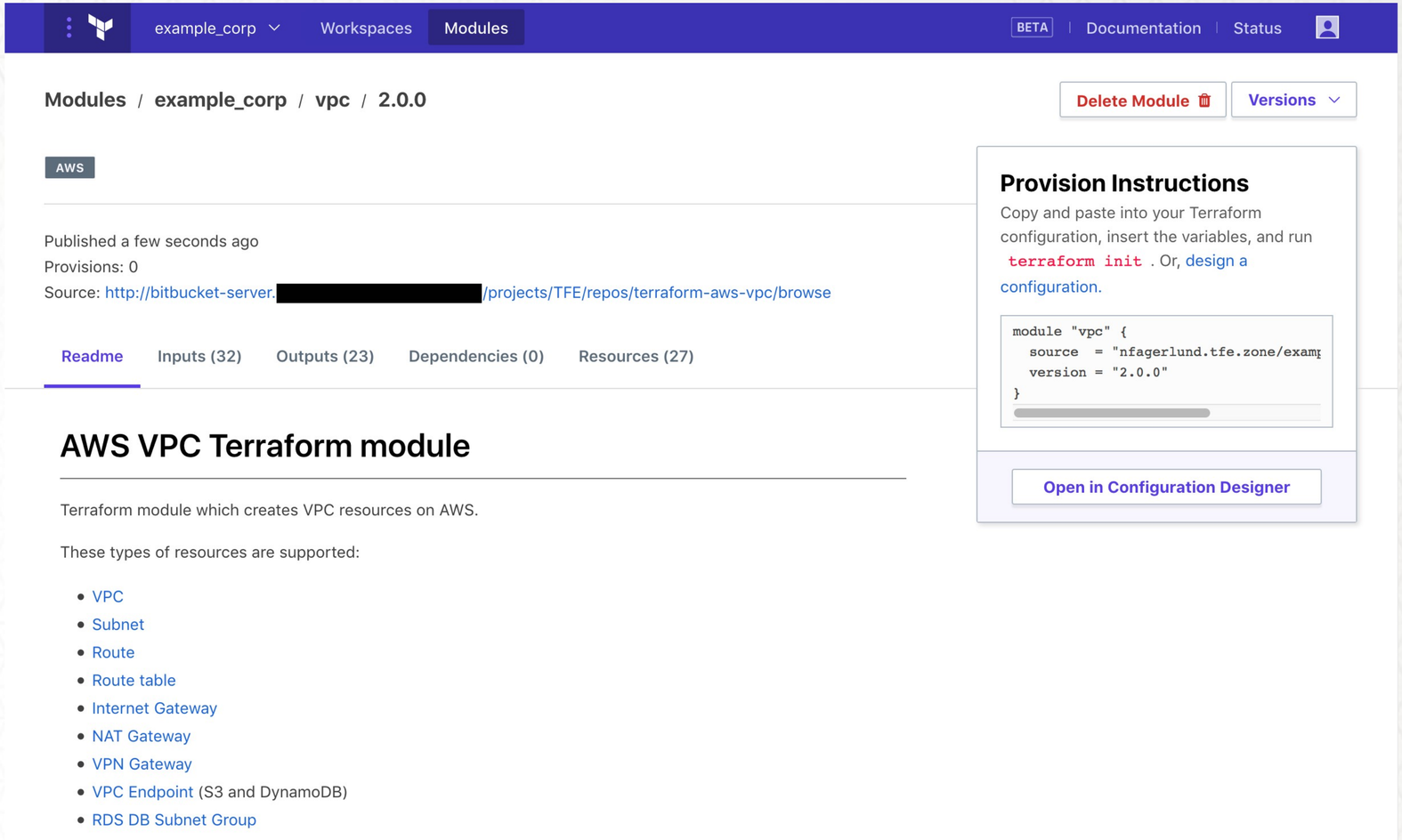
General Workflow



Private Module Registry



Private Module Registry



Terraform Cloud Limitations

- 1 Terraform Run at a time
- 5 users max (Free)
- Two paid tiers
- No published list of public IP's

https://www.hashicorp.com/products/terraform/pricing#compare

Terraform Enterprise Features

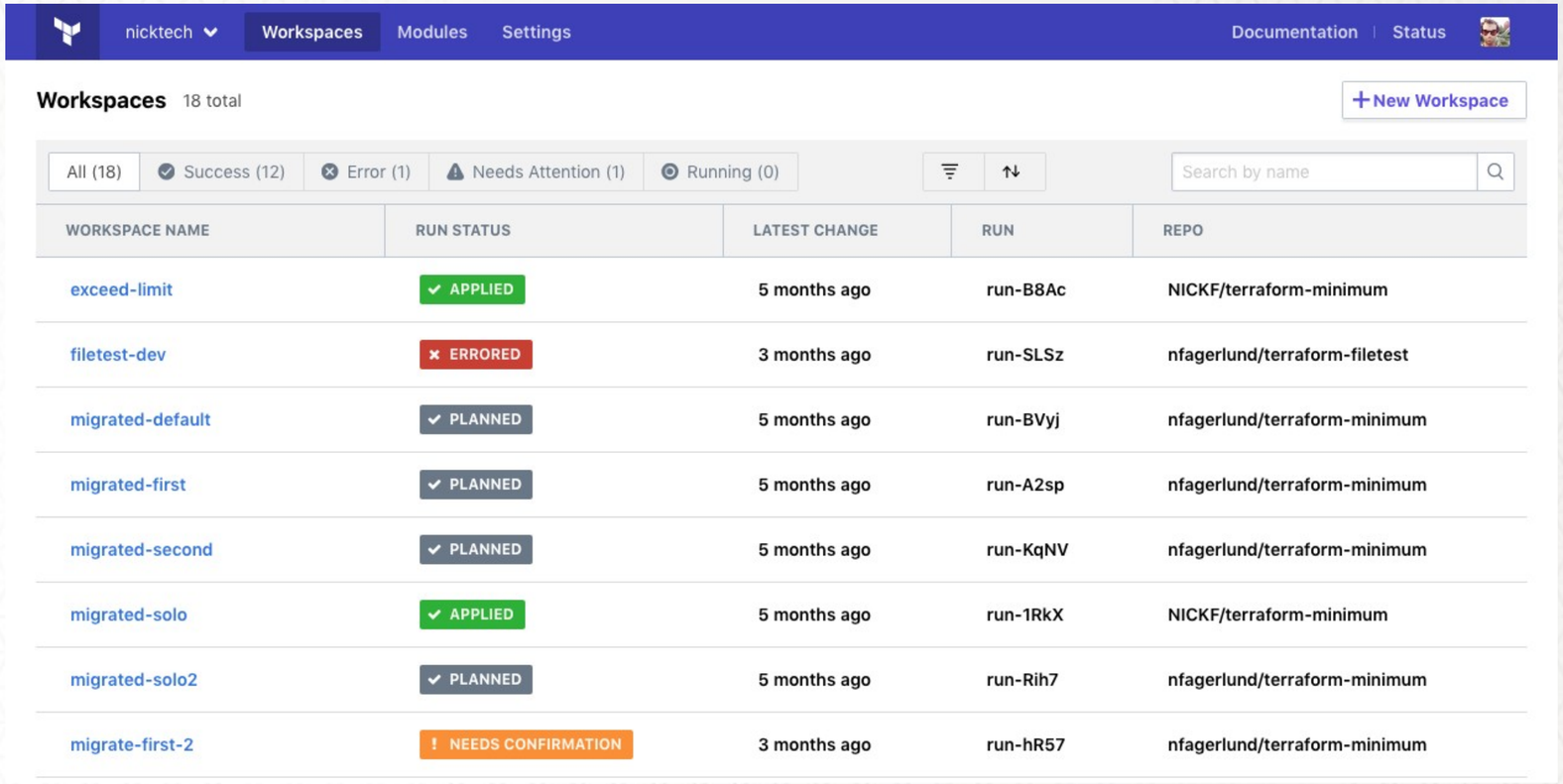
- All the features of TFC
- SAML/SSO
- Self-Hosted
- Standalone & Clustering (Future Work)

Demo

Terraform Enterprise

Workspaces

Workspaces


nicktech
Workspaces
Modules
Settings
Documentation
Status
Workspaces 18 total
+ New Workspace
All (18)
Success (12)
Error (1)
Needs Attention (1)
Running (0)
Search by name
WORKSPACE NAME
RUN STATUS
LATEST CHANGE
RUN
REPO
exceed-limit
APPLIED
5 months ago
run-B8Ac
NICKF/terraform-minimum
filetest-dev
ERRORED
3 months ago
run-SLSz
nfagerlund/terraform-filetest
migrated-default
PLANNED
5 months ago
run-BVyj
nfagerlund/terraform-minimum
migrated-first
PLANNED
5 months ago
run-A2sp
nfagerlund/terraform-minimum
migrated-second
PLANNED
5 months ago
run-KqNV
nfagerlund/terraform-minimum
migrated-solo
APPLIED
5 months ago
run-1RkX
NICKF/terraform-minimum
migrated-solo2
PLANNED
5 months ago
run-Rih7
nfagerlund/terraform-minimum
migrate-first-2
NEEDS CONFIRMATION
3 months ago
run-hR57
nfagerlund/terraform-minimum

What is in a Workspace

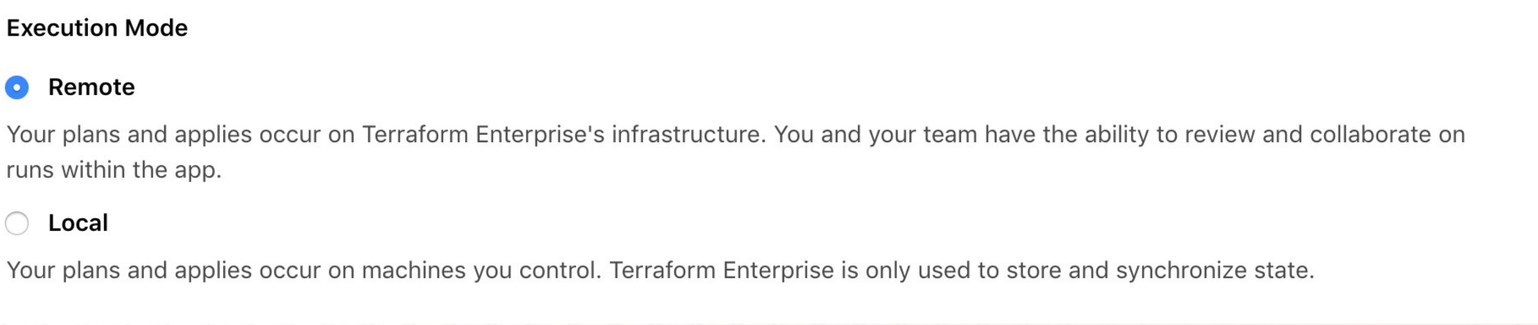
- Name
- VCS Connection
- Workspace Variables
- Environment Variables
- State
- Runs

TFE Workspaces and VCS Integration

Terraform Enterprise

Workflows

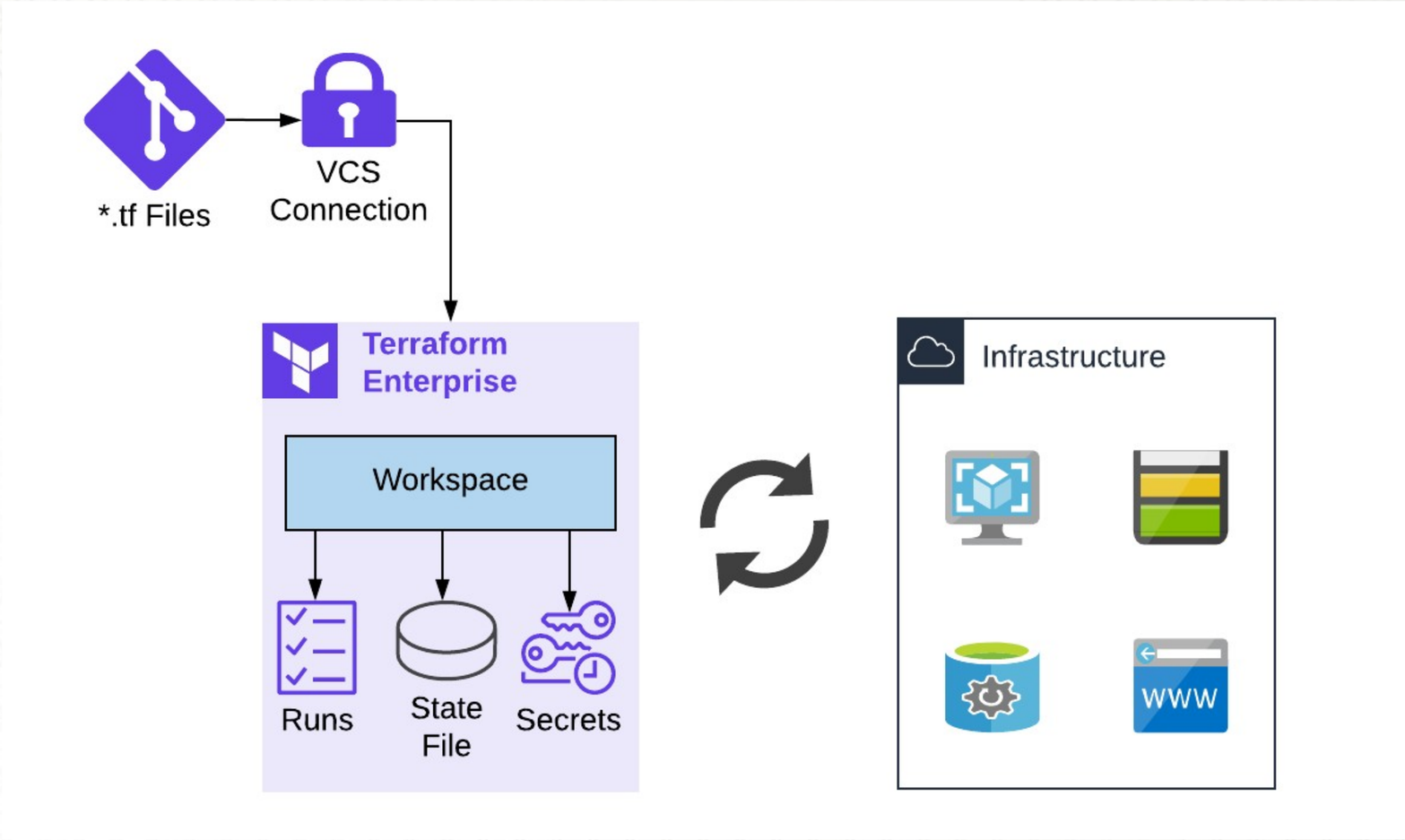
Execution Mode



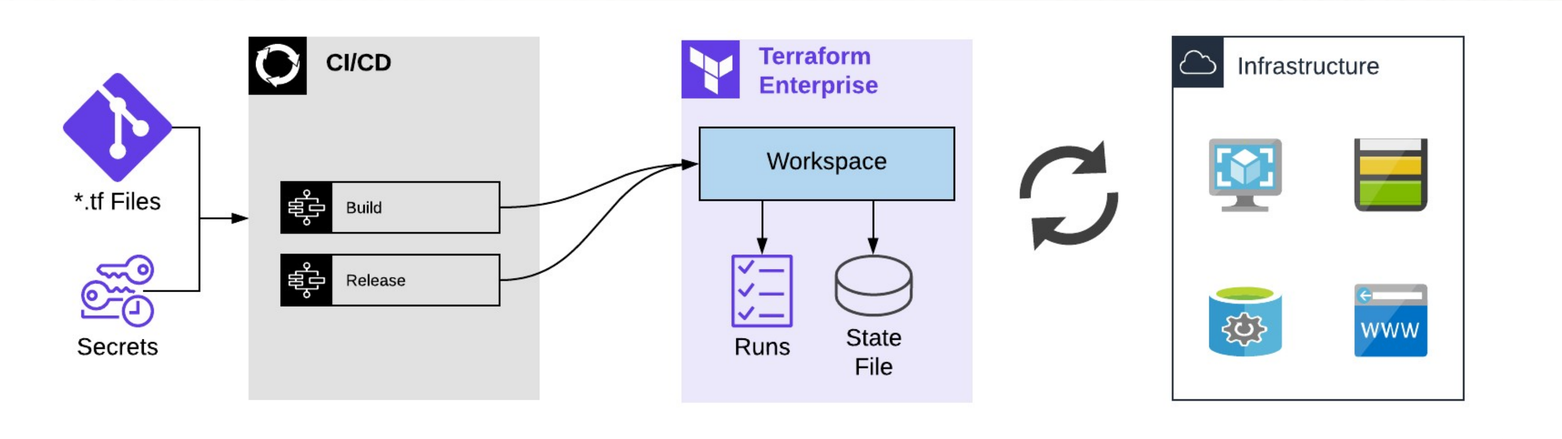
Workspace Interactions

- GUI
- REST API https://www.terraform.io/docs/cloud/api
- `terraform` CLI
- `tfh` CLI, TF Helper https://github.com/hashicorp-community/tf-helper

VCS Driven


*.tf Files
VCS
Connection
Terraform
Enterprise
Workspace
Runs
State
File
Secrets
Infrastructure
WWW

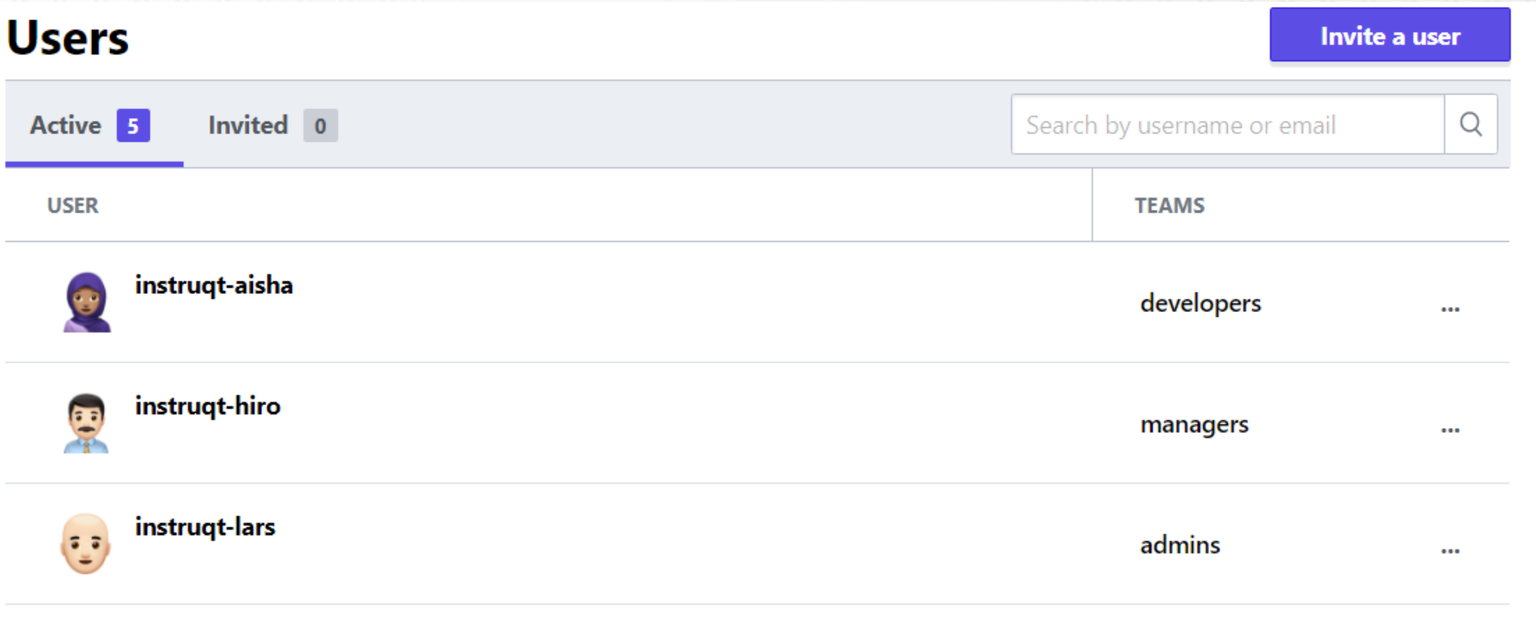
API Driven


CI/CD
*.tf Files
Secrets
Build
Release
Terraform Enterprise
Workspace
Runs
State File
Infrastructure
WWW

Terraform Enterprise

Team Management

Teams


Users
Invite a user
Active 5
Invited 0
Search by username or email
USER
TEAMS
instruqt-aisha
developers
instruqt-hiro
managers
instruqt-lars
admins

Team Types

- `site-admins` - Platform Admins
- `owners` - Organization Owners
- `<owners defined>` - Owners/Admin Managed

Team Permissions

Organization Access

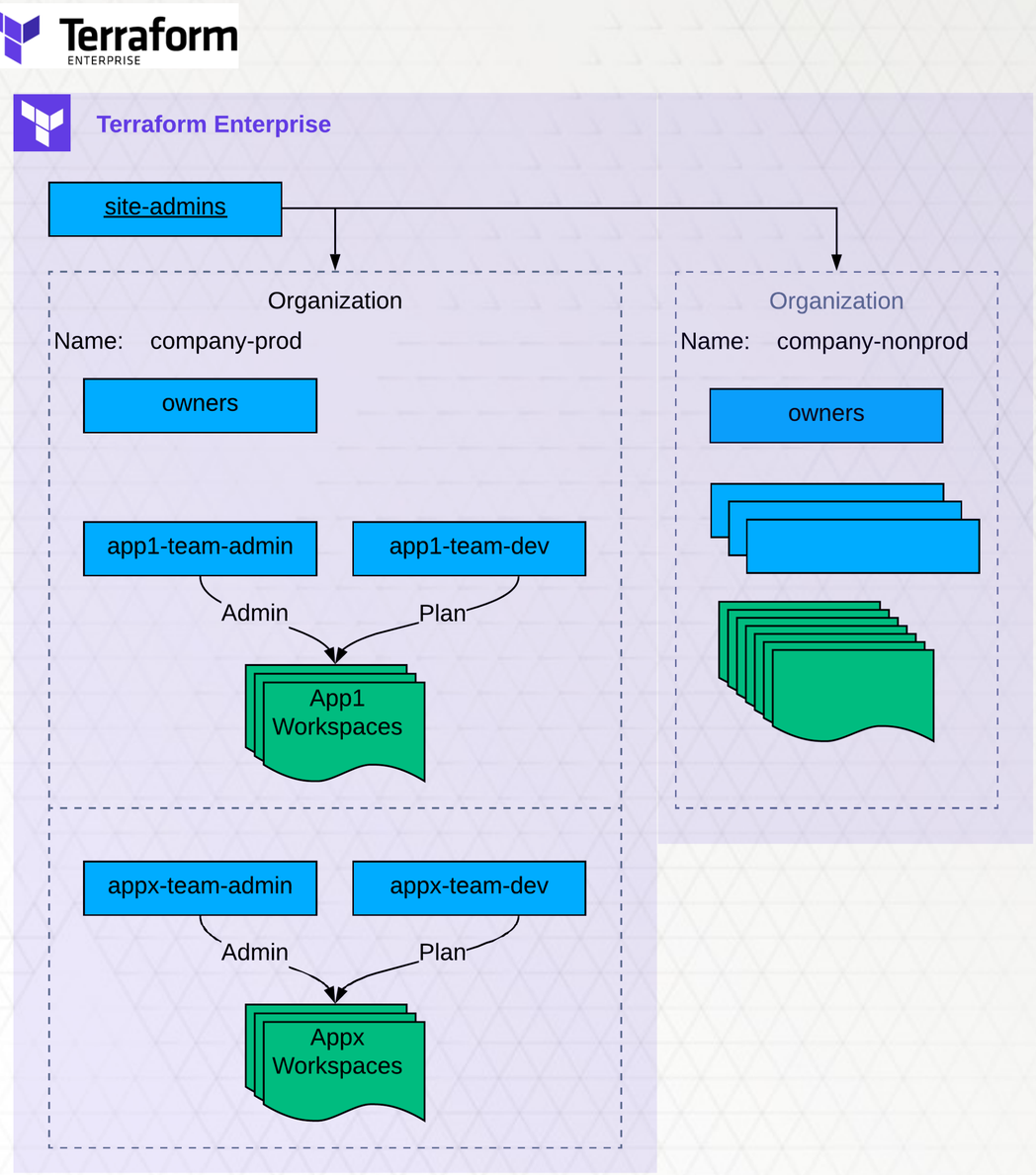
- Manage Policies
- Manage Workspaces
- Manage VCS Settings
- Team Visibility

Team Permissions

Workspace Access

Read Plan

Write Admin

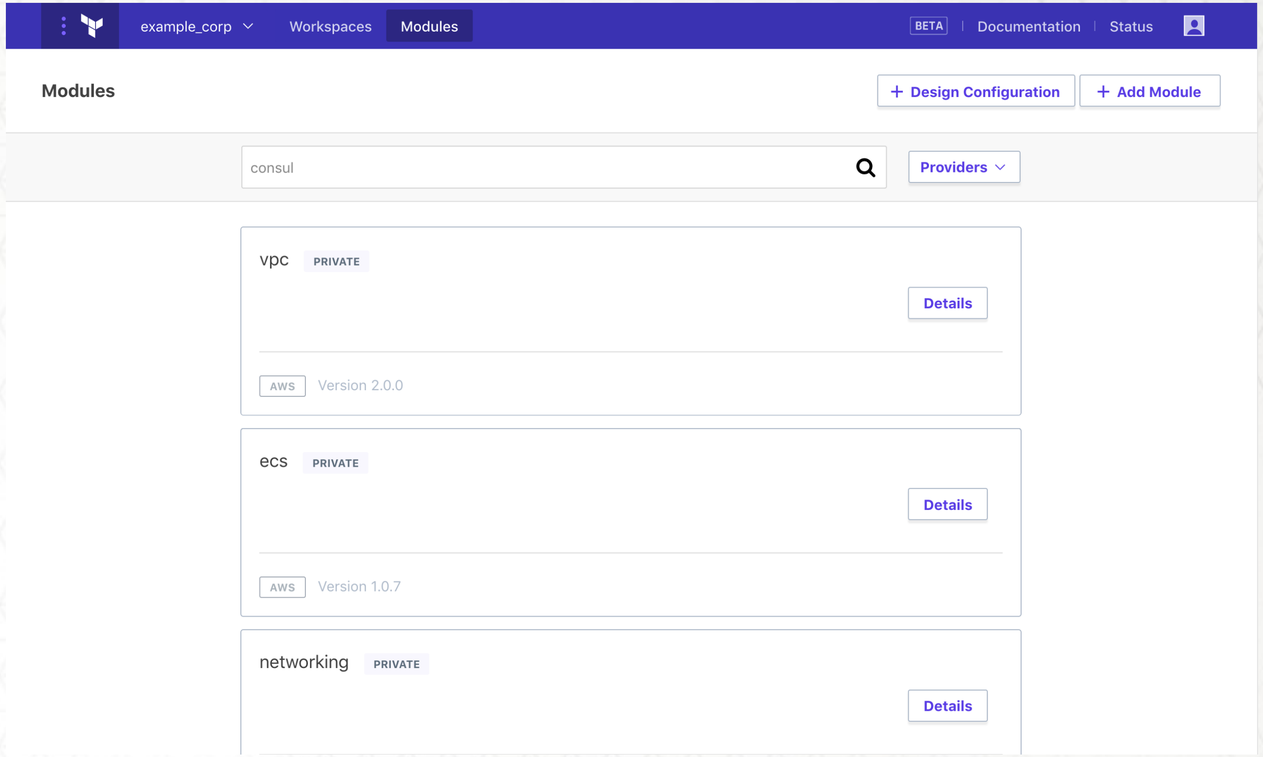

Terraform
ENTERPRISE
Terraform Enterprise
site-admins
Organization
Name: company-prod
owners
app1-team-admin
app1-team-dev
Admin
Plan
App1
Workspaces
appx-team-admin
appx-team-dev
Admin
Plan
Appx
Workspaces
Organization
Name: company-nonprod
owners

Demo

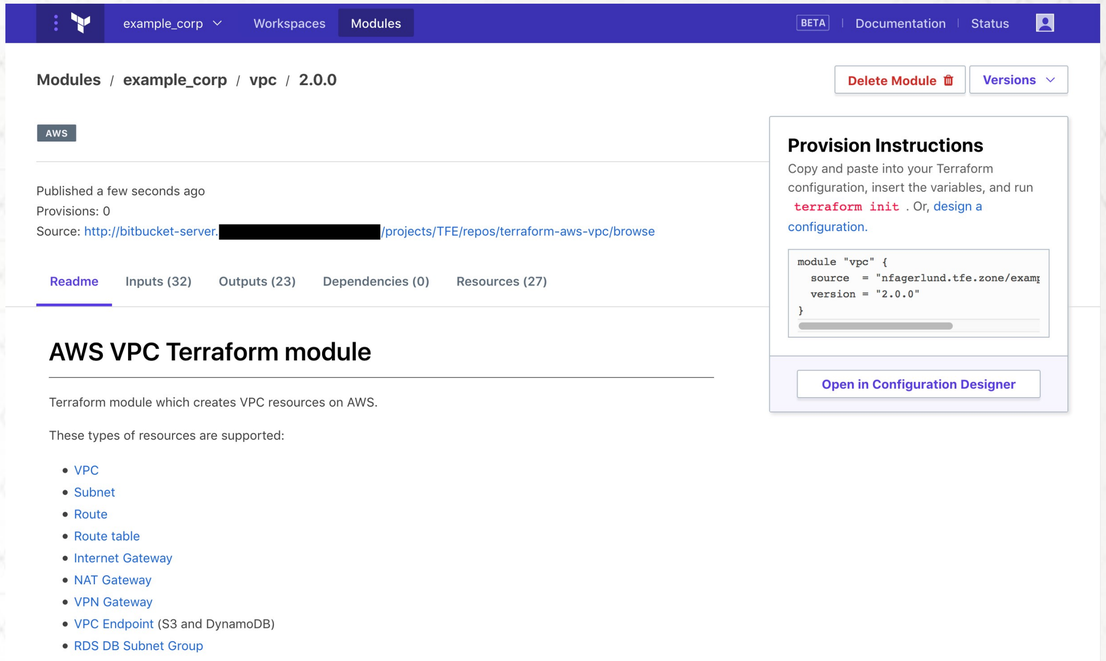
Terraform Enterprise

Private Module Registry

Modules


example_corp
Workspaces
Modules
BETA
Documentation
Status
Modules
+ Design Configuration
+ Add Module
consul
Providers
vpc
PRIVATE
Details
AWS
Version 2.0.0
ecs
PRIVATE
Details
AWS
Version 1.0.7
networking
PRIVATE
Details

Module Definition



Module Publishing

- Repo Name: `terraform-<provider>-<name>`
- Version Tag: `v1.0.4`

Using PMR

```
module "vpc" {
 source    = "tfe.mycompany.com/terraform-aws-mod/vpc/aws"
 version = "2.17.0"
 ...
}
```

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

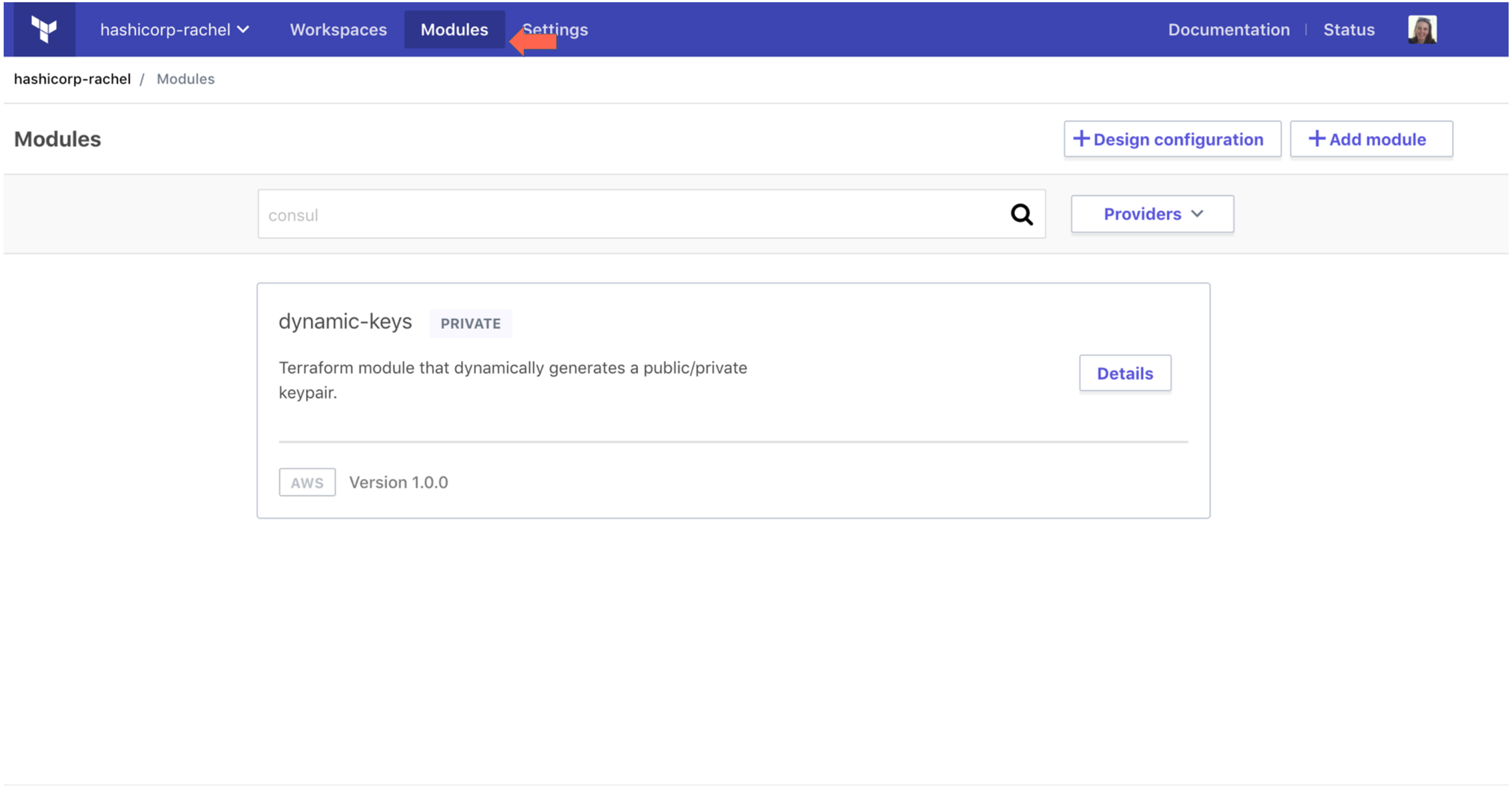
GitHub
Bitbucket (private)

Terraform Registry

Versioning, Documentation,
Search

EXAMPLES

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


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

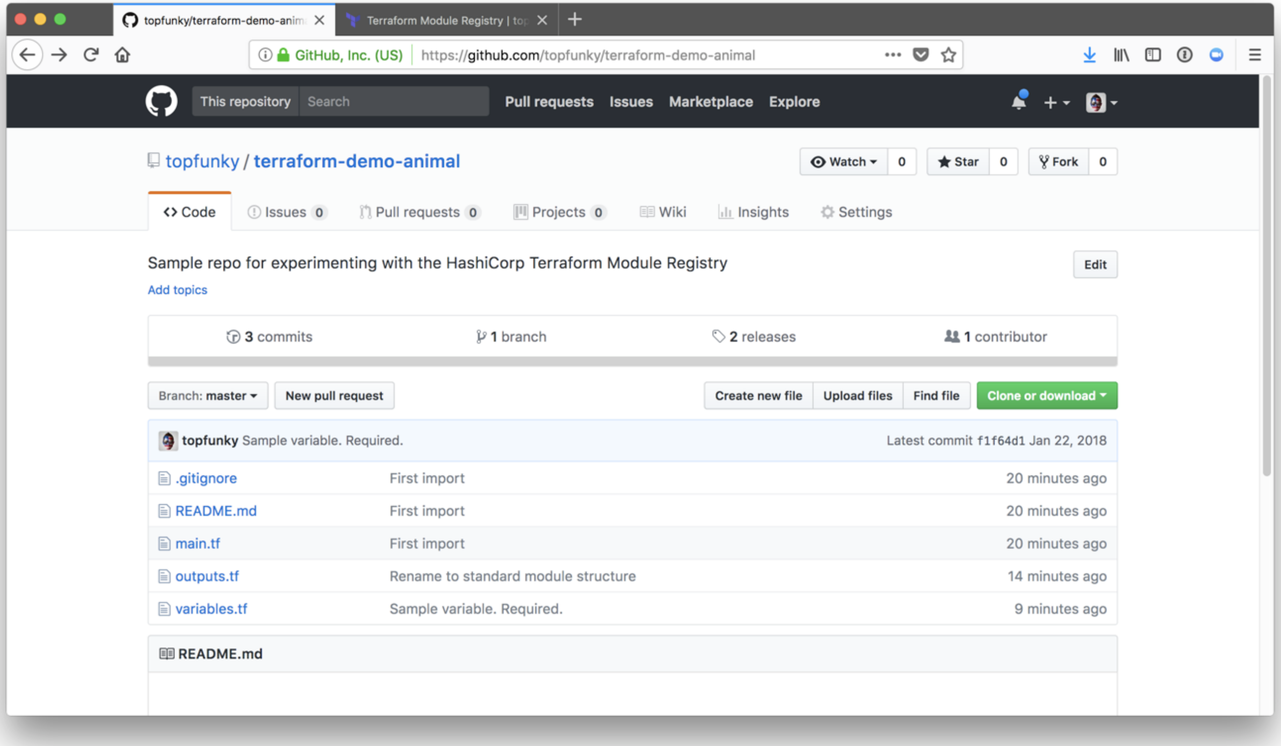
Push code to GitHub or Bitbucket. The repository must be named in the format:

terraform-PROVIDER-NAME

Examples:

terraform-aws-peering

terraform-azure-autoscaling

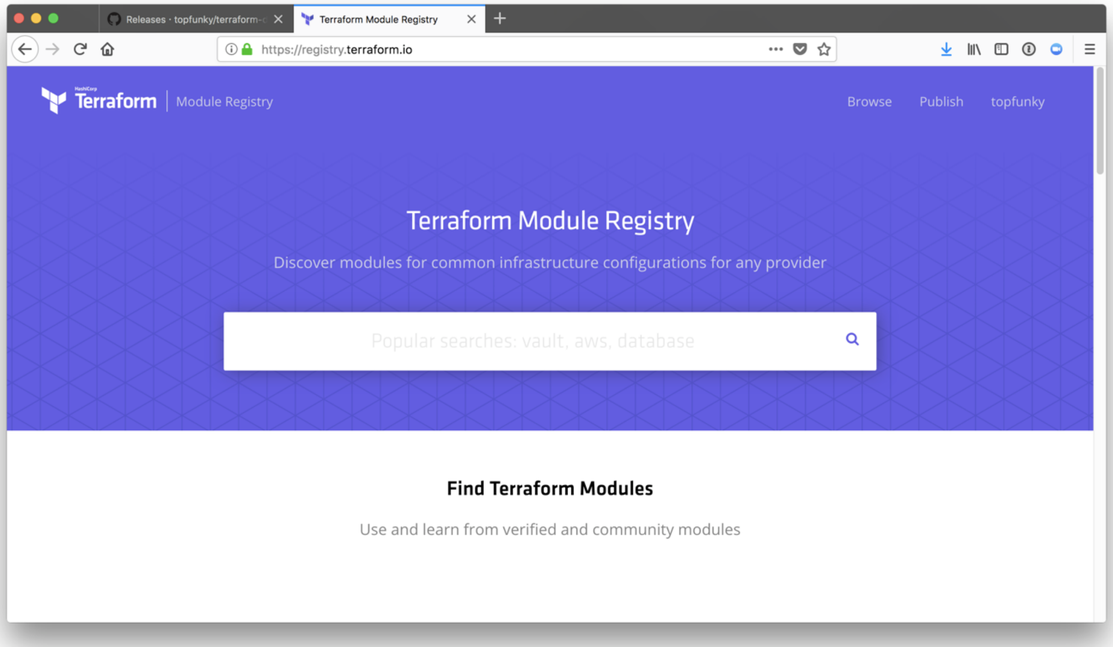


Module Registry Workflow: Registry

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

Tagged commits will be synced automatically.

v1.0.6



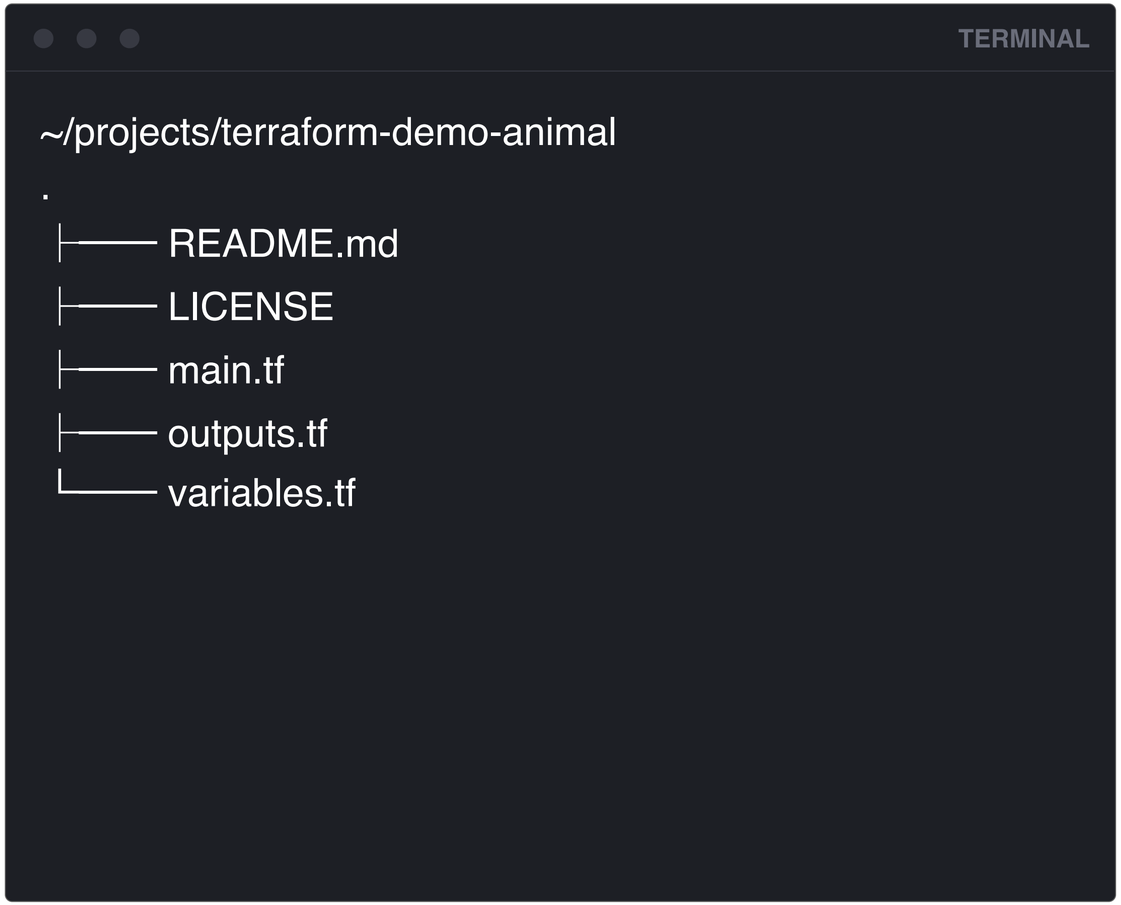
Module Registry Workflow: Config

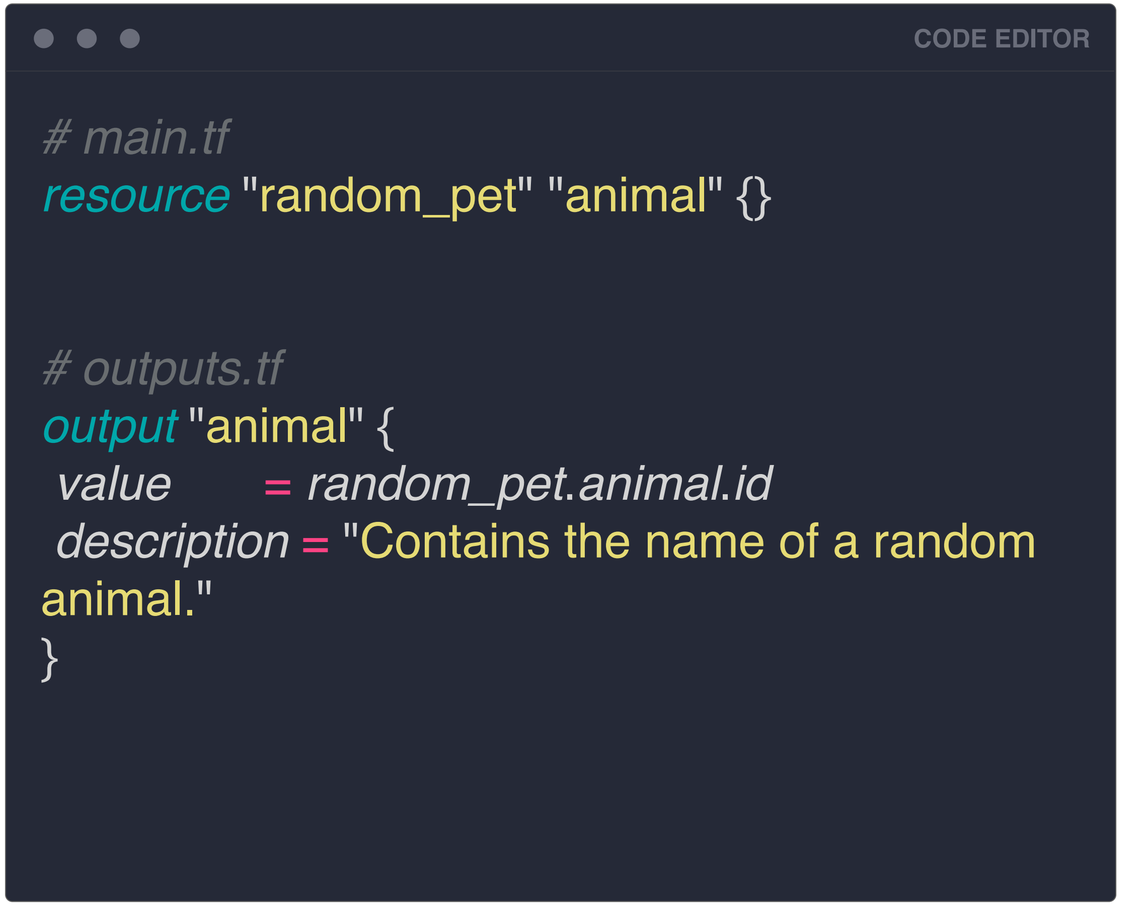
Use the code snippet from the registry to reference the module from your code. Provide all necessary variables.

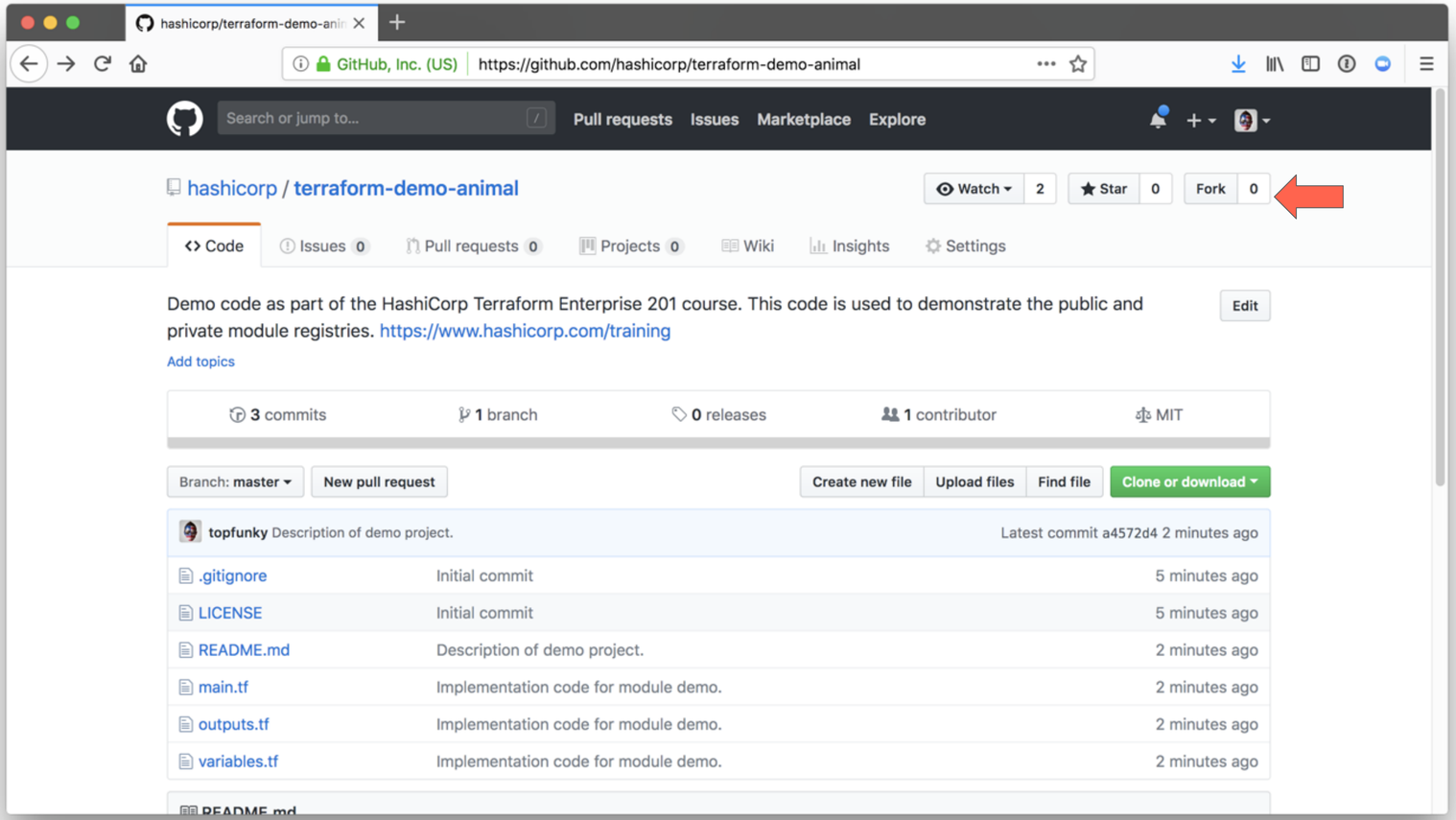
Run init to pull in the correct version of the module.

```
terraform init
```

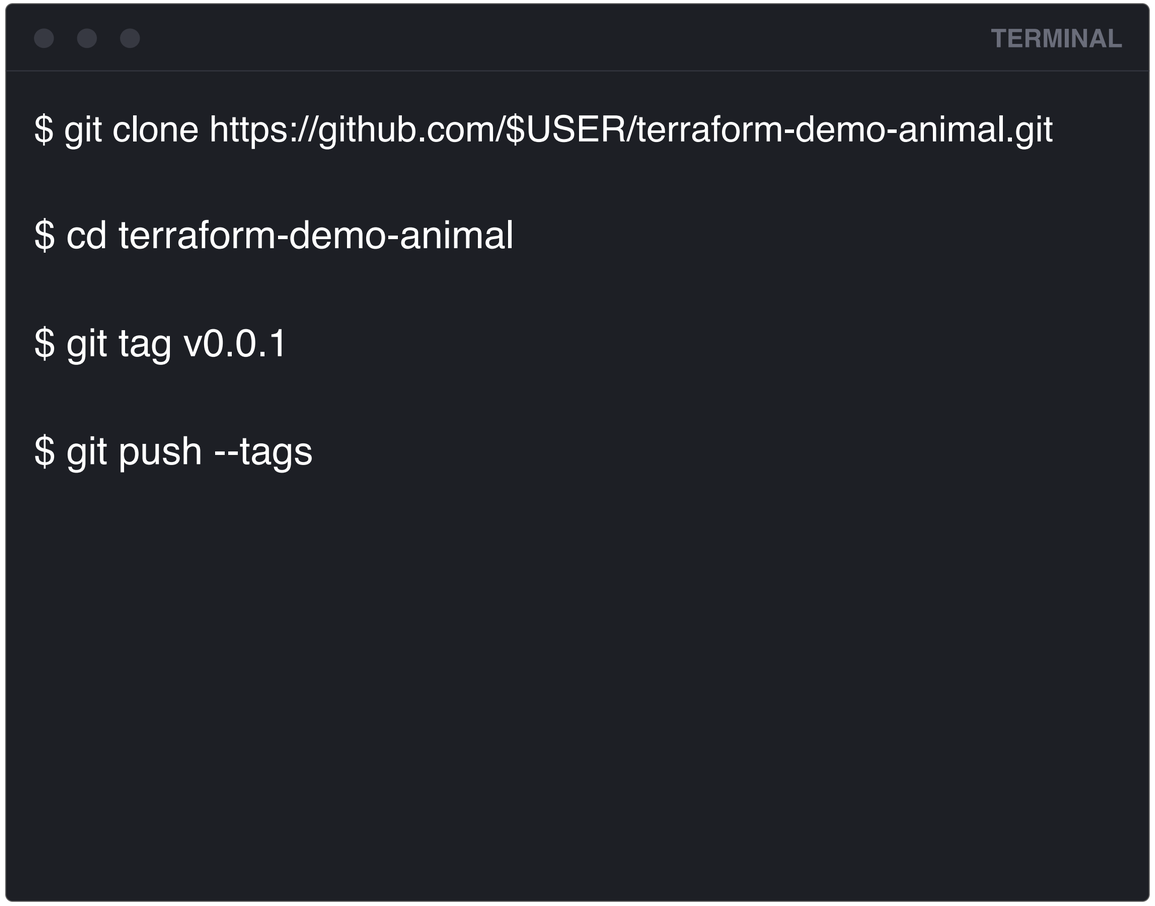
Example of Modules for Registry






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

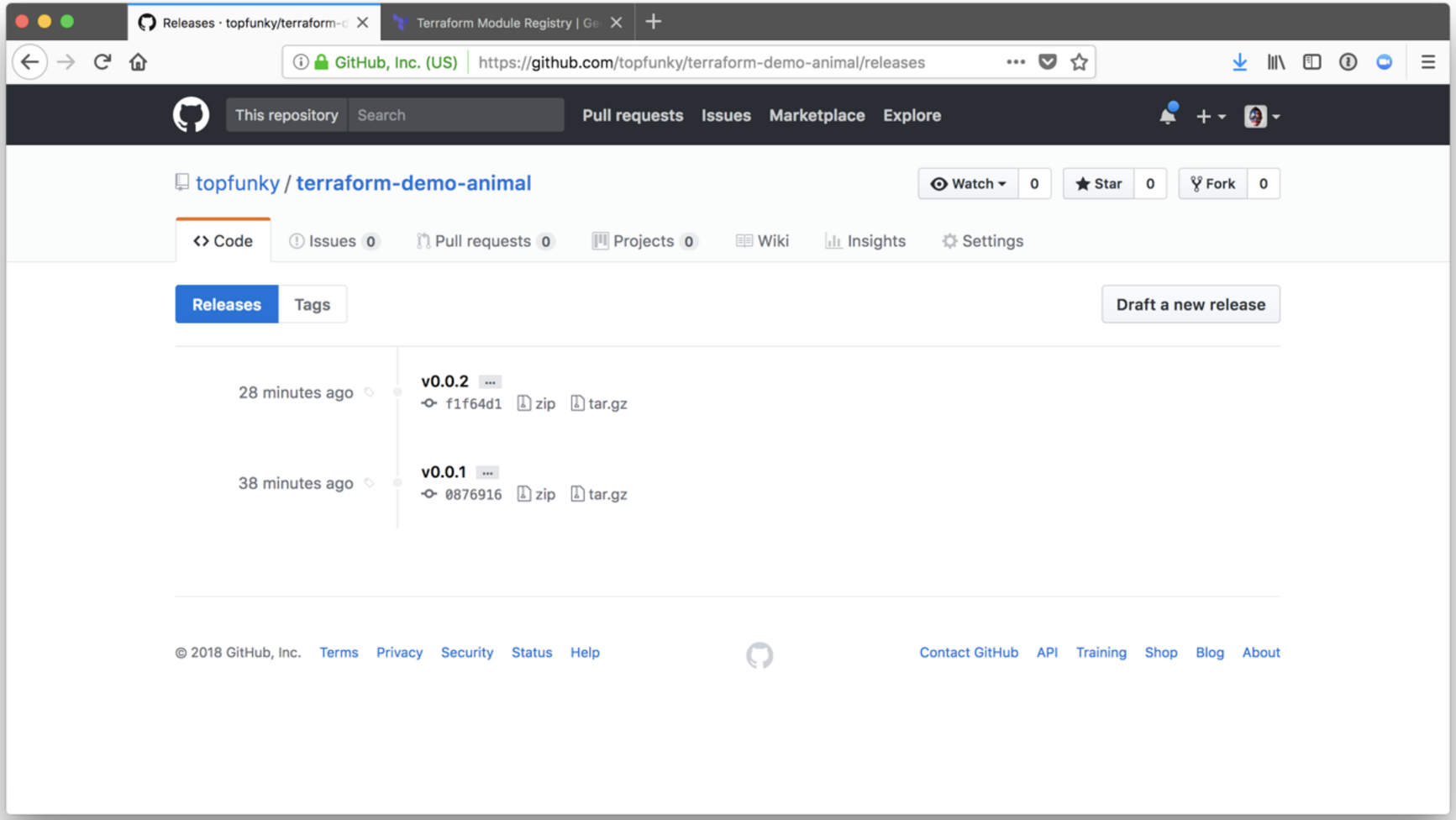


Publish the Code on Your Private Module Registry

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-c
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

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}
```

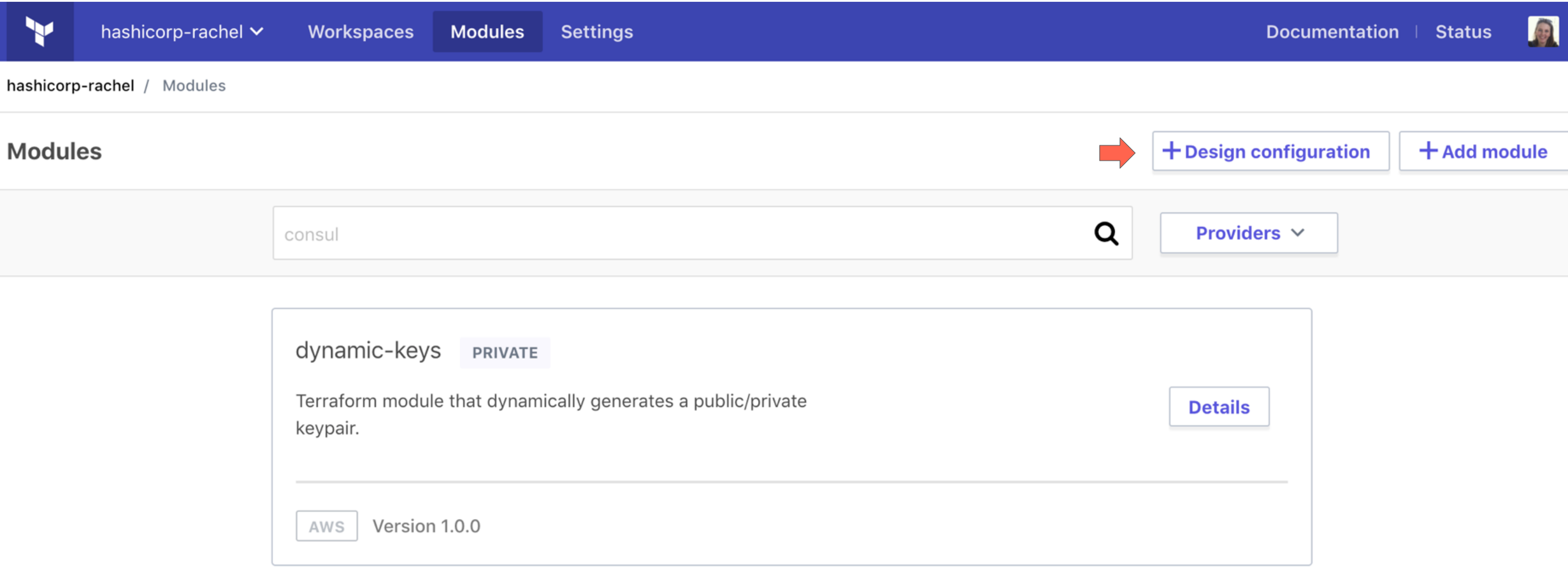
Terraform 201

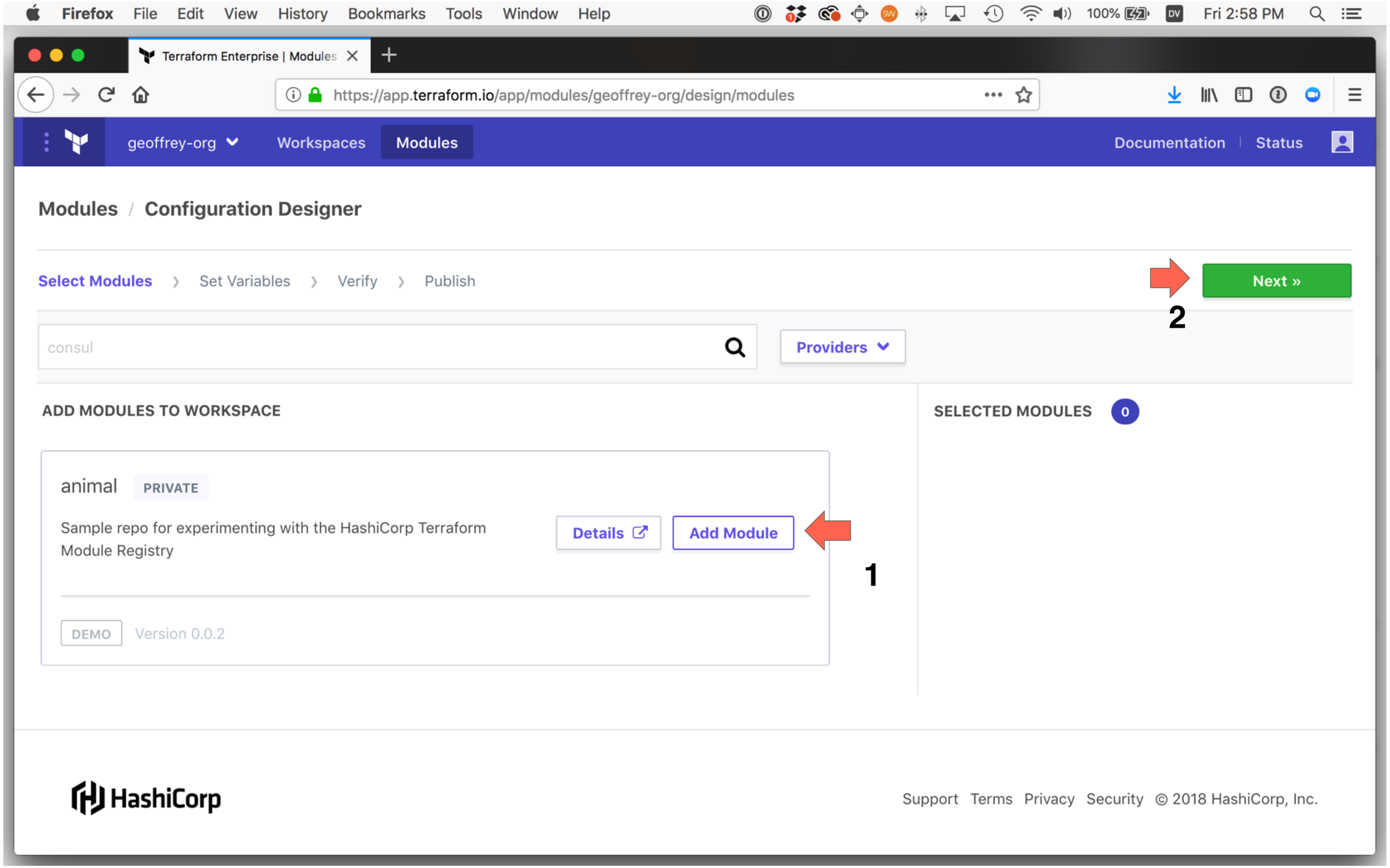
Configuration Designer

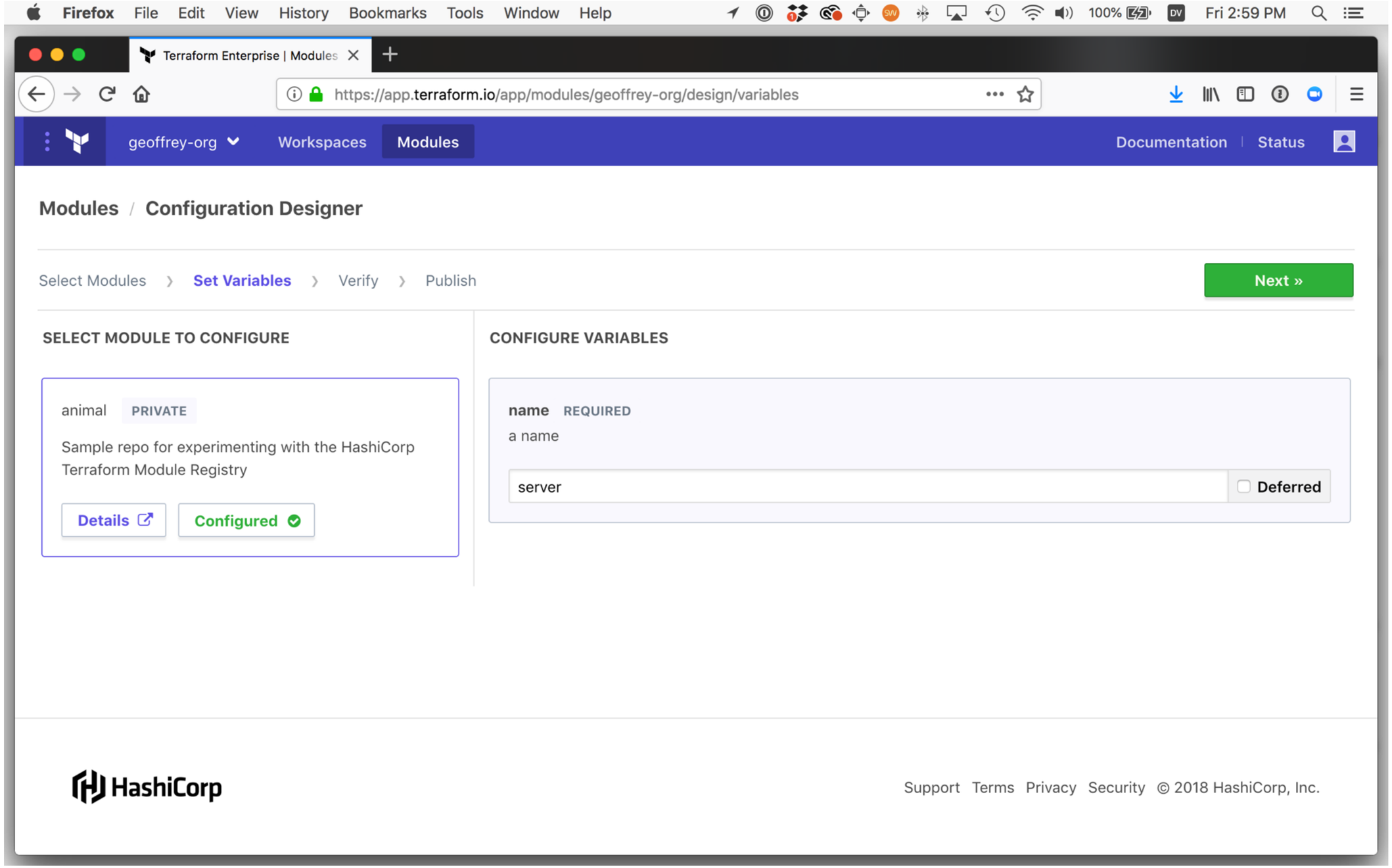
Configuration Designer

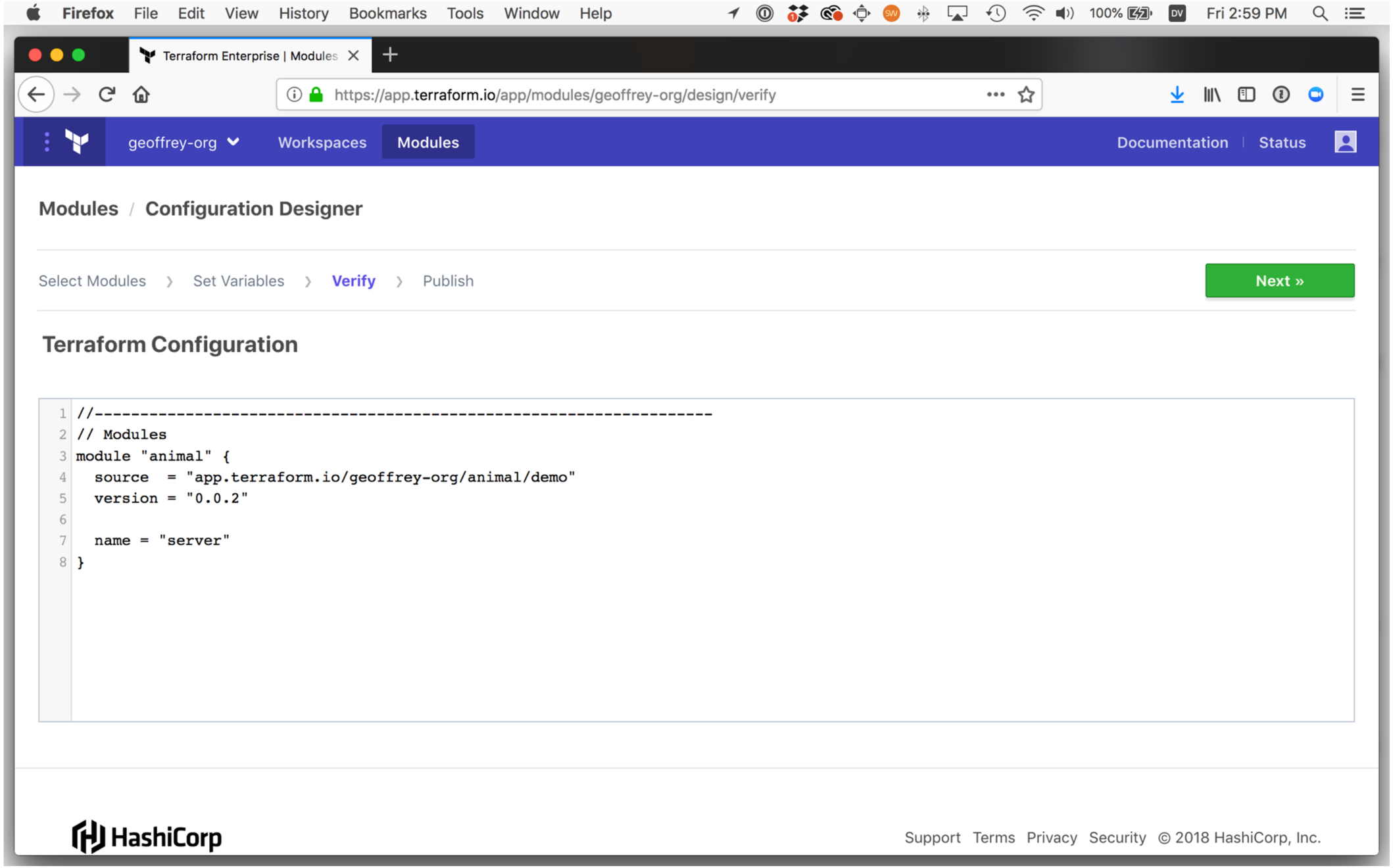
User interface for selecting modules and specifying their versions and required values.

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.

Demo

Terraform 201

Sentinel

Intro Video to 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

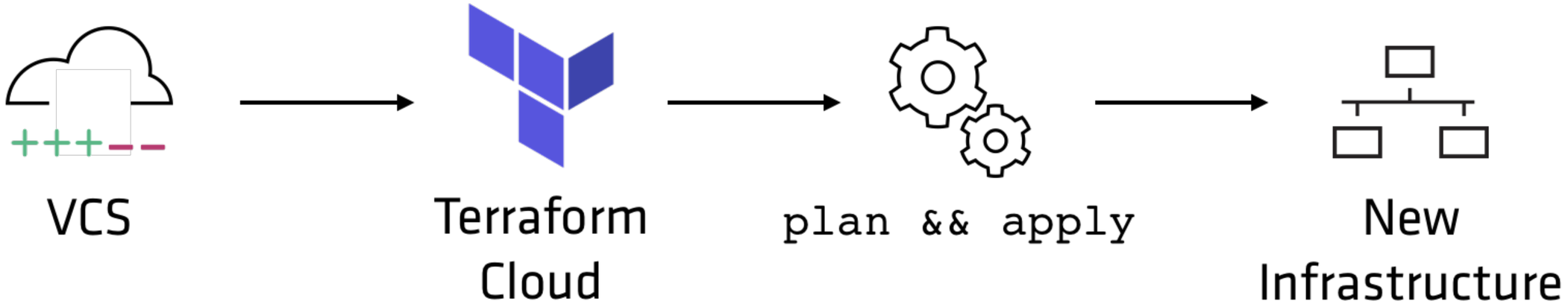
Analyze executable contents of a VM/container/etc.

Limit runtime actions of deployed applications

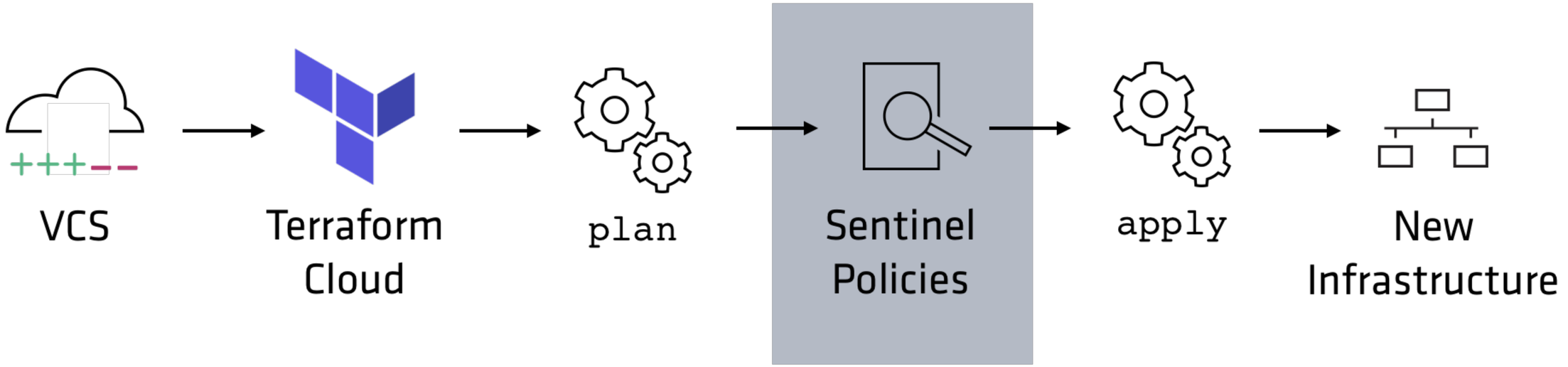
In its current implementation, Sentinel is a tool for preventing mistakes by operators acting in good faith.

– 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

Terraform 201

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.

Syntax - Conditionals

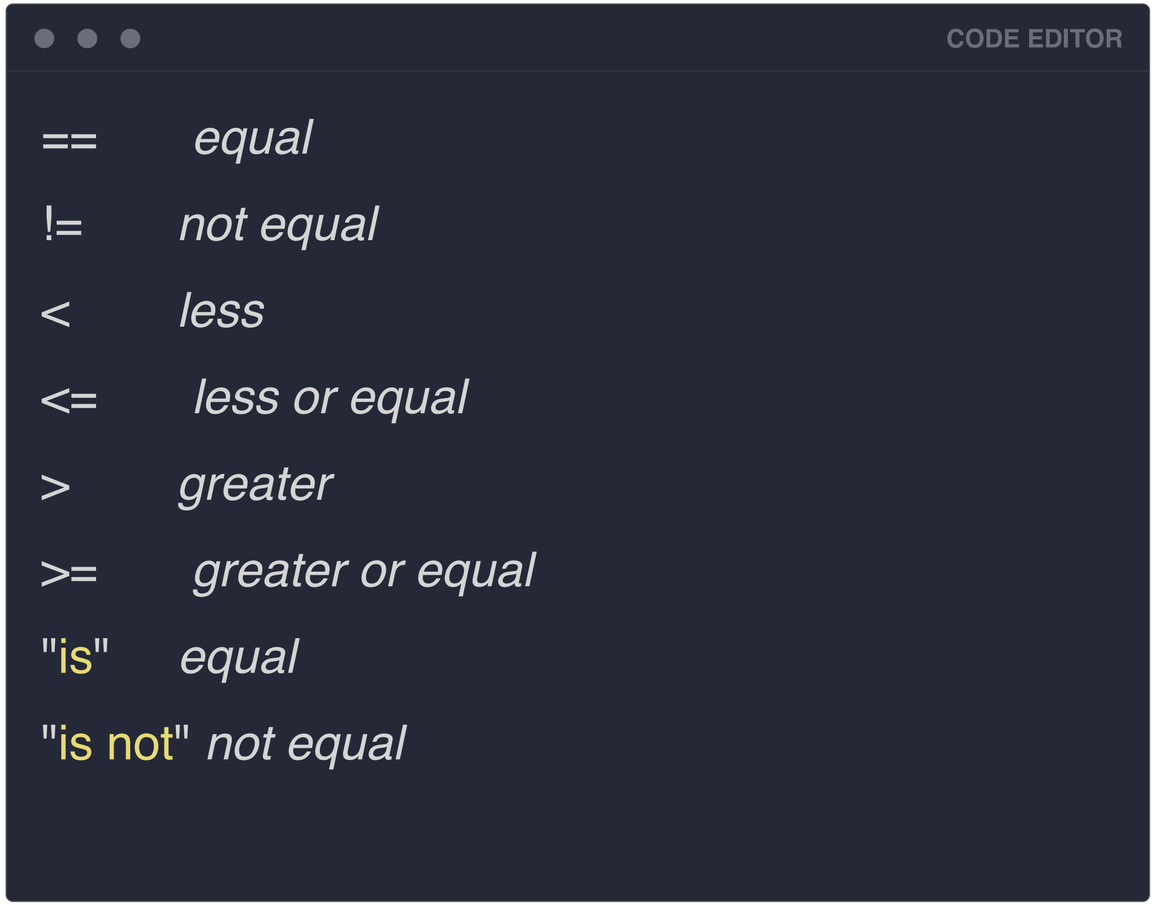
- Rules can also be guarded by conditionals, which is important when writing a policy against a diverse set of infrastructure resources.

- 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.

Lab

Terraform 201

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.

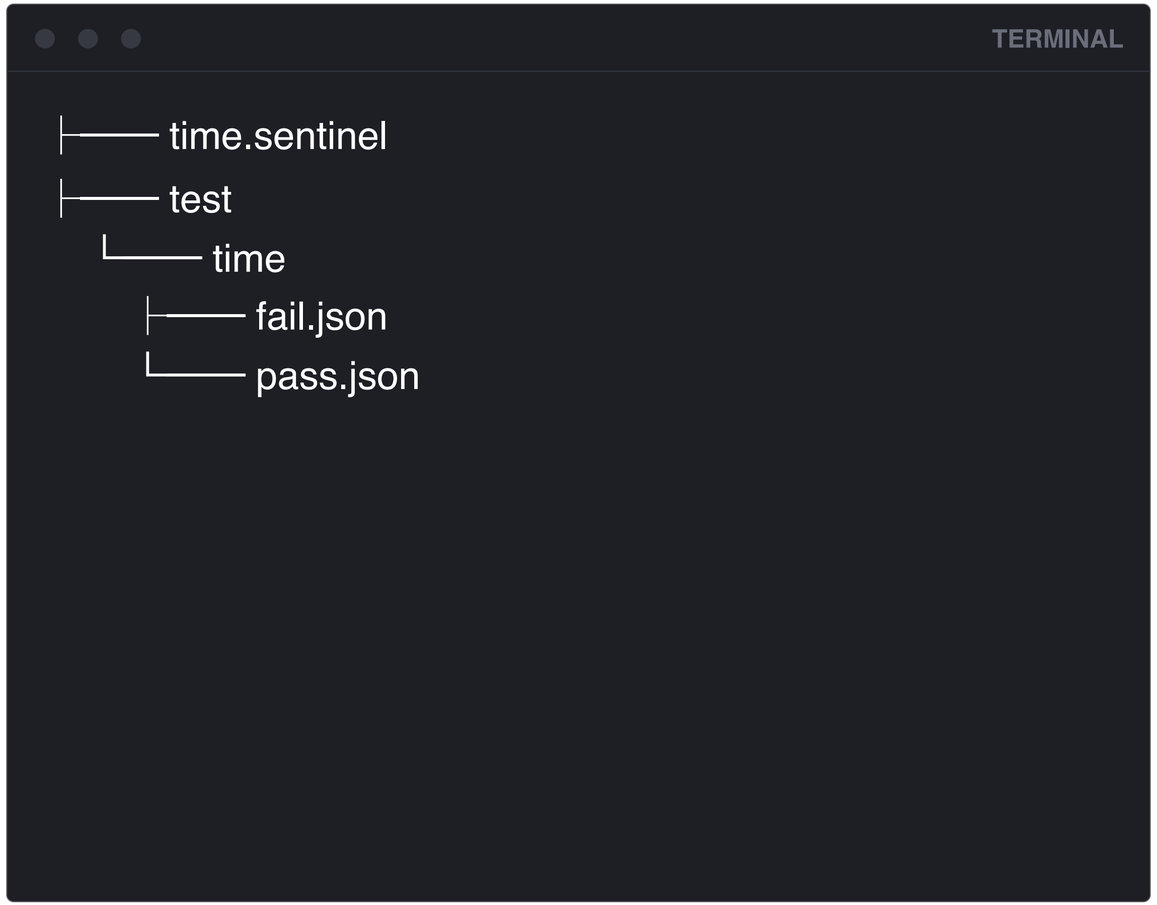
Multiple JSON test files may be present for each policy. Helps test policies against multiple fixtures with varying results.

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).

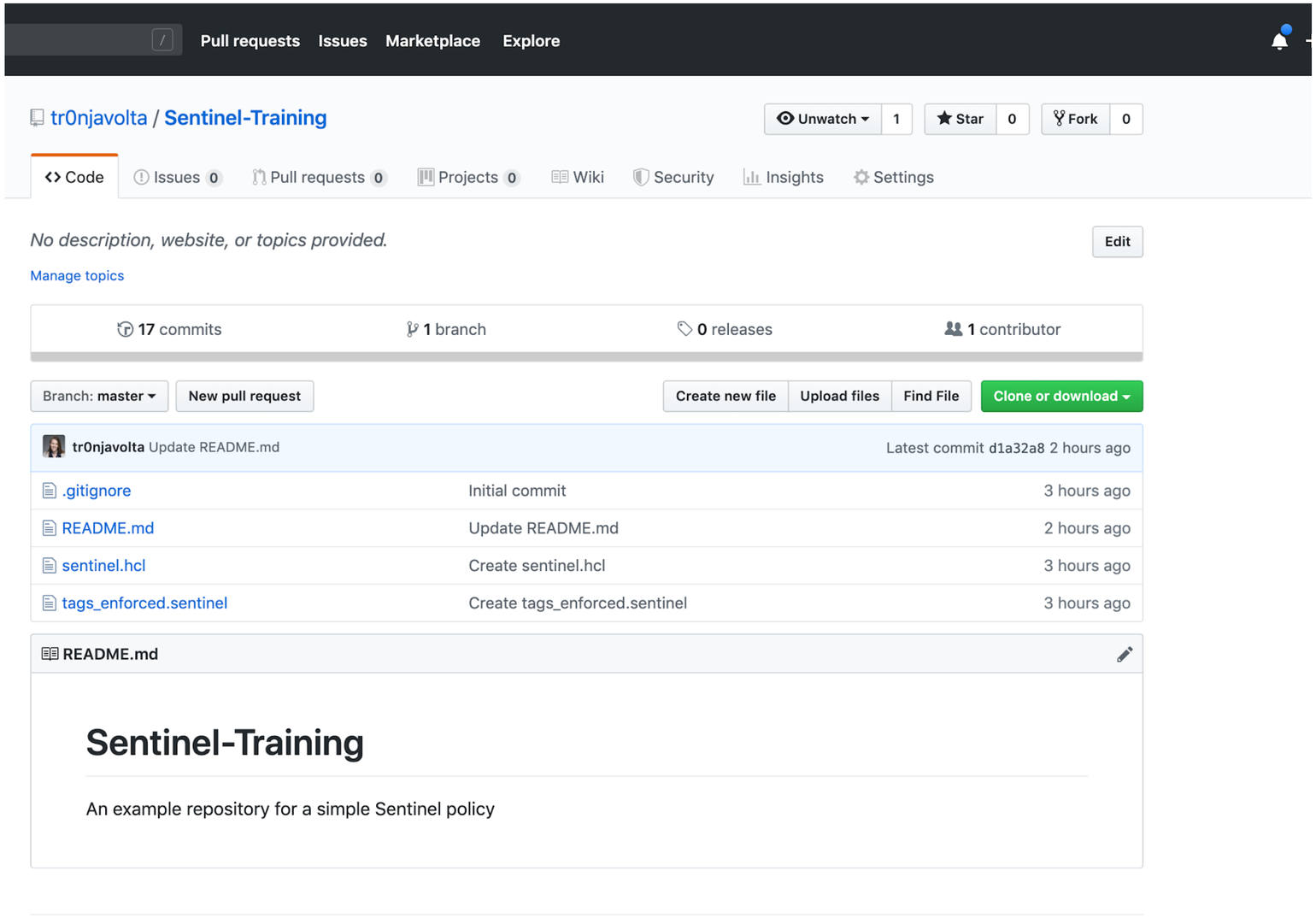
Terraform 201

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

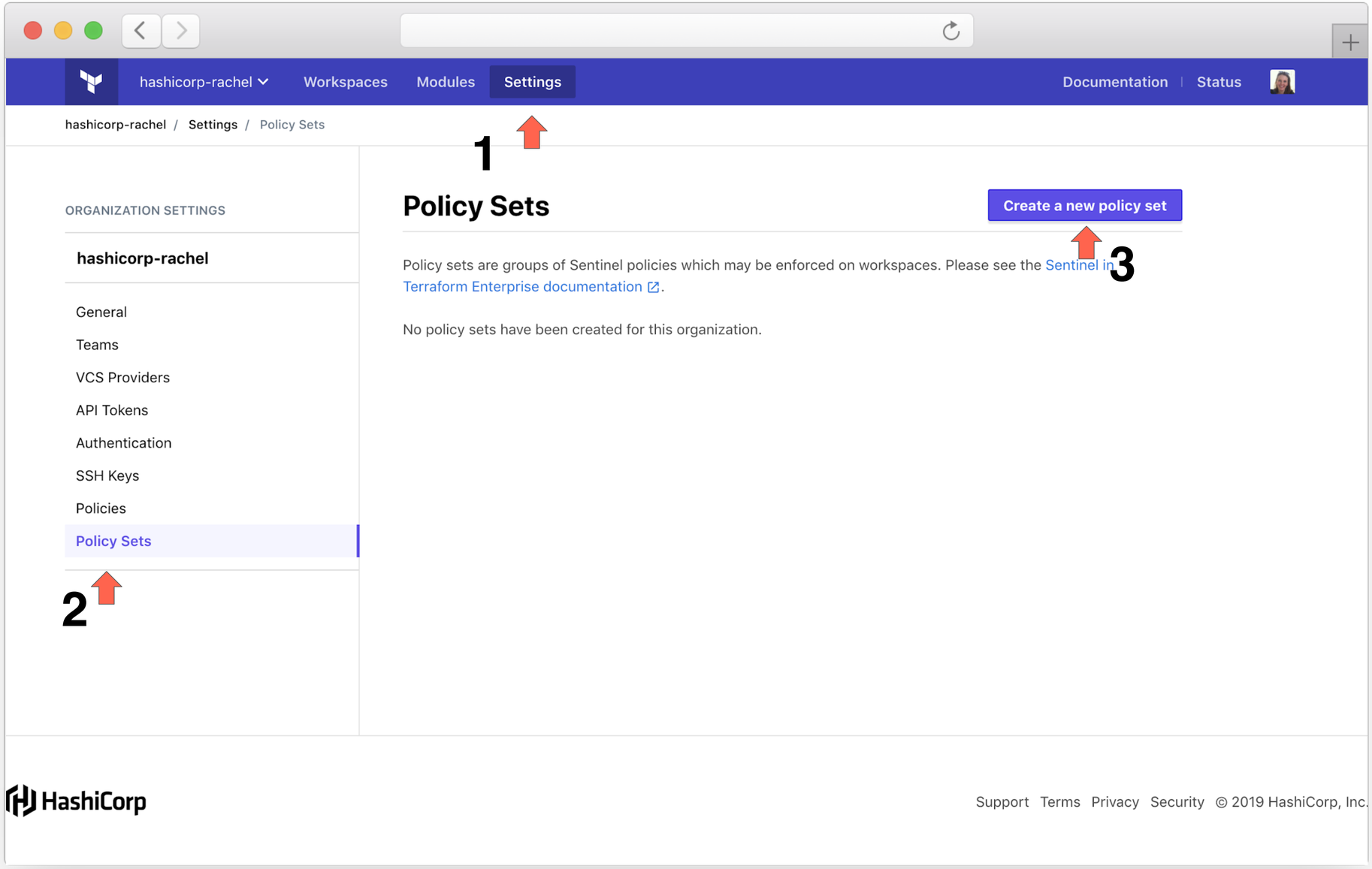
```
policy "tags_enforced" {
  enforcement_level = "hard-mandatory"
}
```

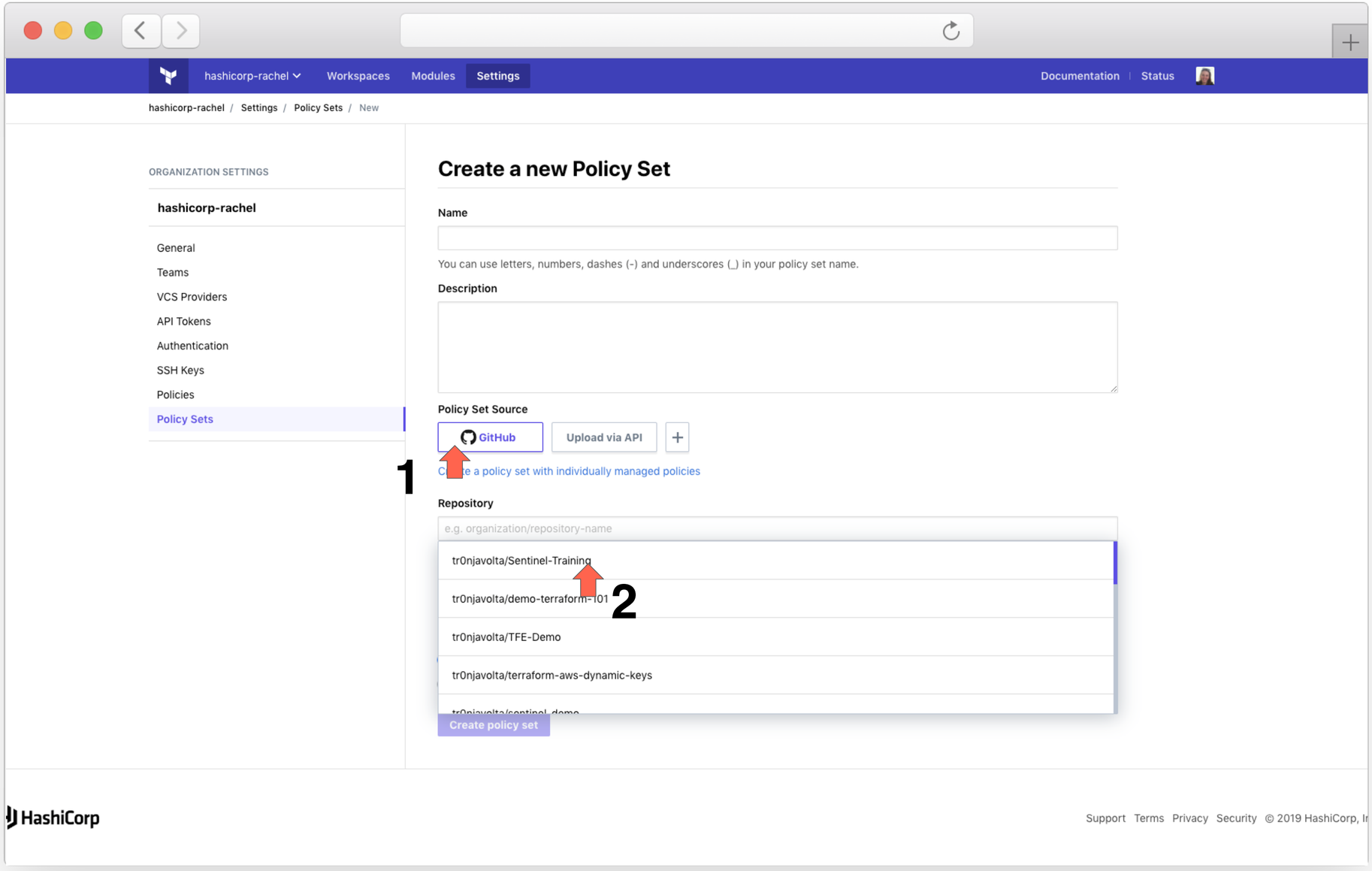
Enforcement Levels

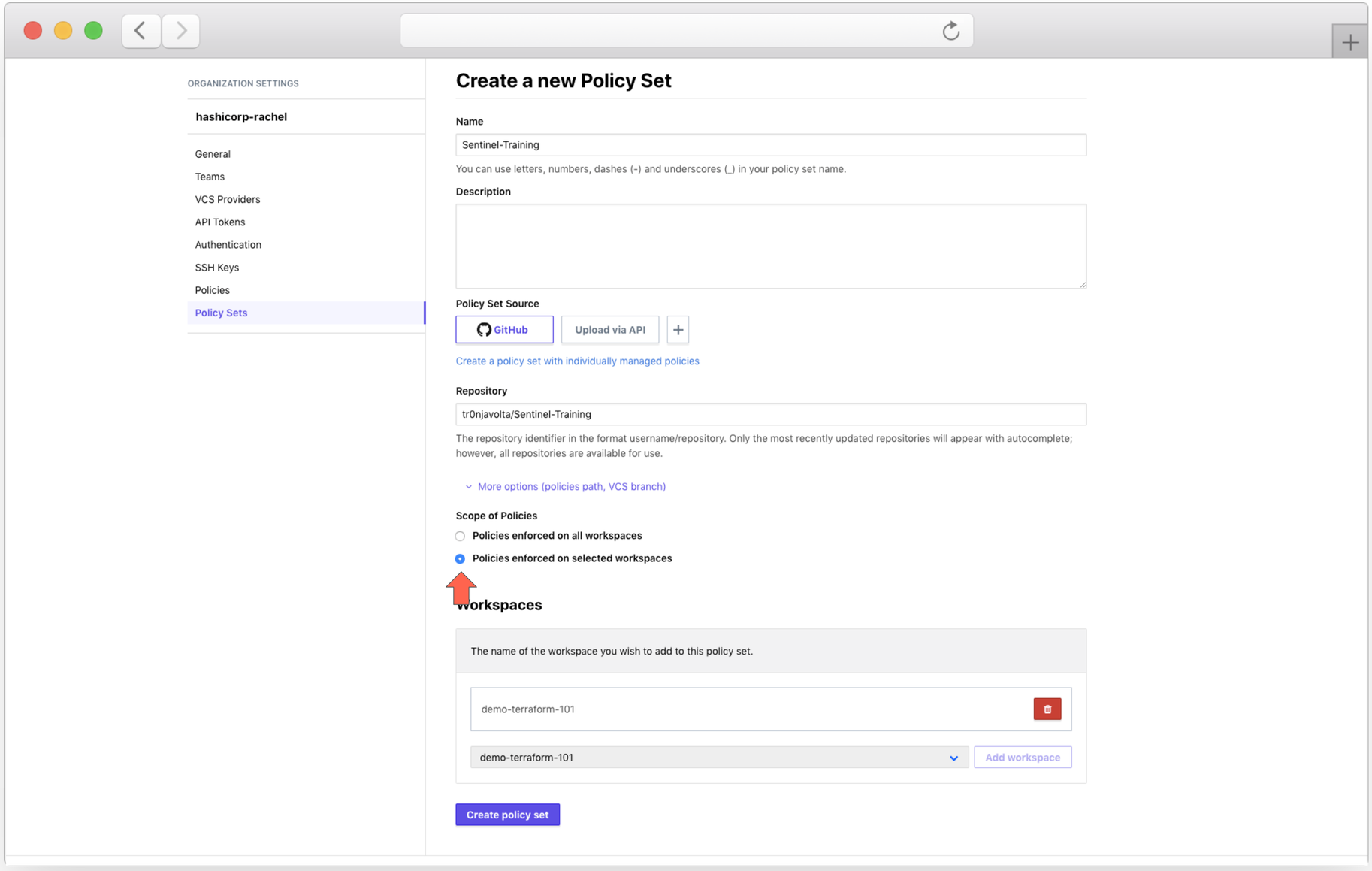
hard-mandatory (permanent)

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
2
tr0njavolta/TFE-Demo
tr0njavolta/terraform-aws-dynamic-keys
Create policy set
HashiCorp
Support
Terms
Privacy
Security
© 2019 HashiCorp, I


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

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

Excercise

Challenge

Use Sentinel Simulator to Test and Apply Your Sentinel Policy to Terraform Cloud

Download & Test a simple policy in the Simulator

Create a more complex policy with more real-world data structures

Create a GitHub repo and upload & run your policy

Overview

Questions?

Exercise

Sentinel Syntax

Challenge

Terraform Cloud & GitHub workflow

Overview

Testing & Mocking in Sentinel

Questions?

Exercise

Challenge

Overview

Questions?

What are some policies that would benefit your organization?

Take a few minutes to discuss with your neighbor.