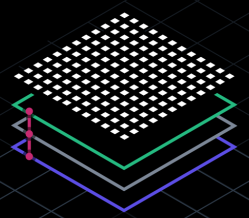
HashiCorp

CLOUD INFRASTRUCTURE AUTOMATION



HashiCorp

Terraform

Introduction

Gabe Maentz

- Married with 4 kids (Marion, Victoria, Ellie, Rhys & Ian and new puppy - Zoey)
- Live in the Pittsburgh, PA
- Grew up in Massachusetts, Georgia, Florida, Minnesota and Wisconsin
- Traveling and exploring new cities, soccer, cooking.
- Been working in IT field for almost 20 years: enterprises and startups. (Epic, Kimberly-Clark, Dick's Sporting Goods, DataGravity)
- Run a local community called vBrisket: Fusion between Technology, BBQ and Craft Beer.
- River Point Technology - Specialized HashiCorp Partner
- 7x VMware vExpert, HashiCorp Terraform & Vault Certified

Tell us about you…

- Name/Role
- Company
- Terraform/Infrastructure as Code experience
- Which clouds are you working with?
- Expectations for class

Course Agenda

Terraform Fundamentals

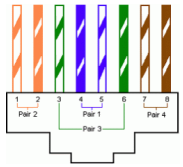
Students will walk away with a solid understanding of HashiCorp Terraform.

Course Agenda:

- Evolution of Infrastructure - VCS & Remote Driven
- Terraform Concepts – Deploy, Init, Apply
- Terraform Configuration
- Outputs
- Console
- Variables
- Auto-Formatting
- Modules
- Provisioners
- Meta Arguments
- Destroy

Progression of Infrastructure Management

Maturity



Manual Everything

```
#!/bin/bash
```

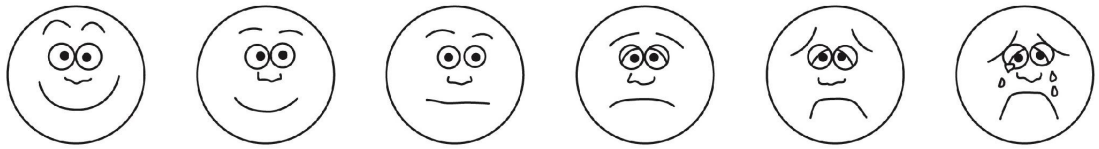
Basic Automation

Machine Virtualization

*aaS

Datacenter-as-Computer

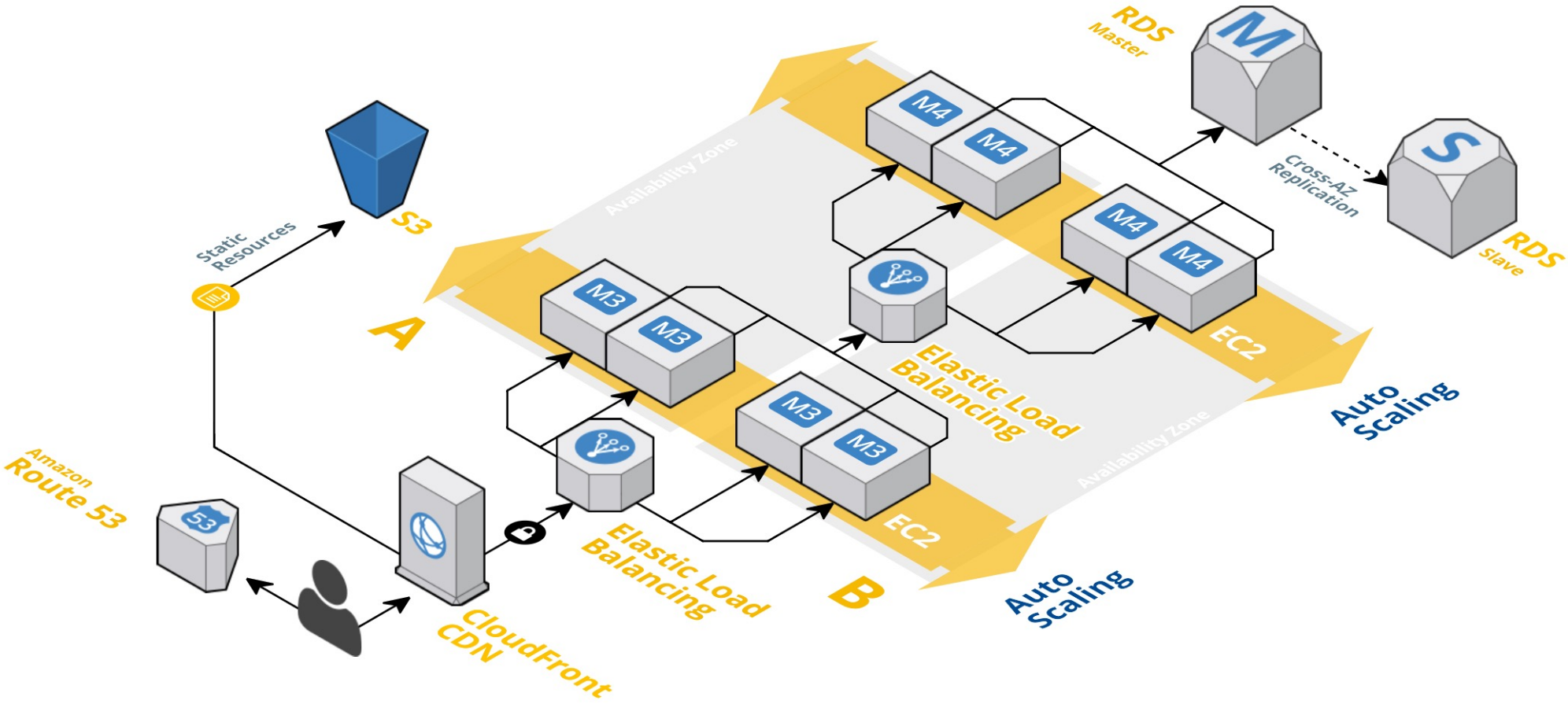
Maturity

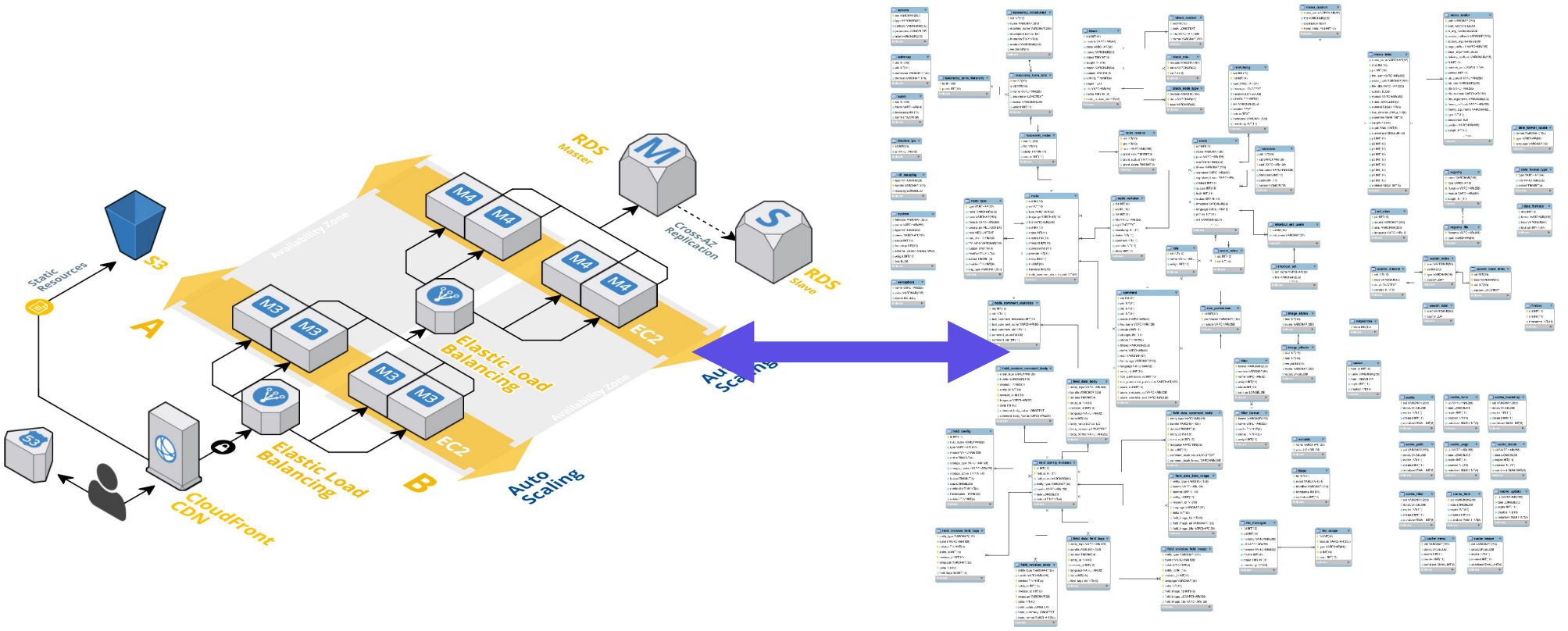


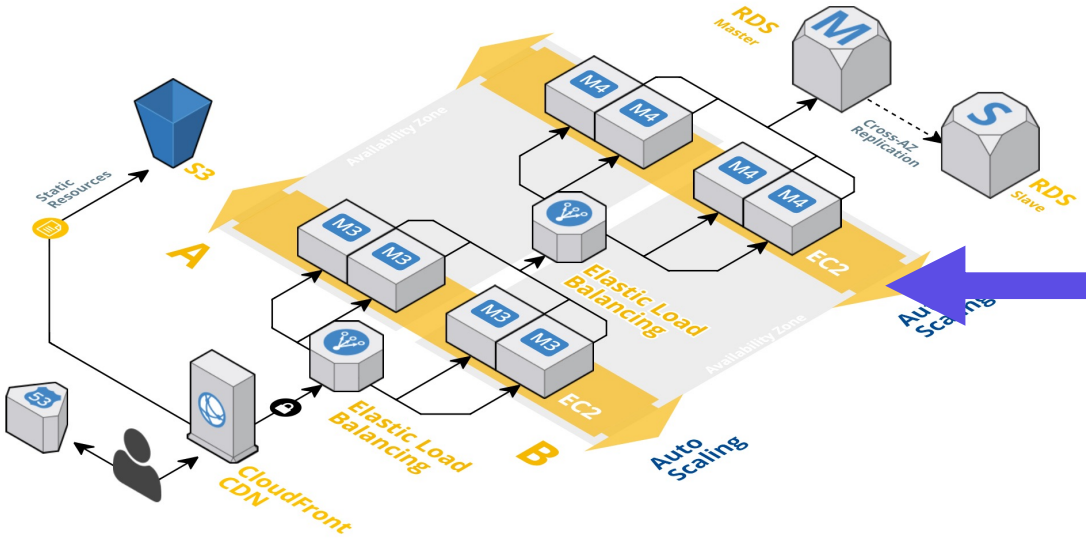
0	2	4	6	8	10
No Pain	A Little Pain	A Little More Pain	Even More Pain	A Whole Lot Of Pain	Worst Pain

Capability

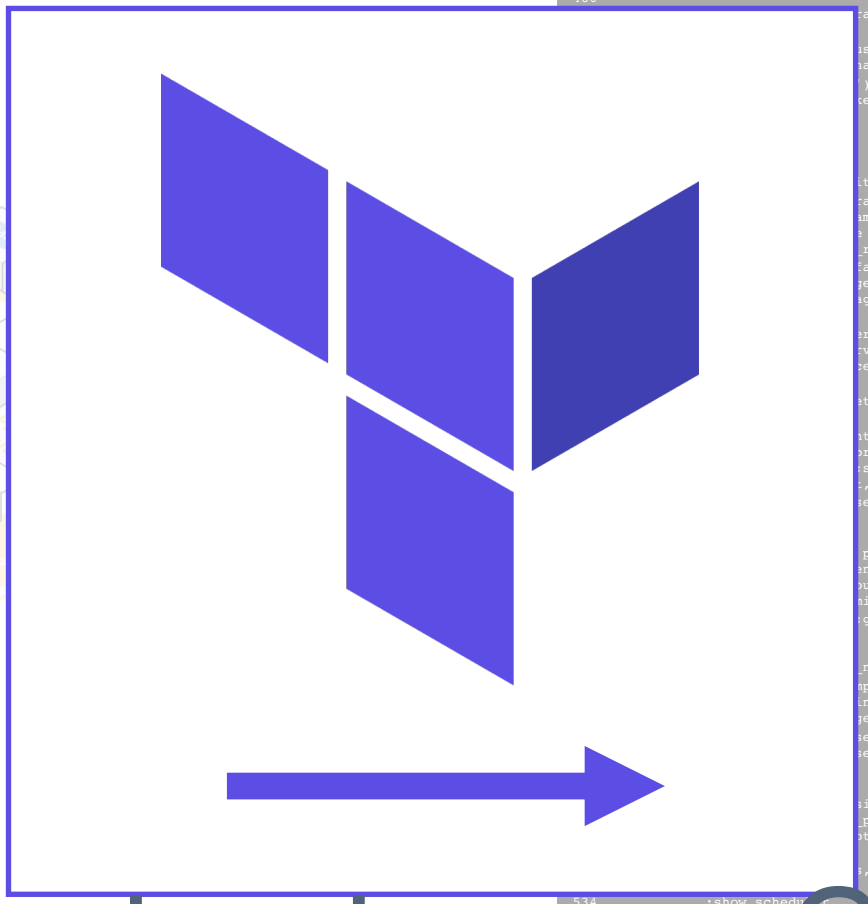
Complexity


RDS
Master
M
Cross-AZ
Replication
S
RDS
Slave
S3
Static
Resources
Availability Zone
M4
M4
M4
M4
M3
M3
M3
M3
A
B
Elastic Load
Balancing
EC2
Auto
Scaling
Amazon
Route 53
53
CloudFront
CDN
Elastic Load
Balancing
EC2
Auto
Scaling
Availability Zone


RDS
Master
Cross-AZ
Replication
RDS
Slave
S3
Static
Resources
Availability Zone
A
M4
M3
Elastic Load
Balancing
EC2
Auto
Scaling
B
CloudFront
CDN




```ruby
def update
  return update_api if api_request?

  if authorized_action(@account, @current_user, :manage_account_settings)
    respond_to do |format|

      custom_help_links = params[:account].delete :custom_help_links
      if custom_help_links
        @account.settings[:custom_help_links] = custom_help_links.select{|k, h| h['state'] != 'delete
          hash = index_with_hash[1]
          hash.delete('state')
          hash.assert_valid_keys ["text", "subtext", "url", "available_to"]
          hash
        end
      end

      params[:account][:turnitin_host] = validated_turnitin_host(params[:account][:turnitin_host])
      enable_user_notes = params[:account].delete :enable_user_notes
      allow_sis_import = params[:account].delete :allow_sis_import
      params[:account].delete :default_user_storage_quota_mb unless @account.root_account? && !@accou
      unless @account.grants_right? @current_user, :manage_storage_quotas
        [:storage_quota, :default_storage_quota, :default_storage_quota_mb,
         :default_user_storage_quota, :default_user_storage_quota_mb,
         :default_group_storage_quota, :default_group_storage_quota_mb].each { |key| params[:account]
      end
      if params[:account][:services]
        params[:account][:services].slice(*Account.services_exposed_to_ui_hash(nil, @current_user, @a
          @account.set_service_availability(key, value == '1')
        end
        params[:account].delete :services
      end
      if @account.grants_right?(@current_user, :manage_site_settings)
        # If the setting is present (update is called from 2 different settings forms, one for notifi
        if params[:account][:settings] && params[:account][:settings][:outgoing_email_default_name_op
          # If set to default, remove the custom name so it doesn't get saved
          params[:account][:settings][:outgoing_email_default_name] = '' if params[:account][:setting
        end

        google_docs_domain = params[:account][:settings].try(:delete, :google_docs_domain)
        if @account.feature_enabled?(:google_docs_domain_restriction) &&
           @account.root_account? &&
           !@account.site_admin?
          @account.settings[:google_docs_domain] = google_docs_domain.present? ? google_docs_domain :
        end

        @account.enable_user_notes = enable_user_notes if enable_user_notes
        @account.allow_sis_import = allow_sis_import if allow_sis_import && @account.root_account?
        if @account.site_admin? && params[:account][:settings]
          # these shouldn't get set for the site admin account
          params[:account][:settings].delete(:enable_alerts)
          params[:account][:settings].delete(:enable_eportfolios)
        end
      else
        # must have :manage_site_settings to update these
        [ :admins_can_change_passwords,
          :admins_can_view_notifications,
          :enable_alerts,
          :enable_eportfolios,
          :enable_profiles,
          :show_scheduler,
          :global_includes,
          :gmail_domain
        ].each do |key|
          params[:account][:settings].try(:delete, key)
```



Infrastructure as Code

HashiCorp

Terraform Concepts

Terraform's Goals

Unify the view of resources using infrastructure as code

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

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

Provide a workflow that is technology agnostic

Manage anything with an API

Terraform vs. Other Tools

Provides a high-level abstraction of infrastructure (IaC)

Allows for composition and combination

Supports parallel management of resources (graph, fast)

Separates planning from execution (dry-run)

Infrastructure as Code

Provide a codified workflow to create infrastructure

Expose a workflow for managing updates to existing infrastructure

Integrate with application code workflows (Git, SCM, Code Review)

Provide modular, sharable components for separation of concerns

Infrastructure as Code (Terraform)

Human-readable configuration (HCL) is designed for human consumption so users can quickly interpret and understand their infrastructure configuration

Terraform can also parse configuration in JSON for machine-generated configurations

Configuration format is very VCS friendly with support for multi-line lists, trailing commas, and auto-formatting

GIT DIFF EXAMPLE.TF

```
 resource "azure_resource_group" "training" {
   name           = "myResourceGroup"
-  location       = "East US"
+  location       = "West US"
 }
```

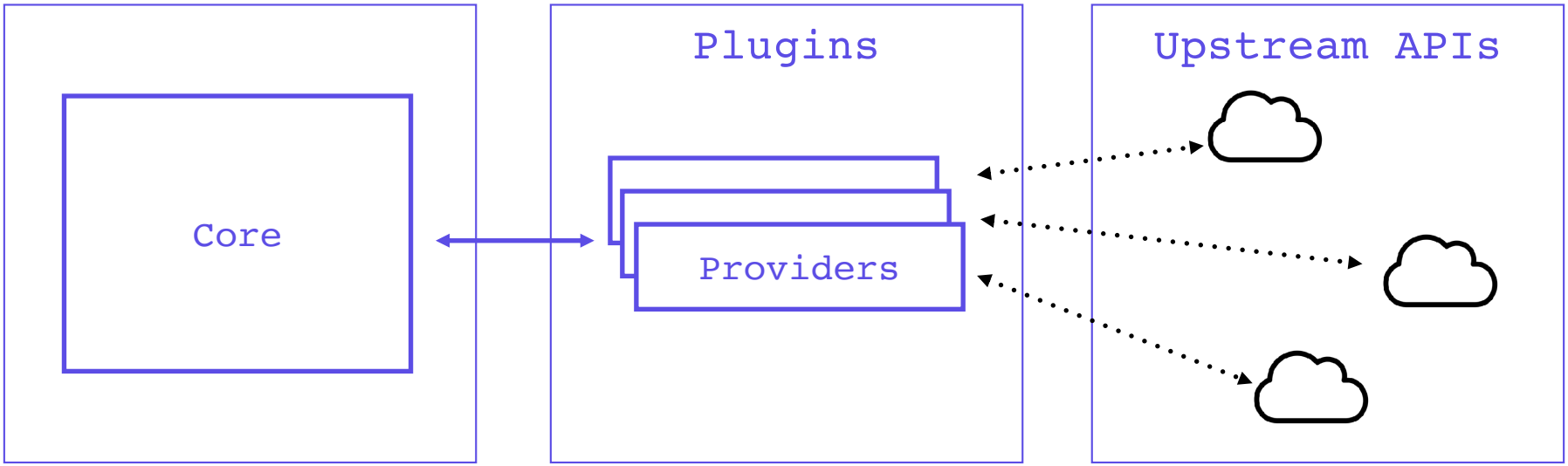
Infrastructure as Code (Terraform)

Reference values from other resources, building the implicit dependency graph.

EXAMPLE.TF

```
resource "azurerm_network_interface" "web" {
  name                = "myWebServerNetworkInterface"
  ip_configuration= {
          private_ip_address = “10.0.2.5”
        }
}

resource "dnsimple_record" "web" {
  domain = "hashicorp.com"
  name   = "web"
  ttl    = "3600"
  type   = "A"
  value  = azurerm_network_interface.web.private_ip_address
}
```

Terraform's Internals: Structure



Terraform's Internals: Core Concepts

Config: Target Reality

State: Current Reality

Diff: {Config - State}

Plan: Presents Diff

Apply: Resolves Diff

```
provider "azurerm" {
  features {}
}

resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                            ”environment" = "production”
                          }
}
```

Command Line Interface

All interactions with Terraform occur via the CLI

Terraform is a local tool (runs on the current machine)

The terraform ecosystem also includes providers for many cloud services, and a module repository

Hashicorp also has products to help teams manage Terraform: Terraform Cloud and Terraform Enterprise

TERMINAL

```
> terraform help
Common commands:
    apply               Builds or changes infrastructure
    console             Interactive console for Terraform interpolations
    destroy             Destroy Terraform-managed infrastructure
    env                 Workspace management
    fmt                 Rewrites config files to canonical format
    get                 Download and install modules for the configuration
    graph               Create a visual graph of Terraform resources
    import              Import existing infrastructure into Terraform
    init                Initialize a Terraform working directory
    output              Read an output from a state file
    plan                Generate and show an execution plan
    providers           Prints a tree of the providers used in the config
    push                Upload this Terraform module to Atlas to run
    refresh             Update local state file against real resources
    show                Inspect Terraform state or plan
    taint               Manually mark a resource for recreation
```

Terraform Init

You must run `terraform init` to initialize a new Terraform working directory, and after changing provider configuration

You **do not** need to run `terraform init` before each command

Similar to `git init`

TERMINAL

```
> terraform init
Initializing provider plugins...
- Checking for available provider plugins on https://releases...
- Downloading plugin for provider "azurerm" (3.8)...

The following providers do not have any version constraints in
configuration, so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain
breaking changes, it is recommended to add version = "..." constraints to
the corresponding provider blocks in configuration, with the constraint
strings suggested below.

* provider.azurerm: version = "~> 3.8"

Terraform has been successfully initialized!
```

Command: `terraform plan`

The plan shows you what will happen

You can save plans to guarantee what will happen

Plans show reasons for certain actions (such as re-create)

Prior to Terraform, users had to guess change ordering, parallelization, and rollout effect

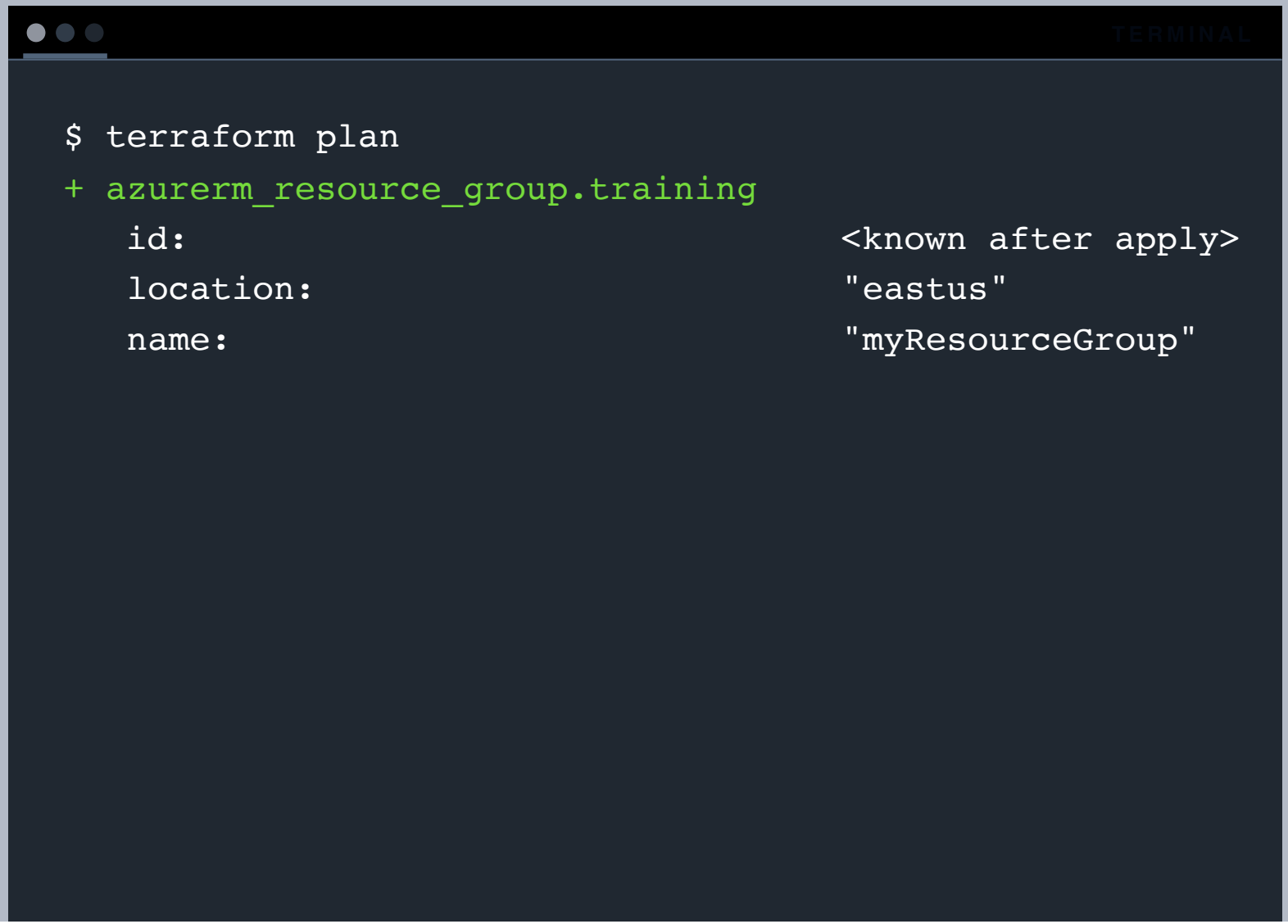
Command: `terraform plan`

`+` resource will be created

`-` resource will be destroyed

`~` resource will be updated in-place

`-/+` resources will be destroyed and re-created


TERMINAL
$ terraform plan
+ azurerm_resource_group.training
id: <known after apply>
location: "eastus"
name: "myResourceGroup"

Command: `terraform apply`

Executes changes in order based on the resource graph

Parallelizes changes when possible

Handles and recovers transient errors

Runs `plan` first unless given a serialized plan file

Command: `terraform apply`

Updates existing resources when updates are allowed

Re-creates existing resources when updates are not allowed

TERMINAL

```
> terraform apply

# ...

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

azurerm_resource_group.training: Creating...
azurerm_resource_group.training: Still creating... [10s elapsed]
azurerm_resource_group.training: Still creating... [20s elapsed]
azurerm_resource_group.training: Still creating... [30s elapsed]
azurerm_resource_group.training: Creation complete after 31s [id=i-
0b8bf14603ebdc264]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Command: `terraform show`

Displays human-friendly state

Can be used to get on-the-fly information

EXERCISE

In-Place Update

Edit the `main` Terraform file and add at least two tags to the Azure Resource Group.

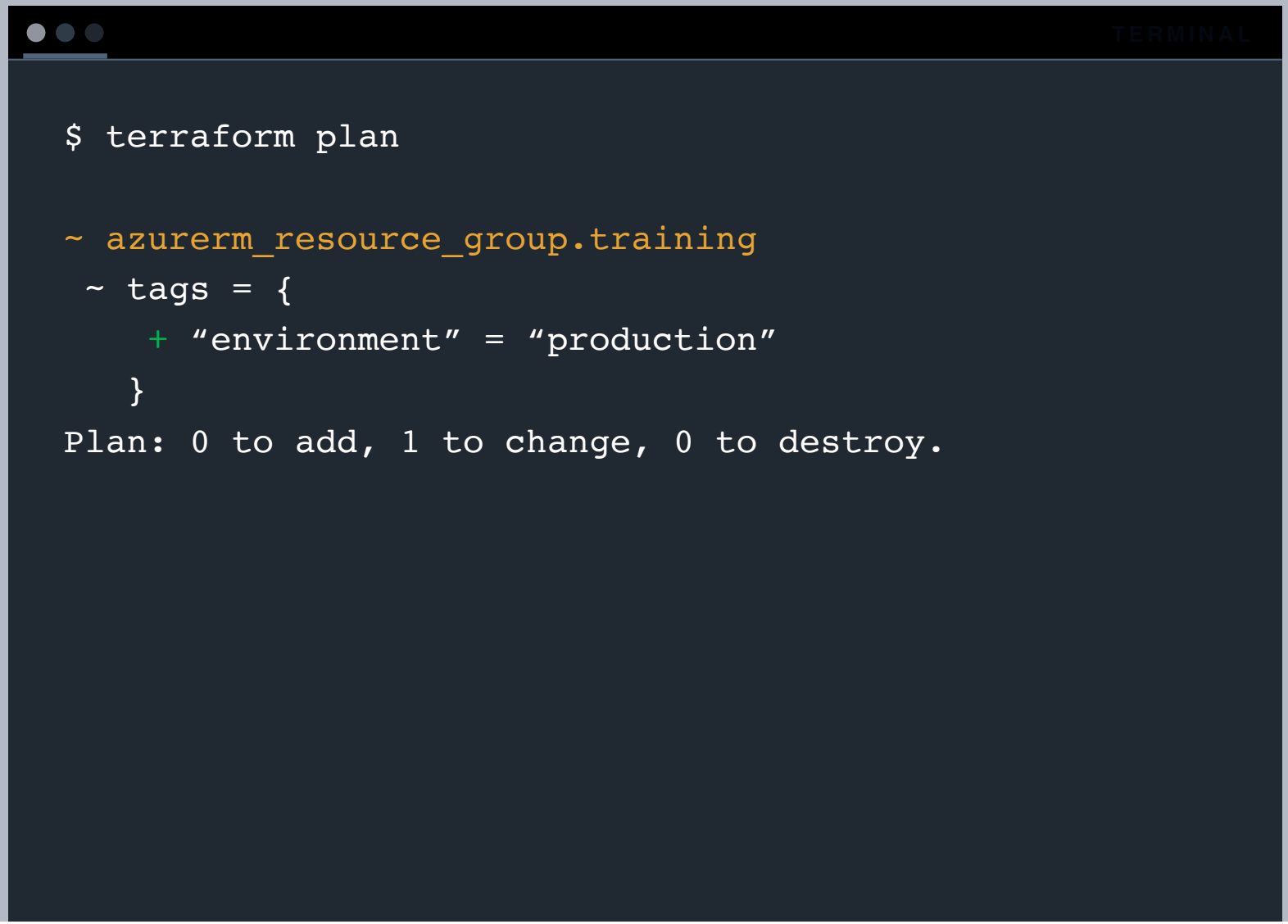
Built-By

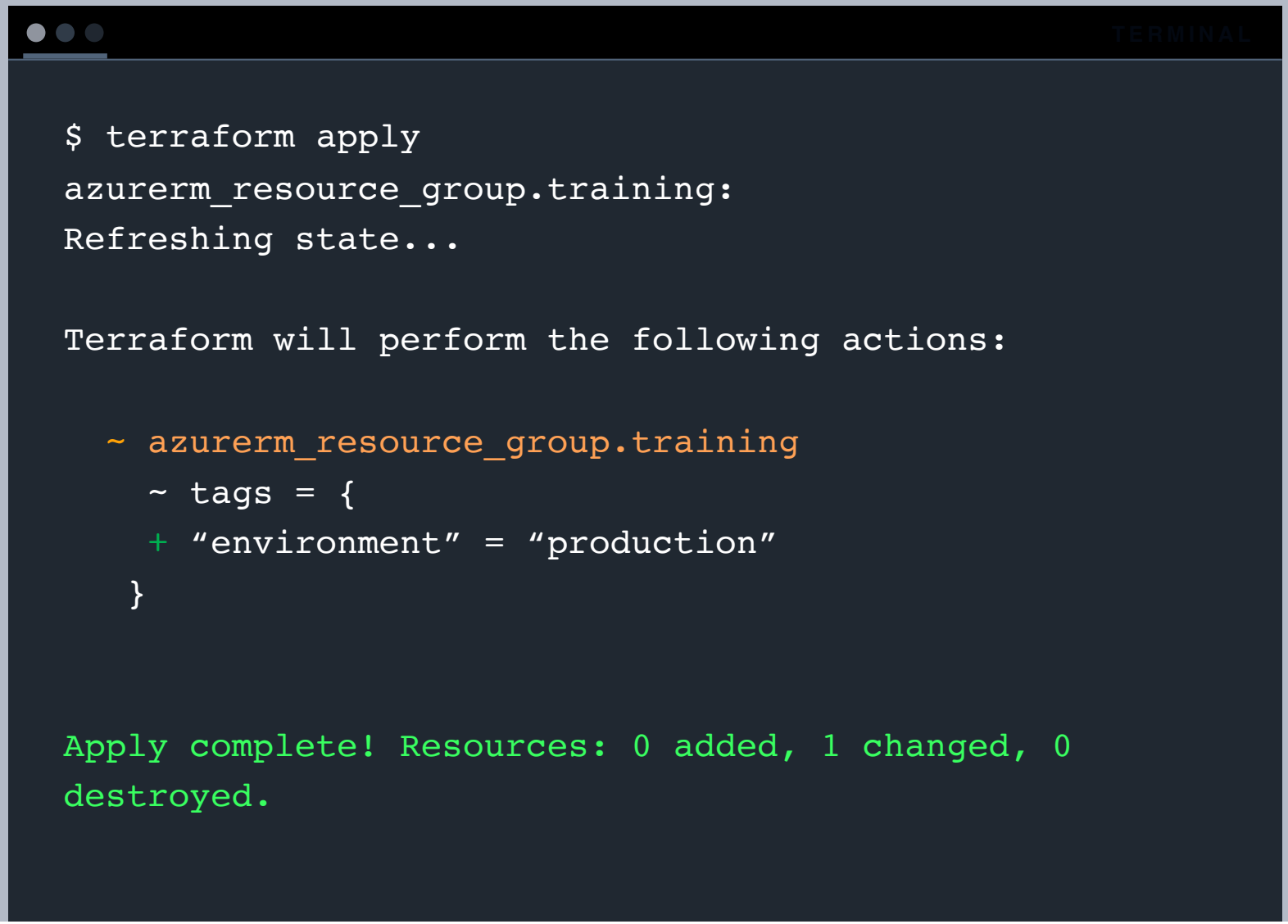
Environment

HINT: You may need to look at Terraform's documentation for the syntax (terraform.io).

MAIN.TF

```
provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" ”training" {
  name                        = "myResourceGroup"
  location                    = "East US”
  tags                        = {
                                   ”environment” = “production”
                              }
}
```


TERMINAL
$ terraform plan
~ azurerm_resource_group.training
~ tags = {
+ “environment” = “production”
}
Plan: 0 to add, 1 to change, 0 to destroy.


TERMINAL
$ terraform apply
azurerm_resource_group.training:
Refreshing state...
Terraform will perform the following actions:
~ azurerm_resource_group.training
~ tags = {
+ “environment” = “production”
}
Apply complete! Resources: 0 added, 1 changed, 0
destroyed.

TIP

Use Version Control Tools For Rollback

Terraform only knows about your configuration and the state of the infrastructure it has built.

If you need to rollback, use the capabilities of your version control software to revert to an earlier version of `main.tf`, then run `terraform apply` on it.

What You Learned

- Login to your workstation
- Create a basic Terraform configuration
- Initialize and apply the configuration (create infrastructure)
- Change and re-apply the configuration

EXERCISE

Lab 0: Instruqt Overview

Duration: 10 minutes

Goal: Familiarize yourself with your VSCode and Instruqt Labs

EXERCISE

Lab 2 & 3: Terraform Basic Configuration and Virtual Machine

Duration: 25 minutes

Goal: Familiarize yourself with your workstation and the Terraform CLI

Terraform Configuration

Configuration Syntax

Configurations are written in HashiCorp Configuration Language (HCL)

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

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

Load Order & Semantics

Configuration can be in a single file or split across multiple files. Terraform will process all files in the current working directory which end in `.tf` or `.tf.json`

Sub-folders are not included (non-recursive)

Files are processed in lexicographical (dictionary) order

Any files with a different extension are ignored (e.g. `.jpg` or `.tf.old`)

Generally, the order in which things are defined doesn't matter.

Load Order & Semantics

Parsed configurations are *appended* to each other, not merged

Resources with the same name are **not** merged (this will produce an error)

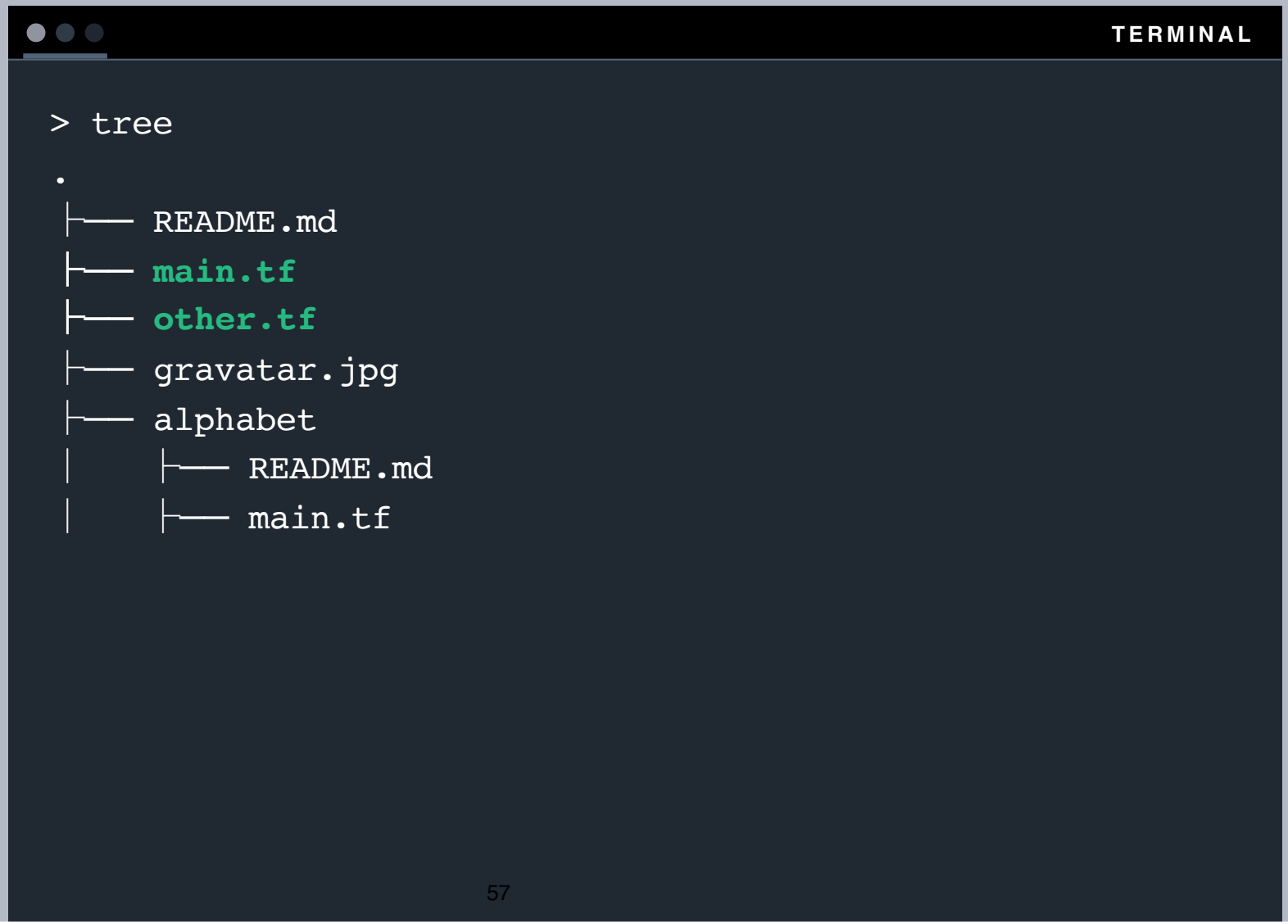
Configuration syntax is declarative, so references to other resources do not depend on the order they are defined

EXERCISE

Which Files Does Terraform Use?

Which of the following files would be parsed by Terraform when a command is issued?

```
> tree
.
├── README.md
├── main.tf
├── other.tf
├── gravatar.jpg
├── alphabet
│   ├── README.md
│   ├── main.tf
```


TERMINAL
> tree
.
├── README.md
├── main.tf
├── other.tf
├── gravatar.jpg
├── alphabet
│   ├── README.md
│   ├── main.tf

Configuration Organization Patterns

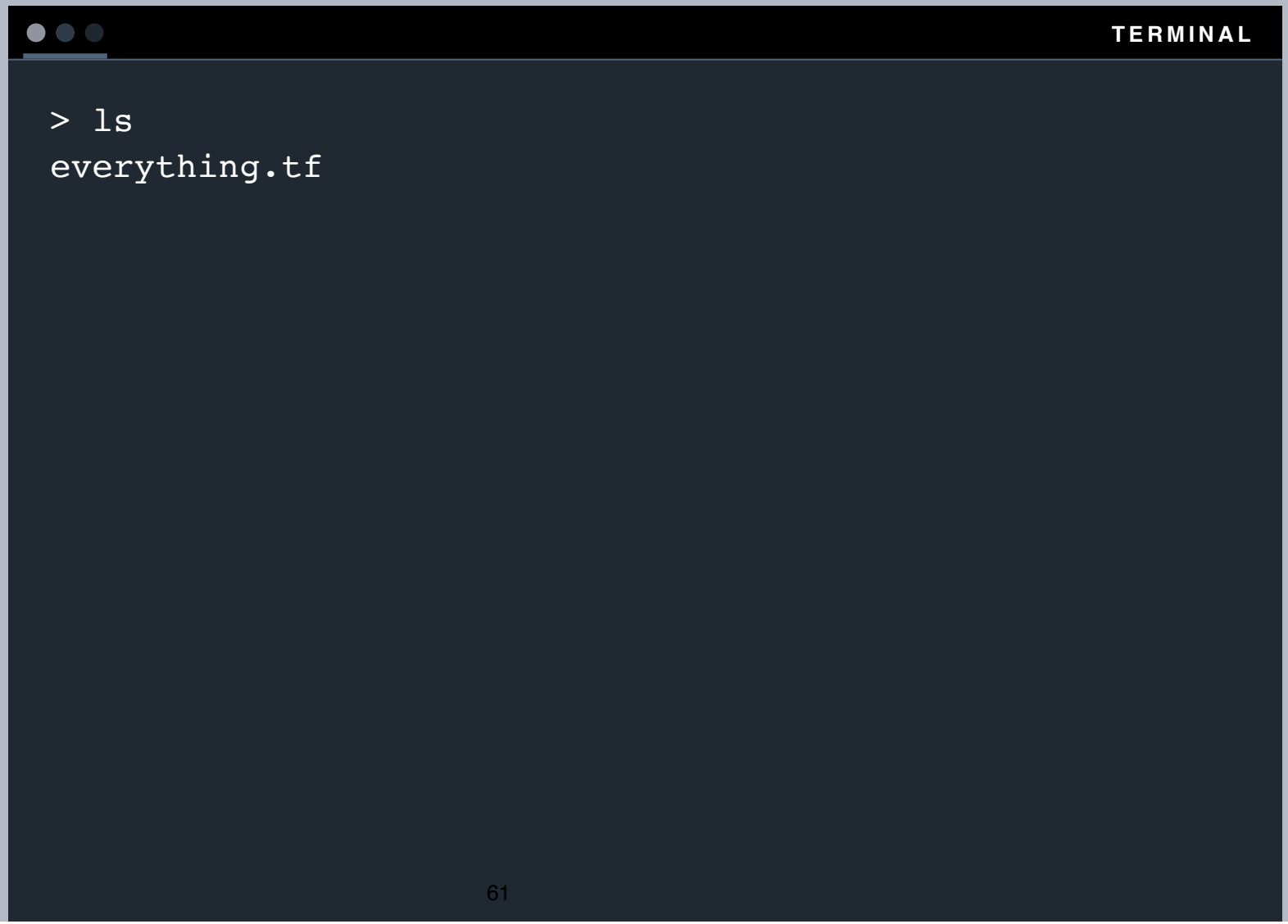
Teams choose to organize their configurations in many different patterns.

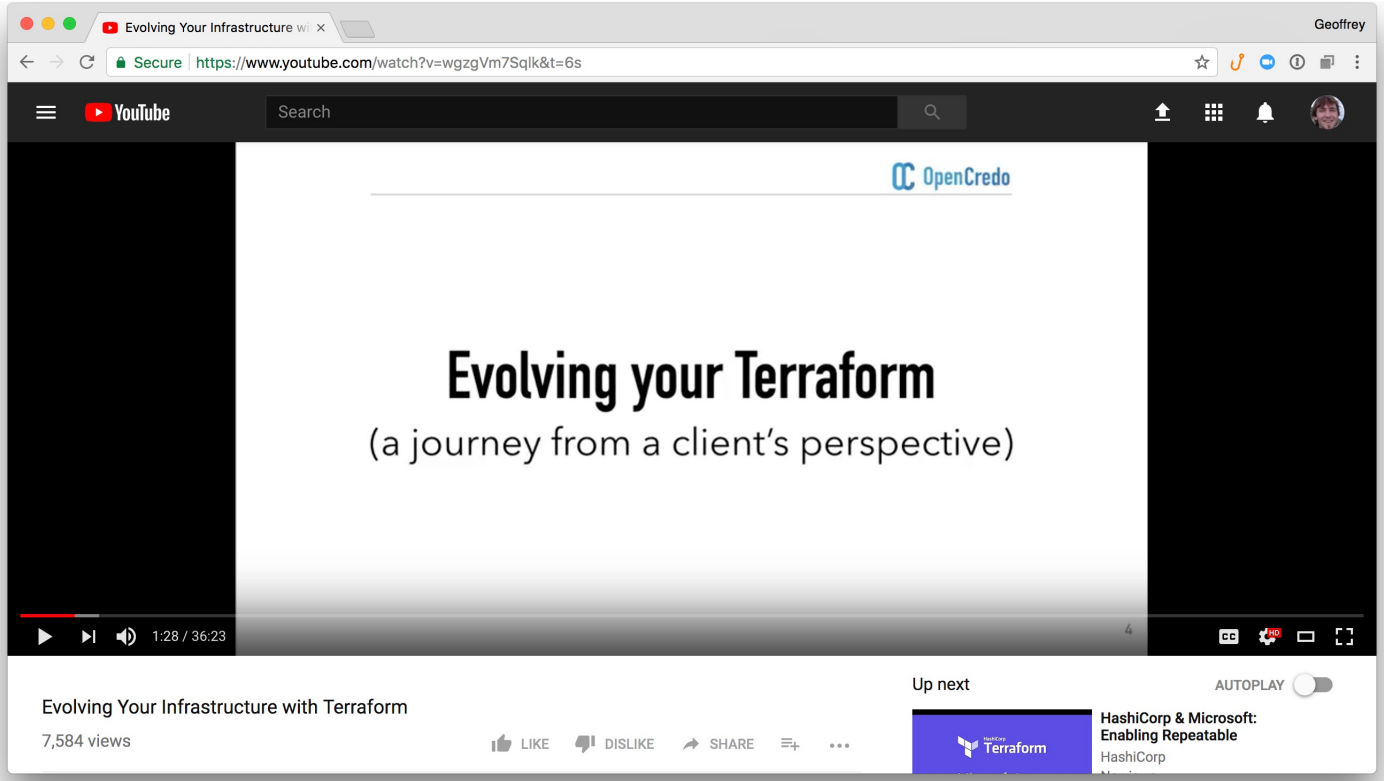
TERMINAL

```
> ls
main.tf
outputs.tf
variables.tf
```

TERMINAL

```
> ls
instances.tf
load-balancers.tf
shared.tf
```


TERMINAL
> ls
everything.tf


Evolving Your Infrastructure w
Geoffrey
Secure
https://www.youtube.com/watch?v=wgzgVm7Sqlk&t=6s
YouTube
Search
OpenCredo
Evolving your Terraform
(a journey from a client's perspective)
1:28 / 36:23
Evolving Your Infrastructure with Terraform
7,584 views
LIKE
DISLIKE
SHARE
Up next
AUTOPLAY
Terraform
HashiCorp & Microsoft:
Enabling Repeatable
HashiCorp

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                           ”environment” = production”
                          }
}
```

Top Level Keywords

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

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                           ”environment” = production”
                          }
}
```

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                           ”environment” = production”
                          }
}
```

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size                 = ”Standard_F2"
  tags                  = {
                            ”environment” = production”
                          }
}

resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size                 = ”Standard_F2"
  tags                  = {
                            ”environment” = production”
                          }
}
```

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                      = "my_server"
  location                  = "East US"
  resource_group_name       = "my_rg"
  network_interface_ids     = ["azure_network_interface.main.id"]
  vm_size                   = ”Standard_F2"
  tags                      = {
                               ”environment” = production”
                              }
}

resource "azurerm_virtual_machine" "web-2" {
  name                      = "my_server"
  location                  = "East US"
  resource_group_name       = "my_rg"
  network_interface_ids     = ["azure_network_interface.main.id"]
  vm_size                   = ”Standard_F2"
  tags                      = {
                               ”environment” = production”
                              }
}
```

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                           ”environment” = production”
                          }
}

resource "digitalocean_droplet" "web" {
  image  = "ubuntu-18-04-x64"
  name   = "web"
  size   = "s-1vcpu-1gb"
  region = "nyc2"
}
```

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                           ”environment” = production”
                          }
}
```

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                           ”environment” = production”
                          }
}
```

MAIN.TF

```
resource "azurerm_network_interface" "web" {
  name                = "myWebServerNetworkInterface"
  ip_configuration = {
          private_ip_address = “10.0.2.5”
        }
}

resource "dnsimple_record" "web" {
  domain = "hashicorp.com"
  name   = "web"
  ttl    = "3600"
  type   = "A"
  value = azurerm_network_interface.web.private_ip_address
}
```

MAIN.TF

```
# This is a comment

resource "azurerm_network_interface" "web" {
  name                = "myWebServerNetworkInterface"
  ip_configuration = {
        private_ip_address = “10.0.2.5”
      }
}

/* This is a multiline
comment */

resource "dnsimple_record" "web" {
  # ...
}
```

MAIN.TF

```
variable "thing" {
  description = <<EOF
This is a
multiline
string
EOF
}
```

Syntax Highlighting

Plugins for Terraform/HCL exist for most major editors, but `ruby` tends to work best if one does not exist.

Outputs

What You'll Learn

- Print dynamic data with an output
- Query specific values from the output

Outputs

Outputs define useful values that will be highlighted to the user when Terraform applies: IP addresses, usernames, generated keys

Outputs can be queried using the `terraform output` command

Outputs are a way to easily extract and query information from all of Terraform's collected data

MAIN.TF

```
resource "azurerm_resource_group" "training" {
# ...
}

output "location" {
  value = azurerm_resource_group.training.location
}

# Prior to Terraform 0.12 syntax:

output "location" {
  value = "${azurerm_resource_group.training.location}"
}
```

MAIN.TF

```
resource "azurerm_resource_group" "training" {
# ...
}

output "location" {
  value = azurerm_resource_group.training.location
}

output ”name" {
  value = azurerm_resource_group.training.name
}
```

TERMINAL

```
> terraform refresh
azurerm_resource_group.training: Refreshing state... [id=i-
0b8bf14603ebdc264]

Outputs:

location = "eastus”
name     = "myResourceGroup”
```

TERMINAL

```
> terraform output
location = "eastus”
name     = "myResourceGroup”
```

TERMINAL

```
> terraform output location
"eastus"
```

What You Learned

- Print dynamic data with an output
- Query specific values from the output

EXERCISE

Lab 4: Outputs

Duration: 10 minutes

Goal: Configure specific outputs and learn to query your configuration for those.

Console

What You'll Learn

- Interact with live data in the console
- Use the console from the command line

Command: `terraform console`

The `terraform console` command creates an interactive console for interacting with Terraform

This is useful for experimenting with interpolations and interacting with Terraform state

TERMINAL

```
> terraform console
> azurerm_resource_group.training.location
eastus
> azurerm_resource_group.training.name
myResourceGroup
> azurerm_resource_group.training.tags
{
  "environment" = "production"
}
> <Ctrl-C>
```

TERMINAL

```
> terraform console
> exit
```

TERMINAL

```
> echo "azurerm_resource_group.training.location" |
terraform console

"eastus”
```

Functions

Terraform has built-in functions to achieve common data manipulation tasks

Functions categories include strings, collections, encoding, and encryption.

Functions can be tested using the `terraform console` command

MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  name                  = "my_server"
  location              = "East US"
  resource_group_name   = "my_rg"
  network_interface_ids = ["azure_network_interface.main.id"]
  vm_size               = ”Standard_F2"
  tags                  = {
                           ”environment” = production”
                          }
}

output "server_name" {
  value = lower(azurerm_virtual_machine.web.name)
}
```

EXERCISE

Lab 5: Console

Duration: 10 minutes

Goal: The Read-Evaluate-Print-Loop is useful for understanding interpolation and finding specific information.

Variables

What You'll Learn

- Use variables to pass values to your configuration
- Refactor existing configuration to use variables
- Keep sensitive data out of source code control
- Pass variables to Terraform in several ways

Variables

Work a lot like variables from programming languages

Allow you to remove hard coded values and pass parameters to your configuration

Can help make configuration easier to understand and re-use

Must always have a value. Variables are never optional, but they can have a default value

EXAMPLE.TF

```
variable "animal" {}
```

EXAMPLE.TF

```
variable "animal" {
  type        = "string"
  default     = "moose"
  description = "An animal, such as whale, elephant, or
bear"
}
```

Variable Types

The most common variable types are `string`, `number`, `list`, and `map`

Other support types include `bool` (true/false), `set`, `object`, and `tuple`

If omitted, the type is inferred from the default value

If neither type nor default is provided, the type is assumed to be `string`

VARIABLES.TF

```
variable "resource_group_name" {}
variable "admin_password" {}
variable "location" {
  default = "eastus"
}
```

```
variable "resource_group_name" {
  type = "string"
}

variable "admin_password" {
  type = "string"
}

variable "location" {
  type    = "string"
  default = "eastus"
}
```

VARIABLES.TF

```
variable "resource_group_name" {
  type        = "string"
  description = "Azure Resource Group Name"
}

variable "admin_password" {
  type        = "string"
  description = "Virtual Machine Password"
}

variable "location" {
  type        = "string"
  default     = "eastus"
  description = "Azure Location"
}
```

Using Variables

Variables can be referred to by using the syntax ***var.<name>***

Variables' values can also be interpolated into a string using the syntax ***"${var.<name>}"***

MAIN.TF

```
resource "azurerm_resource_group" ”training" {
  name          = var.resource_group_name
  location      = var.location
  tags          = {
                  ”environment” = “production”
                  }
}
```

TERMINAL

```
> terraform plan
var.resource_group_name
  Enter a value:

<Ctrl + C>
```

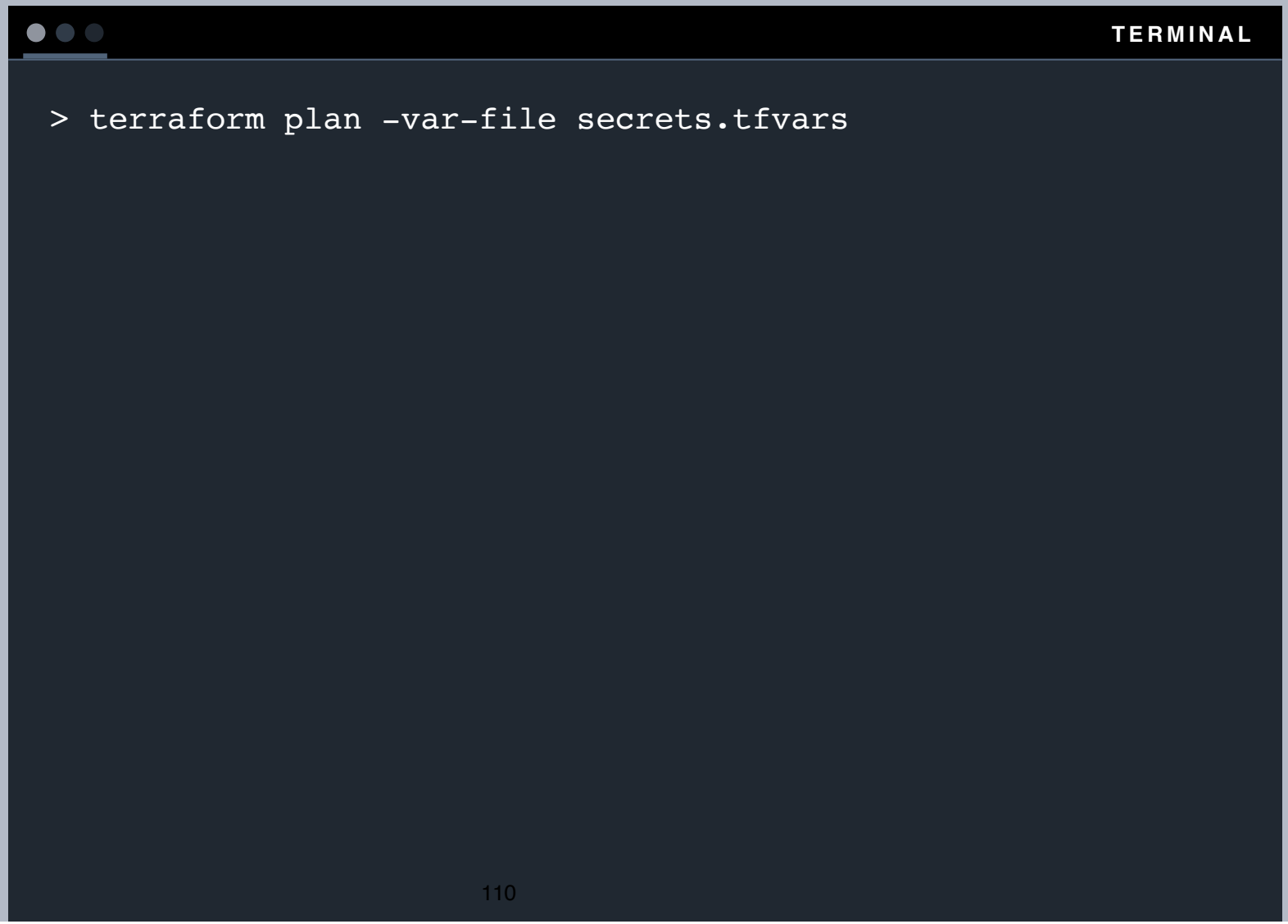
TERMINAL

```
> export TF_VAR_resource_group_name=<rg_name>

> terraform plan
var.admin_password
  Enter a value:

<Ctrl + C>
```

```
resource_group_name   = ”MyResourceGroupName”
admin_password        = ”Password1234!”
location              = ”East US”
# EnvironmentTag      = "staging"
# prefix              = “appA”
# computer_name     = "myserver"
# admin_username      = "testadmin"
```

TERMINAL

```
> terraform plan -var admin_password=<secret_key>
azurerm_virtual_machine.web: Refreshing state... (ID: i-
02f5717f1a84502ed)

# ...

No changes. Infrastructure is up-to-date.

# ...
```


TERMINAL
> terraform plan -var-file secrets.tfvars

Specifying Variable Values

1. Default values
2. Environment variables
3. Saved in variable definition files (`terraform.tfvars`)
4. Using the ***`-var`*** or ***`-var-file`*** command line option
5. Configured in a Terraform Enterprise or Terraform Cloud workspace
6. Entered via CLI prompts

```
variable "num_webservers" {
  type    = "number"
  default = 5
}

variable "regions" {
  type    = "list"
  default = ["eastus", "westus"]
}

variable "image_ids" {
  type    = "map"
  default = {
    us-east-1 = "image-1234"
    us-west-2 = "image-4567"
  }
}
```

WARNING

Variables & Security

Never put secret values, like passwords or access tokens, in `.tf` files or other files that are checked into source control.

Locals

Locals are values defined inside a Terraform configuration.

Locals can construct values from variables, resources, data source, and other locals.

Locals are scoped to their module.

```
locals {
  naming_prefix = "ant"
  default_name = "${local.naming_prefix}-${var.company}"
  number_list = [5,7,9]
}
```

What You Learned

- Use variables to store sensitive values
- Refactor existing configuration to use variables
- Keep sensitive data out of source code control systems
- Populate variables to Terraform in several ways

EXERCISE

Lab 6: Variables

Duration: 15 minutes

Goal: Remove all hardcoded values in your current configuration and use variables instead.

Auto-Formatting/File Layout

Auto-Formatting

Terraform has built-in support for auto-formatting configuration files to match the HCL specification.

There is no need to manually align things.

This helps keep configurations VCS-friendly and reduces bike-shed arguments over formatting styles.

Plugins exist for most major editors.

Command: `terraform fmt`

The `terraform fmt` command formats configuration files.

It does not prompt for confirmation or

It also reports on syntax errors, if they exist.

EXERCISE

Format Your Code

Run the `terraform fmt` command with no arguments. By default it will choose all Terraform configurations in the current working directory non-recursively.

Open the `main.tf` file and see that it has been formatted.

Command: `terraform fmt`

Please feel free to use the `terraform fmt` command through this training and at your leisure, but we will not explicitly discuss it anymore.

Command: `terraform validate`

The `terraform validate` checks the syntax and logic of your configuration

Checks both core terraform and provider resources

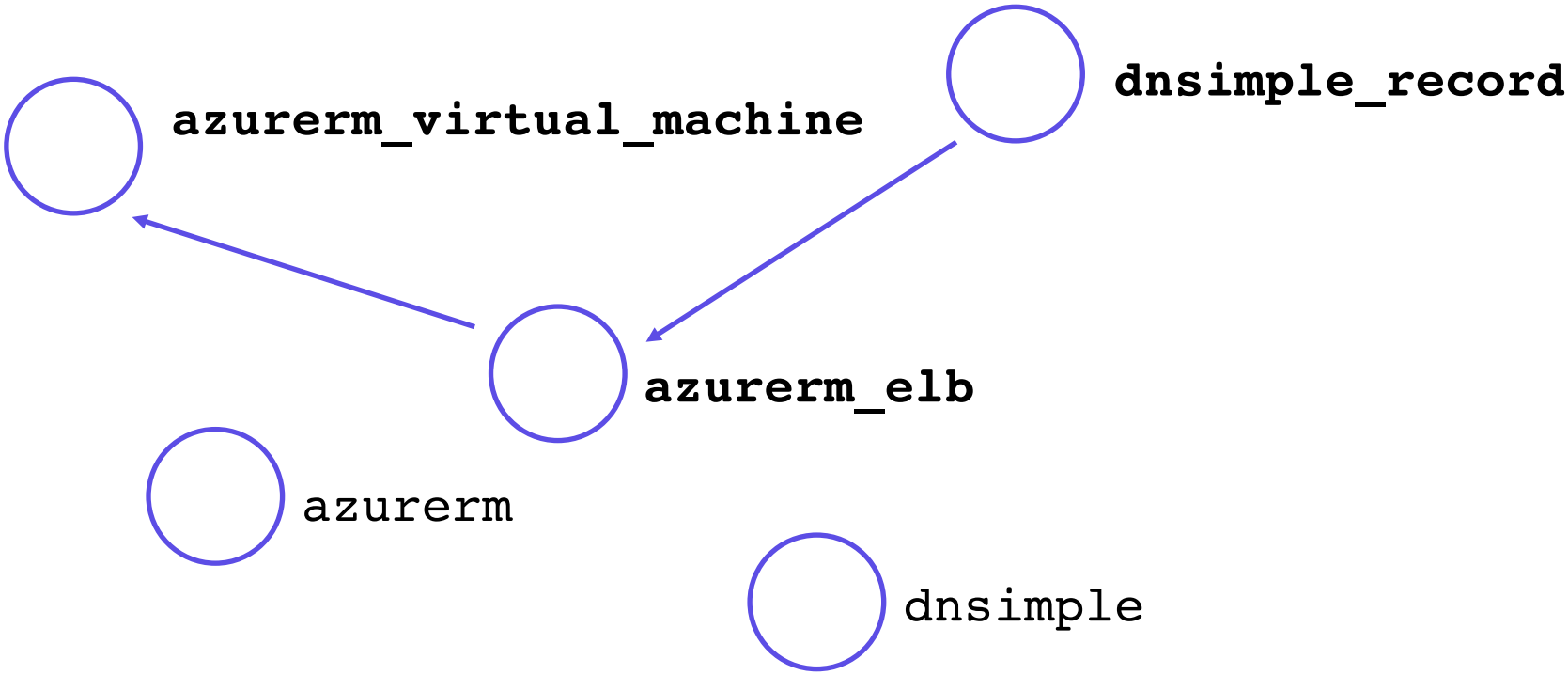
Requires initialization before running

Graph Theory

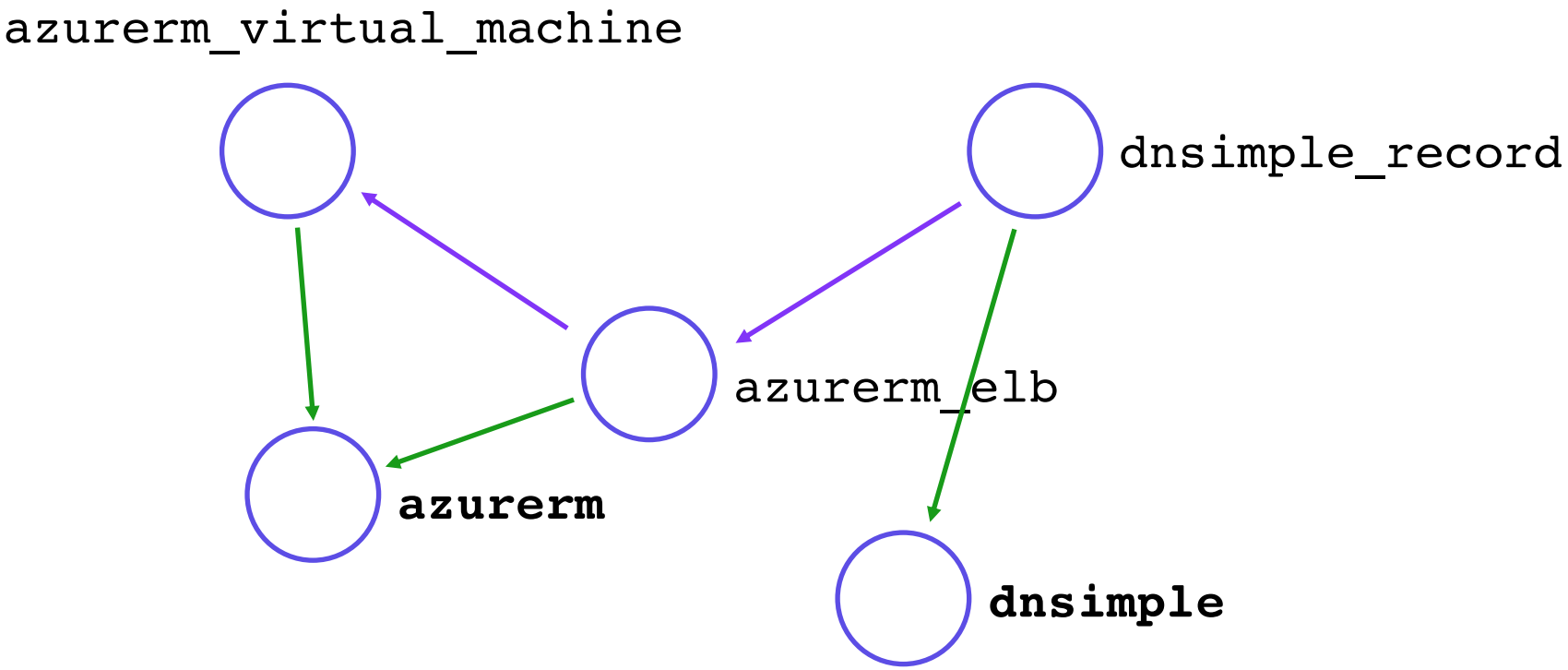
What You'll Learn

- Understand how graph theory makes Terraform more effective
- Generate a graph of your infrastructure

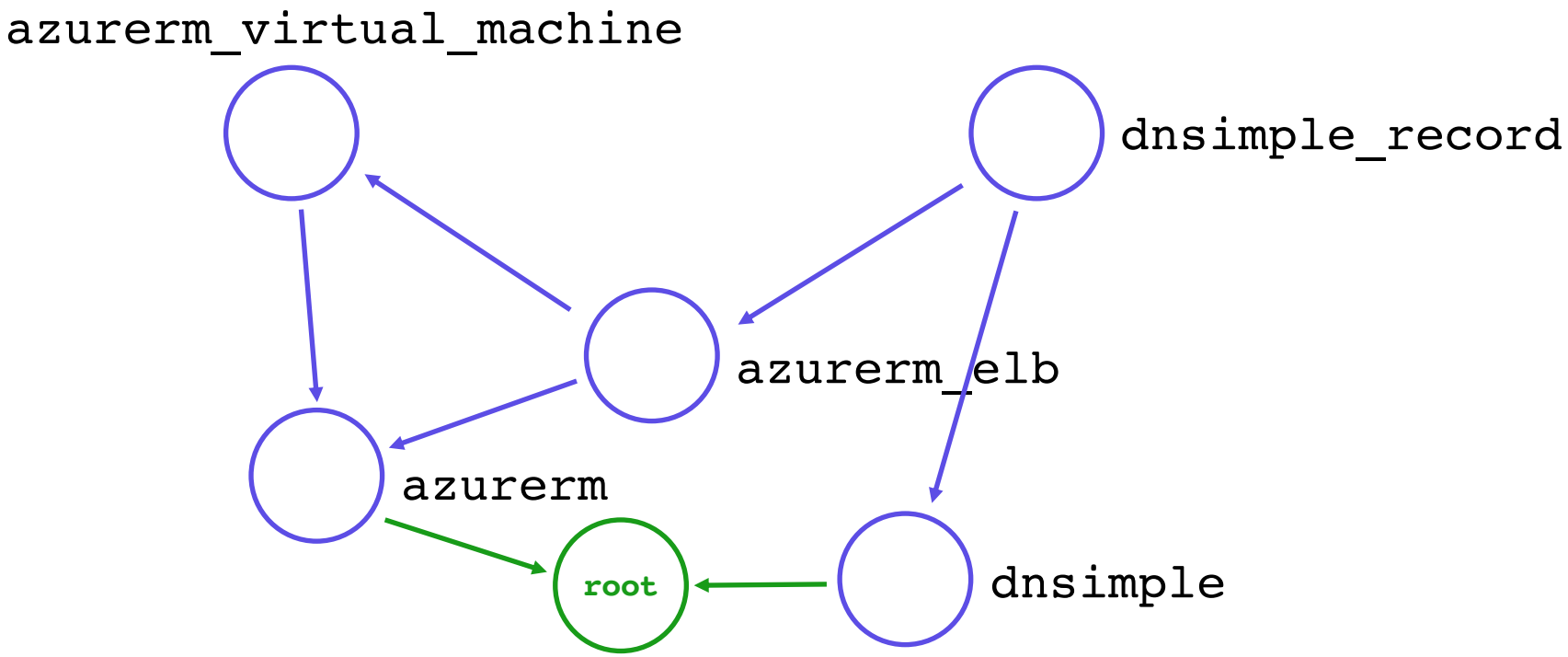
Building the Graph



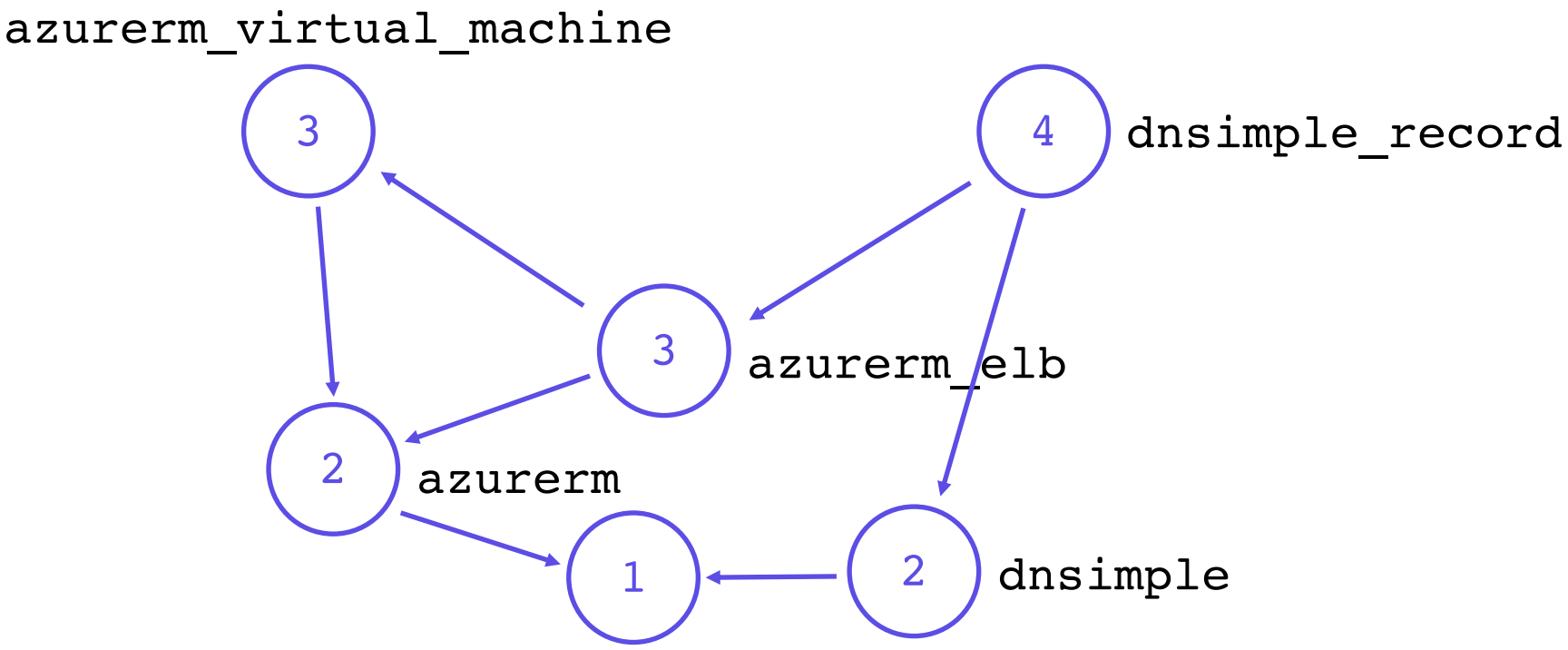
Building the Graph



Building the Graph



Walking the Graph



Resource Graph

The resource graph is an internal representation of all resources and their dependencies.

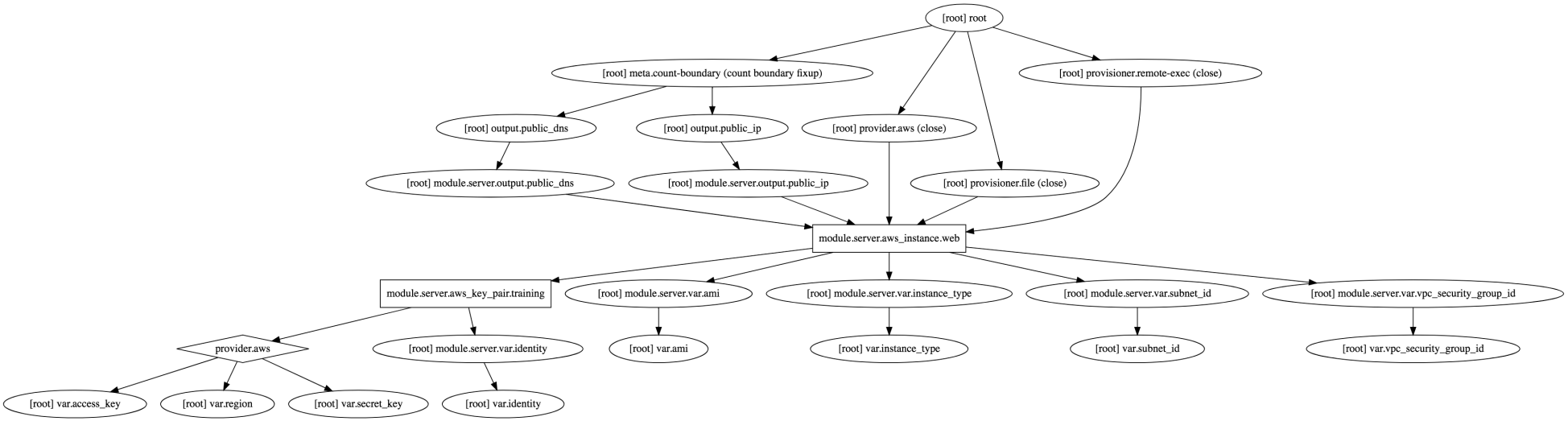
A human-readable graph can be generated using the `terraform graph` command.

Can optionally draw cycles (advanced).

TERMINAL

```
> terraform graph
digraph {
    compound = "true"
    newrank = "true"
    subgraph "root" {
        "[root] azurerm_virtual_machine.web" [label =
"azurerm_virtual_machine.web", shape = "box"]
        "[root] provider.azurerm" [label = "provider.azurerm", shape = "diamond"]
        "[root] azurerm_virtual_machine.web" -> "[root] provider.azurerm"
        "[root] meta.count-boundary (count boundary fixup)" -> "[root] ..."
        "[root] provider.azurerm (close)" -> "[root] azurerm_virtual_machine.web"
        "[root] root" -> "[root] meta.count-boundary (count boundary fixup)"
        "[root] root" -> "[root] provider.azurerm (close)"
    }
}
```

terraform graph



Terraform Graph

Useful for visualizing infrastructure and dependencies

Builds upon existing visualization technologies and open formats such as DOT

Can optionally draw cycles (resources that depend on each other in a circle)

EXERCISE

Lab 7: Graphs

Duration: 10 minutes

Goal: Visualize the dependencies in a Terraform configuration with the graph command.

Meta Arguments

What You'll Learn

- Understand meta arguments
- Use `count` to create multiple instances
- Manage your own metadata

Meta-Arguments

Meta-arguments allow for higher-level control flow and lifecycle management in Terraform

These don’t map directly to cloud resources or APIs, but help you control Terraform’s actions

Meta-Arguments

Count

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

Meta-Arguments

For-Each

The `for_each` argument allows for N number of resources to be created based on a map or set. Similar to `count`, the `for_each` argument gives greater flexibility over each created resource.

Meta-Arguments

Depends On

The `depends_on` argument allows for declaration of explicit dependencies. This is useful where interpolation is not required, but explicit ordering is desired

Meta-Arguments

Provider

The `provider` argument allows for a resource to explicitly use a provider by name or alias. This is most useful when there are multiple providers in a single configuration, such as `aws.west` and `aws.east`

Meta-Arguments

Lifecycle

The `lifecycle` argument allow explicit configuration of resource lifecycle such as preventing destruction or ignoring property changes. This is an advanced option and is not recommended for users who are getting started with Terraform

SERVER/SERVER.TF

```
# ...

resource "azurerm_virtual_machine" "web" {
  count               = 2

  name                = "myServer-${count.index}"
  location            = "eastus"

# ...
}
```

```
# ...


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

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

EXERCISE

Lab 8: Meta Arguments

Duration: 10 minutes

Goal: Modify the number instances created with count.

TERMINAL

```
> terraform apply
# module.server.azurerm_virtual_machine.web[1] will be
created
+ resource "azurerm_virtual_machine" "web" {
    + name:                "myServer-1"
    + location:            "useast”
# ...

Plan: 1 to add, 0 to change, 0 to destroy.

# ...

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

```
variable "num_webservers" {
  count   = number
  default = 2
}

# ...

resource "azurerm_virtual_machine" "web" {
  count = var.num_webservers

  name              = "myServer-${count.index}
  location          = "eastus"


  tags {
    # ...
    "Name" = "web ${count.index+1}/${var.num_webservers}"
  }
```

Provisioners

What You'll Learn

- How to create a connection to our server
- Specify the key for connecting to the server
- Provision the server with files and commands

WARNING

The Server We Created is Useless

We created a server without any running code.

No useful services are running on it.

In this section, we'll see how to provision or instance.

SERVER/MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  # . . .

  connection = {
    type        = "ssh"
    user        = "ubuntu"
    host        = self.public_ip
    private_key = module.keypair.private_key_pem
  }
}
```

SERVER/MAIN.TF

```
resource "azurerm_virtual_machine" "web" {
  # . . .

  provisioner "file" {
    source      = "assets"
    destination = "/tmp/"
  }

}
```

https://www.terraform.io/docs/provisioners/file.html

```
resource "azure_virtual_machine" "web" {
  # . . .

  provisioner "remote-exec" {
    inline = [
      "sudo sh /tmp/assets/setup-web.sh"
    ]
  }
}
```

https://www.terraform.io/docs/provisioners/remote-exec.html

TIP

Force Provisioning With the `–replace` argument

If we run `apply` after adding provisioners, it won't change the server.

Force it to be destroyed and recreated with the `-replace` argument:

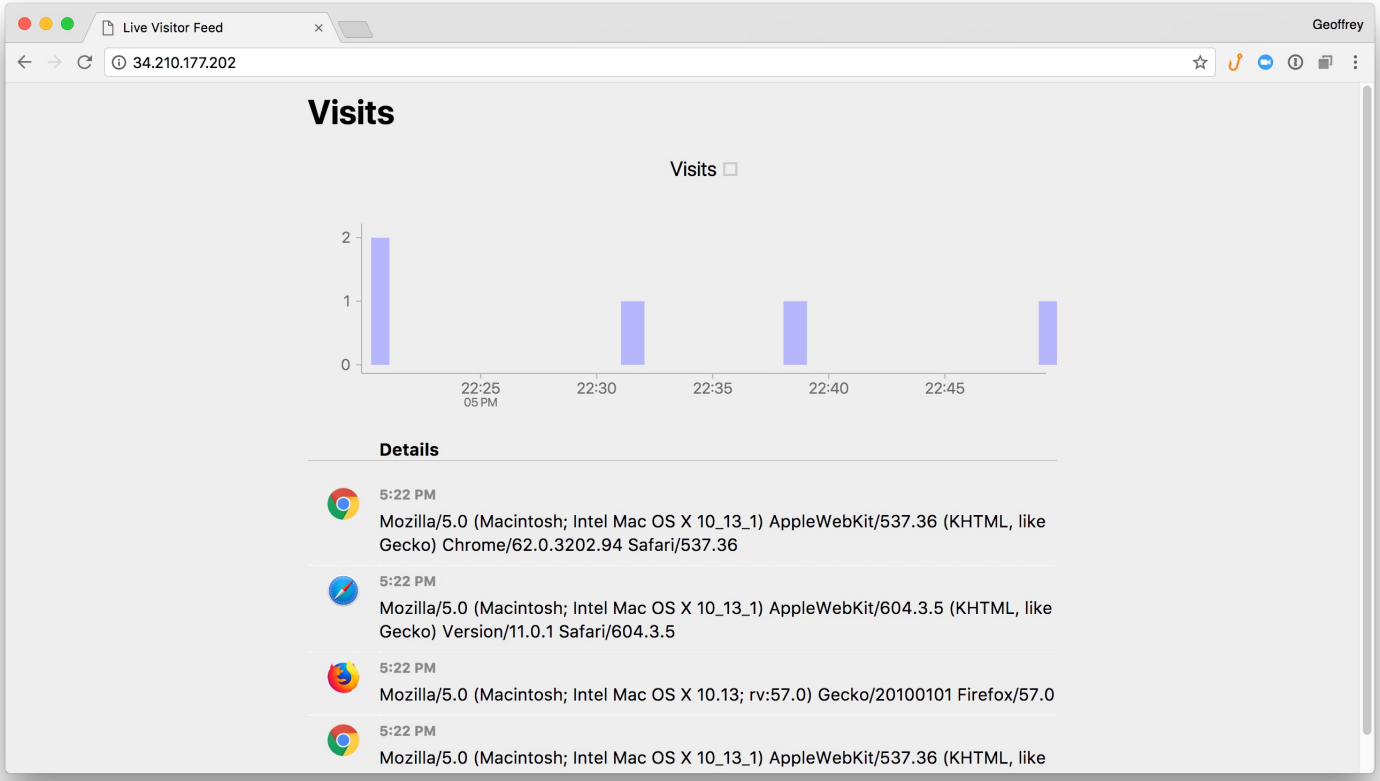
```
terraform apply -replace azurerm_virtual_machine.web
```

If you're unsure of the correct ID for the resource you want to replace, you can use terraform show:

```
> terraform show
azurerm_virtual_machine.web:
  id = " "/subscriptions/9c3c5800-dc5b-4723-8b57-
205b827d6a06/resourceGroups/myResourceGroup/providers/Microsoft.C
ompute/virtualMachines/web_vm"
  name = myWebServer
  . . .
```

TERMINAL

```
> terraform apply –replace azurerm_virtual_machine.web
# ...
Plan: 1 to add, 0 to change, 1 to destroy.
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

# ...

web: Provisioning with 'remote-exec'...
web (remote-exec): Connecting to remote host via SSH...
web (remote-exec):   Host: 35.162.171.124
web (remote-exec):   User: ubuntu
web (remote-exec):   Password: false
web (remote-exec):   Private key: true
web (remote-exec):   SSH Agent: false
web (remote-exec): Connected!
```


Live Visitor Feed
Geoffrey
34.210.177.202
Visits
Visits
2
1
0
22:25
05 PM
22:30
22:35
22:40
22:45
Details
5:22 PM
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36
5:22 PM
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_1) AppleWebKit/604.3.5 (KHTML, like Gecko) Version/11.0.1 Safari/604.3.5
5:22 PM
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:57.0) Gecko/20100101 Firefox/57.0
5:22 PM
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_1) AppleWebKit/537.36 (KHTML, like

Provisioning Best Practices

Good

Use `remote-exec` provisioner on a base AMI to run a few commands on instance creation.

Better

Automate the creation of infrastructure and initialization of instances with user_data or AWS/Azure cloud-init.

Best

Build AMIs with Packer so that minimal configuration is needed.

What You Learned

- How to create a connection to our server
- Specify the key for connecting to the server
- Provision the server with files and commands

EXERCISE

Lab 9: Provisioners

Duration: 10 minutes

Goal: Provisioners allow us to create instances that are ready to use at runtime. Use your SSH key from the last lab to connect to and install our web app on a new instance.

Modules

What You'll Learn

- The Terraform Public Module Registry
- Use a module from `main.tf`
- Understand how modules are stored on disk

Modules

Portable Terraform configurations (packages)

Allow separation of concerns and responsibilities among teams

Parallels: Chef Cookbook, Puppet Module, Ruby gem

Modules

Modules are just Terraform configurations inside a folder - there's nothing special about them.

Complex configurations, team projects, and multi-repository codebases will benefit from modules. Get into the habit of using them wherever it makes sense.

TERMINAL

```
> tree my-module
my-module
└── main.tf
```

MY-MODULE/MAIN.TF

```
variable "location" {}
variable "name" {}

resource "azurerm_network_interface" "web" {
  name                = "myWebServerNetworkInterface"
  ip_configuration = {
        private_ip_address = “10.0.2.5”
      }
}

output "ip_address" {
  value =
azurerm_network_interface.web.ipconfiguration.privat
e_ip_address
}
```

MAIN.TF

```
module "my-module" {
  # Source can be any URL or file path
  source = "../../my-module"

}
```

MAIN.TF

```
module "my-module" {
  # Source can be any URL or file path
  source   = "../../my-module"
  location = ”eastus"
  name = "myServer"
}
```

MAIN.TF

```
module "my-module" {
  # Source can be any URL or file path
  source   = "../../my-module"
  location = ”eastus"
  name     = "myServer"
}

output "ip-address" {
  value    = module.my-module.ip_address
}
```

Modules

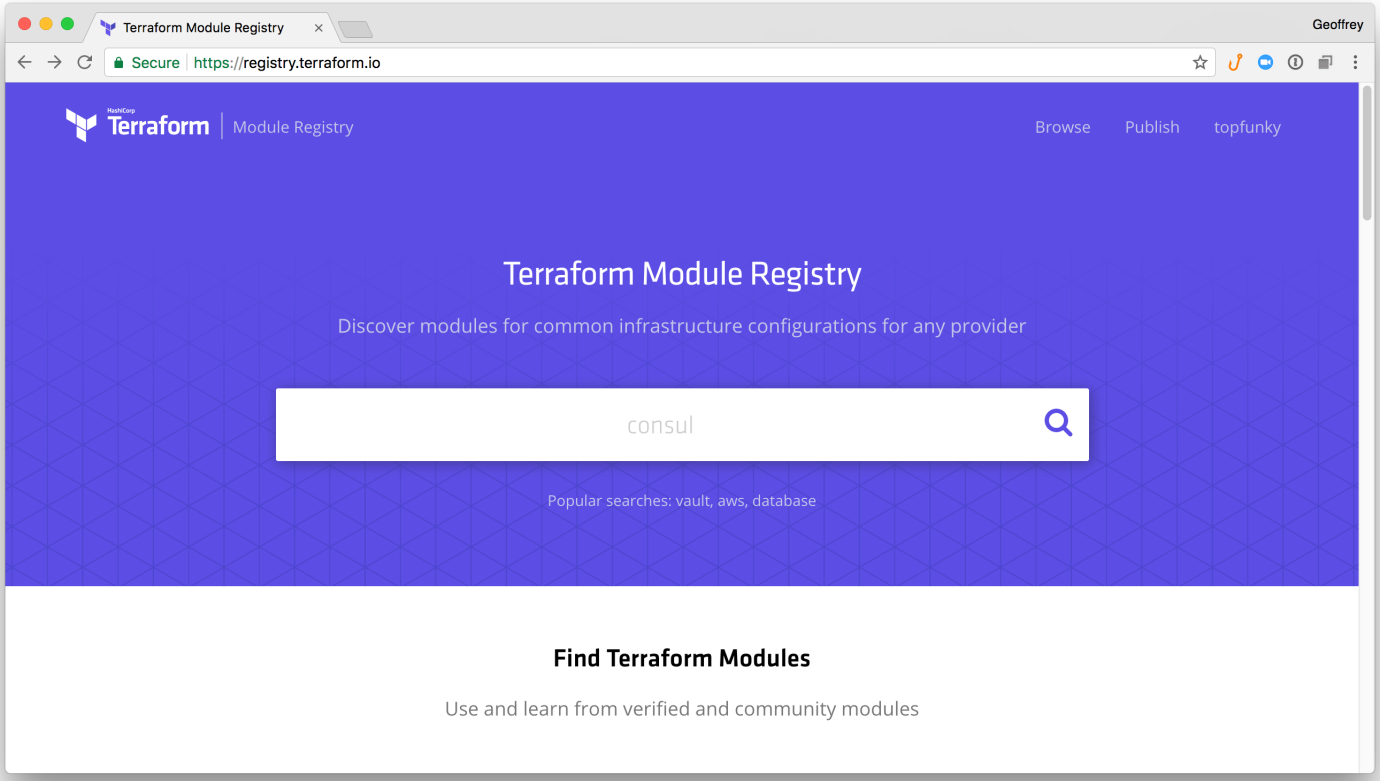
Variables defined in a module become arguments to module block

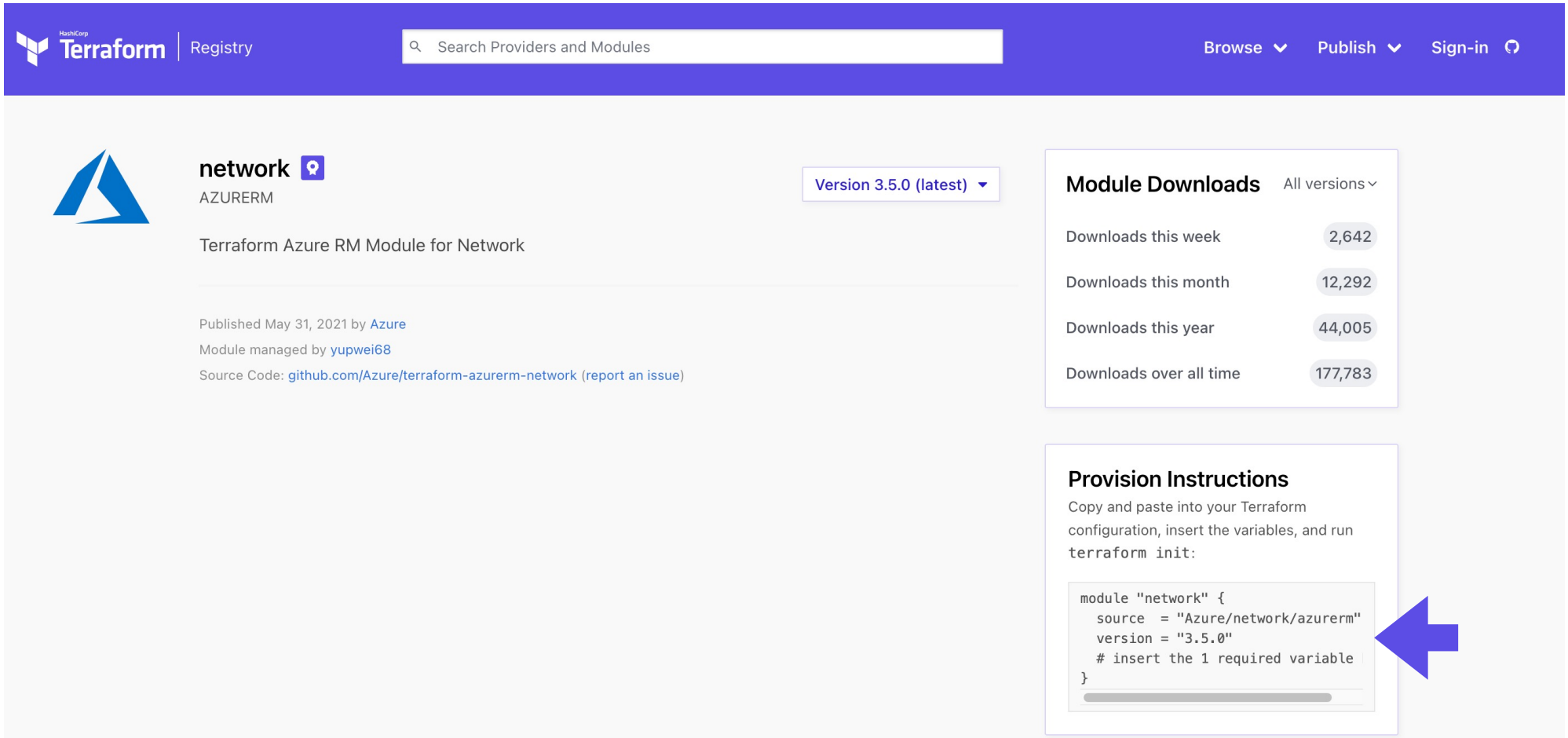
Outputs defined in module become attributes from module definition

Configuration inside a module is a "black box" from the outside

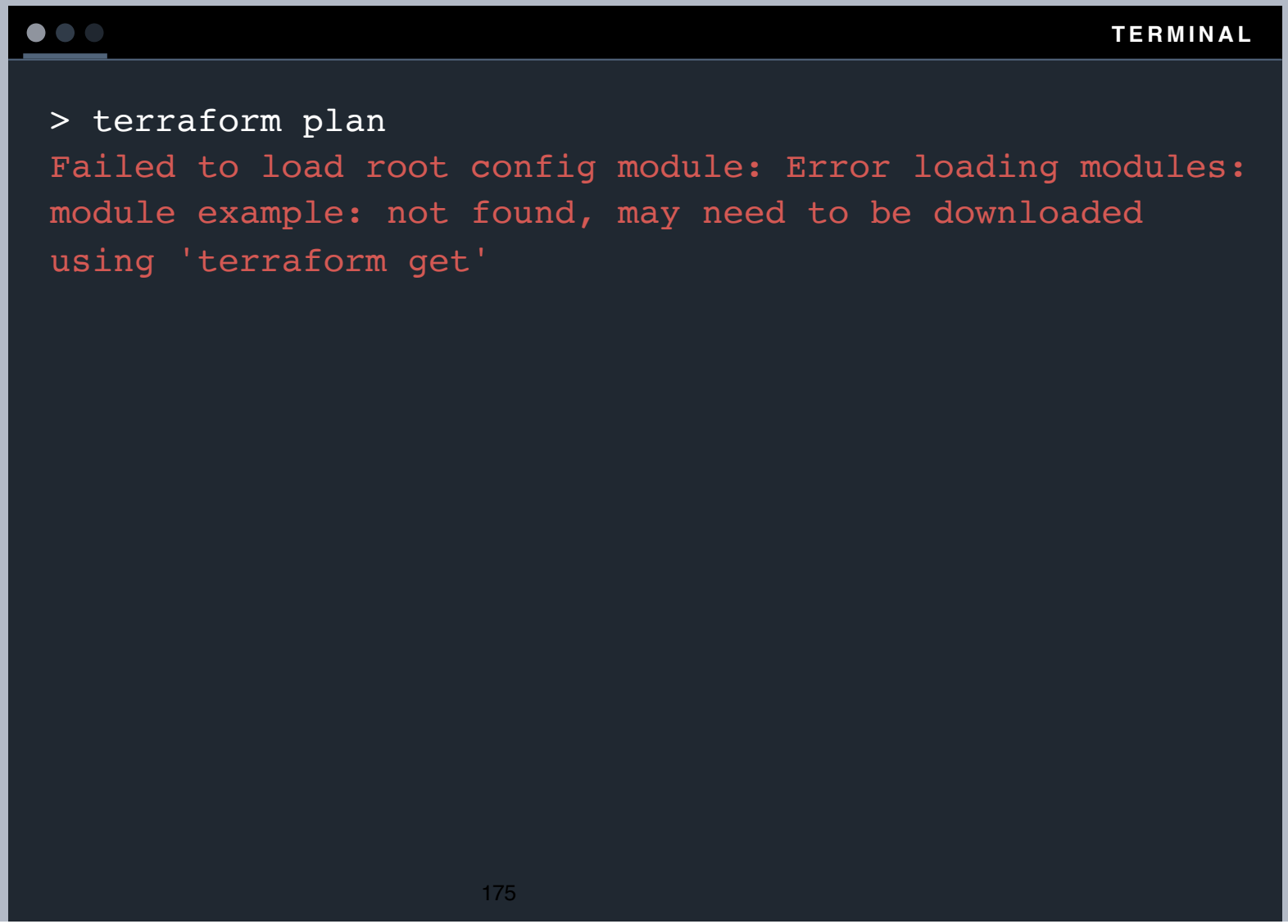
Individual resources are not accessible outside the module

```
module "my-module" {
  # Source can be any URL or file path
  source = "../../my-module"

  argument_1 = "value"
  argument_2 = "value"
}

output "example" {
  value = module.my-module.azure_network.web.public_ip
}
```


Terraform Module Registry
Geoffrey
Secure https://registry.terraform.io
Terraform | Module Registry
Browse
Publish
topfunky
Terraform Module Registry
Discover modules for common infrastructure configurations for any provider
consul
Popular searches: vault, aws, database
Find Terraform Modules
Use and learn from verified and community modules


Terraform | Registry
Search Providers and Modules
Browse
Publish
Sign-in
network
AZURERM
Version 3.5.0 (latest)
Terraform Azure RM Module for Network
Published May 31, 2021 by Azure
Module managed by yupwei68
Source Code: github.com/Azure/terraform-azurerm-network (report an issue)
Module Downloads
All versions
Downloads this week
2,642
Downloads this month
12,292
Downloads this year
44,005
Downloads over all time
177,783
Provision Instructions
Copy and paste into your Terraform configuration, insert the variables, and run terraform init:
module "network" {
source = "Azure/network/azurerm"
version = "3.5.0"
# insert the 1 required variable
}

```
module "network" {
  source              = "Azure/network/azurerm"
  version             = ”3.5.0"
  resource_group_name = azurerm_resource_group.name
}
```


TERMINAL
> terraform plan
Failed to load root config module: Error loading modules:
module example: not found, may need to be downloaded
using 'terraform get'

TERMINAL

```
> terraform plan

Error: Could not satisfy plugin requirements

# ...

Error: provider.tls: no suitable version installed
  version requirements: "(any version)"
  versions installed: none


Error: provider.local: no suitable version installed
  version requirements: "(any version)"
  versions installed: none
```

TERMINAL

```
> terraform init
Initializing modules...

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "null" (terraform-
providers/null) 2.1.2...
- Downloading plugin for provider "tls" (terraform-
providers/tls) 2.0.1...
- Downloading plugin for provider "local" (terraform-
providers/local) 1.3.0...

# ...
```

TIP

Module References Are Stored in `.terraform`

A hidden directory is built at the root of your project

You can use the `tree` command or `ls -l` to see the contents

Note that local modules are symlinked. This explains why changes to module code are available immediately

TERMINAL

```
> tree .terraform

.terraform/
├── modules
│   ├── 56e6098163e5b2675d14437a2a723a54
│   │   └── terraform-azurerm-network-37d94b3
│   │       ├── main.tf
│   │       ├── outputs.tf
│   │       ├── README.md
│   │       └── variables.tf
│   └── modules.json
└── plugins
    └── linux_amd64
        ├── lock.json
        ├── terraform-provider-azurerm_v1.23.0_x4
        ├── terraform-provider-local_v1.1.0_x4
        ├── terraform-provider-null_v1.0.0_x4
        └── terraform-provider-tls_v1.1.0_x4

4 directories, 10 files
```

TERMINAL

```
> terraform apply
azurerm_resource_group.training: Refreshing state... [id=i-
03724b60e0ff853bc]

# ...

  # module.network.azurerm_network_interface.training will be created
  + resource "azurerm_resource_group" "training" {
      + id                  = (known after apply)
      + resource_group_name = "myResourceGroup"
}

# ...

Plan: 5 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

What You Learned

- Import a module from the Terraform Public Module Registry
- Use a module from `main.tf`
- Understand how modules are stored on disk

EXERCISE

Lab 11: Modules

Duration: 30 minutes

Goal: Refactor your existing code into a separate module and connect a module from the Public Registry.

Destroy

What You'll Learn

- Destroy infrastructure that Terraform has created

Command: `terraform destroy`

Destroys running infrastructure

Does not touch infrastructure not managed by Terraform

TERMINAL

```
> terraform destroy
azurerm_resource_group.training: Refreshing state...
[id=i-0ec4ffe55abd300a5]

An execution plan has been generated and is shown below.
Resource actions are indicated with the following
symbols:
  - destroy

Terraform will perform the following actions:

  # azurerm_resource_group.training will be destroyed
  - resource "azurerm_resource_group "training" {
# ...
```

EXERCISE

Lab 10: Destroy

Duration: 5 minutes

Goal: Use the destroy command to destroy your infrastructure

Course Agenda

Terraform Fundamentals

Students will walk away with a solid understanding of HashiCorp Terraform.

Course Agenda:

- Day 1 Recap
- Data Sources
- Terraform State
- Examining Terraform State
- Import
- Terraform Cloud Basics

Course Agenda

Terraform Fundamentals

Students will walk away with a solid understanding of HashiCorp Terraform.

Course Agenda:

- Lifecycles
- Template Files and Rendering
- Debug
- Certification Prep

Azure Authentication

Authentication to Azure

Azure CLI

az login

- Uses your own credentials
- Great for local development
- Also how the Azure Cloud Shell works

Authentication to Azure

Service Principal

- Preferred for shared environments
- Works well in automation
- Also how Terraform Enterprise works

Authentication to Azure

Service Principal

```
provider "azurerm" {

  subscription_id = "SUBSCRIPTION-ID"
  client_id       = "CLIENT-ID"
  client_secret   = "CLIENT-SECRET"
  tenant_id       = "TENANT-ID"
}
```


Authentication to Azure

Service Principal

```
export ARM_TENANT_ID=
export ARM_SUBSCRIPTION_ID=
export ARM_CLIENT_ID=
export ARM_CLIENT_SECRET=
```


Authentication to Azure

Managed Service Identity (MSI)

<no configuration needed>

Azure Location

```
resource "azurerm_resource_group” "main" {
  location = "eastus"
}
```

```
resource "azurerm_resource_group" "main" {
  location = "East US"
}
```


Data Sources

Data Sources

Data sources let you query information defined outside of Terraform.

Similar to a resource, data sources are part of a provider.

Data sources are read-only and refreshed during a plan or apply.

MAIN.TF

```
data "azurerm_subnet" "web_subnet" {
  name                 = "web"
  virtual_network_name = "data-source-network"
  resource_group_name  = "data-source-network"
}

# ...

resource "azurerm_network_interface" "web" {
  name                = "web-nic"
  location            = "eastus"
  resource_group_name = data.azurerm_subnet.web_subnet.resource_gr
oup_name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = data.azurerm_subnet.web_subnet
.id
    private_ip_address_allocation = "Dynamic"
  }
}
```

EXERCISE

Lab 2: Data Sources

Duration: 10 minutes

Goal: Use a data source for the subnet ID of a network interface

State

What You'll Learn

- Understand how state is managed
- Understand where state can be stored
- Understand Terraform Cloud and Terraform Enterprise

State

Terraform stores the state of your managed infrastructure from the last time Terraform was run.

Terraform uses this state to create plans and make changes to your infrastructure.

It is critical that this state is maintained appropriately so future runs operate as expected.

TERMINAL

```
> head terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.0.7",
  "serial": 3,
  "lineage": "067aac2b-fc72-0bad-92a4-bd48547e28e5",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "azurerm_resource_group",
```

TERMINAL

```
> terraform state list

azurerm_resource_group.data_source
azurerm_virtual_network.data_source
```

State Resolutions

Configuration	State	Reality	**Operation**
azurerm_virtual_machine.web			
azurerm_virtual_machine.web	azurerm_virtual_machine.web		
azurerm_virtual_machine.web	azurerm_virtual_machine.web	azurerm_virtual_machine.web	
	azurerm_virtual_machine.web	azurerm_virtual_machine.web	
		azurerm_virtual_machine.web	
azurerm_virtual_machine.web		azurerm_virtual_machine.web	
	azurerm_virtual_machine.web		

State Resolutions

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

State Locking

If supported, the state backend will "lock" to prevent concurrent modifications which could cause corruption.

Not all backends support locking - Terraform's documentation identifies which backends support this functionality (Terraform Enterprise, Terraform Cloud, S3, GCS, Azure Storage do)

Where is State?

Local State (default)

Stored locally in a JSON format. (`terraform.tfstate`)

Remote State

Stored on a remote source (Terraform Cloud, Terraform Enterprise, S3, GCS, Azure Storage, etc).

Local State

State is stored locally on one machine

It is generally acceptable for individuals and small teams

No good way to back up or share state

Tends not to scale for larger teams

Requires a more "mono repo" pattern

Remote State

State is on a remote source like S3, Terraform Cloud, Terraform Enterprise, or Consul

Remote storage is responsible for handling merging and locking

Can be queried for information in other Terraform configurations

Removes the risk of losing state to a hard drive crash or other data loss

Sensitive Data in State

State can contain sensitive data depending on the resources used

Sometimes it can contain initial database passwords or other secret data returned by a provider

Some resources support PGP encrypting the values in the state, but this is implemented on a per-resource basis

We recommend never keeping state in source control, for example

Sensitive Data in State

Local state (JSON) is not encrypted

Remote state encryption is backend-specific

State is only held in memory when remote state is used

Example: S3 bucket can be encrypted + IAM + TLS connection

Example: TFE encrypted in transit and rest + full audit log

State as a Data Source

Terraform state data can be a data source for another configuration using `terraform_remote_state`

Only outputs are available from the remote state data

Read access to state data is required

MAIN.TF

```
data "terraform_remote_state" "network" {
  backend    = "local"
  config = {
    path = "..."
  }
}
```

EXERCISE

Lab 3: State

Duration: 10 minutes

Goal: Read state from another Terraform project.

State Import

Terraform only manages previous or imported resources.

Terraform can `import` state for existing resources. If you have already spun up an instance with the web console, you may need to import it so Terraform knows about it.

This only writes the state file, not the Terraform configuration.

State Import

Critical Thinking

After importing an existing resource, what operation(s) would the `terraform plan` show?

State Import

Critical Thinking

After importing an existing resource, what operation(s) would the `terraform plan` show?

Answer

Destroy (state exists, no configuration)

EXERCISE

Lab 4: Import

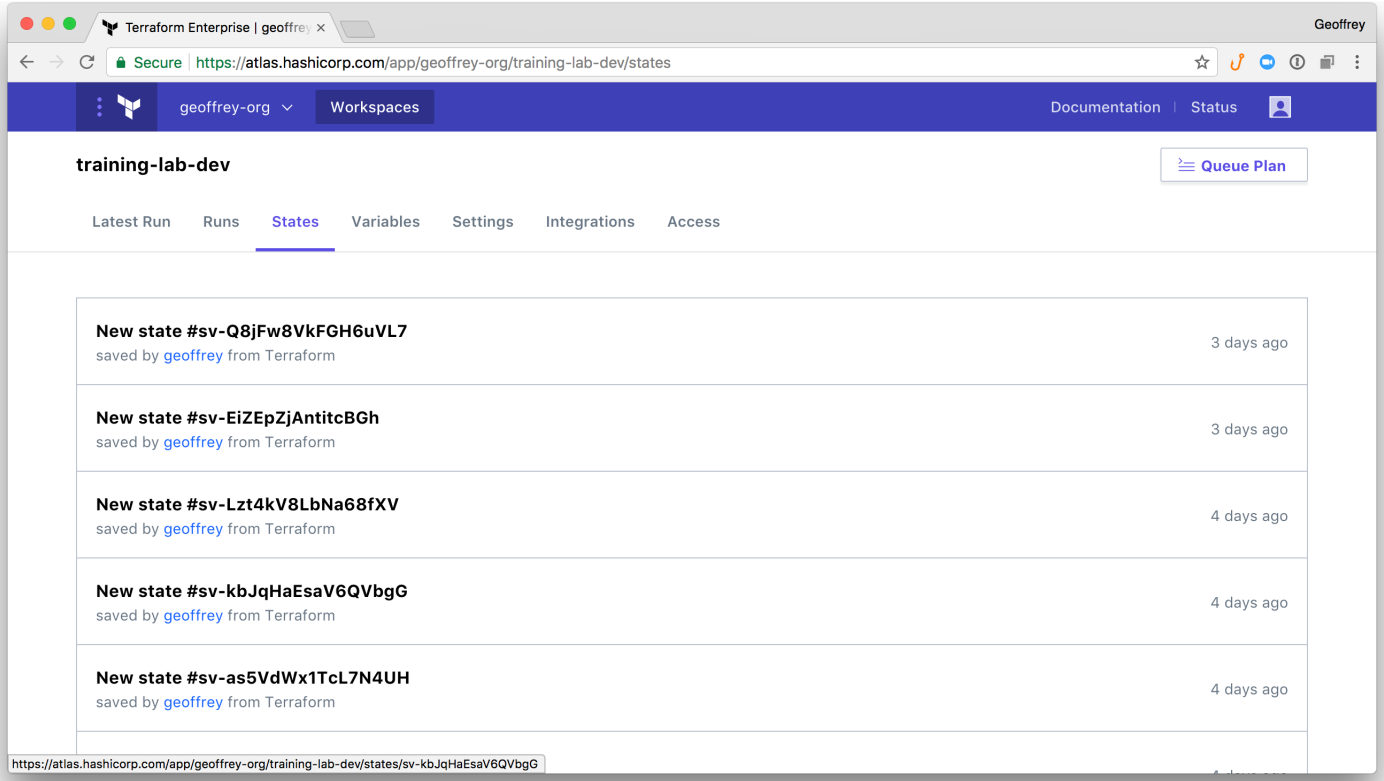
Duration: 10 minutes

Goal: Use terraform import to manage existing resources.

Terraform Cloud

Terraform Ecosystem

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


Terraform Enterprise | geoffrey
Geoffrey
Secure
https://atlas.hashicorp.com/app/geoffrey-org/training-lab-dev/states
geoffrey-org
Workspaces
Documentation
Status
training-lab-dev
Queue Plan
Latest Run
Runs
States
Variables
Settings
Integrations
Access
New state #sv-Q8jFw8VkFGH6uVL7
saved by geoffrey from Terraform
3 days ago
New state #sv-EiZEpZjAntitcBGh
saved by geoffrey from Terraform
3 days ago
New state #sv-Lzt4kV8LbNa68fXV
saved by geoffrey from Terraform
4 days ago
New state #sv-kbJqHaEsaV6QVbgG
saved by geoffrey from Terraform
4 days ago
New state #sv-as5VdWx1TcL7N4UH
saved by geoffrey from Terraform
4 days ago
https://atlas.hashicorp.com/app/geoffrey-org/training-lab-dev/states/sv-kbJqHaEsaV6QVbgG

Transitioning from Local to Remote

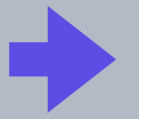
Transitioning is a one-time operation

After configured, Terraform will no longer store local state

The `terraform` stanza declares the configuration alongside resources

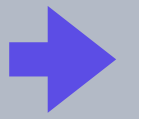
```
terraform {
  cloud {
    organization = "example-company"

# Single workspace:
    workspaces {
      name = "my-app-prod"
    }

# Or use multiple workspaces:
#    workspaces {
#      tags = ["app1"]
#    }
  }
}
```

```
data "terraform_remote_state" "vpc" {
  backend = "remote"

  config {
    organization = "example-org"
    hostname = "app.terraform.io"
  }

  workspaces {
    name = "example-workspace"
  }
}

resource “azurerm_network_interface" "example" {
  # ...
  subnet_id = data.terraform_remote_state.web.outputs.subnet_id
}
```



~/.TERRAFORMRC

```
credentials "app.terraform.io" {
  token = "<TOKEN>"
}
```



```
data "terraform_remote_state" "vpc" {
  backend = "remote"

  config {
    organization = "example-org"
    hostname = "app.terraform.io"
    access_token = "xxxxxx"
  }
}

resource “azurerm_network_interface" "example" {
  # ...
  subnet_id = data.terraform_remote_state.web.outputs.subnet_id
}
```

Backend Initialization

Remote State backends must be initialized before use

Initialization prepares the remote backend for storage

Initialization will prompt for any missing values and cache them locally in `.terraform` (which should be ignored from source)

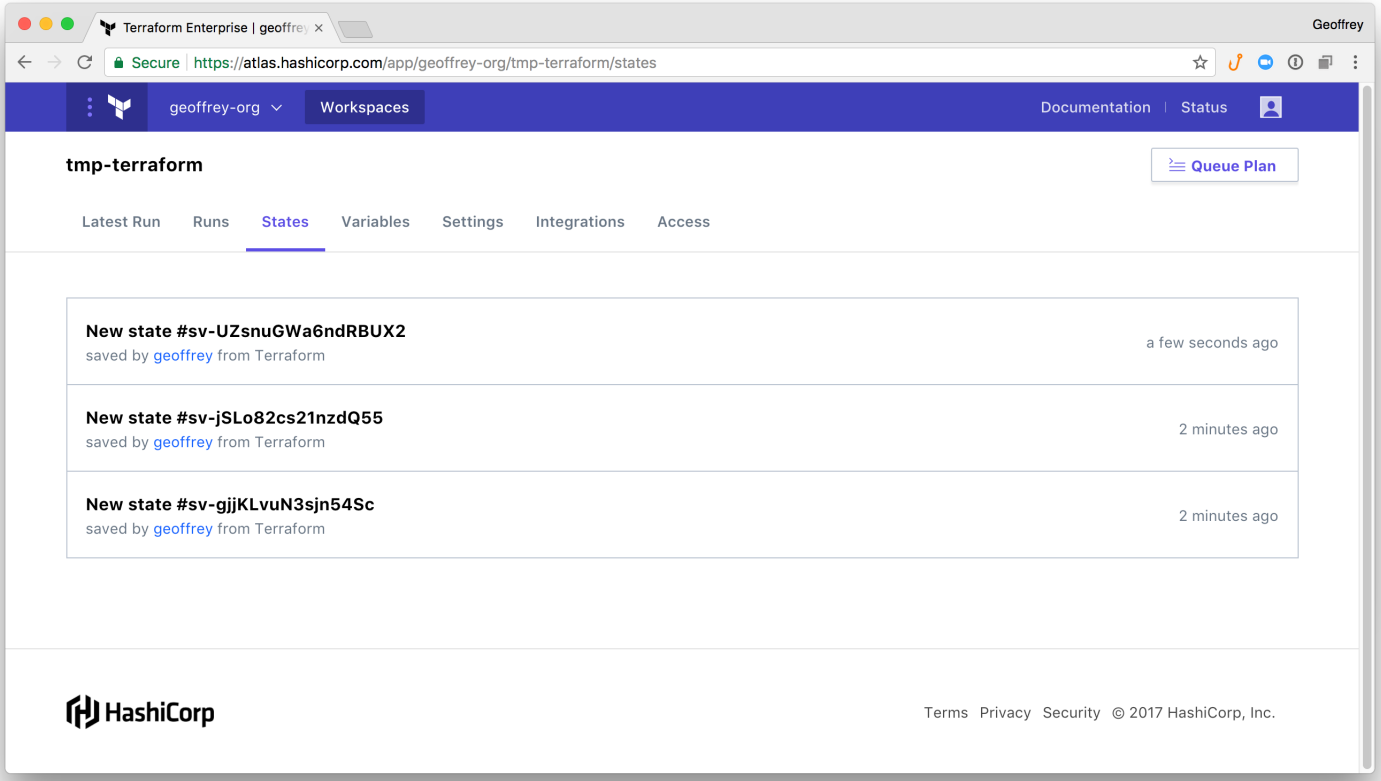
Initialization will also optionally migrate state

TERMINAL

```
> terraform init

Initializing the backend...
Acquiring state lock. This may take a few moments...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the
previous "local" backend to the newly configured "remote"
backend. No existing state was found in the newly
configured "remote" backend.
  Do you want to copy this state to the new "remote"
backend? Enter "yes" to copy and "no" to start with an
empty state.

  Enter a value:
```

TERMINAL

```
Successfully configured the backend "remote"! Terraform
will automatically use this backend unless the backend
configuration changes.
```


Terraform Enterprise | geoffrey
Geoffrey
Secure https://atlas.hashicorp.com/app/geoffrey-org/tmp-terraform/states
geoffrey-org
Workspaces
Documentation
Status
tmp-terraform
Queue Plan
Latest Run
Runs
States
Variables
Settings
Integrations
Access
New state #sv-UZsnuGWa6ndRBUX2
saved by geoffrey from Terraform
a few seconds ago
New state #sv-jSLo82cs21nzdQ55
saved by geoffrey from Terraform
2 minutes ago
New state #sv-gjjKLvuN3sjn54Sc
saved by geoffrey from Terraform
2 minutes ago
HashiCorp
Terms Privacy Security © 2017 HashiCorp, Inc.

Remote State

Terraform will no longer write to the `.tfstate` file

Everything else about your local workflow remains the same

TERMINAL

```
> terraform apply
azurerm_resource_group.web: Refreshing state...

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

EXERCISE

Lab 5: Terraform Cloud Remote State

Duration: 20 minutes

Goal: Use Terraform Cloud to store your state data

EXERCISE

Lab 6: Secure Variables

Duration: 20 minutes

Goal: Securely migrate and store variables to Terraform Cloud

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
```

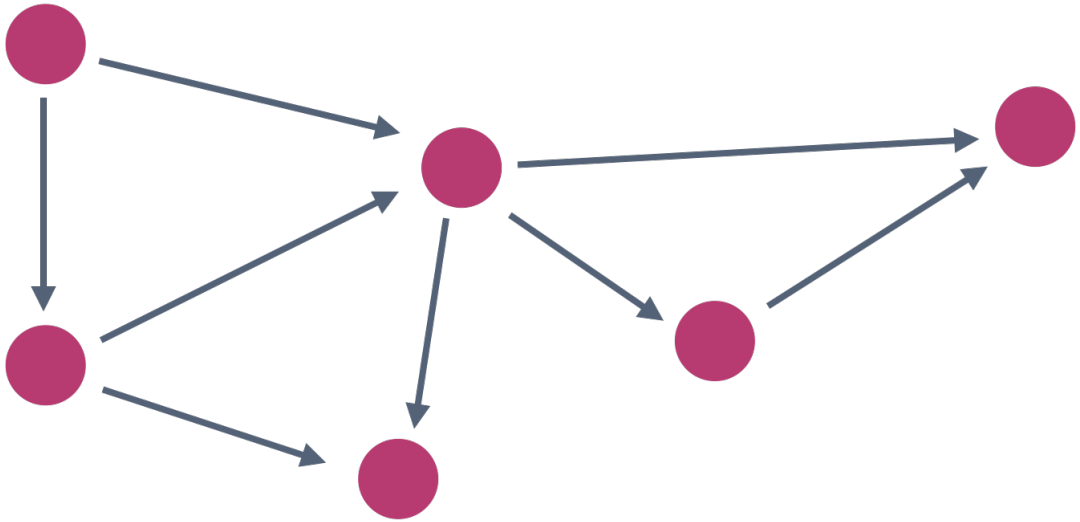
The `lifecycle` block

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.
```

EXERCISE

Lab 7: Lifecycles

Duration: 10 minutes

Goal: Demonstrate how to apply the lifecycle directives that we've discussed so far

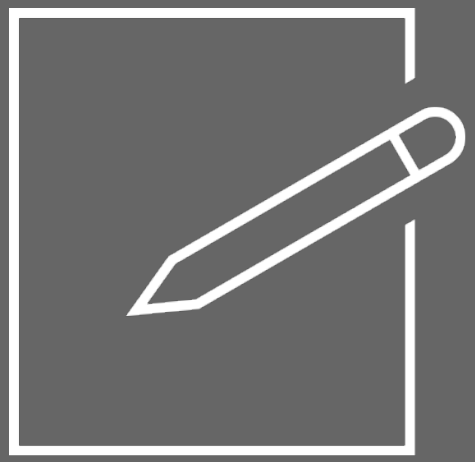
Template Files

What You'll Learn

- Use the templatefile function
- Understand real world scenarios for using templates

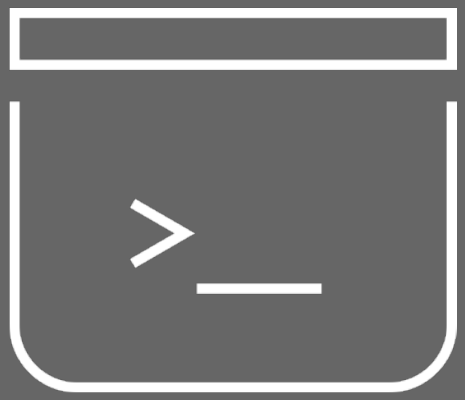
What is a Template File in Terraform?

- 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



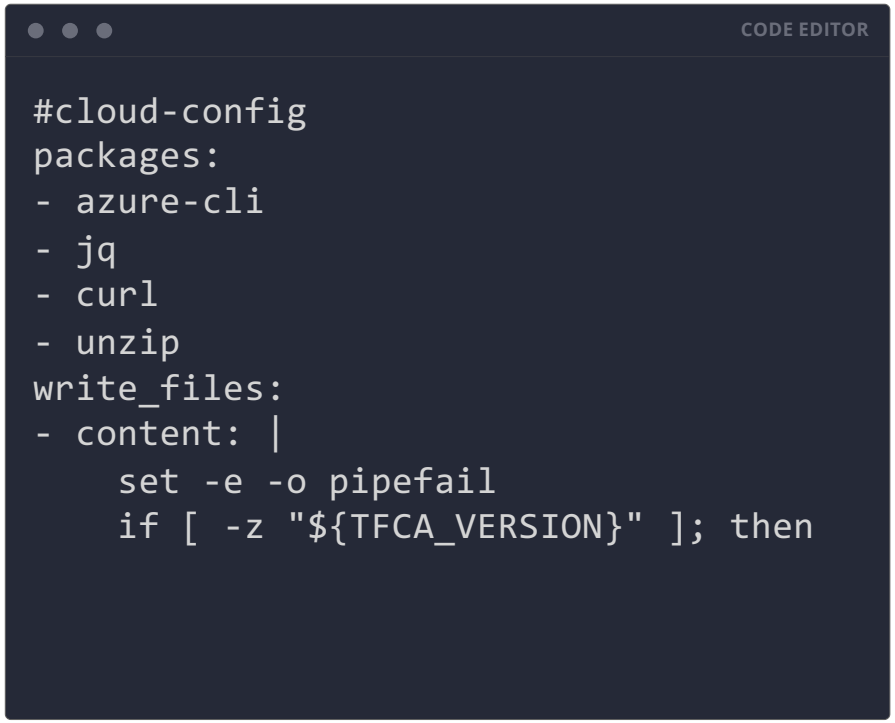
Plain Text
Template

Variables from Cloud or
Other Resources



Render with
Terraform

Creating a Template File



Interpolation in Template Files

We can replace values in the template for more dynamic rendering

CODE EDITOR

```
runcmd:
- ["/bin/bash", "-c",
"TFCA_VERSION='${tfca_version}'
/opt/tfca-install/tfca-download.sh"]
```

CODE EDITOR

```
dashboard_properties = templatefile("${path.module}/templates/dash.tpl",
   {
     md_content = "Variable content here!",
     video_link = "https://www.youtube.com/watch?v=4Ttjci5KPQo",
     sub_id     = data.azurerm_subscription.current.subscription_id
 })
```

Real world scenarios

Pass custom scripts to a VM's custom_data

Create a custom Azure RBAC role with

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 Custom Data startup script

CODE EDITOR

```
resource "azurerm_linux_virtual_machine" "web" {
  name = "my-vm"

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

EXERCISE

Lab 8: Template Files

Duration: 10 minutes

Goal: Define and deploy a template file within a Terraform configuration

Debug

EXERCISE

Lab 9: Debug

Duration: 10 minutes

Goal: Set Terraform configuration to display different levels of debugging output.

Terraform Associate Certifcation Course

Q & A
General Discussion