

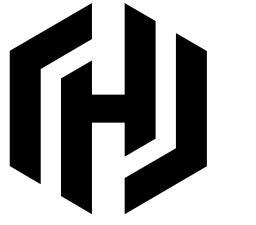


---

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



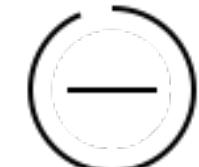
## MODIFY

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.



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

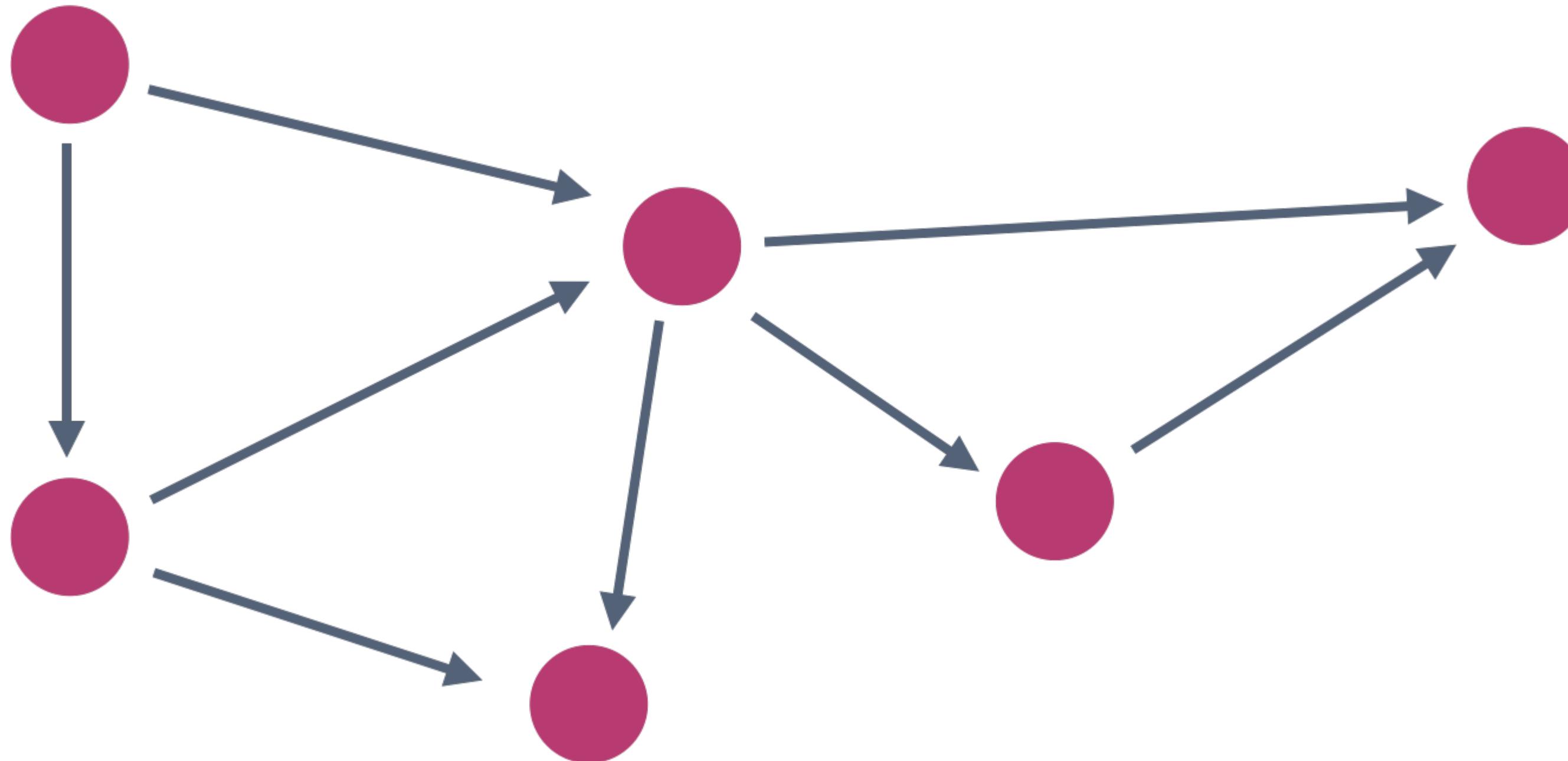
To control dependency errors



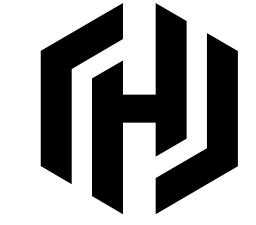
A dark-themed code editor window titled "CODE EDITOR" at the top right. The window shows a snippet of configuration code:

```
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 deposited 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  
}
```



TERMINAL

## 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`.

```
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

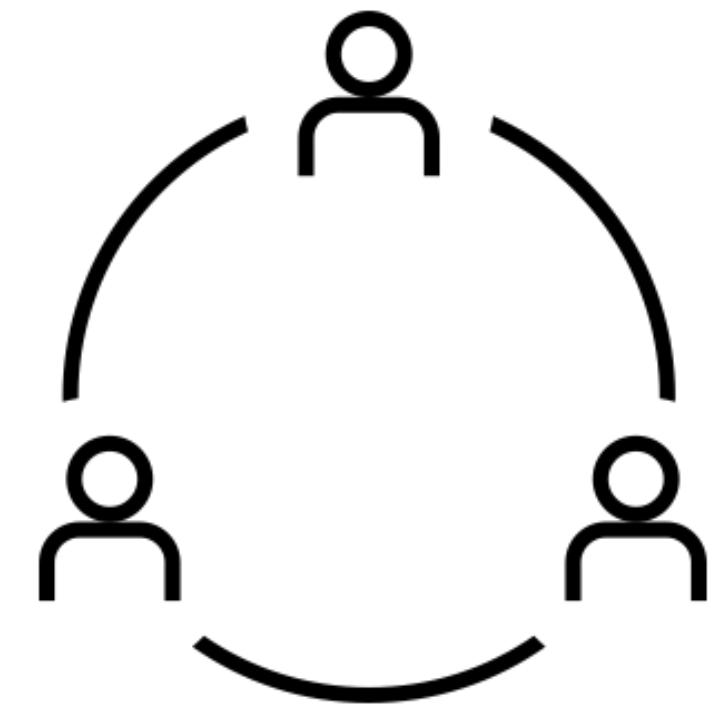
# Code Organization for Teams

# What You'll Learn

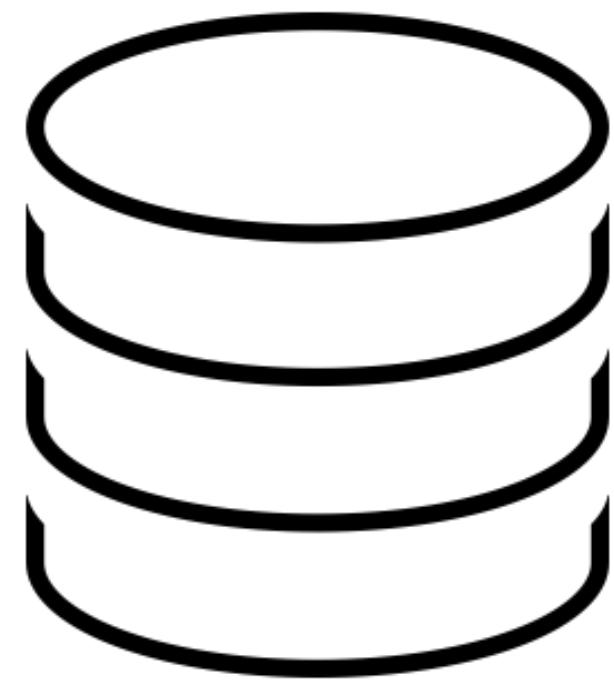


- ❖ How Terraform projects evolve
- ❖ When to use modules vs a new project
- ❖ How to use shared state to manage complexity

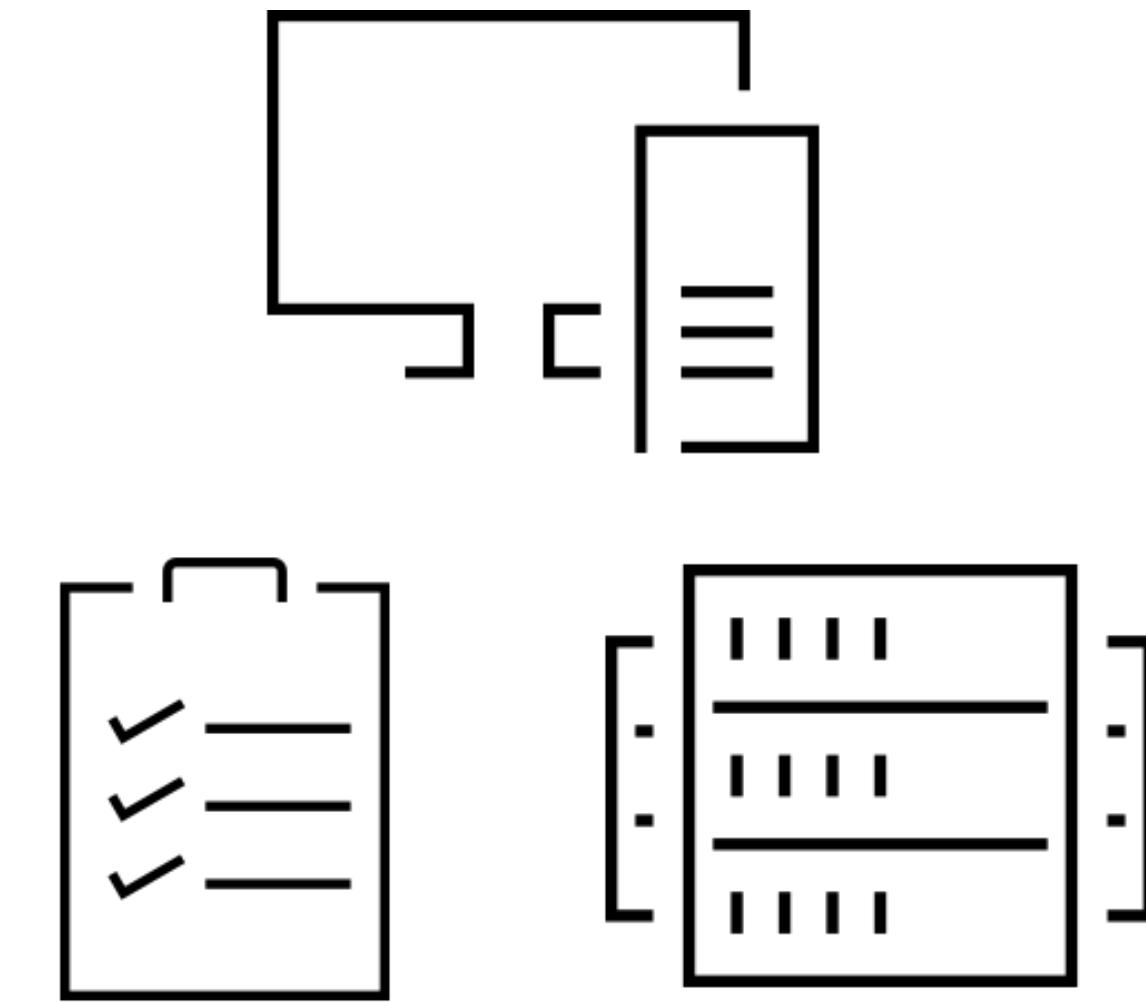
# Ways a Project Grows



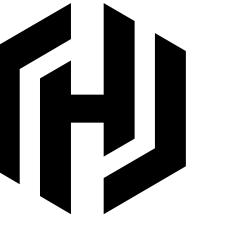
Larger Team



More Resources

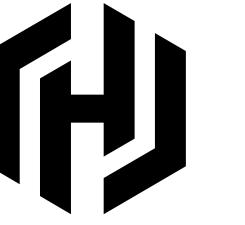


Dev, Test, Staging, Prod



# 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

Dynamic

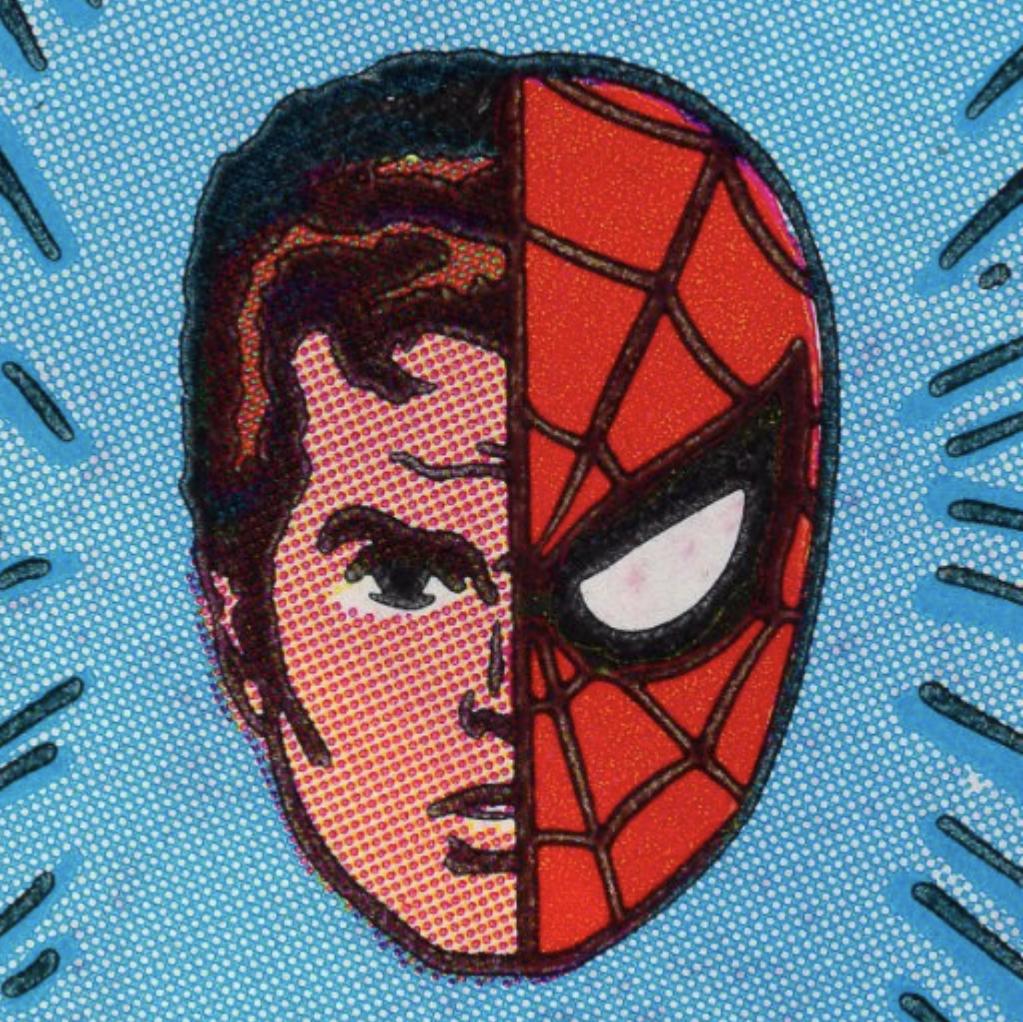
Explicit

Shared

Customizable

Creative

Simple



## Mature Use

Stable

DRY

Isolated

Reusable

Consistent

Complex

# The Growth Continuum



## Early Adoption

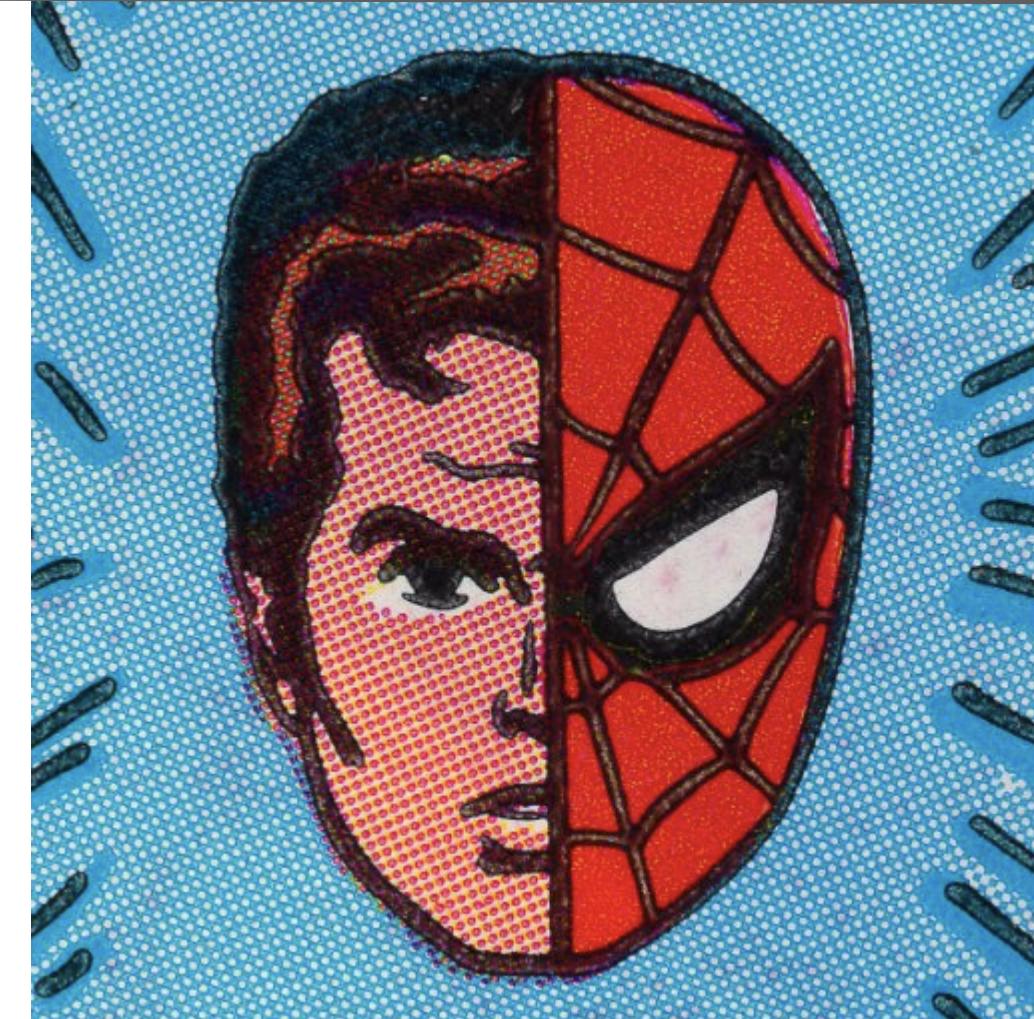
Single Local State

Root Module

Single Directory

Freeform Code

Full Control



## Mature Use

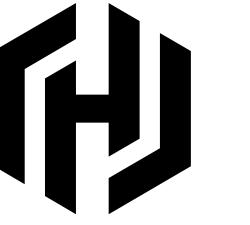
Many Remote State Files

Versioned Modules

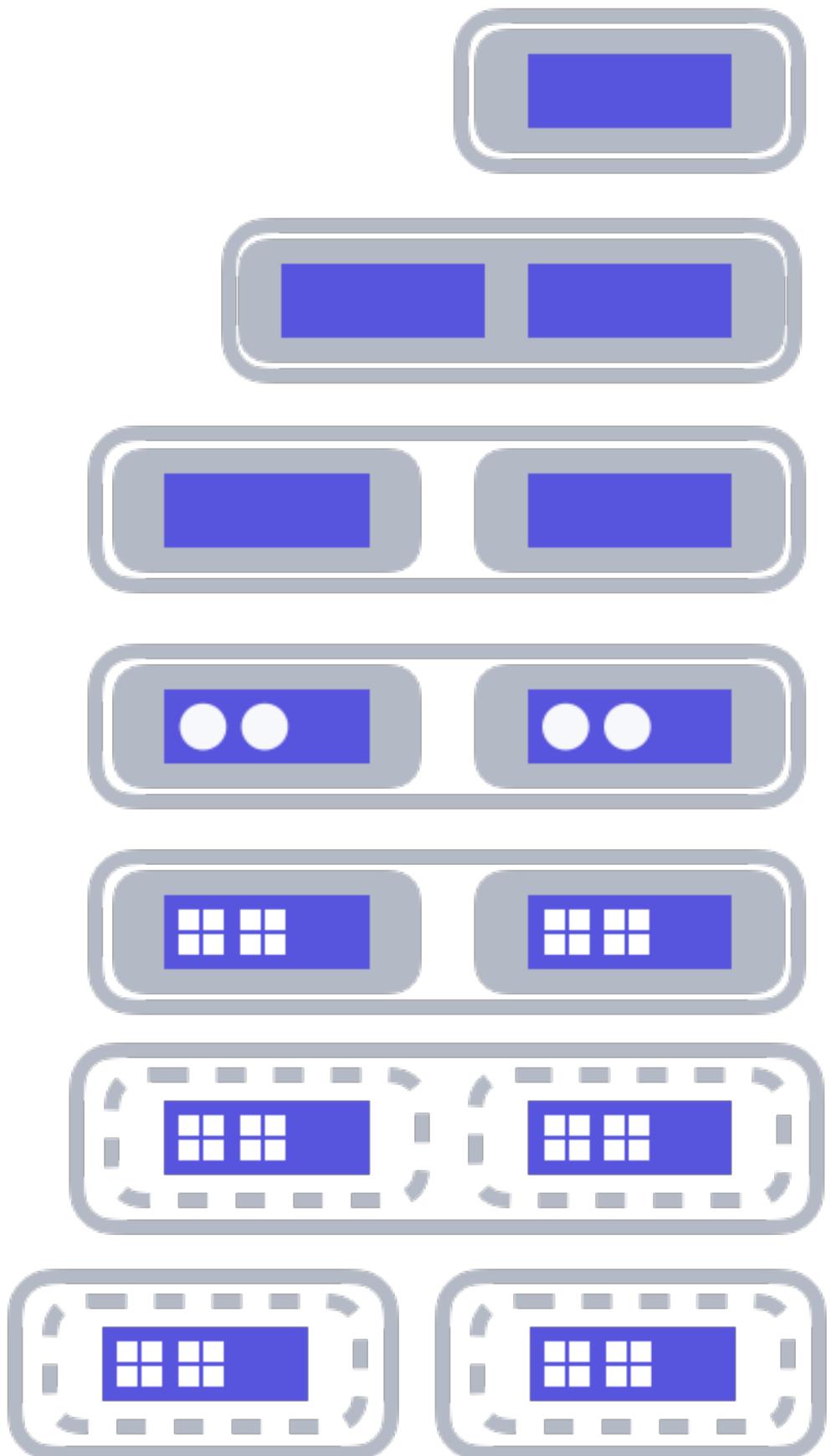
Many Repositories

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

A dark-themed terminal window titled "TERMINAL". Inside, a file tree is displayed:

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

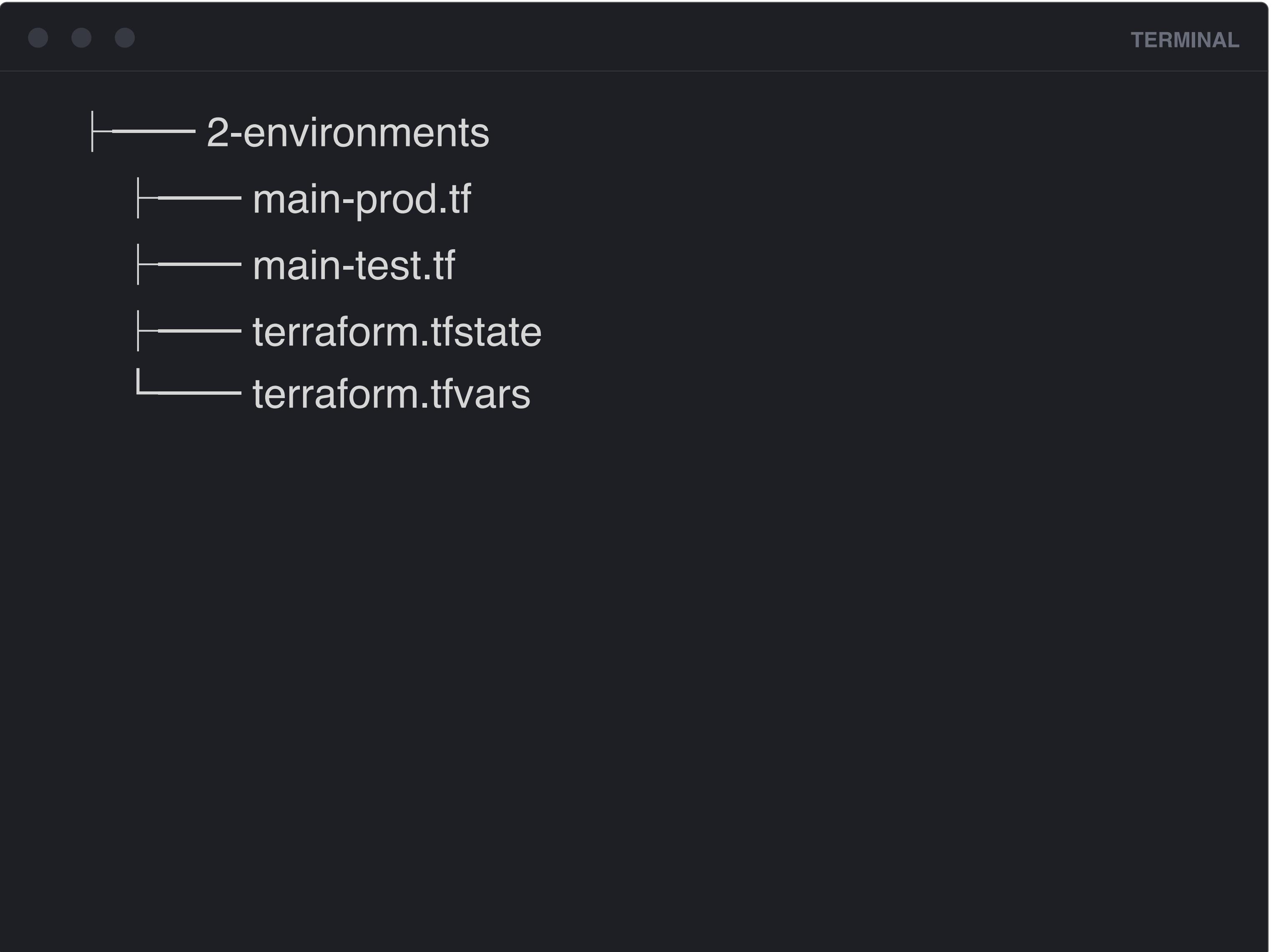


# Multiple Environments

May contain test and prod  
("web\_test", "web\_prod")

Possibly in different files  
within a single repo.

Lowest barrier to entry for  
testing changes.



A terminal window titled "TERMINAL" showing a file tree structure. The root directory is "2-environments", which contains four files: "main-prod.tf", "main-test.tf", "terraform.tfstate", and "terraform.tfvars". The "main-prod.tf" and "main-test.tf" files are shown with a double-line prefix, indicating they are primary configuration files, while "terraform.tfstate" and "terraform.tfvars" are shown with a single-line prefix, indicating they are state and variable files respectively.

```
2-environments
├── main-prod.tf
├── main-test.tf
└── terraform.tfstate
└── terraform.tfvars
```



TERMINAL

# Separate States

Shrinks “blast radius” of changes between infrastructures.

Control timing and release schedule of provisioning

Provides ability to isolate resources and secrets.

```
  ┌── 3-states
  |   ├── prod
  |   |   ├── main.tf
  |   |   ├── terraform.tfstate
  |   |   └── terraform.tfvars
  |   └── test
        ├── main.tf
        ├── terraform.tfstate
        └── terraform.tfvars
```



# Resources in Modules

Improves long term maintenance

Eases reuse

Encourages consistency in design

TERMINAL

```
— 4-modules
  |   +-- envs
  |   |   +-- prod
  |   |   |   +-- main.tf
  |   |   |   +-- terraform.tfstate
  |   |   |   +-- terraform.tfvars
  |   |   +-- test
  |   |   |   +-- main.tf
  |   |   |   +-- terraform.tfstate
  |   |   |   +-- terraform.tfvars
  |   +-- modules
  |       +-- core
  |       |   +-- input.tf
  |       |   +-- main.tf
  |       |   +-- output.tf
  +-- db
  +-- network
  +-- webapp
```



TERMINAL

# Nested Modules

Groups functionality of modules into logical slices.

Allows hierarchical organization of Terraform code.

Provides the ability to effectively namespace modules.

Allows using broad or concise pieces of larger modules.





TERMINAL

# 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.

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

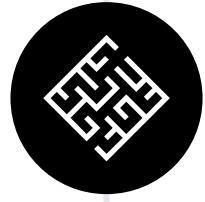
# Discussion Conclusions



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



# Challenge Discussion



## Challenge

**How would you architect a project?**

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



## Overview

How would you divide the project into smaller configurations?



## Questions?

What values would need to be emitted by each configuration?

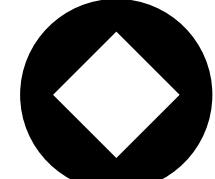
What values would need to be read by each?



# Challenge Discussion



Challenge



Overview



Questions?

**How Terraform projects evolve**

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

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



# Challenge Discussion



Challenge



Overview



Questions?



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



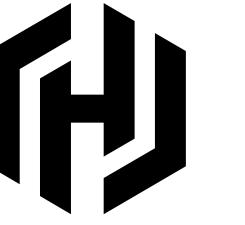
**Take a few minutes to discuss with your neighbor.**



---

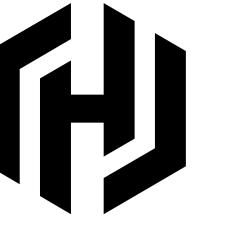
Terraform 201

# 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?

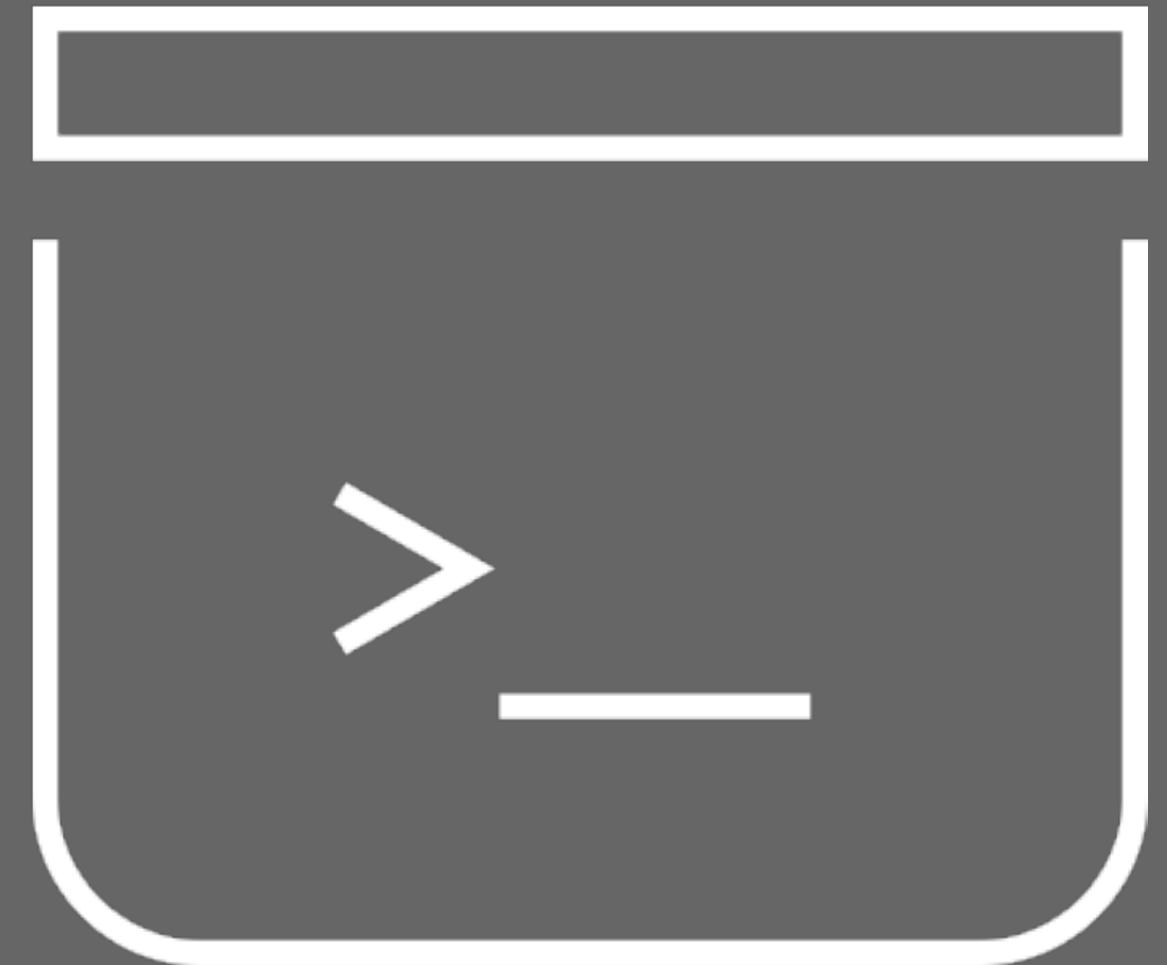
- ◆ Reads a file at a given path and renders its content as a template using a supplied set of variables
- ◆ Invoked using the keyword `templatefile`
- ◆ Uses the same interpolation syntax as strings



Plain Text  
Template



Variables from Cloud or  
Other Resources



Render with  
Terraform

# Creating a Template File



TERMINAL

```
$ mkdir -p tfproject/templates && cd $_  
~/tfproject/templates  
$ touch iampolicy.tpl.json
```

CODE EDITOR

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "RestrictInstanceLifecycle",  
      "Effect": "Allow",  
      "Action": [  
        "ec2:StartInstances",  
        "ec2:StopInstances",  
        "ec2:TerminateInstances"  
      ],  
      "Resource": [  
        "arn:aws:ec2:us-west-1a:1234567890:instance/*"  
      ]  
    }  
  ]  
}
```



# Interpolation in Template Files

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

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





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

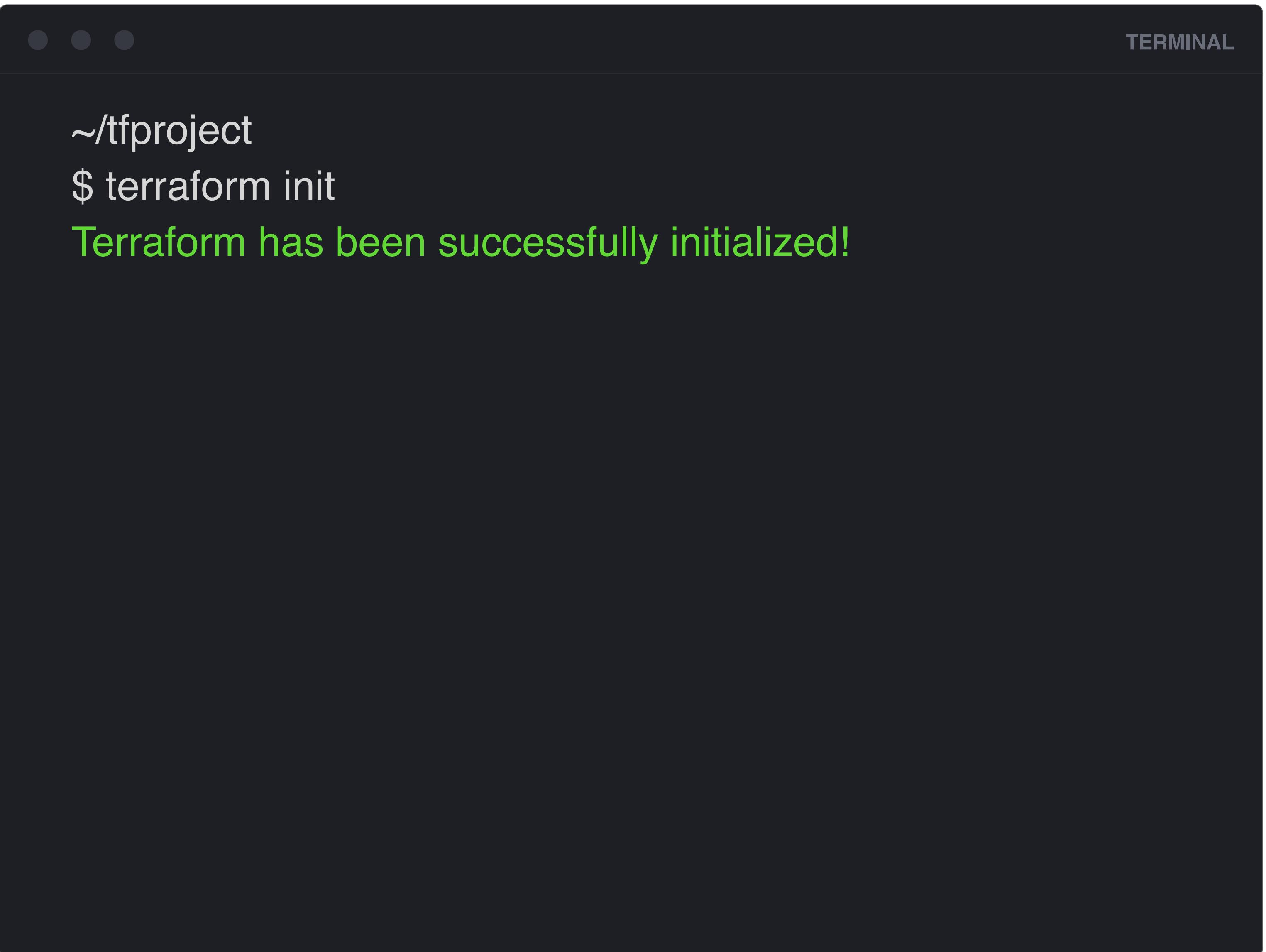
variable "owner_id" {
    default = "1234567890"
}

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



A screenshot of a terminal window titled "TERMINAL". The window shows the command "\$ terraform init" being run in a directory "~/tfproject". The output of the command, "Terraform has been successfully initialized!", is displayed in green text. The terminal has a dark background with light-colored text and icons.



TERMINAL

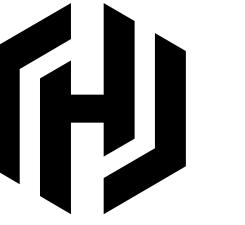
# terraform apply

Rendered file will be reported as an output

```
$ 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

```
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

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

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

}
```



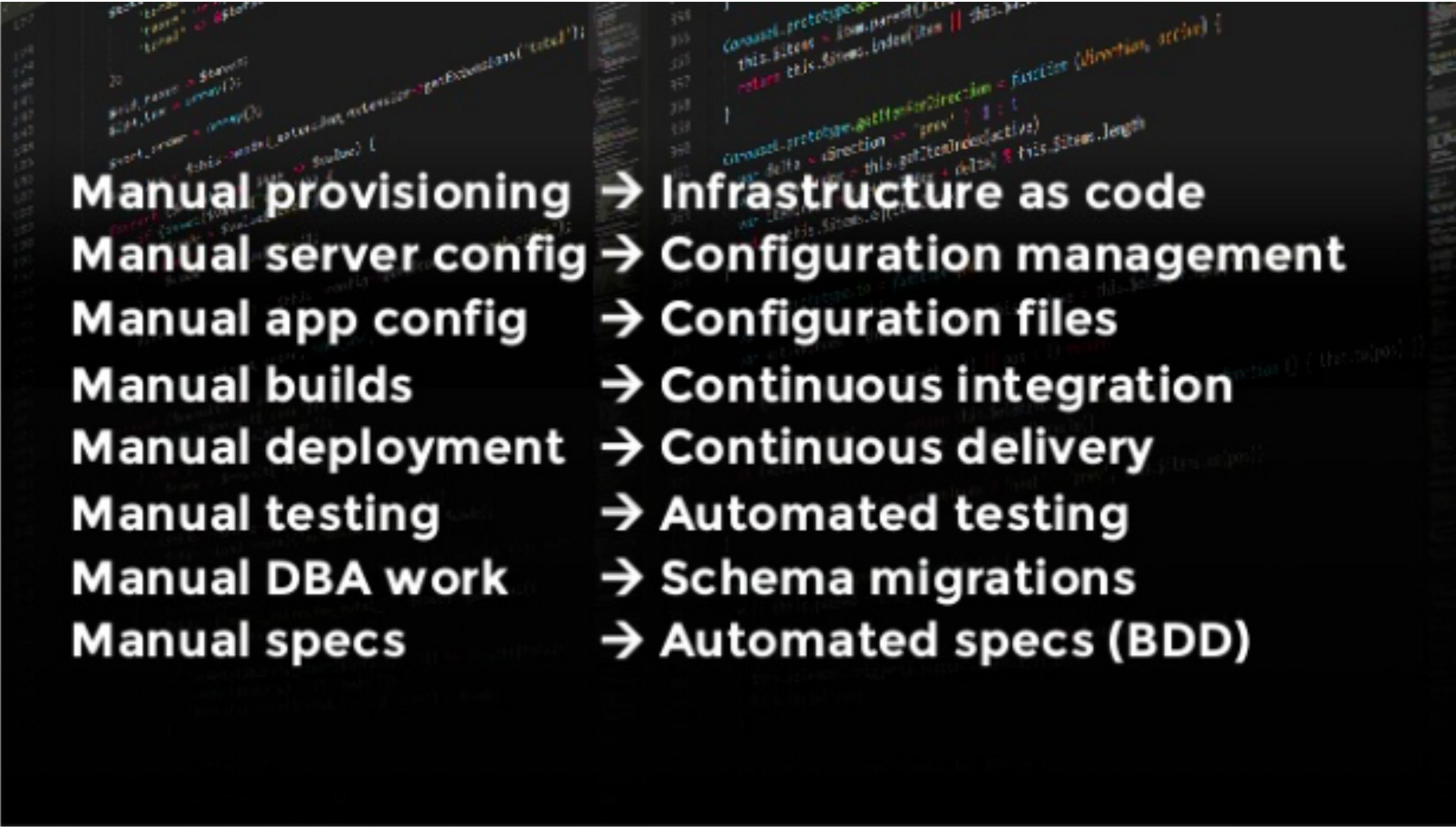
---

Terraform 201

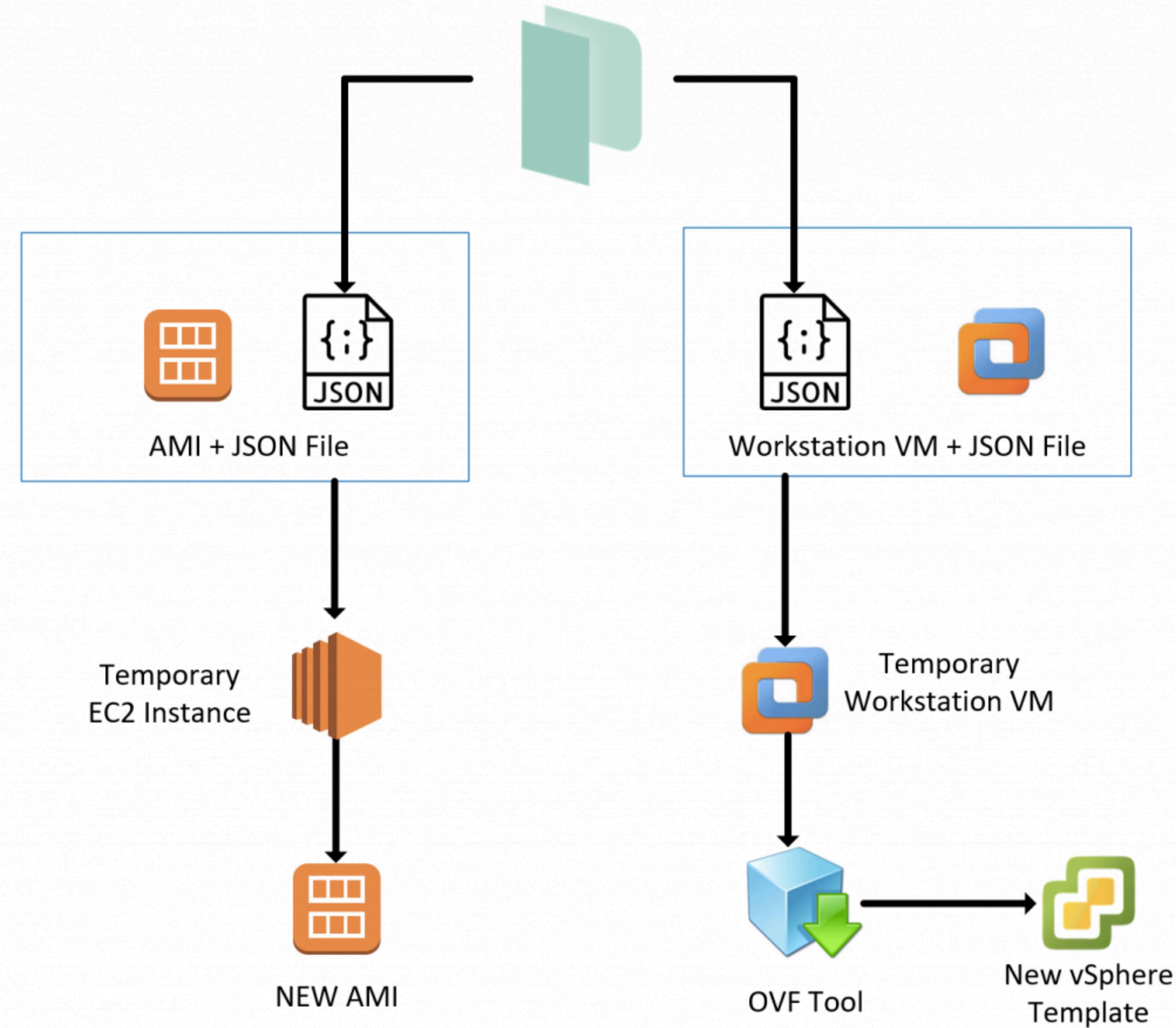
# Creating Images with Packer

# Packer

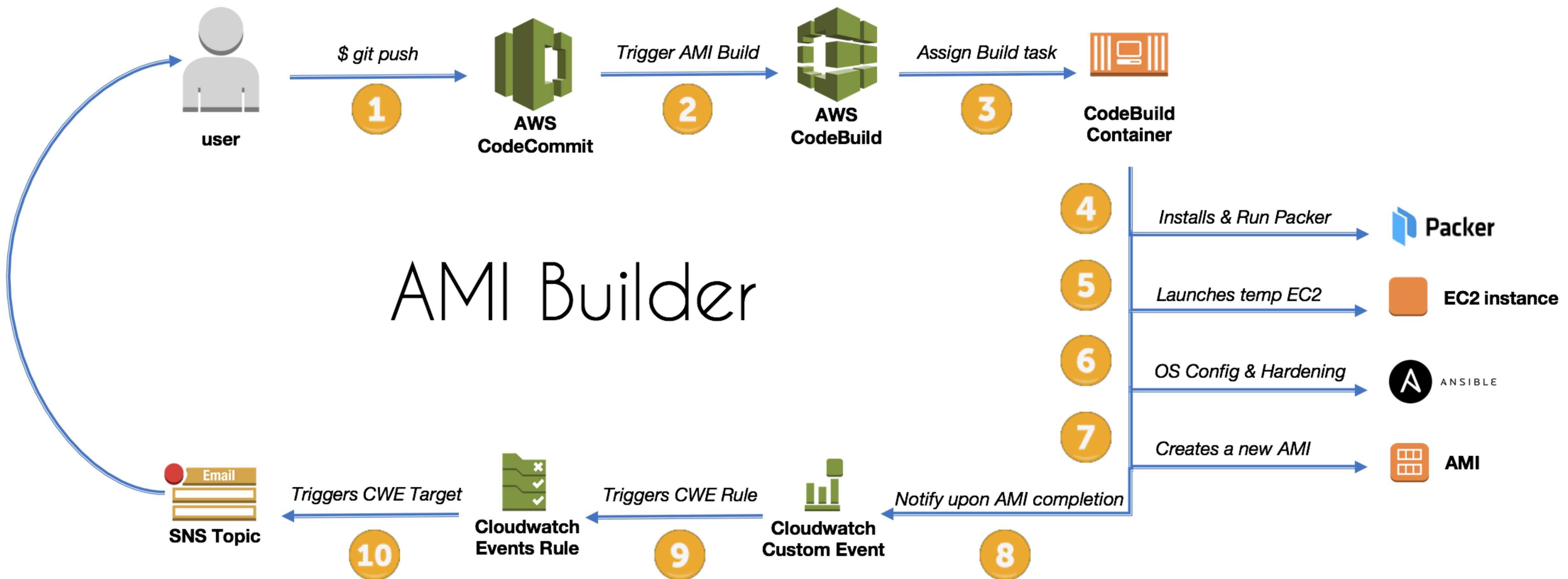
- Automate building machine images and it supports VMware, Amazon EC2, Docker, Google Compute, and others
- Building for a heterogenous cloud environment

- 
- 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



# Automation the Build of AMIs in AWS



# Examples

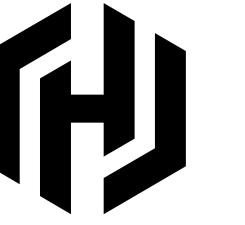
- Tons of Examples of Packer Templates
- <https://github.com/chef/bento>



---

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

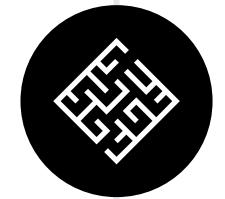
data "github_ip_ranges" "test" {}

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

### 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



## Overview



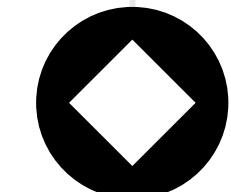
## Questions?



# Lab Exercise 5



Challenge



Overview



Questions?

**How to configure multiple providers**

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



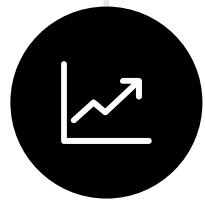
# Lab Exercise 5



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			<b>create</b>
aws_instance.web	aws_instance.web		<b>create</b>
aws_instance.web	aws_instance.web	aws_instance.web	<b>noop</b>
	aws_instance.web	aws_instance.web	<b>delete</b>
		aws_instance.web	<b>noop</b>
aws_instance.web		aws_instance.web	<b>re-create*</b>
	aws_instance.web		<b>update state</b>

# 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

```
1 ami = var.aws_ami
2 instance_type = var.instance_size
3 subnet_id = "subnet-0bb1c79de3EXAMPLE"
4 tags = {
5   Name = "HelloWorld"
6 }
```

C

put arguments in alphabetical order

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

B

place a space in between each line

```
1 type = "A"
2
3 ttl = "300"
4
5 zone_id = aws_route53_zone.primary.zone_id
```

D

align their equals signs

```
1 ami = "abc123"
2 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

```
1 ami = var.aws_ami  
2 instance_type = var.instance_size  
3 subnet_id = "subnet-0bb1c79de3EXAMPLE"  
4 tags = {  
5   Name = "HelloWorld"  
6 }
```

C

put arguments in alphabetical order

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

B

place a space in between each line

```
1 type = "A"  
2  
3 ttl = "300"  
4  
5 zone_id = aws_route53_zone.primary.zone_id
```

D

align their equals signs

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

## Explanation

HashiCorp style conventions suggest you 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:-

```
1 provider "aws" {  
2   region = "us-east-1"  
3 }  
4  
5 provider "aws" {  
6   region = "us-west-1"  
7 }
```

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

```
1 Error: Duplicate provider configuration  
2  
3   on main.tf line 5:  
4     5: provider "aws" {  
5  
6     A default provider configuration for "aws" was already given at  
7     main.tf:1,1-15. If multiple configurations are required, set the "_____"  
8     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:-

```
1 provider "aws" {  
2   region = "us-east-1"  
3 }  
4  
5 provider "aws" {  
6   region = "us-west-1"  
7 }
```

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

```
1 Error: Duplicate provider configuration  
2  
3   on main.tf line 5:  
4     5: provider "aws" {  
5  
6 A default provider configuration for "aws" was already given at  
7 main.tf:1,1-15. If multiple configurations are required, set the "_____"  
8 argument for alternative configurations.
```

### Explanation

An **alias** meta-argument 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 depenaencies 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 depenaencies 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 9: Why might a user opt to include the following snippet in their configuration file?

```
1 |   terraform {  
2 |     required_version = ">= 0.12"  
3 |   }
```

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?

```
1 |   terraform {  
2 |     required_version = ">= 0.12"  
3 |   }
```

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/cf-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:**

```
1  terraform {  
2      required_providers {  
3          aws = "~> 1.2.0"  
4      }  
5  }
```

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:

```
1  terraform {  
2      required_providers {  
3          aws = "~> 1.2.0"  
4      }  
5  }
```

### Explanation

$\sim >$   
 $1.2.0$   
will match any non-beta version of the provider  
between  $\geq$  and  $<$ . For example,  
 $1.2.X$      $1.2.0$      $1.3.0$

A

1.2.9

B

1.3.1

C

1.3.0

D

1.2.3

**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 known-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 known-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.html>

**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.

```
1 resource "aws_eip" "public_ip" {
2   vpc = true
3   instance = aws_instance.web_server.id
4 }
5
6 resource "aws_instance" "web_server" {
7   ami           = "ami-2757f631"
8   instance_type = "t2.micro"
9   depends_on   = [aws_s3_bucket.company_data]
10 }
```

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.

```
1 resource "aws_eip" "public_ip" {
2   vpc = true
3   instance = aws_instance.web_server.id
4 }
5
6 resource "aws_instance" "web_server" {
7   ami           = "ami-2757f631"
8   instance_type = "t2.micro"
9   depends_on   = [aws_s3_bucket.company_data]
10 }
```

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 `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](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 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**

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.

B

**TF\_LOG**

C

**TF\_TRACE**

You can set **TF\_LOG** to one of the log levels **TRACE**, **DEBUG**, **INFO**, **WARN**, or **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.

D

**TF\_DEBUG**

<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?**

```
1  $ terraform init
2
3  Initializing the backend...
4
5  Initializing provider plugins...
6  - Checking for available provider plugins...
7  - Downloading plugin for provider "aws" (hashicorp/aws) 2.44.0...
8  - Downloading plugin for provider "random" (hashicorp/random) 2.2.1...
9
10 :
11
12 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?**

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

```
1 $ terraform init
2
3 Initializing the backend...
4
5 Initializing provider plugins...
6 - Checking for available provider plugins...
7 - Downloading plugin for provider "aws" (hashicorp/aws) 2.44.0...
8 - Downloading plugin for provider "random" (hashicorp/random) 2.2.1...
9
10 :
11
12 Terraform has been successfully initialized!
```

**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 or False, `terraform init` cannot automatically download Community providers?

The screenshot shows the Terraform website's 'Community Providers' section. The page header includes the HashiCorp logo, a 'Learn how Terraform fits into the HashiCorp Suite' link, and navigation links for Intro, Learn, Docs, Community, Enterprise, Download, GitHub, and Sign In. On the left, there is a sidebar titled 'Terraform CLI' with links for Configuration Language, Commands (CLI), Import, State, and Providers. The 'Providers' section is expanded, listing categories like Major Cloud, Cloud, Infrastructure Software, Network, VCS, Monitor & System Management, Database, Misc., and Community, each with a list of provider names. The main content area is titled 'Community Providers' and contains a paragraph explaining that these providers are built by the community and not officially maintained by HashiCorp. It also provides a link to a community providers form.

## Community Providers

The providers listed below have been built by the community of Terraform users and vendors. These providers are not tested nor officially maintained by HashiCorp, and are listed here in order to help users find them easily.

If you have built a provider and would like to add it to this community list, please fill out this [community providers form](#).

Category	Provider
Major Cloud	1Password
	Gandi
	Abiquo
	Active Directory - adlerrobert
	Active Directory - GSLabDev
	Aiven
	AlienVault
	AnsibleVault
	Apigee
	GoCD
Cloud	oVirt
	Pass
	Generic Rest API
	PHPiPAM
	Git
	Github Code Owners
	Pingdom
	Pivotal Tracker
	Proxmox
	Puppet CA
Infrastructure Software	GitHub File
	GitInfo
	Glue
	GoCD
Network	Glue
	GoCD
	Glue
	GoCD
VCS	Glue
	GoCD
	Glue
	GoCD
Monitor & System Management	Glue
	GoCD
	Glue
	GoCD
Database	Glue
	GoCD
	Glue
	GoCD
Misc.	Glue
	GoCD
	Glue
	GoCD
Community	Glue
	GoCD
	Glue
	GoCD

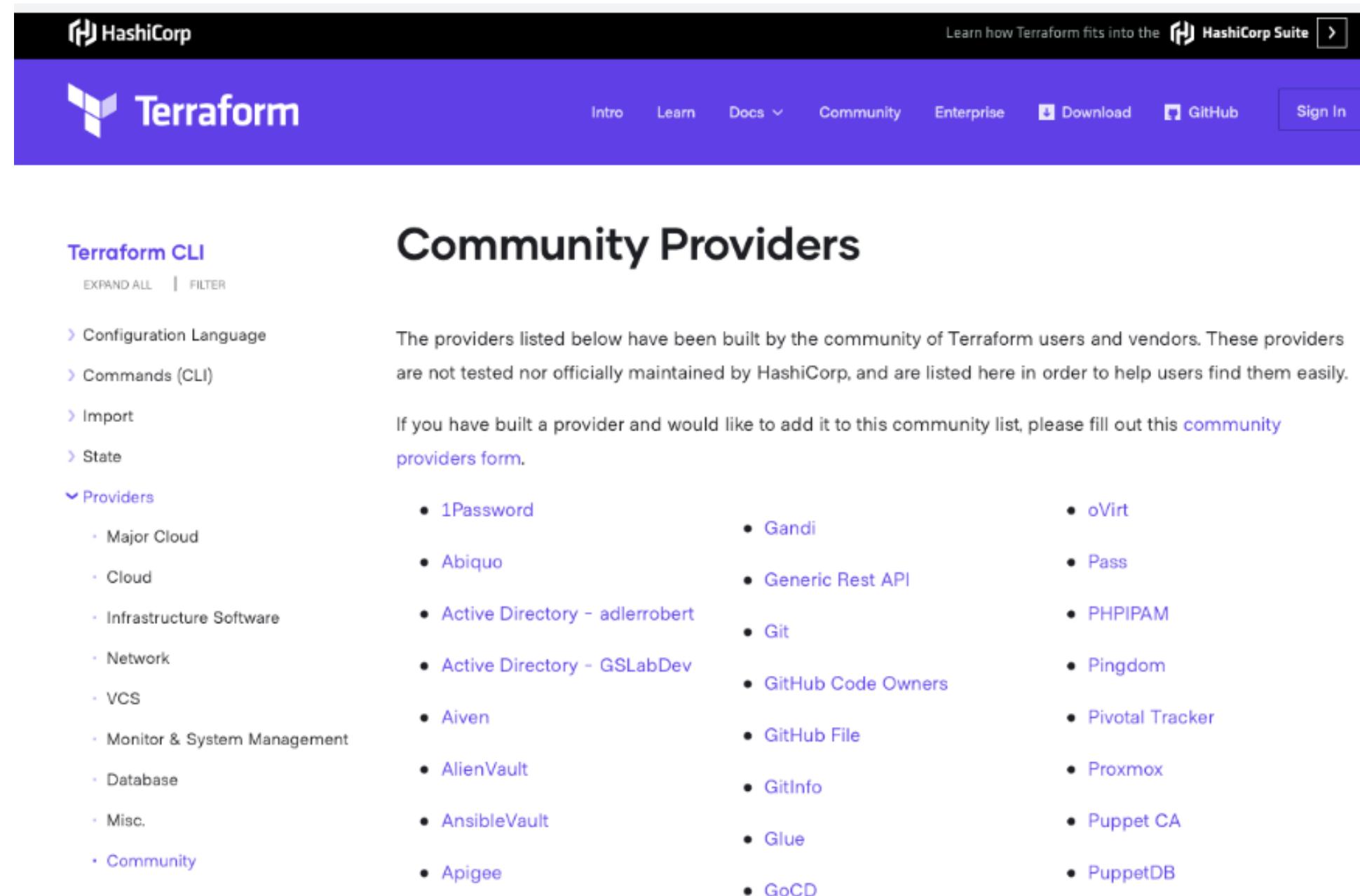
A

False

B

True

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



The screenshot shows the Terraform website's 'Community Providers' page. The left sidebar has a 'Providers' section expanded, listing categories like Major Cloud, Cloud, Infrastructure Software, Network, VCS, Monitor & System Management, Database, Misc., and Community. The main content area is titled 'Community Providers' and contains a list of third-party providers, many of which are links:

- 1Password
- Abiquo
- Active Directory - adllerobert
- Active Directory - GSLabDev
- Aiven
- AlienVault
- AnsibleVault
- Apigee
- Gandi
- Generic Rest API
- Git
- GitHub Code Owners
- GitHub File
- GitInfo
- Glue
- GoCD
- oVirt
- Pass
- PHPIPAM
- Pingdom
- Pivotal Tracker
- Proxmox
- Puppet CA
- PuppetDB

## 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>

A

False

B

True

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

```
1 | resource "aws_instance" "example" {
2 |   ami           = "ami-2757f631"
3 |   instance_type = "t2.micro"
4 |   depends_on   = [aws_s3_bucket.company_data]
5 | }
```

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?**

```
1 | resource "aws_instance" "example" {
2 |   ami           = "ami-2757f631"
3 |   instance_type = "t2.micro"
4 |   depends_on   = [aws_s3_bucket.company_data]
5 | }
```

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

```
1  variable "subnet_details" {
2    type = list(object({
3      cidr          = string
4      subnet_name   = string
5      route_table_name = string
6      aznum         = number
7    }))
8  }
9
10 locals {
11   route_tables_all = distinct([for s in var.subnet_details : s.route_table_name ])
12 }
```

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

```
1  variable "subnet_details" {  
2      type = list(object({  
3          cidr          = string  
4          subnet_name    = string  
5          route_table_name = string  
6          aznum         = number  
7      }))  
8  }  
9  
10 locals {  
11     route_tables_all = distinct([for s in var.subnet_details : s.route_table_name ])  
12 }
```

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:-

```
1 resource "aws_instance" "db" {  
2     ami           = "ami-123456"  
3     instance_type = "t2.micro"  
4 }
```

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:-

```
1 resource "aws_instance" "db" {  
2   ami           = "ami-123456"  
3   instance_type = "t2.micro"  
4 }
```

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>

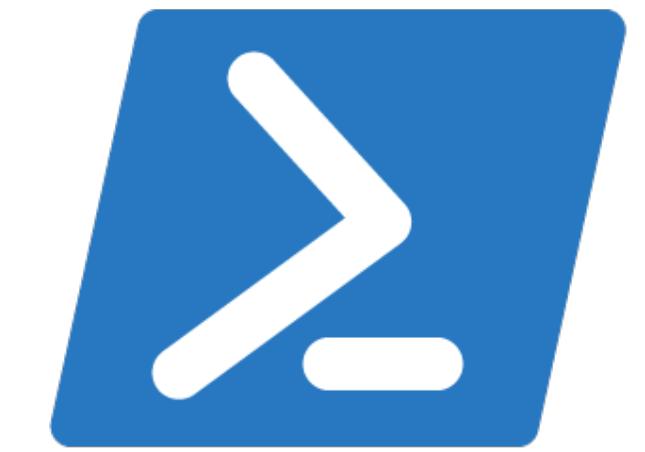
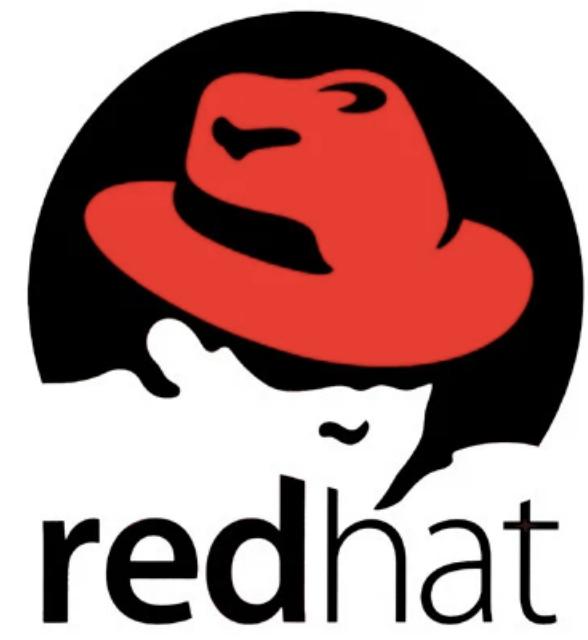
# RPT Architecture and Installation Services

Experts in the Architecture, Strategy, Installation, and Configuration of HashiCorp Software



# Who Is RPT?

- **Tiger Team of Automation Experts** enabling organizations to effectively adopt, run and manage a cloud operating model.
- **Deep Datacenter Experience** (VMware vExperts, Windows, Linux)
- **Cloud Engineers** (AWS, Azure, Expedient)
- **Scripting and Development** (Powershell, Bash, Python)
- **Automation Management** (Ansible, Packer, Terraform)





# Thank You

[www.riverpointtechnology.com](http://www.riverpointtechnology.com)

# Your Current Configuration and Provisioning Practices

## \*Q1. How do you currently manage your infrastructure?

1. Through a UI or CLI. This might seem like the easiest option for one-off tasks, but for recurring operations it is a big consumer of valuable engineering time. It's also difficult to track and manage changes.
2. Through reusable command line scripts, or a combination of UI and infrastructure as code. This is faster and more reliable than pure ad-hoc management and makes recurring operations repeatable, but the lack of consistency and versioning makes it difficult to manage over time.
3. Through an infrastructure as code tool (Terraform, CloudFormation). Infrastructure as code enables scalable, repeatable, and versioned infrastructure. It dramatically increases the productivity of each operator and can enforce consistency across environments when used appropriately.
4. Through a general-purpose automation framework (i.e. Jenkins + scripts / Jenkins + Terraform). This centralizes the management workflow, albeit with a tool that isn't built specifically for provisioning tasks.

# Your Current Configuration and Provisioning Practices

\*Q2. What topology is in place for your service provider accounts?

1. Flat structure, single account. All infrastructure is provisioned within the same account.
2. Flat structure, multiple accounts. Infrastructure is provisioned using different infrastructure providers, with an account per environment.
3. Tree hierarchy. This features a master billing account, an audit/security/logging account, and project/environment-specific infrastructure accounts.

# Your Current Configuration and Provisioning Practices

\*Q3. How do you manage the infrastructure for different environments?

1. Manual. Everything is manual, with no configuration management in place.
2. Siloed. Each application team has its own way of managing infrastructure — some manually, some using infrastructure as code or custom scripts.
3. Infrastructure as code with different code bases per environment. Having different code bases for infrastructure as code configurations can lead to untracked changes from one environment to the other if there is no promotion within environments.
4. Infrastructure as code with a single code base and differing environment variables. All resources, regardless of environment, are provisioned with the same code, ensuring that changes promote through your deployment tiers in a predictable way.

# Your Current Configuration and Provisioning Practices

\*Q4. How do teams collaborate and share infrastructure configuration and code?

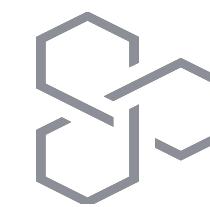
1. N/A - Infrastructure as code is not used.
2. Locally. Infrastructure configuration is hosted locally and shared via email, documents or spreadsheets.
3. Ticketing system. Code is shared through journal entries in change requests or problem/incident tickets.
4. Centralized without version control. Code is stored on a shared filesystem and secured through security groups. Changes are not versioned. Rollbacks are only possible through restores from backups or snapshots.
5. Configuration stored and collaborated in a version control system (VCS) (Git repositories, etc.). Teams collaborate on infrastructure configurations within a VCS workflow and can review infrastructure changes before they enter production. This is the most mature approach, as it offers the best record-keeping and cross-department/cross-team visibility.

# Your Current Configuration and Provisioning Practices

\*Q5. Do you use reusable modules for writing infrastructure as code?

1. Everything is manual. No infrastructure as code currently used.
2. No modularity. Infrastructure as code is used, but primarily as one-off configurations. Users usually don't share or re-use code.
3. Teams use modules internally but do not share them across teams.
4. Modules are shared organization-wide. Similar to shared software libraries, a module for a common infrastructure pattern can be updated once and the entire organization benefits.

# What Are People Looking For



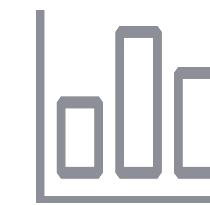
**Orchestration / Automation and  
Infrastructure as Code**



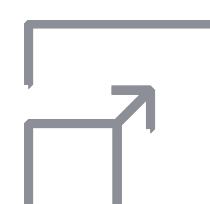
**Diverse Tech Stack**



**DevOps**



**Agile Release Cadences**



**Microservices and Cloud and  
Containers**

# Orchestration / Automation

## Differentiators

- Tags
- Time to Value
- Event Overlay



## Discover / Qualification Questions:

- Are you using anything like Puppet, Chef, Ansible today?
- If you're using containers, are you using Kubernetes, ECS, Swarm, etc?
- Are you doing automated deployments through something like Jenkins today?
- If you're not doing any automation, are you looking at doing this in the future? Which tools?

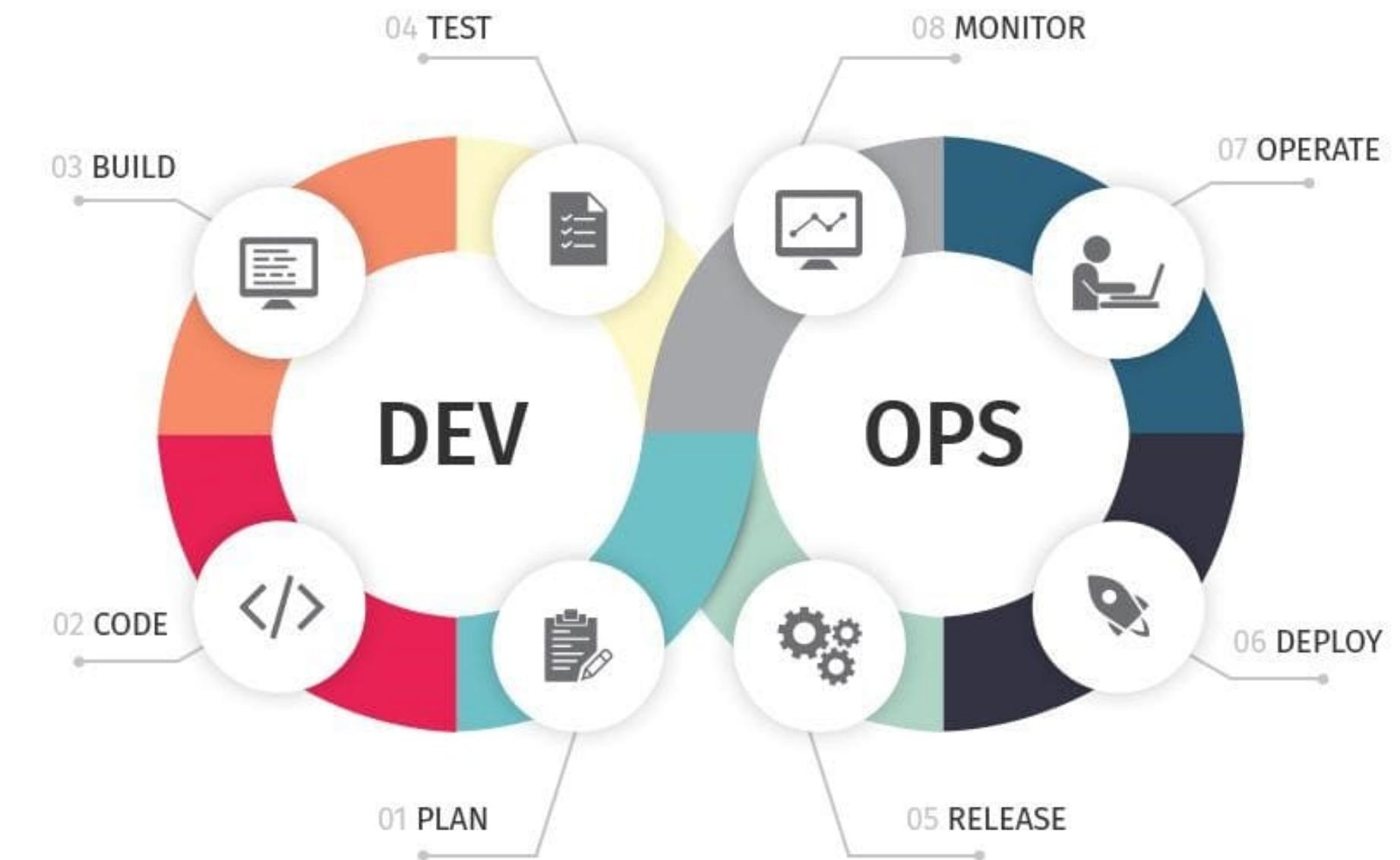
# DevOps

## Differentiators

- Collaboration
- Smart and Predictive Alerting
- History and Resolution of Data
- Event Overlay and Correlation

## Questions

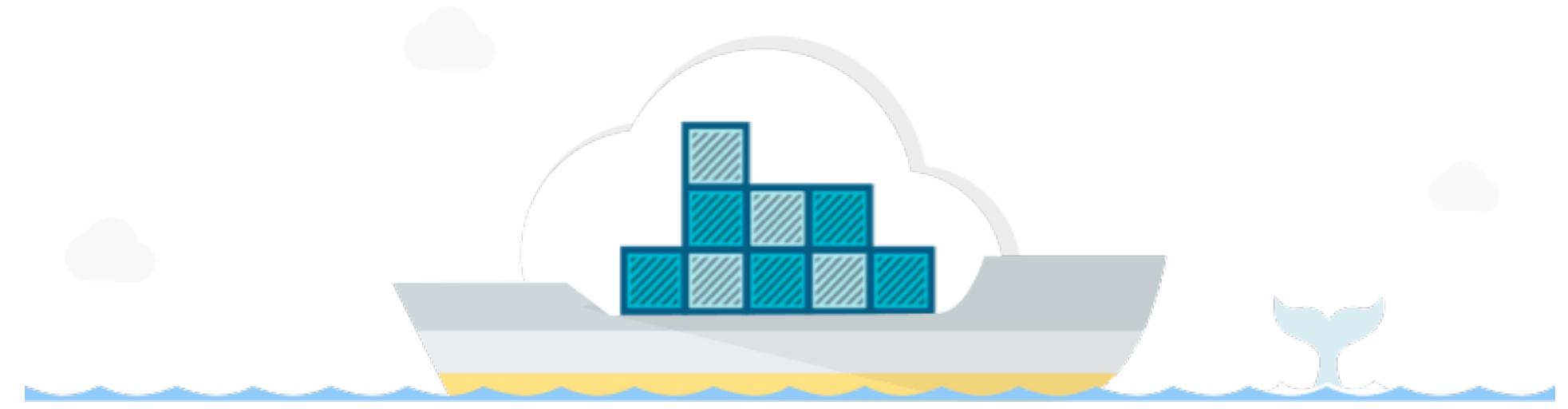
- Has your team / company adopted a DevOps model?
- If not, what's the process between Developers / Engineers and deployment.
- If so, can you talk a bit about your Continuous Integration / Continuous Deployment (CI/CD) process? Are there any manual steps or are you completely automated?



# Cloud and Containers

## Differentiators

- Built for Cloud Scale
- Breadth and Resolution of Data
- Event Overlay and Correlation
- Cost Optimization and Right Sizing



## Questions

- Are you in the public cloud today? AWS, GCP, Azure? Multiple? Other?
- If not in the public cloud, are you using private cloud (OpenStack, CloudFoundry, etc)?
- Are you running Docker, Kubernetes, ECS, or other container technology today?
- If so, what kind of visibility do you have into your containers today?

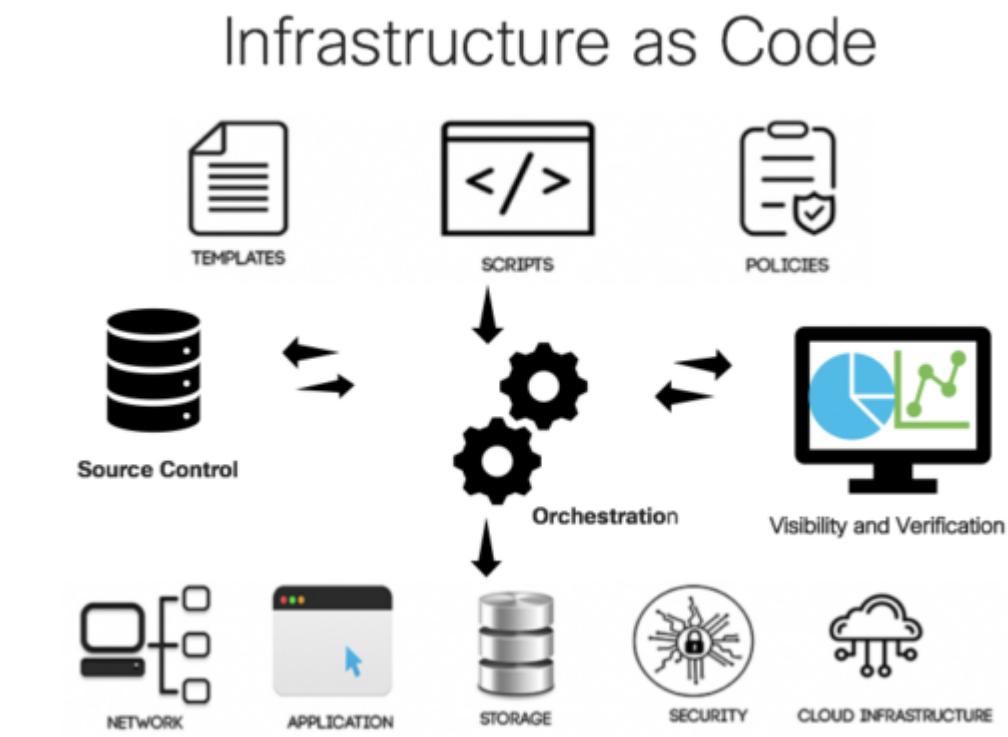
# Infrastructure as Code

## Differentiators

- Tags and Descriptive Meta-Data
- Monitoring as Code
- Time to Value

## Questions

- Are you using anything to define your infrastructure today? Terraform? Ansible? etc..
- If so, do you version control this?
- If so, what environments are you defining with these tools?
- How long does it take you today to deploy your monitoring solution?





# Thank You

[www.riverpointtechnology.com](http://www.riverpointtechnology.com)