# Experiment No. 06

**Practical Name :** Build, change, and destroy AWS / GCP /Microsoft Azure/ DigitalOcean infrastructure Using Terraform.

**Aim :** To Build, change, and destroy AWS / GCP /Microsoft Azure/ DigitalOcean infrastructure Using Terraform.

# Theory :

* **What Is Infrastructure as Code?**

Infrastructure as code is a means by which engineers define the computer systems and managing the operations environment in the same way one does applications or other code for general release.

Infrastructure as Code (IaC) grew as a response to the difficulty posed by utility computing and second-generation web frameworks. When you use a high-level descriptive coding language to automate the IT infrastructure, it eliminates the need for engineers to manually provision and manage servers, operating systems, database connections, storage, and other infrastructure elements when they want to develop, test, or deploy a software application.

To deploy infrastructure with Terraform:

* + **Scope** - Identify the infrastructure for your project.
  + **Author** - Write the configuration for your infrastructure.
  + **Initialize** - Install the plugins Terraform needs to manage the infrastructure.
  + **Plan** - Preview the changes Terraform will make to match your configuration.
  + **Apply** - Make the planned changes.

# Plan :

The terraform plan command evaluates a Terraform configuration to determine the desired state of all the resources it declares, then compares that desired state to the real infrastructure objects being managed with the current working directory and workspace. It uses state data to determine which real objects correspond to which declared resources, and checks the current state of each resource using the relevant infrastructure provider's API.

Once it has determined the difference between the current state and the desired state, terraform plan presents a description of the changes necessary to achieve the desired state. It does not perform any actual changes to real world infrastructure objects; it only presents a plan for making changes.

Plans are usually run to validate configuration changes and confirm that the resulting actions are as expected. However, terraform plan can also save its plan as a runnable artifact, which terraform apply can use to carry out those exact changes.

# Apply :

The terraform apply command performs a plan just like terraform plan does, but then actually carries out the planned changes to each resource using the relevant infrastructure provider's API. It asks for confirmation from the user before making any changes, unless it was explicitly told to skip approval.

By default, terraform apply performs a fresh plan right before applying changes, and displays the plan to the user when asking for confirmation. However, it can also accept a plan file produced by terraform plan in lieu of running a new plan. You can use this to reliably perform an exact set of pre-approved changes, even if the configuration or the state of the real infrastructure has changed in the minutes since the original plan was created.

# Change infrastructure :

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

# Destroy infrastructure :

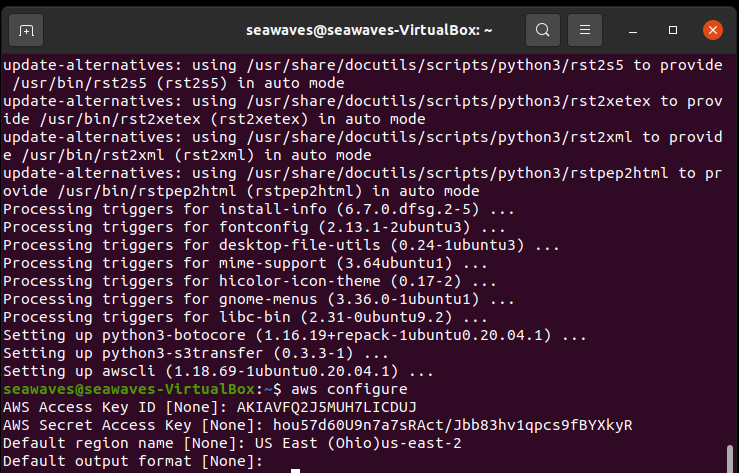
The terraform destroy command destroys all of the resources being managed by the current working directory and workspace, using state data to determine which real world objects correspond to managed resources. Like terraform apply, it asks for confirmation before proceeding.

A destroy behaves exactly like deleting every resource from the configuration and then running an apply, except that it doesn't require editing the configuration. This is more convenient if you intend to provision similar resources at a later date.

# Steps to build an infrastructure :

1. Configure the AWS CLI from your terminal. Follow the prompts to input your AWS Access Key ID and Secret Access Key.

# $ aws configure



1. Write configuration

The set of files used to describe infrastructure in Terraform is known as a Terraform configuration. You will write your first configuration to define a single AWS EC2 instance. Each Terraform configuration must be in its own working directory. Create a directory for your configuration.

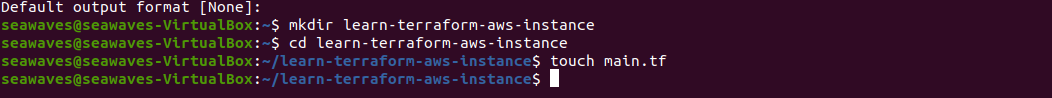
# $ mkdir learn-terraform-aws-instance

1. Change into the directory.

# $ cd learn-terraform-aws-instance

1. Create a file to define your infrastructure.

# $ touch main.tf



Open main.tf in your text editor, paste in the configuration below, and save the file.

terraform { required\_providers { aws = {

source = "hashicorp/aws" version = "~> 3.27"

}

}

required\_version = ">= 0.14.9"

}

provider "aws" { profile = "default" region = "us-west-2"

}

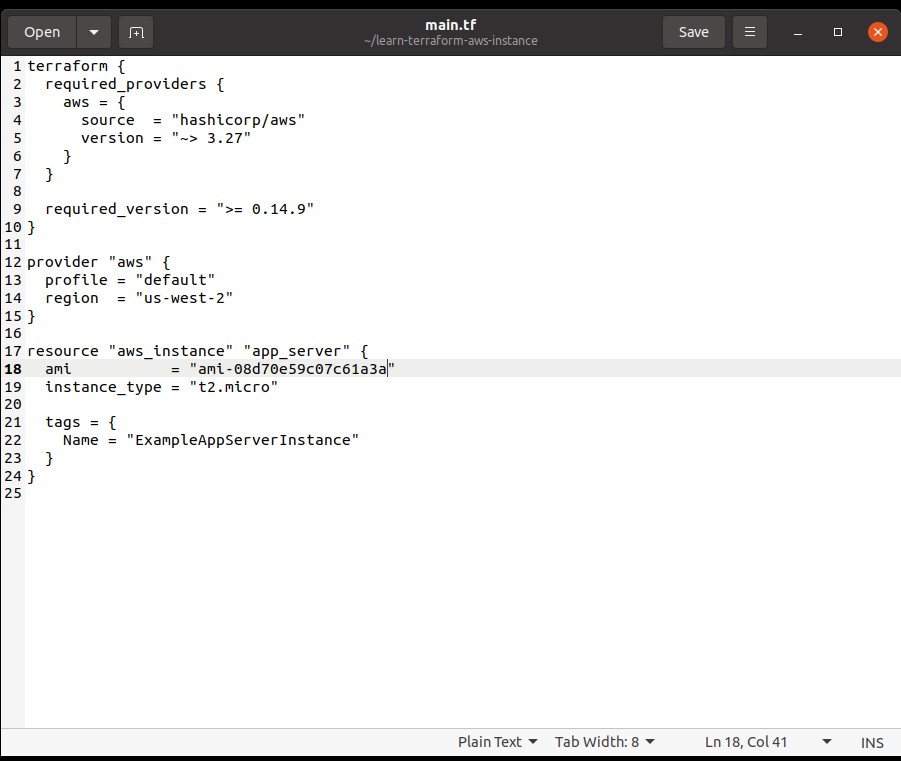
resource "aws\_instance" "app\_server" { ami = "ami-830c94e3" instance\_type = "t2.micro"

tags = {

Name = "ExampleAppServerInstance"

}

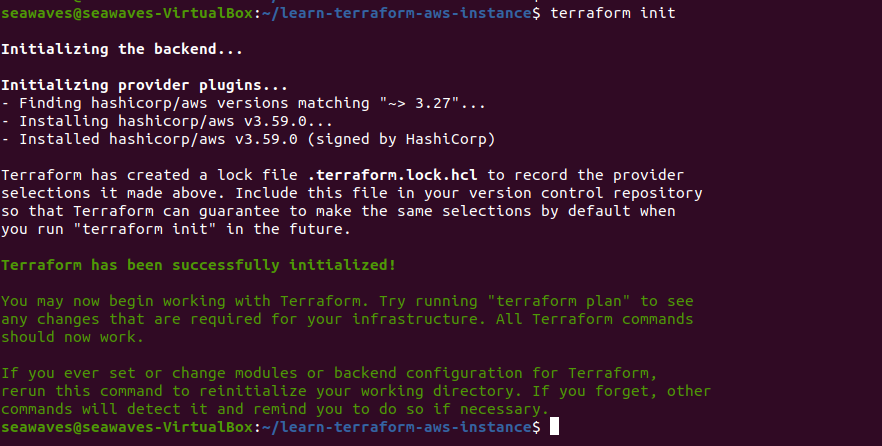
}



1. Initialize the directory

When you create a new configuration — or check out an existing configuration from version control — you need to initialize the directory with terraform init. Initializing a configuration directory downloads and installs the providers defined in the configuration, which in this case is the aws provider.

# $ terraform init



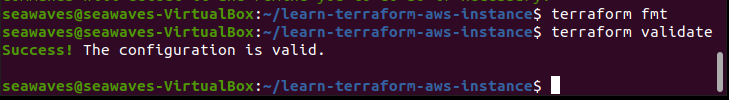
1. Format and validate the configuration

Format your configuration. Terraform will print out the names of the files it modified, if any. In this case, your configuration file was already formatted correctly, so Terraform won't return any file names.

# $ terraform fmt

1. Validate your configuration. The example configuration provided above is valid, so Terraform will return a success message.

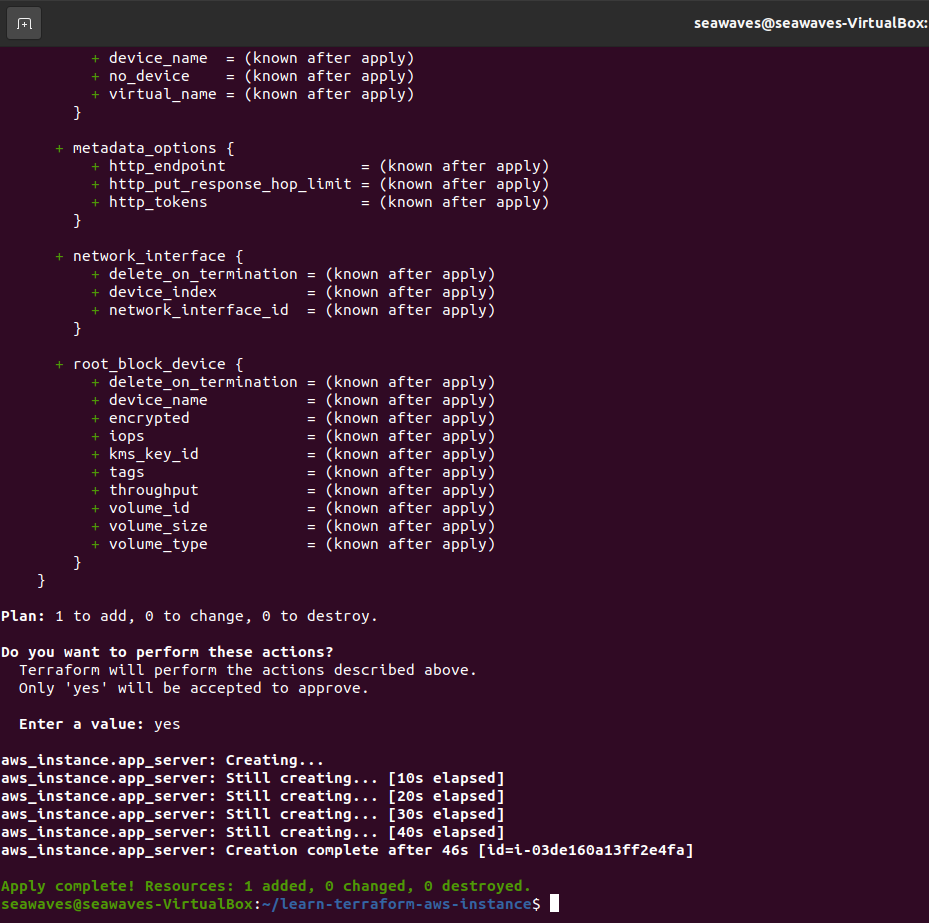
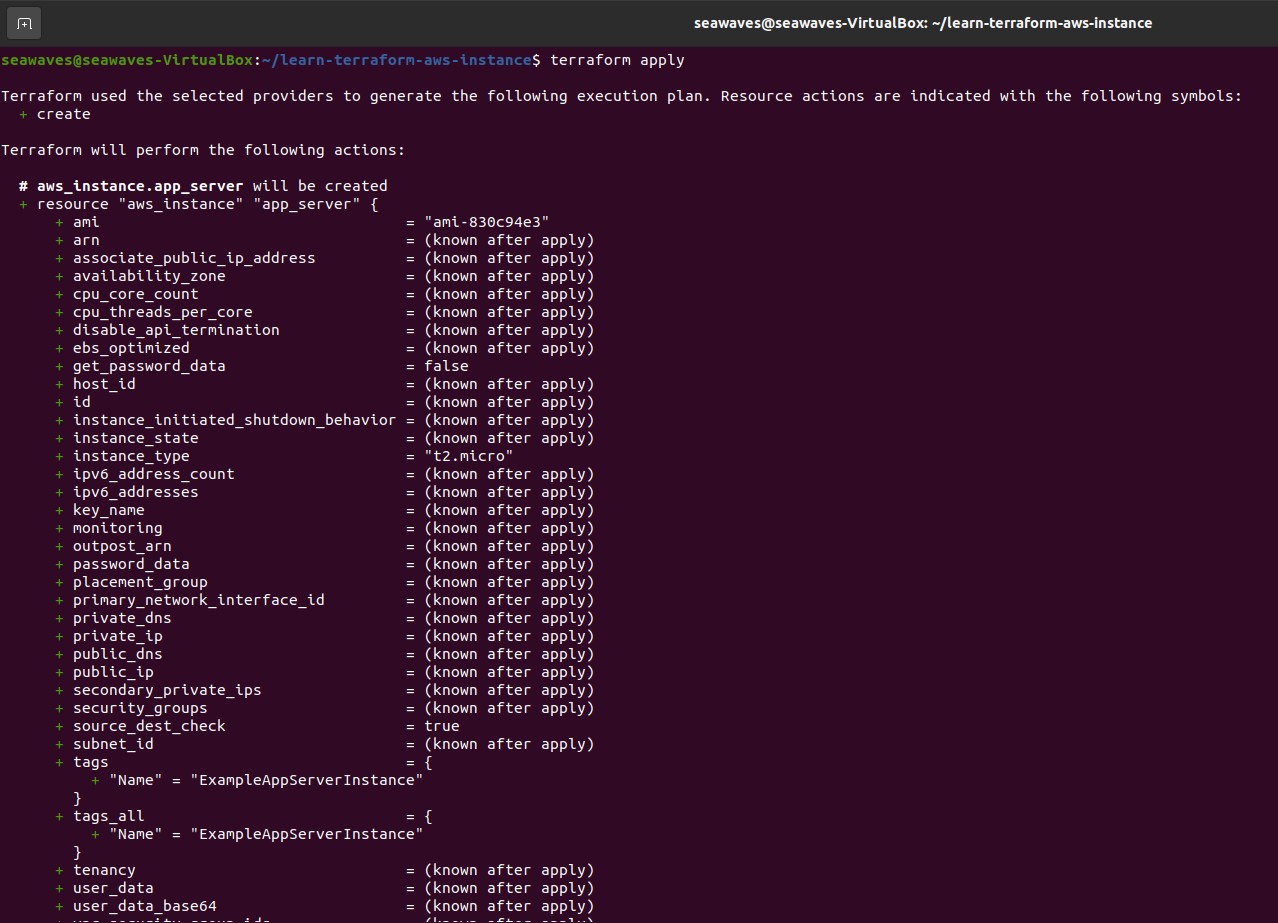
# $ terraform validate

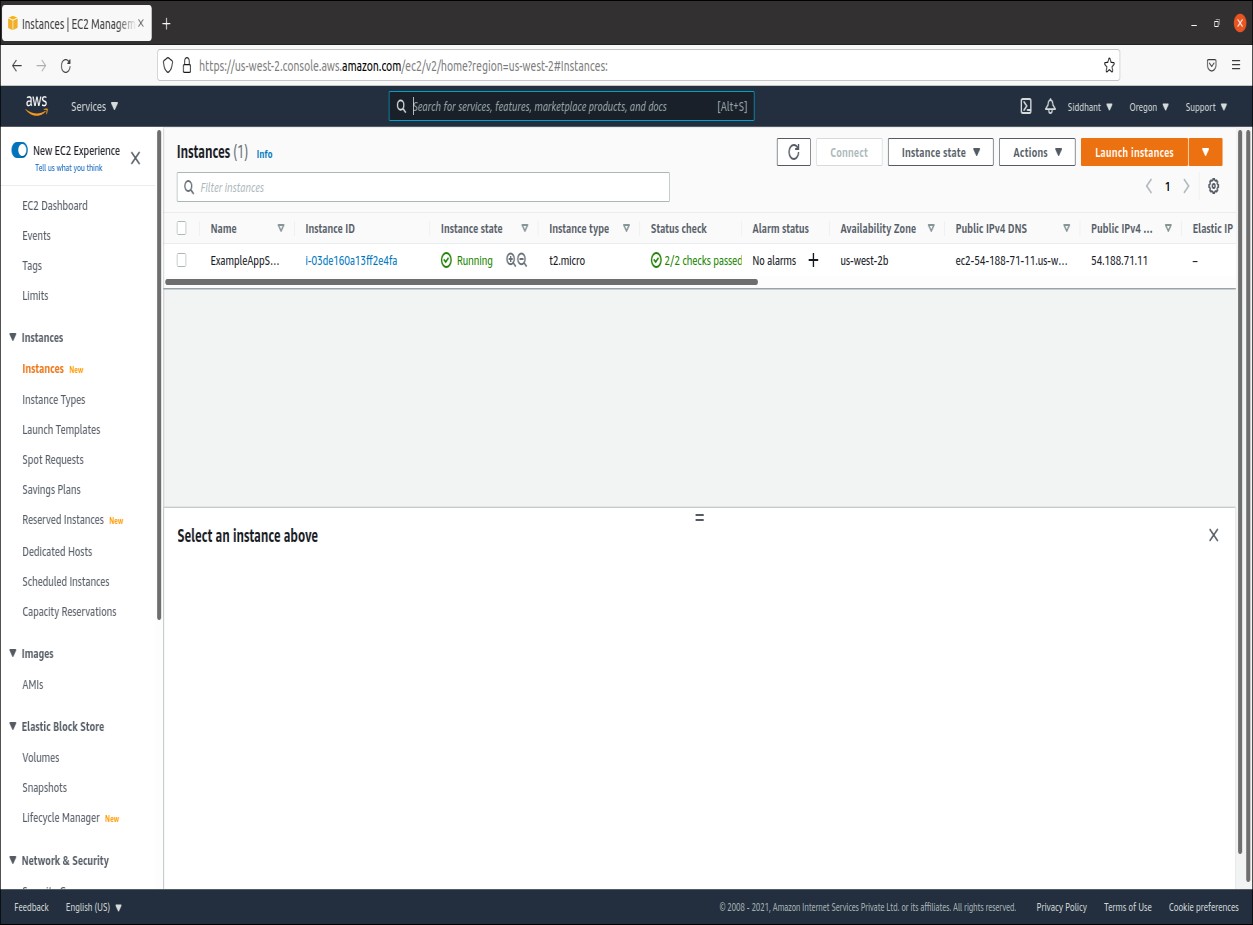


1. Create infrastructure

Apply the configuration now with the terraform apply command. Terraform will print output similar to what is shown below. We have truncated some of the output to save space.

# $ terraform apply





* **Steps to change an infrastructure :**

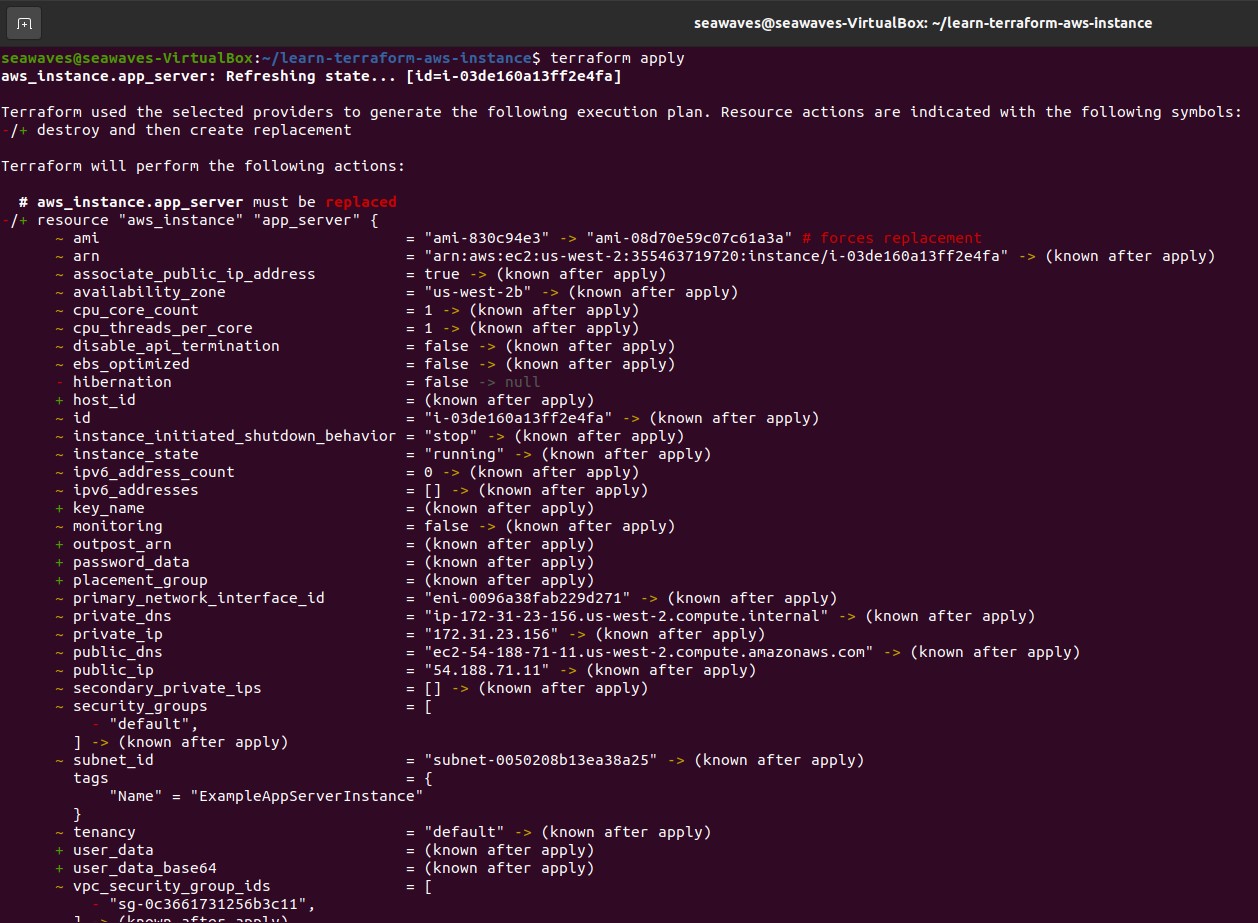
1. Configuration

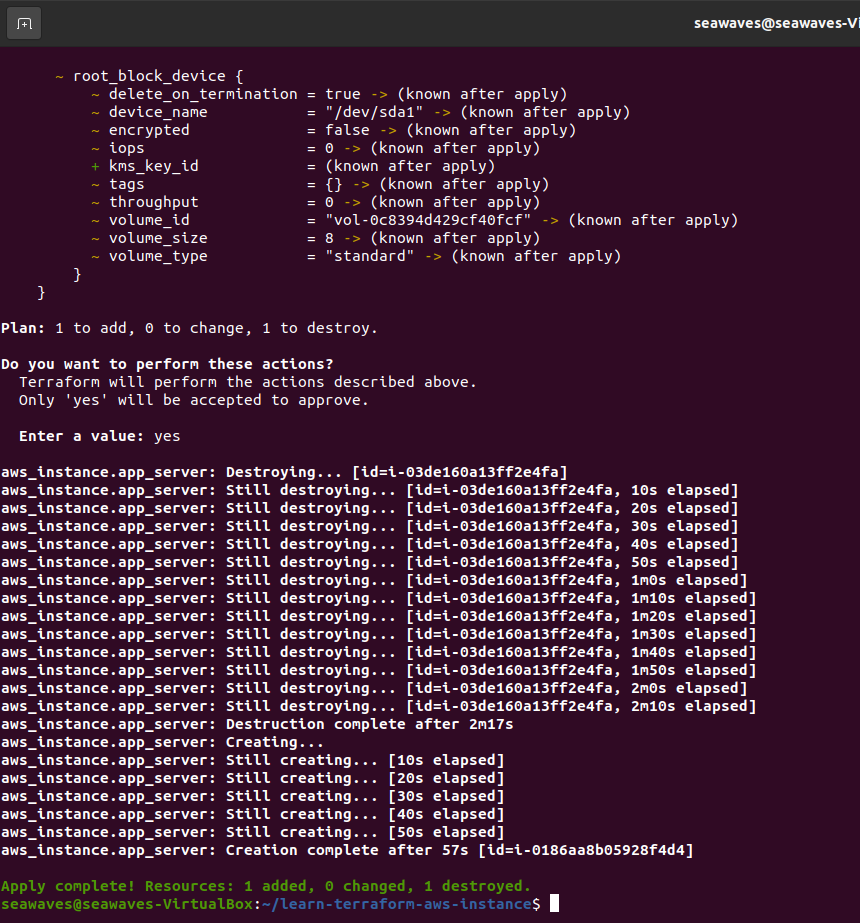
Now update the ami of your instance. Change the aws\_instance.app\_server resource under the provider block in main.tf by replacing the current AMI ID with a new one. Replace "ami- 830c94e3" with "ami-08d70e59c07c61a3a".

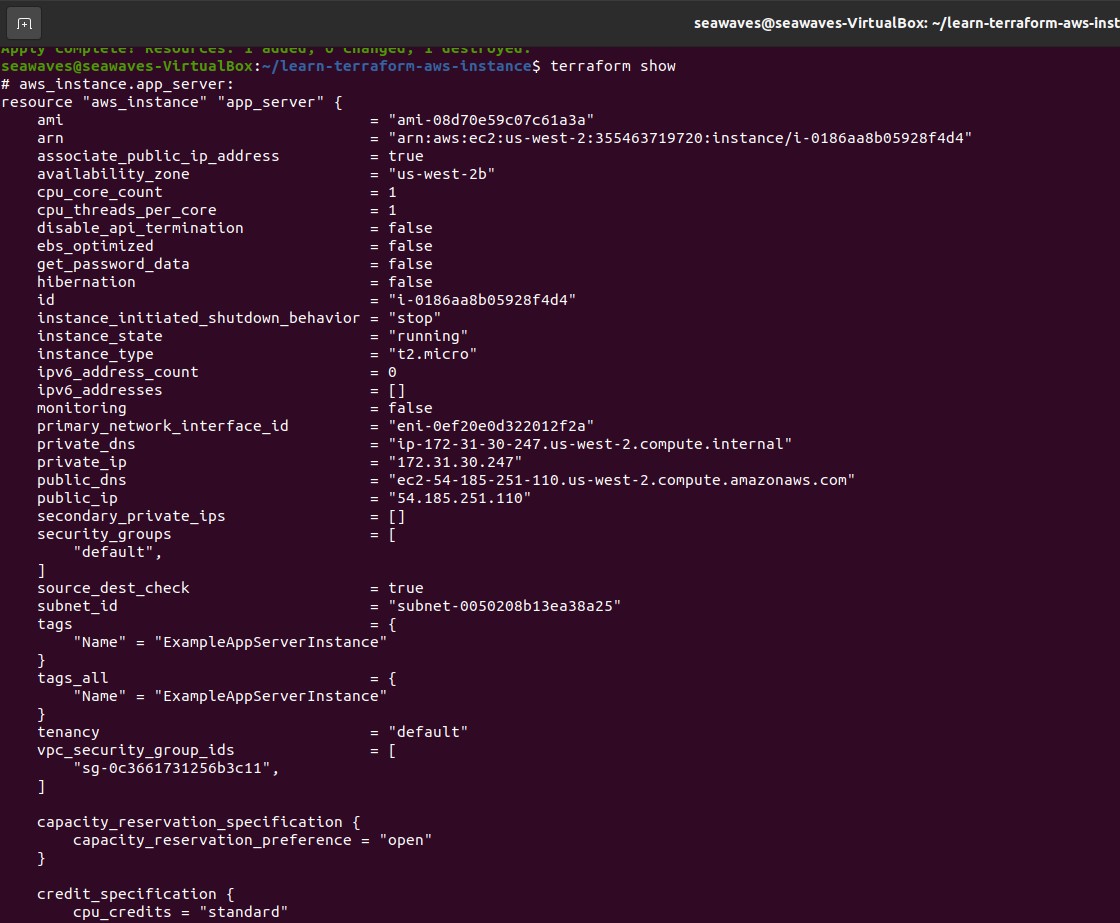
1. Apply Changes

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

# $ terraform apply



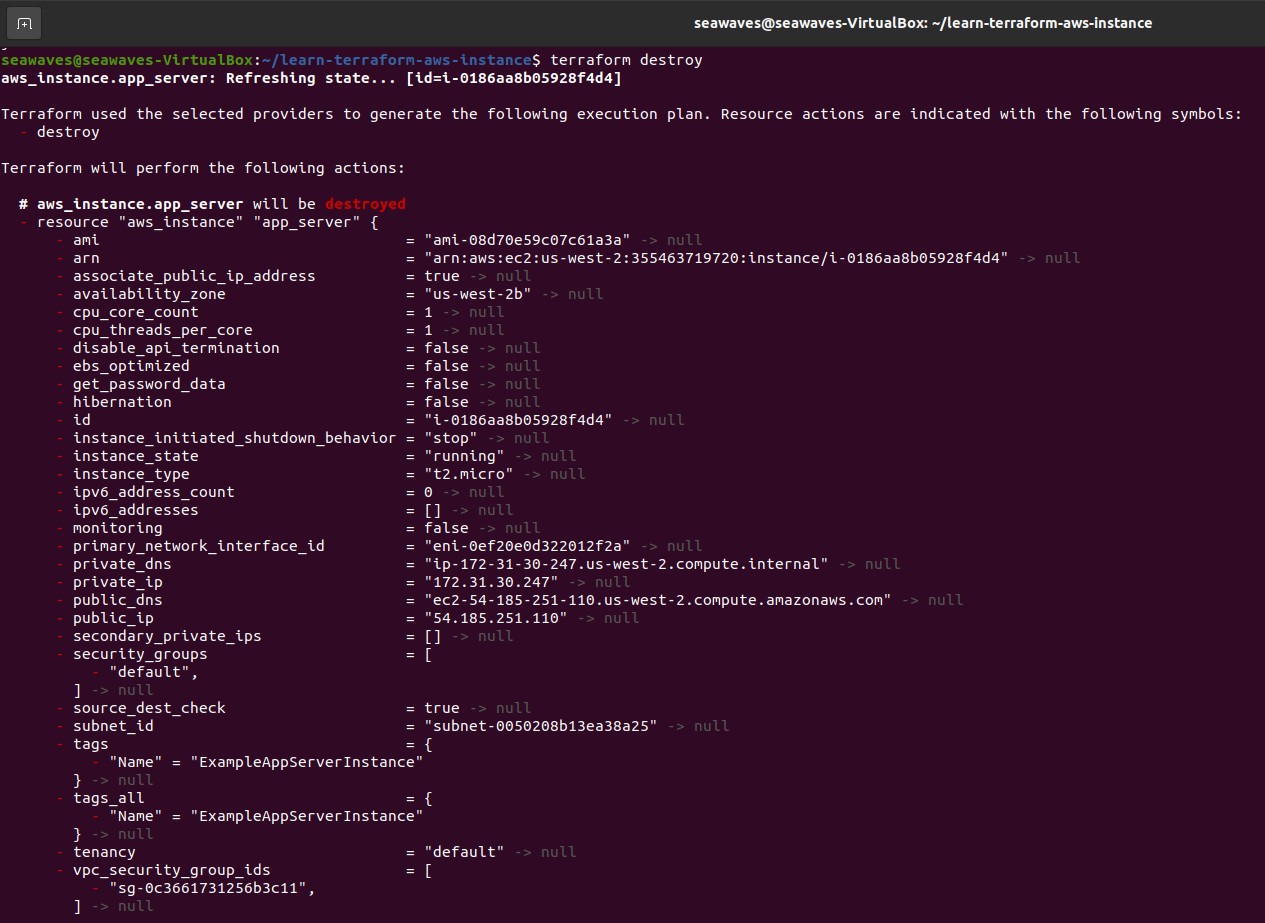


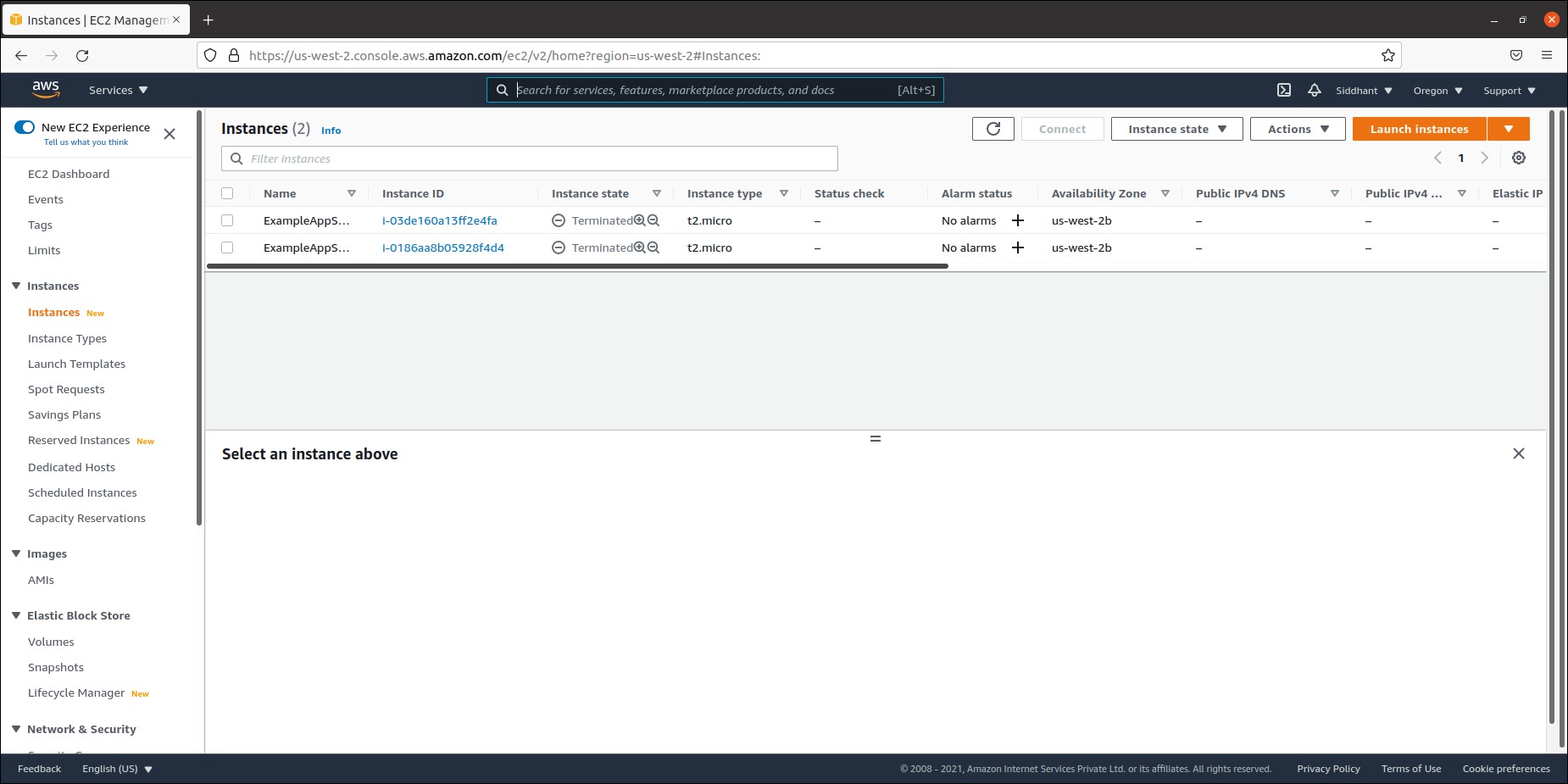


* **Steps to destroy an infrastructure :**

The terraform destroy command terminates resources managed by your Terraform project. This command is the inverse of terraform apply in that it terminates all the resources specified in your Terraform state. It does not destroy resources running elsewhere that are not managed by the current Terraform project.

# $ terraform destroy





**Conclusion :**

Successfully built, changed and destroyed an AWS infrastructure using terraform.