**Using Terraform to Manage Applications and Infrastructure**

1. **Introduction**
2. **About This Course(cont.)**

**Course Topics:**

* **Introduction**
* **About Terraform**
* **Deploying infrastructure with Terraform**
* **Terraform Basics**
* **Terraform Modules**
* **Terraform and Docker**
* **Using Terraform in a CI/CD Environment**
* **Terraform and AWS**
* **Troubleshooting**
* **Terraform State**
* **Terraform and Kubernetes**
* **Terraform 0.12 Beta**

**What is Terraform?**

* **Terraform is a tool for building infrastructure.**
* **Terraform is a tool for building, changing and versioning infrastructure safely and effectively.**
* **Terraform manages all this through state files which have to go version control.**
* **Allows simple version control**
* **Available as open-source or enterprise software**
* **Enterprise provides advanced collaboration and governance**
* **Supports many popular service providers such as**
* **AWS**
* **OpenStack**
* **Azure**
* **GCP**
* **Kubernetes**

**Terraform Features**

* **Primary Terraform features:**
* **Infrastructure as Code (IaC)**
* **Idempotent: you can apply your scripts multiple times but the result will be same if the configuration already in place**
* **Ex: If you go and create s3 bucket, you can go and run your script again the buckets already there it can’t do anything.**
* **High-level syntax**
* **Easily reusable code**
* **Execution Planes**
* **Show the intent of the deploy**
* **Can help ensure everything in the development is intentional**
* **Resource graph**
* **Illustrates all changes and dependencies**

## Install Terraform

Install Terraform 0.11.13 on the Swarm manager:

sudo curl -O https://releases.hashicorp.com/terraform/0.11.13/terraform\_0.11.13\_linux\_amd64.zip

sudo yum install -y unzip

sudo unzip terraform\_0.11.13\_linux\_amd64.zip -d /usr/local/bin/

Test the Terraform installation:

terraform version

### Deploying Infrastructure with Terraform

## Terraform Commands

In this lesson, we begin working with Terraform commands. We will start by creating a very simple Terraform file that will pull down the an image from Docker Hub.

List the Terraform commands:

terraform

**Common commands:**

apply: Builds or changes infrastructure  
console: Interactive console for Terraform interpolations  
destroy: Destroys Terraform-managed infrastructure  
fmt: Rewrites configuration files to canonical format  
get: Downloads and installs modules for the configuration  
graph: Creates a visual graph of Terraform resources  
import: Imports existing infrastructure into Terraform  
init: Initializes a new or existing Terraform configuration  
output: Reads an output from a state file  
plan: Generates and shows an execution plan  
providers: Prints a tree of the providers used in the configuration  
push: Uploads this Terraform module to Terraform Enterprise to run  
refresh: Updates local state file against real resources  
show: Inspects Terraform state or plan  
taint: Manually marks a resource for recreation  
untaint: Manually unmarks a resource as tainted  
validate: Validates the Terraform files  
version: Prints the Terraform version  
workspace: Workspace management

Set up the environment:

mkdir -p terraform/basics

cd terraform/basics

Create a Terraform script:

vi main.tf

main.tf contents:

# Download the latest Ghost image

resource "docker\_image" "image\_id" {

name = "ghost:latest"

}

Initialize Terraform:

Init: Initializes a new or existing Terraform configuration

Provider is responsible for understanding api interaction and exposing resources

The init knows to install the docker provider

terraform init

Validate the Terraform file:

terraform validate

**$ls –la**

**.terraform in this folder plugins got installed**

List providers in the folder:

ls .terraform/plugins/linux\_amd64/

List providers used in the configuration:

# Docker Provider

The Docker provider is used to interact with Docker containers and images. It uses the Docker API to manage the lifecycle of Docker containers. Because the Docker provider uses the Docker API, it is immediately compatible not only with single server Docker but Swarm and any additional Docker-compatible API hosts.

terraform providers

**$ ls .terraform/plugins/linux\_amd64/**

**$terraform providers**

**Provider.docker got installed**

Terraform Plan:

terraform plan

**this command is used to go create and execution plan**

**Terraform will perform the following actions: it will download the required software’s and installed**

Useful flags for plan:  
-out=path: Writes a plan file to the given path. This can be used as input to the "apply" command.  
-var 'foo=bar': Set a variable in the Terraform configuration. This flag can be set multiple times.

Terraform Apply:

terraform apply

Useful flags for apply:  
-auto-approve: This skips interactive approval of plan before applying.  
-var 'foo=bar': This sets a variable in the Terraform configuration. It can be set multiple times.

Confirm your apply by typing **yes**. The apply will take a bit to complete

**This command is used to go and apply to changes require to desired state of our configuration.**

List the Docker images:

docker image ls

Terraform Show: **Inspect current Terraform state/plan**

terraform show

Terraform Destroy: destroy Terraform infrastructure

terraform destroy

Confirm your destroy by typing **yes**.

Useful flags for destroys:  
-auto-approve: Skip interactive approval of plan before applying.

Re-list the Docker images:

docker image ls

Using a plan:

terraform plan -out=tfplan

$ls – we can see the Terraform tfplan file

Applying a plan:

terraform apply tfplan

Show the Docker Image resource:

terraform show

Destroy the resource once again:

terraform destroy

## HashiCorp Configuration Language

Resources are a component of your infrastructure. It might be some low level component such as a physical server, virtual machine, or container. Or it can be a higher level component such as an email provider, DNS record, or database provider

**Description**

The resource block creates a resource of the given TYPE (first parameter) and NAME (second parameter). The combination of the type and name must be unique.

Example:

resource "aws\_instance" "web" {

ami **=** "ami-408c7f28"

instance\_type **=** "t1.micro"

}

## Syntax reference:

* Single line comments start with #.
* Multi-line comments are wrapped with /\* and \*/.
* Values are assigned with the syntax of key = value.
* Strings are in double-quotes.
* Strings can interpolate other values using syntax wrapped in ${}, for example ${var.foo}.
* Numbers are assumed to be base 10.
* Boolean values: true, false
* Lists of primitive types can be made with square brackets ([]), for example ["foo", "bar", "baz"].
* Maps can be made with braces ({}) and colons (:), for example { "foo": "bar", "bar": "baz" }.

## Style Conventions:

* Indent two spaces for each nesting level.
* With multiple arguments, align their equals signs.

Setup the environment:

cd terraform/basics

The interpolation syntax is powerful and allows you to reference variables, attributes of resources, call functions, etc.

vi main.tf

main.tf contents:

# Download the latest Ghost image

resource "docker\_image" "image\_id" {

name = "ghost:latest"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "ghost\_blog"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "2368"

external = "80"

}

}

Validate main.tf:

terraform validate

Terraform Plan:

terraform plan

Apply the changes to main.tf:

terraform apply

Confirm the apply by typing **yes**.

List the Docker containers:

docker container ls

## Cleaning up the environment

Reset the environment:

terraform destroy

Confirm the destroy by typing **yes**.

## Tainting and Updating Resources

## Tainting and Untainting Resources

Terraform commands:  
taint: Manually mark a resource for recreation untaint: Manually unmark a resource as tainted

Tainting a resource:

terraform taint [NAME]

Untainting a resource:

terraform untaint [NAME]

Set up the environment:

cd terraform/basics

Redeploy the Ghost image:

terraform apply

Taint the Ghost blog resource:

terraform taint docker\_container.container\_id

See what will be changed:

terraform plan

Remove the taint on the Ghost blog resource:

terraform untaint docker\_container.container\_id

Verity that the Ghost blog resource is untainted:

terraform plan

## Updating Resources

Let's edit main.tf and change the image to ghost:alpine.

Open main.tf:

vi main.tf

main.tf contents:

# Download the latest Ghost image

resource "docker\_image" "image\_id" {

name = "ghost:alpine"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "ghost\_blog"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "2368"

external = "80"

}

}

Validate changes made to main.tf:

terraform validate

See what changes will be applied:

terraform plan

Apply image changes:

terraform apply

List the Docker containers:

docker container ls

See what image Ghost is using:

docker image ls | grep [IMAGE]

Check again to see what changes will be applied:

terraform plan

Apply container changes:

terraform apply

See what image Ghost is now using:

docker image ls | grep [IMAGE]

## Cleaning up the environment

Reset the environment:

terraform destroy

Confirm the destroy by typing **yes**.

List the Docker images:

docker image ls

Remove the Ghost blog image:

docker image rm ghost:latest

Reset main.tf:

vi main.tf

main.tf contents:

# Download the latest Ghost image

resource "docker\_image" "image\_id" {

name = "ghost:latest"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "ghost\_blog"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "2368"

external = "80"

}

}

## Terraform Console and Output

In this lesson, we will use the Terraform Console to view various outputs that we can use for our scripts. The Terraform Console is extremely useful for troubleshooting and planning deployments.

Terraform commands:  
console: Interactive console for Terraform interpolations

Set up the environment:

cd terraform/basics

**$ls**

**There is files**

**Main.tf fle and all**

## Working with the Terraform console

Redeploy the Ghost image and container:

terraform apply

Show the Terraform resources:

terraform show

Start the Terraform console:

terraform console

Type the following in the console to get the container's name:

docker\_container.container\_id.name

Type the following in the console to get the container's IP:

docker\_container.container\_id.ip\_address

Break out of the Terraform console by using **Ctrl+C**. go back to the terminal.

Destroy the environment:

terraform destroy

## Output the name and IP of the Ghost blog container

Edit main.tf:

vi main.tf

main.tf contents:

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "ghost:latest"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "blog"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "2368"

external = "80"

}

}

#Output the IP Address of the Container

output "ip\_address" {

value = "${docker\_container.container\_id.ip\_address}"

description = "The IP for the container."

}

#Output the Name of the Container

output "container\_name" {

value = "${docker\_container.container\_id.name}"

description = "The name of the container."

}

Validate changes:

terraform validate

Apply changes to get output:

terraform apply

## Cleaning up the environment

Reset the environment:

terraform destroy

## Input Variables

Input variables serve as parameters for a Terraform file. A variable block configures a single input variable for a Terraform module. Each block declares a single variable.

Syntax:

variable [NAME] {

[OPTION] = "[VALUE]"

}

## Arguments

Within the block body (between { }) is configuration for the variable, which accepts the following arguments:

* type (Optional): If set, this defines the type of the variable. Valid values are string, list, and map.
* default (Optional): This sets a default value for the variable. If no default is provided, Terraform will raise an error if a value is not provided by the caller.
* description (Optional): A human-friendly description for the variable.

Using variables during an apply:

terraform apply -var 'foo=bar'

Set up the environment:

cd terraform/basics

Edit main.tf:

vi main.tf

main.tf contents:

#Define variables

variable "image\_name" {

description = "Image for container."

default = "ghost:latest"

}

variable "container\_name" {

description = "Name of the container."

default = "blog"

}

variable "int\_port" {

description = "Internal port for container."

default = "2368"

}

variable "ext\_port" {

description = "External port for container."

default = "80"

}

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "${var.image\_name}"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "${var.container\_name}"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "${var.int\_port}"

external = "${var.ext\_port}"

}

}

#Output the IP Address of the Container

output "IP Address" {

value = "${docker\_container.container\_id.ip\_address}"

description = "The IP for the container."

}

output "container\_name" {

value = "${docker\_container.container\_id.name}"

description = "The name of the container."

}

Validate the changes:

terraform validate

Plan the changes:

terraform plan

Apply the changes using a variable:

terraform apply -var 'ext\_port=8080'

Change the container name:

terraform apply -var 'container\_name=ghost\_blog' -var 'ext\_port=8080'

Reset the environment:

terraform destroy -var 'ext\_port=8080'

## Breaking Out Our Variables and Outputs

Setup your environment:

cd terraform/basics

Edit variables.tf:

vi variables.tf

variables.tf contents:

#Define variables

variable "container\_name" {

description = "Name of the container."

default = "blog"

}

variable "image\_name" {

description = "Image for container."

default = "ghost:latest"

}

variable "int\_port" {

description = "Internal port for container."

default = "2368"

}

variable "ext\_port" {

description = "External port for container."

default = "80"

}

Edit main.tf:

vi main.tf

main.tf contents:

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "${var.image\_name}"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "${var.container\_name}"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "${var.int\_port}"

external = "${var.ext\_port}"

}

}

Edit outputs.tf:

vi outputs.tf

outputs.tf contents:

#Output the IP Address of the Container

output "IP Address" {

value = "${docker\_container.container\_id.ip\_address}"

description = "The IP for the container."

}

output "container\_name" {

value = "${docker\_container.container\_id.name}"

description = "The name of the container."

}

Validate the changes:

terraform validate

Plan the changes:

terraform plan -out=tfplan -var container\_name=ghost\_blog

Apply the changes:

terraform apply tfplan

Destroy deployment:

terraform destroy -auto-approve -var container\_name=ghost\_blog

## Maps and Lookups

In this lesson, we will create a map to specify different environment variables based on conditions. This will allow us to dynamically deploy infrastructure configurations based on information we pass to the deployment.

Set up the environment:

cd terraform/basics

Edit variables.tf:

vi variables.tf

variables.tf contents:

#Define variables

variable "env" {

description = "env: dev or prod"

}

variable "image\_name" {

type = "map"

description = "Image for container."

default = {

dev = "ghost:latest"

prod = "ghost:alpine"

}

}

variable "container\_name" {

type = "map"

description = "Name of the container."

default = {

dev = "blog\_dev"

prod = "blog\_prod"

}

}

variable "int\_port" {

description = "Internal port for container."

default = "2368"

}

variable "ext\_port" {

type = "map"

description = "External port for container."

default = {

dev = "8081"

prod = "80"

}

}

Validate the change:

terraform validate

Edit main.tf:

vi main.tf

main.tf contents:

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "${lookup(var.image\_name, var.env)}"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "${lookup(var.container\_name, var.env)}"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "${var.int\_port}"

external = "${lookup(var.ext\_port, var.env)}"

}

}

Plan the dev deploy:

terraform plan -out=tfdev\_plan -var env=dev

Apply the dev plan:

terraform apply tfdev\_plan

Plan the prod deploy:

terraform plan -out=tfprod\_plan -var env=prod

Apply the prod plan:

terraform apply tfprod\_plan

Destroy prod deployment:

terraform destroy -var env=prod -auto-approve

Use environment variables:

export TF\_VAR\_env=prod

Open the Terraform console:

terraform console

Execute a lookup:

lookup(var.ext\_port, var.env)

Exit the console:

unset TF\_VAR\_env

## Terraform Workspaces

In this lesson, we will see how workspaces can help us deploy multiple environments. By using workspaces, we can deploy multiple environments simultaneously without the state files colliding.

cd terraform/basics

$ls

$ cat Terraform.tfstate

$ Terraform plan –out=tfdev\_plan –var env=dev

$terraform apply tfdev\_plan

$cat terraform.tfstate

All the information about the infrastructure

$docker container ls

$docker container rm blog\_dev –f

$terraform show

$terraform plan –out=tfdev\_plan –var env=dev

$terraform apply tfdev\_plan

$ Terraform destroy –var env=prod

**Over writing that’s why workspace comes in.**

## Creating a workspace

Terraform commands:

workspace: New, list, select and delete Terraform workspaces

Workspace subcommands:

delete: Delete a workspace

list: List Workspaces

new: Create a new workspace

select: Select a workspace

show: Show the name of the current workspace

Create a dev workspace:

terraform workspace new dev

**$ Terraform workspace show**

**$terraform workspace list**

**$terraform workspace select default**

**$terraform workspace select dev**

Plan the dev deployment:

terraform plan -out=tfdev\_plan -var env=dev

Apply the dev deployment:

terraform apply tfdev\_plan

**$ls**

Will see new directory called **Terraform.tfstate.d**

**$ ls Terraform.tfstate.d**

**Dev**

**$ls Terraform.tfstate.d/dev/**

**Terraform.tfstate**

Change workspaces:

terraform workspace new prod

Plan the prod deployment:

terraform plan -out=tfprod\_plan -var env=prod

Apply the prod deployment:

terraform apply tfprod\_plan

**$ls Terraform.tfstate.d/**

**Dev prod**

Select the default workspace:

terraform workspace select default

Find what workspace we are using:

terraform workspace show

Select the dev workspace:

terraform workspace select dev

Destroy the dev deployment:

terraform destroy -var env=dev

Select the prod workspace:

terraform workspace select prod

Destroy the prod deployment:

terraform destroy -var env=prod

## Null Resources and Local-exec

In this lesson, we will utilize a *Null Resource* in order to perform local commands on our machine without having to deploy extra resources.

# local-exec Provisioner

The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource. See the remote-exec [provisioner](https://www.terraform.io/docs/provisioners/remote-exec.html) to run commands on the resource.

Note that even though the resource will be fully created when the provisioner is run, there is no guarantee that it will be in an operable state - for example system services such as sshd may not be started yet on compute resources.

**Note:** Provisioners should only be used as a last resort. For most common situations there are better alternatives. For more information, see [the main Provisioners page](https://www.terraform.io/docs/provisioners/).

## Example usage

resource "aws\_instance" "web" {

# ...

provisioner "local-exec" {

command **=** "echo ${aws\_instance.web.private\_ip} >> private\_ips.txt"

}

}

Setup the environment:

cd terraform/basics

main.tf contents:

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "${lookup(var.image\_name, var.env)}"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "${lookup(var.container\_name, var.env)}"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "${var.int\_port}"

external = "${lookup(var.ext\_port, var.env)}"

}

}

resource "null\_resource" "null\_id" {

provisioner "local-exec" {

command = "echo ${docker\_container.container\_id.name}:${docker\_container.container\_id.ip\_address} >> container.txt"

}

}

Reinitialize Terraform:

terraform init

Validate the changes:

terraform validate

Plan the changes:

terraform plan -out=tfplan -var env=dev

Apply the changes:

terraform apply tfplan

View the contents of container.txt:

cat container.txt

Destroy the deployment:

terraform destroy -auto-approve -var env=dev

## iv. Terraform Modules

we will start talking about how to use modules in Terraform.

Set up the environment:

mkdir -p modules/image

mkdir -p modules/container

Create files for the image:

cd ~/terraform/basics/module/image

touch main.tf variables.tf outputs.tf

Create files for container:

cd ~/terraform/basics/module/container

touch main.tf variables.tf outputs.tf

## The Image Module

we will create our first Terraform module.

Go to the image directory:

cd ~/terraform/docker/modules/image

Edit main.tf:

vi main.tf

main.tf contents:

# Download the Image

resource "docker\_image" "image\_id" {

name = "${var.image\_name}"

}

Edit variables.tf:

vi variables.tf

variables.tf contents:

variable "image\_name" {

description = "Name of the image"

}

Edit outputs.tf:

vi outputs.tf

outputs.tf: contents:

output "image\_out" {

value = "${docker\_image.image\_id.latest}"

}

Initialize Terraform:

terraform init

Create the image plan:

terraform plan -out=tfplan -var 'image\_name=ghost:alpine'

Deploy the image using the plan:

terraform apply -auto-approve tfplan

Destroy the image:

terraform destroy -auto-approve -var 'image\_name=ghost:alpine'

## The Container Module

we will continue working with Terraform modules by breaking out the container code into it's own module.

Go to the container directory:

cd ~/terraform/basics/modules/container

Edit main.tf:

vi main.tf

main.tf contents:

# Start the Container

resource "docker\_container" "container\_id" {

name = "${var.container\_name}"

image = "${var.image}"

ports {

internal = "${var.int\_port}"

external = "${var.ext\_port}"

}

}

Edit variables.tf:

vi variables.tf

variables.tf contents:

#Define variables

variable "container\_name" {}

variable "image" {}

variable "int\_port" {}

variable "ext\_port" {}

Edit outputs.tf:

vi outputs.tf

outputs.tf contents:

#Output the IP Address of the Container

output "ip" {

value = "${docker\_container.container\_id.ip\_address}"

}

output "container\_name" {

value = "${docker\_container.container\_id.name}"

}

Initialize:

terraform init

Create the image plan:

terraform plan -out=tfplan -var 'container\_name=blog' -var 'image=ghost:alpine' -var 'int\_port=2368' -var 'ext\_port=80'

Deploy container using the plan:

terraform apply tfplan

## The Root Module

we will refactor the root module to use the image and container modules we created in the previous two lessons.

Go to the module directory:

cd ~/terraform/basics/modules/

touch {main.tf,variables.tf,outputs.tf}

Edit main.tf:

vi main.tf

main.tf contents:

# Download the image

module "image" {

source = "./image"

image\_name = "${var.image\_name}"

}

# Start the container

module "container" {

source = "./container"

image = "${module.image.image\_out}"

container\_name = "${var.container\_name}"

int\_port = "${var.int\_port}"

ext\_port = "${var.ext\_port}"

}

Edit variables.tf:

vi variables.tf

variables.tf contents:

#Define variables

variable "container\_name" {

description = "Name of the container."

default = "blog"

}

variable "image\_name" {

description = "Image for container."

default = "ghost:latest"

}

variable "int\_port" {

description = "Internal port for container."

default = "2368"

}

variable "ext\_port" {

description = "External port for container."

default = "80"

}

Edit outputs.tf:

vi outputs.tf

outputs.tf contents:

#Output the IP Address of the Container

output "ip" {

value = "${module.container.ip}"

}

output "container\_name" {

value = "${module.container.container\_name}"

}

Initialize Terraform:

terraform init

Create the image plan:

terraform plan -out=tfplan

Deploy the container using the plan:

terraform apply tfplan

Destroy the deployment:

terraform destroy -auto-approve

## v. Terraform and Docker

## Managing Docker Networks

 we will build on our knowledge of Terraform and Docker by learning about the docker\_network resource.

Set up the environment:

mkdir -p ~/terraform/docker/networks

cd terraform/docker/networks

Create the files:

touch {variables.tf,image.tf,network.tf,main.tf}

Edit variables.tf:

vi variables.tf

variables.tf contents:

variable "mysql\_root\_password" {

description = "The MySQL root password."

default = "P4sSw0rd0!"

}

variable "ghost\_db\_username" {

description = "Ghost blog database username."

default = "root"

}

variable "ghost\_db\_name" {

description = "Ghost blog database name."

default = "ghost"

}

variable "mysql\_network\_alias" {

description = "The network alias for MySQL."

default = "db"

}

variable "ghost\_network\_alias" {

description = "The network alias for Ghost"

default = "ghost"

}

variable "ext\_port" {

description = "Public port for Ghost"

default = "8080"

}

Edit image.tf:

vi image.tf

image.tf contents:

resource "docker\_image" "ghost\_image" {

name = "ghost:alpine"

}

resource "docker\_image" "mysql\_image" {

name = "mysql:5.7"

}

Edit network.tf:

vi network.tf

network.tf contents:

resource "docker\_network" "public\_bridge\_network" {

name = "public\_ghost\_network"

driver = "bridge"

}

resource "docker\_network" "private\_bridge\_network" {

name = "ghost\_mysql\_internal"

driver = "bridge"

internal = true

}

Edit main.tf:

vi main.tf

main.tf contents:

resource "docker\_container" "blog\_container" {

name = "ghost\_blog"

image = "${docker\_image.ghost\_image.name}"

env = [

"database\_\_client=mysql",

"database\_\_connection\_\_host=${var.mysql\_network\_alias}",

"database\_\_connection\_\_user=${var.ghost\_db\_username}",

"database\_\_connection\_\_password=${var.mysql\_root\_password}",

"database\_\_connection\_\_database=${var.ghost\_db\_name}"

]

ports {

internal = "2368"

external = "${var.ext\_port}"

}

networks\_advanced {

name = "${docker\_network.public\_bridge\_network.name}"

aliases = ["${var.ghost\_network\_alias}"]

}

networks\_advanced {

name = "${docker\_network.private\_bridge\_network.name}"

aliases = ["${var.ghost\_network\_alias}"]

}

}

resource "docker\_container" "mysql\_container" {

name = "ghost\_database"

image = "${docker\_image.mysql\_image.name}"

env = [

"MYSQL\_ROOT\_PASSWORD=${var.mysql\_root\_password}"

]

networks\_advanced {

name = "${docker\_network.private\_bridge\_network.name}"

aliases = ["${var.mysql\_network\_alias}"]

}

}

Initialize Terraform:

terraform init

Validate the files:

terraform validate

Build a plan:

terraform plan -out=tfplan -var 'ext\_port=8082'

Apply the plan:

terraform apply tfplan

Destroy the environment:

terraform destroy -auto-approve -var 'ext\_port=8082'

### Fixing main.tf

main.tf contents:

resource "docker\_container" "mysql\_container" {

name = "ghost\_database"

image = "${docker\_image.mysql\_image.name}"

env = [

"MYSQL\_ROOT\_PASSWORD=${var.mysql\_root\_password}"

]

networks\_advanced {

name = "${docker\_network.private\_bridge\_network.name}"

aliases = ["${var.mysql\_network\_alias}"]

}

}

resource "null\_resource" "sleep" {

depends\_on = ["docker\_container.mysql\_container"]

provisioner "local-exec" {

command = "sleep 15s"

}

}

resource "docker\_container" "blog\_container" {

name = "ghost\_blog"

image = "${docker\_image.ghost\_image.name}"

depends\_on = ["null\_resource.sleep", "docker\_container.mysql\_container"]

env = [

"database\_\_client=mysql",

"database\_\_connection\_\_host=${var.mysql\_network\_alias}",

"database\_\_connection\_\_user=${var.ghost\_db\_username}",

"database\_\_connection\_\_password=${var.mysql\_root\_password}",

"database\_\_connection\_\_database=${var.ghost\_db\_name}"

]

ports {

internal = "2368"

external = "${var.ext\_port}"

}

networks\_advanced {

name = "${docker\_network.public\_bridge\_network.name}"

aliases = ["${var.ghost\_network\_alias}"]

}

networks\_advanced {

name = "${docker\_network.private\_bridge\_network.name}"

aliases = ["${var.ghost\_network\_alias}"]

}

}

Build a plan:

terraform plan -out=tfplan -var 'ext\_port=8082'

Apply the plan:

terraform apply tfplan

## Managing Docker Volumes

we will add a volume to our Ghost Blog/MySQL setup.

Destroy the existing environment:

terraform destroy -auto-approve -var 'ext\_port=8082'

Setup an environment:

cp -r ~/terraform/docker/networks ~/terraform/docker/volumes

cd ../volumes/

Create volumes.tf:

vi volumes.tf

volumes.tf contents:

resource "docker\_volume" "mysql\_data\_volume" {

name = "mysql\_data"

}

Edit main.tf:

vi main.tf

main.tf contents:

resource "docker\_container" "mysql\_container" {

name = "ghost\_database"

image = "${docker\_image.mysql\_image.name}"

env = [

"MYSQL\_ROOT\_PASSWORD=${var.mysql\_root\_password}"

]

volumes {

volume\_name = "${docker\_volume.mysql\_data\_volume.name}"

container\_path = "/var/lib/mysql"

}

networks\_advanced {

name = "${docker\_network.private\_bridge\_network.name}"

aliases = ["${var.mysql\_network\_alias}"]

}

}

resource "null\_resource" "sleep" {

depends\_on = ["docker\_container.mysql\_container"]

provisioner "local-exec" {

command = "sleep 15s"

}

}

resource "docker\_container" "blog\_container" {

name = "ghost\_blog"

image = "${docker\_image.ghost\_image.name}"

depends\_on = ["null\_resource.sleep", "docker\_container.mysql\_container"]

env = [

"database\_\_client=mysql",

"database\_\_connection\_\_host=${var.mysql\_network\_alias}",

"database\_\_connection\_\_user=${var.ghost\_db\_username}",

"database\_\_connection\_\_password=${var.mysql\_root\_password}",

"database\_\_connection\_\_database=${var.ghost\_db\_name}"

]s

ports {

internal = "2368"

external = "${var.ext\_port}"

}

networks\_advanced {

name = "${docker\_network.public\_bridge\_network.name}"

aliases = ["${var.ghost\_network\_alias}"]

}

networks\_advanced {

name = "${docker\_network.private\_bridge\_network.name}"

aliases = ["${var.ghost\_network\_alias}"]

}

}

Initialize Terraform:

terraform init

Validate the files:

terraform validate

Build a plan:

terraform plan -out=tfplan -var 'ext\_port=8082'

Apply the plan:

terraform apply tfplan

List Docker volumes:

docker volume inspect mysql\_data

List the data in mysql\_data:

sudo ls /var/lib/docker/volumes/mysql\_data/\_data

Destroy the environment:

terraform destroy -auto-approve -var 'ext\_port=8082'

## Creating Swarm Services

we will convert our Ghost and MySQL containers over to using Swarm services. Swarm services are a more production-ready way of running containers.

Setup the environment:

cp -r volumes/ services

cd services

variables.tf contents:

variable "mysql\_root\_password" {

description = "The MySQL root password."

default = "P4sSw0rd0!"

}

variable "ghost\_db\_username" {

description = "Ghost blog database username."

default = "root"

}

variable "ghost\_db\_name" {

description = "Ghost blog database name."

default = "ghost"

}

variable "mysql\_network\_alias" {

description = "The network alias for MySQL."

default = "db"

}

variable "ghost\_network\_alias" {

description = "The network alias for Ghost"

default = "ghost"

}

variable "ext\_port" {

description = "The public port for Ghost"

}

images.tf contents:

resource "docker\_image" "ghost\_image" {

name = "ghost:alpine"

}

resource "docker\_image" "mysql\_image" {

name = "mysql:5.7"

}

networks.tf contents:

resource "docker\_network" "public\_bridge\_network" {

name = "public\_network"

driver = "overlay"

}

resource "docker\_network" "private\_bridge\_network" {

name = "mysql\_internal"

driver = "overlay"

internal = true

}

volumes.tf contents:

resource "docker\_volume" "mysql\_data\_volume" {

name = "mysql\_data"

}

main.tf contents:

resource "docker\_service" "ghost-service" {

name = "ghost"

task\_spec {

container\_spec {

image = "${docker\_image.ghost\_image.name}"

env {

database\_\_client = "mysql"

database\_\_connection\_\_host = "${var.mysql\_network\_alias}"

database\_\_connection\_\_user = "${var.ghost\_db\_username}"

database\_\_connection\_\_password = "${var.mysql\_root\_password}"

database\_\_connection\_\_database = "${var.ghost\_db\_name}"

}

}

networks = [

"${docker\_network.public\_bridge\_network.name}",

"${docker\_network.private\_bridge\_network.name}"

]

}

endpoint\_spec {

ports {

target\_port = "2368"

published\_port = "${var.ext\_port}"

}

}

}

resource "docker\_service" "mysql-service" {

name = "${var.mysql\_network\_alias}"

task\_spec {

container\_spec {

image = "${docker\_image.mysql\_image.name}"

env {

MYSQL\_ROOT\_PASSWORD = "${var.mysql\_root\_password}"

}

mounts = [

{

target = "/var/lib/mysql"

source = "${docker\_volume.mysql\_data\_volume.name}"

type = "volume"

}

]

}

networks = ["${docker\_network.private\_bridge\_network.name}"]

}

}

Initialize Terraform:

terraform init

Validate the files:

terraform validate

Build a plan:

terraform plan -out=tfplan -var 'ext\_port=8082'

Apply the plan:

terraform apply tfplan

docker service ls

docker container ls

Destroy the environment:

terraform destroy -auto-approve -var 'ext\_port=8082'

## Using Secrets

we'll explore using Terraform to store sensitive data, by using Docker Secrets.

Setup the environment:

mkdir secrets

cd secrets

Encode the password with Base64:

echo 'p4sSWoRd0!' | base64

Create variables.tf:

vi variables.tf

variables.tf contents:

variable "mysql\_root\_password" {

default = "cDRzU1dvUmQwIQo="

}

variable "mysql\_db\_password" {

default = "cDRzU1dvUmQwIQo="

}

Create image.tf:

vi image.tf

image.tf contents:

resource "docker\_image" "mysql\_image" {

name = "mysql:5.7"

}

Create secrets.tf:

vi secrets.tf

secrets.tf contents:

resource "docker\_secret" "mysql\_root\_password" {

name = "root\_password"

data = "${var.mysql\_root\_password}"

}

resource "docker\_secret" "mysql\_db\_password" {

name = "db\_password"

data = "${var.mysql\_db\_password}"

}

Create networks.tf:

vi networks.tf

networks.tf contents:

resource "docker\_network" "private\_overlay\_network" {

name = "mysql\_internal"

driver = "overlay"

internal = true

}

Create volumes.tf:

vi volumes.tf

volumes.tf contents:

resource "docker\_volume" "mysql\_data\_volume" {

name = "mysql\_data"

}

Create main.tf:

vi main.tf

main.tf contents:

resource "docker\_service" "mysql-service" {

name = "mysql\_db"

task\_spec {

container\_spec {

image = "${docker\_image.mysql\_image.name}"

secrets = [

{

secret\_id = "${docker\_secret.mysql\_root\_password.id}"

secret\_name = "${docker\_secret.mysql\_root\_password.name}"

file\_name = "/run/secrets/${docker\_secret.mysql\_root\_password.name}"

},

{

secret\_id = "${docker\_secret.mysql\_db\_password.id}"

secret\_name = "${docker\_secret.mysql\_db\_password.name}"

file\_name = "/run/secrets/${docker\_secret.mysql\_db\_password.name}"

}

]

env {

MYSQL\_ROOT\_PASSWORD\_FILE = "/run/secrets/${docker\_secret.mysql\_root\_password.name}"

MYSQL\_DATABASE = "mydb"

MYSQL\_PASSWORD\_FILE = "/run/secrets/${docker\_secret.mysql\_db\_password.name}"

}

mounts = [

{

target = "/var/lib/mysql"

source = "${docker\_volume.mysql\_data\_volume.name}"

type = "volume"

}

]

}

networks = [

"${docker\_network.private\_overlay\_network.name}"

]

}

}

Initialize Terraform:

terraform init

Validate the files:

terraform validate

Build a plan:

terraform plan -out=tfplan

Apply the plan:

terraform apply tfplan

Find the MySQL container:

docker container ls

Use the exec command to log into the MySQL container:

docker container exec -it [CONTAINER\_ID] /bin/bash

Access MySQL:

mysql -u root -p

Destroy the environment:

terraform destroy -auto-approve

## vi. Using Terraform in a CI/CD Environment

## Building a Custom Jenkins Image

we will learn how to build a Jenkins Docker image that has Docker and Terraform baked in. We will be using this image throughout the remainder of this section.

Setup the environment:

mkdir -p jenkins

Create Dockerfile:

vi Dockerfile

Dockerfile contents:

FROM jenkins/jenkins:lts

USER root

RUN apt-get update -y && apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

RUN curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey

RUN add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") $(lsb\_release -cs) stable"

RUN apt-get update -y

RUN apt-get install -y docker-ce docker-ce-cli containerd.io

RUN curl -O https://releases.hashicorp.com/terraform/0.11.13/terraform\_0.11.13\_linux\_amd64.zip && unzip terraform\_0.11.13\_linux\_amd64.zip -d /usr/local/bin/

USER ${user}

Build the Image:

docker build -t jenkins:terraform .

List the Docker images:

docker image ls

## Setting Up Jenkins

we will take the Jenkins image we built in the previous lesson, and deploy a Docker container using Terraform.

Edit main.tf:

vi main.tf

main.tf contents:

# Jenkins Volume

resource "docker\_volume" "jenkins\_volume" {

name = "jenkins\_data"

}

# Start the Jenkins Container

resource "docker\_container" "jenkins\_container" {

name = "jenkins"

image = "jenkins:terraform"

ports {

internal = "8080"

external = "8080"

}

volumes {

volume\_name = "${docker\_volume.jenkins\_volume.name}"

container\_path = "/var/jenkins\_home"

}

volumes {

host\_path = "/var/run/docker.sock"

container\_path = "/var/run/docker.sock"

}

}

Initialize Terraform:

terraform init

Plan the deployment:

terraform plan -out=tfplan

Deploy Jenkins:

terraform apply tfplan

Get the Admin password:

docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword

## Creating a Jenkins Job

We will start working with Jenkins by creating a simple build job. This job will deploy a Docker container using Terraform, list the container, and then destroy it.

In the Jenkins dashboard, Click **New Item**.  
Select **Freestyle Project**, and enter an item name of **DeployGhost**. Click **Ok**.

Under *Source Code Management*, select **Git**. Enter a *Repository URL* of [**https://github.com/linuxacademy/content-terraform-docker.git**](https://github.com/linuxacademy/content-terraform-docker.git)

In the *Build* section, click **Add build step** and select **Execute shell** from the dropdown.

Add the following in the *Command* area:

terraform init

terraform plan -out=tfplan

terraform apply tfplan

docker container ls

terraform destroy -auto-approve

Click **Save**.

Now, if we click **Build Now** in the left-hand menu, our project will start building. Clicking the little dropdown arrow next to *#1* will give us a menu. Select **Console Output** to watch things build. Once we get a Finished: SUCCESS message, we're done.

## Building a Jenkins Pipeline Part 1

we will create the first Jenkins Pipeline that will deploy out a Ghost blog.

In the Jenkins dashboard, click **New Item** Enter an item name of **PipelinePart1**, and select **Pipeline**. Click **Ok.**

Check the box for *This project is parameterized*. Click **Add Parameter** and select *Choice Parameter*. Give it a *Name* of **action**. For *Choices*, enter **Deploy** and **Destroy**, and make sure they are on separate lines. Enter **The action that will be executed** as the *Description*.

Click **Add Parameter** and select **Choice Parameter** again. This time, name it **image\_name**. Enter **ghost:latest** and **ghost:alpine** in the *Choices* box, making sure they are on separate lines. Enter **The image Ghost Blog will deploy** as a *Description*.

Click **Add Parameter** a third time, and select **String Parameter**. Give it a *Name* of **ext\_port**. Set the *Default Value* to **80**. Enter **The Public Port** as the *Description*.

Down in the *Pipeline* section, give a *Definition* of **Pipeline script**, and add the following to the *Script*:

node {

git 'https://github.com/linuxacademy/content-terraform-docker.git'

if(action == 'Deploy') {

stage('init') {

sh """

terraform init

"""

}

stage('plan') {

sh label: 'terraform plan', script: "terraform plan -out=tfplan -input=false -var image\_name=${image\_name} -var ext\_port=${ext\_port}"

script {

timeout(time: 10, unit: 'MINUTES') {

input(id: "Deploy Gate", message: "Deploy environment?", ok: 'Deploy')

}

}

}

stage('apply') {

sh label: 'terraform apply', script: "terraform apply -lock=false -input=false tfplan"

}

}

if(action == 'Destroy') {

stage('plan\_destroy') {

sh label: 'terraform plan destroy', script: "terraform plan -destroy -out=tfdestroyplan -input=false -var image\_name=${image\_name} -var ext\_port=${ext\_port}"

}

stage('destroy') {

script {

timeout(time: 10, unit: 'MINUTES') {

input(id: "Destroy Gate", message: "Destroy environment?", ok: 'Destroy')

}

}

sh label: 'Destroy environment', script: "terraform apply -lock=false -input=false tfdestroyplan"

}

}

}

Click **Save**

## Building a Jenkins Pipeline Part 2

we will create a Jenkins Pipeline to deploy out a Swarm service.

In the Jenkins dashboard, click **New Item** Enter an item name of **PipelinePart2**, and select **Pipeline**. Click **Ok.**

Check the box for *This project is parameterized*. Click **Add Parameter** and select *Choice Parameter*. Give it a *Name* of **action**. For *Choices*, enter **Deploy** and **Destroy**, and make sure they are on separate lines. Enter **The action that will be executed** as the *Description*.

Click **Add Parameter** and select **Choice Parameter** again. This time, name it **image\_name**. Enter **ghost:latest** and **ghost:alpine** in the *Choices* box, making sure they are on separate lines. Enter **The image Ghost Blog will deploy** as a *Description*.

Click **Add Parameter** a third time, and select **String Parameter**. Give it a *Name* of **ghost\_ext\_port**. Set the *Default Value* to **80**. Enter **The Public Port** as the *Description*.

Down in the *Pipeline* section, give a *Definition* of **Pipeline script**, and add the following to the *Script*:

node {

git 'https://github.com/linuxacademy/content-terraform-docker-service.git'

if(action == 'Deploy') {

stage('init') {

sh label: 'terraform init', script: "terraform init"

}

stage('plan') {

sh label: 'terraform plan', script: "terraform plan -out=tfplan -input=false -var image\_name=${image\_name} -var ghost\_ext\_port=${ghost\_ext\_port}"

script {

timeout(time: 10, unit: 'MINUTES') {

input(id: "Deploy Gate", message: "Deploy environment?", ok: 'Deploy')

}

}

}

stage('apply') {

sh label: 'terraform apply', script: "terraform apply -lock=false -input=false tfplan"

}

}

if(action == 'Destroy') {

stage('plan\_destroy') {

sh label: 'terraform plan', script: "terraform plan -destroy -out=tfdestroyplan -input=false -var image\_name=${image\_name} -var ghost\_ext\_port=${ghost\_ext\_port}"

}

stage('destroy') {

script {

timeout(time: 10, unit: 'MINUTES') {

input(id: "Destroy Gate", message: "Destroy environment?", ok: 'Destroy')

}

}

sh label: 'terraform apply', script: "terraform apply -lock=false -input=false tfdestroyplan"

}

stage('cleanup') {

sh label: 'cleanup', script: "rm -rf terraform.tfstat"

}

}

}

Click **Save**

## Building a Jenkins Pipeline Part 3

we will complete working with Jenkins by creating a pipeline that will create a MySQL Swarm service that uses Docker Secrets.

In the Jenkins dashboard, click **New Item** Enter an item name of **PipelinePart3**, and select **Pipeline**. Click **Ok.**

Check the box for *This project is parameterized*. Click **Add Parameter** and select *Choice Parameter*. Give it a *Name* of **action**. For *Choices*, enter **Deploy** and **Destroy**, and make sure they are on separate lines. Enter **The action that will be executed** as the *Description*.

Click **Add Parameter** and select **String Parameter**. This time, name it **mysql\_root\_password**. Enter **P4ssW0rd0!** as a *Default Value*box, making sure they are on separate lines. Enter **MySQL root password.** as a *Description*.

Down in the *Pipeline* section, give a *Definition* of **Pipeline script**, and add the following to the *Script*:

node {

git 'https://github.com/linuxacademy/content-terraform-docker-secrets.git'

if(action == 'Deploy') {

stage('init') {

sh label: 'terraform init', script: "terraform init"

}

stage('plan') {

def ROOT\_PASSWORD = sh (returnStdout: true, script: """echo ${mysql\_root\_password} | base64""").trim()

def USER\_PASSWORD = sh (returnStdout: true, script: """echo ${mysql\_user\_password} | base64""").trim()

sh label: 'terraform plan', script: "terraform plan -out=tfplan -input=false -var mysql\_root\_password=${ROOT\_PASSWORD} -var mysql\_db\_password=${USER\_PASSWORD}"

script {

timeout(time: 10, unit: 'MINUTES') {

input(id: "Deploy Gate", message: "Deploy ${params.project\_name}?", ok: 'Deploy')

}

}

}

stage('apply') {

sh label: 'terraform apply', script: "terraform apply -lock=false -input=false tfplan"

}

}

if(action == 'Destroy') {

stage('plan\_destroy') {

def ROOT\_PASSWORD = sh (returnStdout: true, script: """echo ${mysql\_root\_password} | base64""").trim()

def USER\_PASSWORD = sh (returnStdout: true, script: """echo ${mysql\_user\_password} | base64""").trim()

sh label: 'terraform plan', script: "terraform plan -destroy -out=tfdestroyplan -input=false -var mysql\_root\_password=${ROOT\_PASSWORD} -var mysql\_db\_password=${USER\_PASSWORD}"

}

stage('destroy') {

script {

timeout(time: 10, unit: 'MINUTES') {

input(id: "Destroy Gate", message: "Destroy ${params.project\_name}?", ok: 'Destroy')

}

}

sh label: 'terraform apply', script: "terraform apply -lock=false -input=false tfdestroyplan"

}

stage('cleanup') {

sh label: 'cleanup', script: "rm -rf terraform.tfstat"

}

}

}

Click **Save**

## Terraform and AWS

## Storage Part 1: The S3 Bucket and Random ID

 We will start working with AWS by creating a S3 Terraform module.

Environment setup:

mkdir -p ~/terraform/AWS/storage

cd ~/terraform/AWS/storage

Create main.tf:

vi main.tf

main.tf:

#---------storage/main.tf---------

# Create a random id

resource "random\_id" "tf\_bucket\_id" {

byte\_length = 2

}

# Create the bucket

resource "aws\_s3\_bucket" "tf\_code" {

bucket = "${var.project\_name}-${random\_id.tf\_bucket\_id.dec}"

acl = "private"

force\_destroy = true

tags {

Name = "tf\_bucket"

}

}

Create variables.tf:

vi variables.tf

variables.tf:

#----storage/variables.tf----

variable "project\_name" {}

Coutputs.tf:

vi outputs.tf

outputs.tf:

#----storage/outputs.tf----

output "bucketname" {

value = "${aws\_s3\_bucket.tf\_code.id}"

}

Initialize Terraform:

terraform init

Validate your files:

terraform validate

Plan the deployment:

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]]"

export AWS\_DEFAULT\_REGION="us-east-1"

terraform plan -out=tfplan -var project\_name=la-terraform

Deploy the S3 bucket:

terraform apply tfplan

Destroy S3 bucket:

terraform destroy -auto-approve -var project\_name=la-terraform

## Storage Part 2: The Root Module

we will start working on our root module. We'll start off by adding the storage module created in the previous lesson.

Environment setup:

cd ~/terraform/AWS

touch {main.tf,variables.tf,outputs.tf,terraform.tfvars}

Edit main.tf:

vi main.tf

main.tf:

#----root/main.tf-----

provider "aws" {

region = "${var.aws\_region}"

}

# Deploy Storage Resources

module "storage" {

source = "./storage"

project\_name = "${var.project\_name}"

}

Edit variables.tf:

vi variables.tf

variables.tf:

#----root/variables.tf-----

variable "aws\_region" {}

#------ storage variables

variable "project\_name" {}

Edit terraform.tfvars:

vi terraform.tfvars

terraform.tfvars:

aws\_region = "us-east-1"

project\_name = "la-terraform"

Edit outputs.tf:

vi outputs.tf

outputs.tf:

#----root/outputs.tf-----

#----storage outputs------

output "Bucket Name" {

value = "${module.storage.bucketname}"

}

Initialize terraform:

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]]"

terraform init

Validate code:

terraform validate

Deploy the S3 bucket:

terraform apply -auto-approve

Destroy S3 bucket:

terraform destroy -auto-approve

## Networking Part 1: VPC, IGW, and Route Tables

we will start our networking resource deployment and we will deploy our Internet Gateway and route tables.

Environment setup:

mkdir -p ~/terraform/AWS/networking

cd ~/terraform/AWS/networking

Touch the files:

touch {main.tf,variables.tf,outputs.tf,terraform.tfvars}

Edit main.tf:

vi main.tf

main.tf:

#----networking/main.tf----

data "aws\_availability\_zones" "available" {}

resource "aws\_vpc" "tf\_vpc" {

cidr\_block = "${var.vpc\_cidr}"

enable\_dns\_hostnames = true

enable\_dns\_support = true

tags {

Name = "tf\_vpc"

}

}

resource "aws\_internet\_gateway" "tf\_internet\_gateway" {

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

tags {

Name = "tf\_igw"

}

}

resource "aws\_route\_table" "tf\_public\_rt" {

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = "${aws\_internet\_gateway.tf\_internet\_gateway.id}"

}

tags {

Name = "tf\_public"

}

}

resource "aws\_default\_route\_table" "tf\_private\_rt" {

default\_route\_table\_id = "${aws\_vpc.tf\_vpc.default\_route\_table\_id}"

tags {

Name = "tf\_private"

}

}

resource "aws\_subnet" "tf\_public\_subnet" {

count = 2

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

cidr\_block = "${var.public\_cidrs[count.index]}"

map\_public\_ip\_on\_launch = true

availability\_zone = "${data.aws\_availability\_zones.available.names[count.index]}"

tags {

Name = "tf\_public\_${count.index + 1}"

}

}

resource "aws\_route\_table\_association" "tf\_public\_assoc" {

count = "${aws\_subnet.tf\_public\_subnet.count}"

subnet\_id = "${aws\_subnet.tf\_public\_subnet.\*.id[count.index]}"

route\_table\_id = "${aws\_route\_table.tf\_public\_rt.id}"

}

resource "aws\_security\_group" "tf\_public\_sg" {

name = "tf\_public\_sg"

description = "Used for access to the public instances"

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

#SSH

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["${var.accessip}"]

}

#HTTP

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["${var.accessip}"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

Edit variables.tf:

vi variables.tf

variables.tf:

#----networking/variables.tf----

variable "vpc\_cidr" {}

variable "public\_cidrs" {

type = "list"

}

variable "accessip" {}

Edit outputs.tf:

vi outputs.tf

outputs.tf:

#-----networking/outputs.tf----

output "public\_subnets" {

value = "${aws\_subnet.tf\_public\_subnet.\*.id}"

}

output "public\_sg" {

value = "${aws\_security\_group.tf\_public\_sg.id}"

}

output "subnet\_ips" {

value = "${aws\_subnet.tf\_public\_subnet.\*.cidr\_block}"

}

terraform.tfvars:

vpc\_cidr = "10.123.0.0/16"

public\_cidrs = [

"10.123.1.0/24",

"10.123.2.0/24"

]

accessip = "0.0.0.0/0"

Initialize Terraform:

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]]"

terraform init

Validate code:

terraform validate

Deploy Network:

terraform apply -auto-approve

Destroy Network:

terraform destroy -auto-approve

Delete terraform.tfvars:

rm terraform.tfvars

## Networking Part 2: Subnets, Security, and the Count Attribute

 we will continue working on the Networking module by adding in code to handle subnets and security groups.

Environment setup:

mkdir -p ~/terraform/AWS/networking

cd ~/terraform/AWS/networking

Touch the files:

touch {main.tf,variables.tf,outputs.tf,terraform.tfvars}

Edit main.tf:

vi main.tf

main.tf:

#----networking/main.tf----

data "aws\_availability\_zones" "available" {}

resource "aws\_vpc" "tf\_vpc" {

cidr\_block = "${var.vpc\_cidr}"

enable\_dns\_hostnames = true

enable\_dns\_support = true

tags {

Name = "tf\_vpc"

}

}

resource "aws\_internet\_gateway" "tf\_internet\_gateway" {

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

tags {

Name = "tf\_igw"

}

}

resource "aws\_route\_table" "tf\_public\_rt" {

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = "${aws\_internet\_gateway.tf\_internet\_gateway.id}"

}

tags {

Name = "tf\_public"

}

}

resource "aws\_default\_route\_table" "tf\_private\_rt" {

default\_route\_table\_id = "${aws\_vpc.tf\_vpc.default\_route\_table\_id}"

tags {

Name = "tf\_private"

}

}

resource "aws\_subnet" "tf\_public\_subnet" {

count = 2

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

cidr\_block = "${var.public\_cidrs[count.index]}"

map\_public\_ip\_on\_launch = true

availability\_zone = "${data.aws\_availability\_zones.available.names[count.index]}"

tags {

Name = "tf\_public\_${count.index + 1}"

}

}

resource "aws\_route\_table\_association" "tf\_public\_assoc" {

count = "${aws\_subnet.tf\_public\_subnet.count}"

subnet\_id = "${aws\_subnet.tf\_public\_subnet.\*.id[count.index]}"

route\_table\_id = "${aws\_route\_table.tf\_public\_rt.id}"

}

resource "aws\_security\_group" "tf\_public\_sg" {

name = "tf\_public\_sg"

description = "Used for access to the public instances"

vpc\_id = "${aws\_vpc.tf\_vpc.id}"

#SSH

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["${var.accessip}"]

}

#HTTP

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["${var.accessip}"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

# Networking Part 3: Variables and Outputs

we will finish off the networking module by creating the variables and outputs Terraform files. Finally we will test the module before integrating it into the root module.

Edit variables.tf:

vi variables.tf

variables.tf:

#----networking/variables.tf----

variable "vpc\_cidr" {}

variable "public\_cidrs" {

type = "list"

}

variable "accessip" {}

Edit outputs.tf:

vi outputs.tf

outputs.tf:

#-----networking/outputs.tf----

output "public\_subnets" {

value = "${aws\_subnet.tf\_public\_subnet.\*.id}"

}

output "public\_sg" {

value = "${aws\_security\_group.tf\_public\_sg.id}"

}

output "subnet\_ips" {

value = "${aws\_subnet.tf\_public\_subnet.\*.cidr\_block}"

}

Edit terraform.tfvars:

vi terraform.tfvars

terraform.tfvars:

vpc\_cidr = "10.123.0.0/16"

public\_cidrs = [

"10.123.1.0/24",

"10.123.2.0/24"

]

accessip = "0.0.0.0/0"

Initialize Terraform:

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]]"

terraform init

Validate code:

terraform validate

Deploy Network:

terraform apply -auto-approve

Destroy Network:

terraform destroy -auto-approve

Delete terraform.tfvars:

rm terraform.tfvars

## Networking Part 4: The Root Module

we will add the networking module to the root module.

Environment setup:

cd ~/terraform/AWS

Edit main.tf:

vi main.tf

main.tf:

provider "aws" {

region = "${var.aws\_region}"

}

# Deploy Storage Resources

module "storage" {

source = "./storage"

project\_name = "${var.project\_name}"

}

# Deploy Networking Resources

module "networking" {

source = "./networking"

vpc\_cidr = "${var.vpc\_cidr}"

public\_cidrs = "${var.public\_cidrs}"

accessip = "${var.accessip}"

}

Edit variables.tf:

vi variables.tf

variables.tf:

#----root/variables.tf-----

variable "aws\_region" {}

#------ storage variables

variable "project\_name" {}

#-------networking variables

variable "vpc\_cidr" {}

variable "public\_cidrs" {

type = "list"

}

variable "accessip" {}

Edit terraform.tfvars:

vi terraform.tfvars

terraform.tfvars:

aws\_region = "us-east-1"

project\_name = "la-terraform"

vpc\_cidr = "10.123.0.0/16"

public\_cidrs = [

"10.123.1.0/24",

"10.123.2.0/24"

]

accessip = "0.0.0.0/0"

Reinitialize Terraform:

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]]"

terraform init

Validate code:

terraform validate

Apply Changes:

terraform apply -auto-approve

Destroy environment:

terraform destroy -auto-approve

## Compute Part 1: AMI Data, Key Pair, and the File Function

we will start working on building out the resources for out AWS compute.

Environment setup:

mkdir -p ~/terraform/AWS/compute

cd ~/terraform/AWS/compute

Touch the files:

touch {main.tf,variables.tf,outputs.tf}

Create a SSH key.

ssh-keygen

Edit main.tf:

vi main.tf

main.tf:

#----compute/main.tf#----

data "aws\_ami" "server\_ami" {

most\_recent = true

owners = ["amazon"]

filter {

name = "name"

values = ["amzn-ami-hvm\*-x86\_64-gp2"]

}

}

resource "aws\_key\_pair" "tf\_auth" {

key\_name = "${var.key\_name}"

public\_key = "${file(var.public\_key\_path)}"

}

Edit variables.tf:

vi variables.tf

variables.tf:

#----compute/variables.tf----

variable "key\_name" {}

variable "public\_key\_path" {}

Initialize Terraform:

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]]"

terraform init

Validate changes:

terraform validate

Plan the changes:

terraform plan -out=tfplan -var 'key\_name=tfkey' -var 'public\_key\_path=/home/cloud\_user/.ssh/id\_rsa.pub'

Apply the changes:

terraform apply -auto-approve

Provide the values for key\_name and public\_key\_path: key\_name: tfkey public\_key\_path: /home/cloud\_user/.ssh/id\_rsa.pub

Destroy environment:

terraform destroy -auto-approve

Provide the values for key\_name and public\_key\_path: key\_name: tfkey public\_key\_path: /home/cloud\_user/.ssh/id\_rsa.pub

## Compute Part 2: The EC2 Instance

we will finish off the Compute module by adding the aws\_instance resource.

Edit main.tf:

vi main.tf

main.tf:

#-----compute/main.tf#-----

data "aws\_ami" "server\_ami" {

most\_recent = true

owners = ["amazon"]

filter {

name = "name"

values = ["amzn-ami-hvm\*-x86\_64-gp2"]

}

}

resource "aws\_key\_pair" "tf\_auth" {

key\_name = "${var.key\_name}"

public\_key = "${file(var.public\_key\_path)}"

}

data "template\_file" "user-init" {

count = 2

template = "${file("${path.module}/userdata.tpl")}"

vars {

firewall\_subnets = "${element(var.subnet\_ips, count.index)}"

}

}

resource "aws\_instance" "tf\_server" {

count = "${var.instance\_count}"

instance\_type = "${var.instance\_type}"

ami = "${data.aws\_ami.server\_ami.id}"

tags {

Name = "tf\_server-${count.index +1}"

}

key\_name = "${aws\_key\_pair.tf\_auth.id}"

vpc\_security\_group\_ids = ["${var.security\_group}"]

subnet\_id = "${element(var.subnets, count.index)}"

user\_data = "${data.template\_file.user-init.\*.rendered[count.index]}"

}

Create userdata.tpl:

vi userdata.tpl

userdata.tpl:

#!/bin/bash

yum install httpd -y

echo "Subnet for Firewall: ${firewall\_subnets}" >> /var/www/html/index.html

service httpd start

chkconfig httpd on

Edit variables.tf:

vi variables.tf

variables.tf:

#-----compute/variables.tf

variable "key\_name" {}

variable "public\_key\_path" {}

variable "subnet\_ips" {

type = "list"

}

variable "instance\_count" {}

variable "instance\_type" {}

variable "security\_group" {}

variable "subnets" {

type = "list"

}

Edit outputs.tf:

vi outputs.tf

outputs.tf"

#-----compute/outputs.tf-----

output "server\_id" {

value = "${join(", ", aws\_instance.tf\_server.\*.id)}"

}

output "server\_ip" {

value = "${join(", ", aws\_instance.tf\_server.\*.public\_ip)}"

## Compute Part 3: The Root Module

we will finish working with the EC2 resources by adding the compute module to the root module.

Edit main.tf:

vi main.tf

main.tf:

provider "aws" {

region = "${var.aws\_region}"

}

# Deploy Storage Resources

module "storage" {

source = "./storage"

project\_name = "${var.project\_name}"

}

# Deploy Networking Resources

module "networking" {

source = "./networking"

vpc\_cidr = "${var.vpc\_cidr}"

public\_cidrs = "${var.public\_cidrs}"

accessip = "${var.accessip}"

}

# Deploy Compute Resources

module "compute" {

source = "./compute"

instance\_count = "${var.instance\_count}"

key\_name = "${var.key\_name}"

public\_key\_path = "${var.public\_key\_path}"

instance\_type = "${var.server\_instance\_type}"

subnets = "${module.networking.public\_subnets}"

security\_group = "${module.networking.public\_sg}"

subnet\_ips = "${module.networking.subnet\_ips}"

}

Edit variables.tf:

vi variables.tf

variables.tf:

#----root/variables.tf-----

variable "aws\_region" {}

#------ storage variables

variable "project\_name" {}

#-------networking variables

variable "vpc\_cidr" {}

variable "public\_cidrs" {

type = "list"

}

variable "accessip" {}

#-------compute variables

variable "key\_name" {}

variable "public\_key\_path" {}

variable "server\_instance\_type" {}

variable "instance\_count" {

default = 1

}

Edit outputs.tf:

vi outputs.tf

outputs.tf:

#----root/outputs.tf-----

#----storage outputs------

output "Bucket Name" {

value = "${module.storage.bucketname}"

}

#---Networking Outputs -----

output "Public Subnets" {

value = "${join(", ", module.networking.public\_subnets)}"

}

output "Subnet IPs" {

value = "${join(", ", module.networking.subnet\_ips)}"

}

output "Public Security Group" {

value = "${module.networking.public\_sg}"

}

#---Compute Outputs ------

output "Public Instance IDs" {

value = "${module.compute.server\_id}"

}

output "Public Instance IPs" {

value = "${module.compute.server\_ip}"

}

terraform.tfvars:

aws\_region = "us-west-1"

project\_name = "la-terraform"

vpc\_cidr = "10.123.0.0/16"

public\_cidrs = [

"10.123.1.0/24",

"10.123.2.0/24"

]

accessip = "0.0.0.0/0"

key\_name = "tf\_key"

public\_key\_path = "/home/cloud\_user/.ssh/id\_rsa.pub"

server\_instance\_type = "t2.micro"

instance\_count = 2

Initialize Terraform:

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]"

terraform init

Validate changes:

terraform validate

Plan the changes:

terraform plan

Apply the changes:

terraform apply

Destroy environment:

terraform destroy

## viii. Troubleshooting Terraform Files

we will talk about troubleshooting issues in the real world. Sometimes it's nothing more than "XYZ is broken. Go fix it!"

Test your troubleshooting abilities:

cd ~/terraform

git clone https://github.com/linuxacademy/content-terraform-labs.git troubleshooting

cd troubleshooting

git checkout troubleshooting-aws

terraform init

## ix. Terraform State

## 

## Terraform Formatting and Remote State

You will learn more about Terraform state and how to version it using a S3 bucket.

## Create an S3 Bucket

Search for S3 in **Find Services**.  
Click **Create Bucket**.  
Enter a **Bucket name**. The bucket name must be unique.  
Make sure the *Region* is **US East (N. Virginia)** Click **Next**.  
Click **Next** again on the Configure options page.  
Click **Next** again on the Set permissions page.  
Click **Create bucket** on the Review page.

## Add the Terraform Folder to the Bucket

Click on the bucket name.  
Click **Create folder**.  
Enter **terraform-aws** for the folder name.  
Click **Save**.

## Add Backend to Your Scripts

From the Docker Swarm Manager navigate to the AWS directory:

cd ~/terraform/AWS

## Set the Environment Variables

export AWS\_ACCESS\_KEY\_ID="[ACCESS\_KEY]"

export AWS\_SECRET\_ACCESS\_KEY="[SECRET\_KEY]]"

export AWS\_DEFAULT\_REGION="us-east-1"

Create terraform.tf:

vi terraform.tf

terraform.tf contents:

terraform {

backend "s3" {

key = "terraform-aws/terraform.tfstate"

}

}

Initialize Terraform:

terraform init -backend-config "bucket=[BUCKET\_NAME]"

Validate changes:

terraform validate

Plan the changes:

terraform plan

Apply the changes:

terraform apply -auto-approve

Destroy environment:

terraform destroy -auto-approve

## Using Remote State with Jenkins

we will and update our CI/CD process to use remote state with our Jenkins Pipelines. This will allow us to create two separate Jenkins Pipelines: one for deploying the infrastructure and one for destroying it.

## Create S3 Bucket

Search for S3 in Find Services.  
Click **Create Bucket**.  
Enter a **Bucket name**. The bucket name must be unique.  
Make sure the *Region* is **US East (N. Virginia)** Click **Next**.  
Click **Next** again on the Configure options page.  
Click **Next** again on the Set permissions page.  
Click **Create bucket** on the Review page.

## Add the Terraform Folder to the Bucket

Click on the bucket name.  
Click **Create folder**.  
Enter **terraform-aws** for the folder name.  
Click **Save**.

## Create the Jenkins Deploy Job

Enter an item name and call it **DeployDockerService**. Select **Pipeline**. Click **Ok**.

Click **Add Parameter** and select **String Parameter**. For the name enter **access\_key\_id**. Set the Default Value to your Access Key Id.

Click **Add Parameter** and select **String Parameter**. For the name enter **secret\_access\_key**. Set the Default Value to your Secret Access Key.

CClick **Add Parameter** and select **String Parameter**. For the name enter **bucket\_name**. Set the Default Value to the name of your S3 Bucket.

Click **Add Parameter** and select **Choice Parameter**. For the name enter **image\_name**. For choices enter ghost:latest and ghost:alpine. Make sure they are on separate lines.

Click **Add Parameter** and select **String Parameter**. For the name enter **ghost\_ext\_port**. Set the Default Value to **80**.

In the Pipeline section add the following to Script:

env.AWS\_ACCESS\_KEY\_ID = "${access\_key\_id}"

env.AWS\_SECRET\_ACCESS\_KEY = "${secret\_access\_key}"

env.AWS\_DEFAULT\_REGION = 'us-east-1'

node {

git (

url: 'https://github.com/linuxacademy/content-terraform-docker-service.git',

branch: 'remote-state'

)

stage('init') {

sh label: 'terraform init', script: "terraform init -backend-config \"bucket=${bucket\_name}\""

}

stage('plan') {

sh label: 'terraform plan', script: "terraform plan -out=tfplan -input=false -var image\_name=${image\_name} -var ghost\_ext\_port=${ghost\_ext\_port}"

}

stage('apply') {

sh label: 'terraform apply', script: "terraform apply -lock=false -input=false tfplan"

}

}

## Create the Jenkins Destroy Job

Enter an item name and call it **DestroyDockerService**.  
In Copy from enter **DeployDockerService**. Click **Ok**.

Change Pipeline section to the following:

env.AWS\_ACCESS\_KEY\_ID = "${access\_key\_id}"

env.AWS\_SECRET\_ACCESS\_KEY = "${secret\_access\_key}"

env.AWS\_DEFAULT\_REGION = 'us-east-1'

node {

git (

url: 'https://github.com/linuxacademy/content-terraform-docker-service.git',

branch: 'remote-state'

)

stage('init') {

sh label: 'terraform init', script: "terraform init -backend-config \"bucket=${bucket\_name}\""

}

stage('plan\_destroy') {

sh label: 'terraform plan', script: "terraform plan -destroy -out=tfdestroyplan -input=false -var image\_name=${image\_name} -var ghost\_ext\_port=${ghost\_ext\_port}"

}

stage('destroy') {

sh label: 'terraform apply', script: "terraform apply -lock=false -input=false tfdestroyplan"

}

}

Once Jenkins is running:

docker container ls

docker exec -it 73575a9ee4ac /bin/bash

## x. Setting up Kubernetes Installing Terraform

we will setup a Kuberentes master and install Terraform.

Add the following to kube-config.yml:

apiVersion: kubeadm.k8s.io/v1beta1

kind: ClusterConfiguration

kubernetesVersion: "v1.13.5"

networking:

podSubnet: 10.244.0.0/16

apiServer:

extraArgs:

service-node-port-range: 8000-31274

Initialize Kubernetes:

sudo kubeadm init --config kube-config.yml

Copy adimin.conf to your home directory:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

Install Flannel:

sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Untaint the Kubernetes Master:

kubectl taint nodes --all node-role.kubernetes.io/master-

## Install Terraform

Terraform will be installed on the Swarm manager.

Install Terraform 0.11.13:

sudo curl -O https://releases.hashicorp.com/terraform/0.11.13/terraform\_0.11.13\_linux\_amd64.zip

sudo unzip terraform\_0.11.13\_linux\_amd64.zip -d /usr/local/bin/

Test the Terraform installation:

terraform version

## Creating a Pod

We will start working with Kubernetes resources by creating a Pod.

Setup your environment:

mkdir -p ~/terraform/pod

cd ~/terraform/pod

vi main.tf:

resource "kubernetes\_pod" "ghost\_alpine" {

metadata {

name = "ghost-alpine"

}

spec {

host\_network = "true"

container {

image = "ghost:alpine"

name = "ghost-alpine"

}

}

}

Initialize Terraform:

terraform init

Validate main.tf:

terraform validate

Plan the deployment:

terraform plan

Deploy the pod:

terraform apply -auto-approve

List the Pods:

kubectl get pods

Reset the environment:

terraform destroy -auto-approve

## Creating a Pod and Service

we will create a pod and service using Terraform.

Setup your environment:

mkdir -p ~/terraform/service

cd ~/terraform/service

Create main.tf:

vi main.tf

main.tf contents:

resource "kubernetes\_service" "ghost\_service" {

metadata {

name = "ghost-service"

}

spec {

selector {

app = "${kubernetes\_pod.ghost\_alpine.metadata.0.labels.app}"

}

port {

port = "2368"

target\_port = "2368"

node\_port = "8081"

}

type = "NodePort"

}

}

resource "kubernetes\_pod" "ghost\_alpine" {

metadata {

name = "ghost-alpine"

labels {

app = "ghost-blog"

}

}

spec {

container {

image = "ghost:alpine"

name = "ghost-alpine"

port {

container\_port = "2368"

}

}

}

}

Initialize Terraform:

terraform init

Validate the files:

terraform validate

Plan the deployment:

terraform plan

Deploy the pod:

terraform apply -auto-approve

List the Pods:

kubectl get pods

List the Services:

kubectl get services

Reset the environment:

terraform destroy -auto-approve

# Creating a Deployment

we will use Terraform to create a Kubernetes deployment and service.

Setup your environment:

mkdir -p ~/terraform/deployment

cd ~/terraform/deployment

Create main.tf:

vi main.tf

main.tf contents:

resource "kubernetes\_service" "ghost\_service" {

metadata {

name = "ghost-service"

}

spec {

selector {

app = "${kubernetes\_deployment.ghost\_deployment.spec.0.template.0.metadata.0.labels.app}"

}

port {

port = "2368"

target\_port = "2368"

node\_port = "8080"

}

type = "NodePort"

}

}

resource "kubernetes\_deployment" "ghost\_deployment" {

metadata {

name = "ghost-blog"

}

spec {

replicas = "1"

selector {

match\_labels {

app = "ghost-blog"

}

}

template {

metadata {

labels {

app = "ghost-blog"

}

}

spec {

container {

name = "ghost"

image = "ghost:alpine"

port {

container\_port = "2368"

}

}

}

}

}

}

Initialize Terraform:

terraform init

Validate the files:

terraform validate

Plan the deployment:

terraform plan

Deploy the pod:

terraform apply -auto-approve

List the Deployments:

kubectl get deployments

kubectl get pods

kubectl delete pod [POD\_ID]

Reset the environment:

terraform destroy -auto-approve