**Change and Destroy Infrastructure**

**Change Infrastructure**

In the previous lesson, you created your first infrastructure with Terraform: **a single EC2 instance**. In this page, we're going to modify that resource, and see how Terraform handles change.

Infrastructure is continuously evolving, and Terraform was built to help manage and enact that change. As you change Terraform configurations, Terraform builds an execution plan that only modifies what is necessary to reach your desired state.

By using Terraform to change infrastructure, you can version control not only your configurations but also your state so you can see how the infrastructure evolved over time.

We already created example.tf file at learn-terraform directory. So we will start with modify the ami of our instance. Edit the aws\_instance.example resource under your provider block in your configuration and change it to the following:

provider "aws" {

profile = "default"

region = "us-east-1"

}

resource "aws\_instance" "example" {

ami = "ami-b374d5a5"

instance\_type = "t2.micro"

}

We've changed the AMI from being an **Ubuntu 16.04 LTS AMI** to being an **Ubuntu 16.10 AMI**. Terraform configurations are meant to be changed like this. You can also completely remove resources and Terraform will know to destroy the old one.

**Apply Changes**

After changing the configuration, run terraform apply again to see how Terraform will apply this change to the existing resources.

$ terraform apply

aws\_instance.example: Refreshing state... [id=i-0bbf06244e44211d1]

An execution plan has been generated and is shown below.

Resource actions are indicated with the following symbols:

-/+ destroy and then create replacement

Terraform will perform the following actions:

# aws\_instance.example must be replaced

-/+ resource "aws\_instance" "example" {

~ ami = "ami-2757f631" -> "ami-b374d5a5" #

          forces replacement

~ arn = "arn:aws:ec2:us-east-1:130490850807

          :instance/i-0bbf06244e44211d1" -> (known after apply)

~ associate\_public\_ip\_address = true -> (known after apply)

~ availability\_zone = "us-east-1c" -> (known after apply)

~ cpu\_core\_count = 1 -> (known after apply)

~ cpu\_threads\_per\_core = 1 -> (known after apply)

- disable\_api\_termination = false -> null

- ebs\_optimized = false -> null

get\_password\_data = false

+ host\_id = (known after apply)

~ id = "i-0bbf06244e44211d1" -> (known after

          apply)

~ instance\_state = "running" -> (known after apply)

instance\_type = "t2.micro"

~ ipv6\_address\_count = 0 -> (known after apply)

~ ipv6\_addresses = [] -> (known after apply)

+ key\_name = (known after apply)

- monitoring = false -> null

+ network\_interface\_id = (known after apply)

+ password\_data = (known after apply)

+ placement\_group = (known after apply)

~ primary\_network\_interface\_id = "eni-0f1ce5bdae258b015" -> (known after

          apply)

~ private\_dns = "ip-172-31-61-141.ec2.internal" ->

          (known after apply)

~ private\_ip = "172.31.61.141" -> (known after apply)

~ public\_dns = "ec2-54-166-19-244.compute-1.amazonaws

          .com" -> (known after apply)

~ public\_ip = "54.166.19.244" -> (known after apply)

~ security\_groups = [

- "default",

] -> (known after apply)

source\_dest\_check = true

~ subnet\_id = "subnet-1facdf35" -> (known after apply

          )

~ tenancy = "default" -> (known after apply)

~ volume\_tags = {} -> (known after apply)

~ vpc\_security\_group\_ids = [

- "sg-5255f429",

] -> (known after apply)

- credit\_specification {

- cpu\_credits = "standard" -> null

}

+ ebs\_block\_device {

+ delete\_on\_termination = (known after apply)

+ device\_name = (known after apply)

+ encrypted = (known after apply)

+ iops = (known after apply)

+ snapshot\_id = (known after apply)

+ volume\_id = (known after apply)

+ volume\_size = (known after apply)

+ volume\_type = (known after apply)

}

+ ephemeral\_block\_device {

+ device\_name = (known after apply)

+ no\_device = (known after apply)

+ virtual\_name = (known after apply)

}

+ network\_interface {

+ delete\_on\_termination = (known after apply)

+ device\_index = (known after apply)

+ network\_interface\_id = (known after apply)

}

~ root\_block\_device {

~ delete\_on\_termination = true -> (known after apply)

~ iops = 100 -> (known after apply)

~ volume\_id = "vol-0079e485d9e28a8e5" -> (known after

              apply)

~ volume\_size = 8 -> (known after apply)

~ volume\_type = "gp2" -> (known after apply)

}

}

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

The **prefix -/+** means that Terraform will destroy and recreate the resource, *rather than updating it in-place*. While some attributes can be updated in-place (which are shown with the ~ prefix), changing the AMI for an EC2 instance requires **recreating** it. Terraform handles these details for you, and the execution plan makes it clear what Terraform will do. You can follow the steps from AWS EC2 console.

*Additionally,*  *the execution plan shows that the AMI change is what required your resource to be replaced. Using this information, you can adjust your changes to possibly avoid destroy/create updates if they are not acceptable in some situations.*

Once again, Terraform prompts for approval of the execution plan before proceeding. Answer yes to execute the planned steps:

aws\_instance.example: Destroying... [id=i-0bbf06244e44211d1]

aws\_instance.example: Still destroying... [id=i-0bbf06244e44211d1, 10s

    elapsed]

aws\_instance.example: Still destroying... [id=i-0bbf06244e44211d1, 20s

    elapsed]

aws\_instance.example: Still destroying... [id=i-0bbf06244e44211d1, 30s

    elapsed]

aws\_instance.example: Destruction complete after 31s

aws\_instance.example: Creating...

aws\_instance.example: Still creating... [10s elapsed]

aws\_instance.example: Still creating... [20s elapsed]

aws\_instance.example: Still creating... [30s elapsed]

aws\_instance.example: Creation complete after 38s [id=i-0589469dd150b453b]

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

As indicated by the execution plan, Terraform first destroyed the existing instance and then created a new one in its place. You can use terraform show again to see the new values associated with this instance.

**Destroy Infrastructure**

We've now seen how to build and change infrastructure. Before we move on to creating multiple resources and showing resource dependencies, we're going to go over how to completely destroy the Terraform-managed infrastructure.

Destroying your infrastructure is a rare event in production environments. But if you're using Terraform to spin up multiple environments such as development, test, QA environments, then destroying is a useful action.

The terraform destroy command terminates resources defined in your Terraform configuration. This command is the *reverse of terraform apply* in that it **terminates all the resources specified by the configuration**. It does not destroy resources running elsewhere that are not described in the current configuration.

$ terraform destroy

# ...

# aws\_instance.example will be destroyed

- resource "aws\_instance" "example" {

- ami = "ami-b374d5a5" -> null

# ...

The - prefix indicates that the instance will be destroyed. As with apply, Terraform shows its execution plan and waits for approval before making any changes.

Answer yes to execute this plan and destroy the infrastructure.

# ...

aws\_instance.example: Destroying... [id=i-0589469dd150b453b]

Destroy complete! Resources: 1 destroyed.

# ...

Formun Üstü