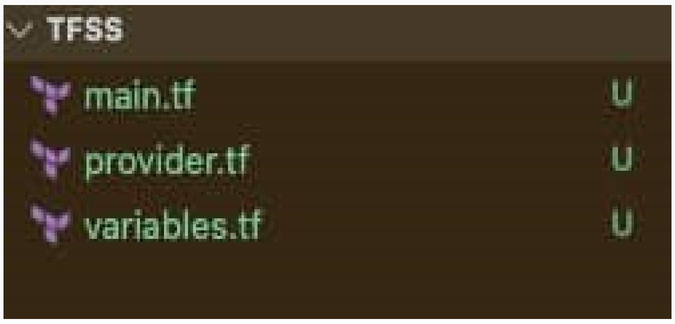
Terraform using AWS provider

we will be using AWS infrastructure during this article, perform the steps below:

1. Install AWS CLI
2. Use an AWS account with credentials that have the rights to create infrastructure on AWS
3. Login to AWS using the AWS CLI – Terraform will use this AWS account to perform its tasks
4. let’s step through the file structure used in Terraform projects with an example. Create a directory in a path of your choice. The example below uses the “terraform” directory. Then, create three files as below.



Terraform project file structure

1. **main.tf** contains the IAC that describes the AWS infrastructure we intend to create
2. **provider.tf** contains the AWS provider configuration. This file can contain multiple provider configs if working in multi-cloud or multi-account environments

3. **Variables.tf** is used to manage any variables used within the Terraform project

   Have defined a code block with the name terraform and have listed aws in the required_providers . We have also created a block that defines the region we want to use – a full list of AWS regions is available at in code which is save as provider.tf file

```
terraform {
 required_providers {
  aws = {
    source  = "hashicorp/aws"
    version = "~> 3.0"
     }
  }
}
provider "aws" {
region = var.region
}
```

The **region** attribute gets its value from a region variable, and we have used **var.region** in the example above. We defined this variable in the **variables.tf file**

```
variable "region" {
  type        = string
  description = "AWS East Region"
  default     = "us-east-1"
}
```

To Create an EC2 Instance open the **main.tf** file and define your EC2 instance as code.

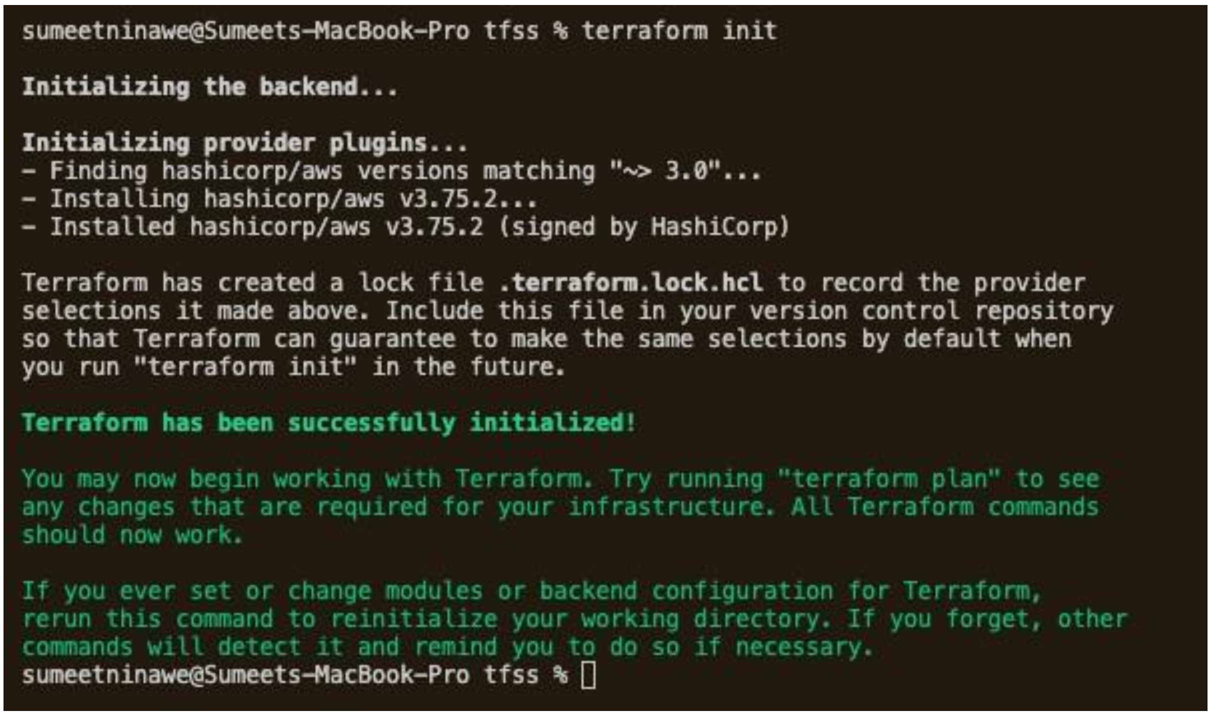
```
resource "aws_instance" "my_vm" {
 ami                       = var.ami
 instance_type             = var.instance_type
  tags = {
   Name = "My EC2 instance"
 }
}
```

At the moment our **main.tf** file contains one code block named **resource**. The resource type we have defined is an **aws_instance**, an EC2 instance, and has been named **my_vm**. This name is the EC2 instance reference within our Terraform project. We can use this block to define attributes such as **ami, instance_type**, and **Name** tag.

```
variable "region" {
type        = string
 description = "AWS East Region"
  default     = "us-east-1"
}
```

```
variable "ami" {
 type        = string
  description = "Ubuntu AMI ID in N. Virginia Region"
   default     = "ami-09d56f8956ab235b3"
}


variable "instance_type" {
   type        = string
   description = "Instance type"
   default     = "t2.micro"
}
```

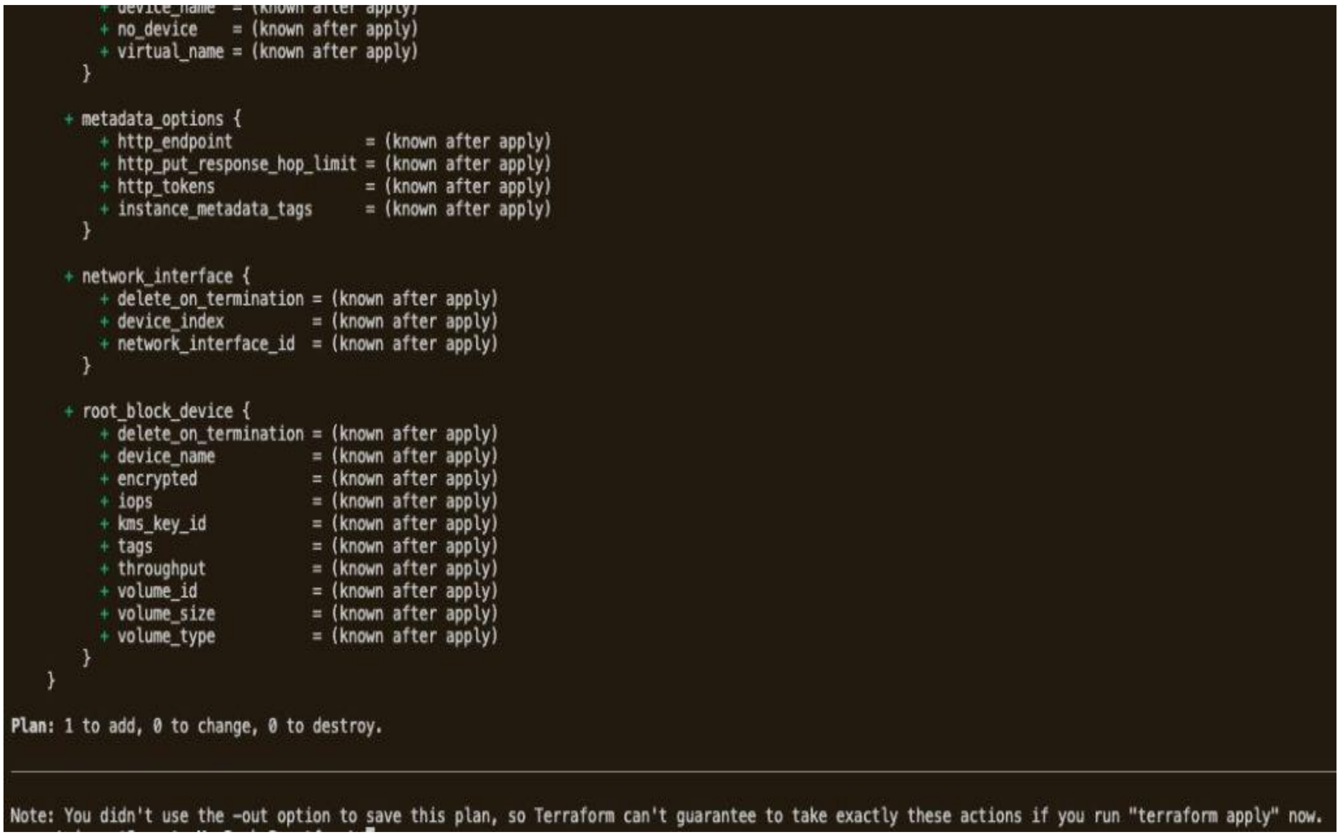
Now, open your terminal in the same directory and initialize Terraform by running terraform init. The output should look like this



```
sumeetninawe@Sumeets-MacBook-Pro tfss % terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 3.0"...
- Installing hashicorp/aws v3.75.2...
- Installed hashicorp/aws v3.75.2 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
sumeetninawe@Sumeets-MacBook-Pro tfss % 
```

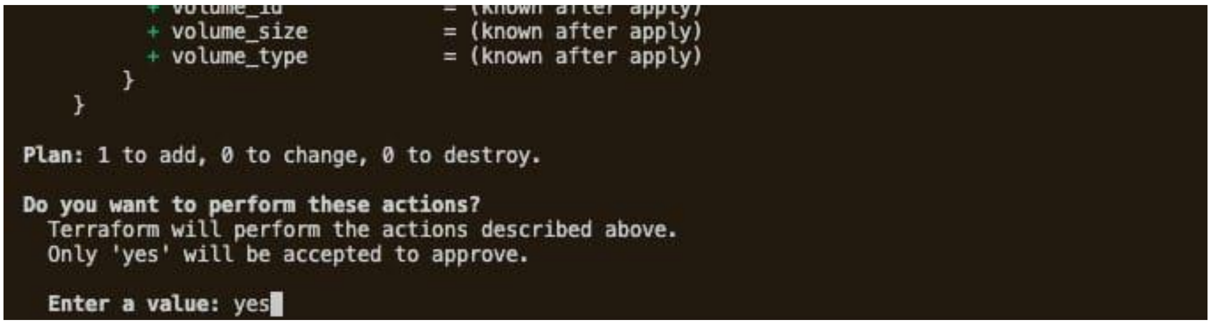
Terminal output on successful initiation of Terraform

We can see below that Terraform has installed the AWS plugin and has initialized the project. To recap, we have written IaC that will create an EC2 instance called **my_vm**. To double-check that Terraform can do this, run the terraform plan command to perform a dry-run.



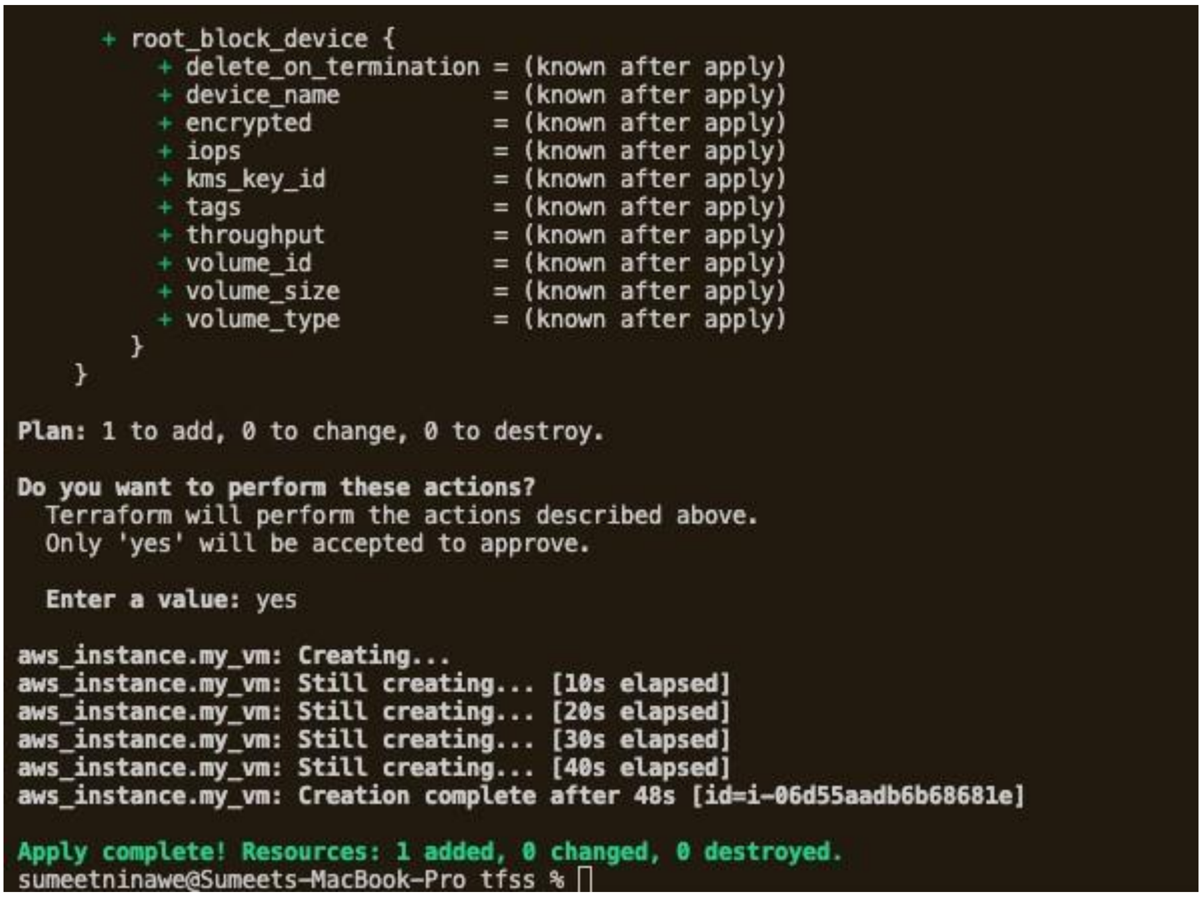
```
          + device_name  = (known after apply)
          + no_device    = (known after apply)
          + virtual_name = (known after apply)
        }

      + metadata_options {
          + http_endpoint               = (known after apply)
          + http_put_response_hop_limit = (known after apply)
          + http_tokens                 = (known after apply)
          + instance_metadata_tags      = (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 = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
    }

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

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
```

let’s go ahead and run terraform apply to deploy the EC2 instance. When you run the apply command, Terraform asks for your confirmation. To confirm, type yes and hit enter.



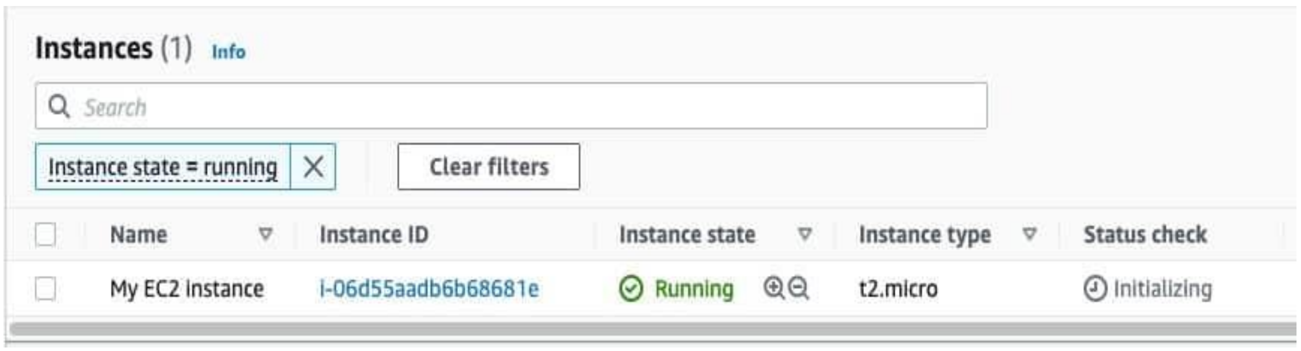
```
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
    }

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
```

Terraform will now create the EC2 instance using the supplied AWS account. The deployment status will update every 10 seconds and notify us when deployment is complete. In our case, it took 48 seconds



```
      + root_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after 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

aws_instance.my_vm: Creating...
aws_instance.my_vm: Still creating... [10s elapsed]
aws_instance.my_vm: Still creating... [20s elapsed]
aws_instance.my_vm: Still creating... [30s elapsed]
aws_instance.my_vm: Still creating... [40s elapsed]
aws_instance.my_vm: Creation complete after 48s [id=i-06d55aadb6b68681e]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
sumeetninawe@Sumeets-MacBook-Pro tfss %
```

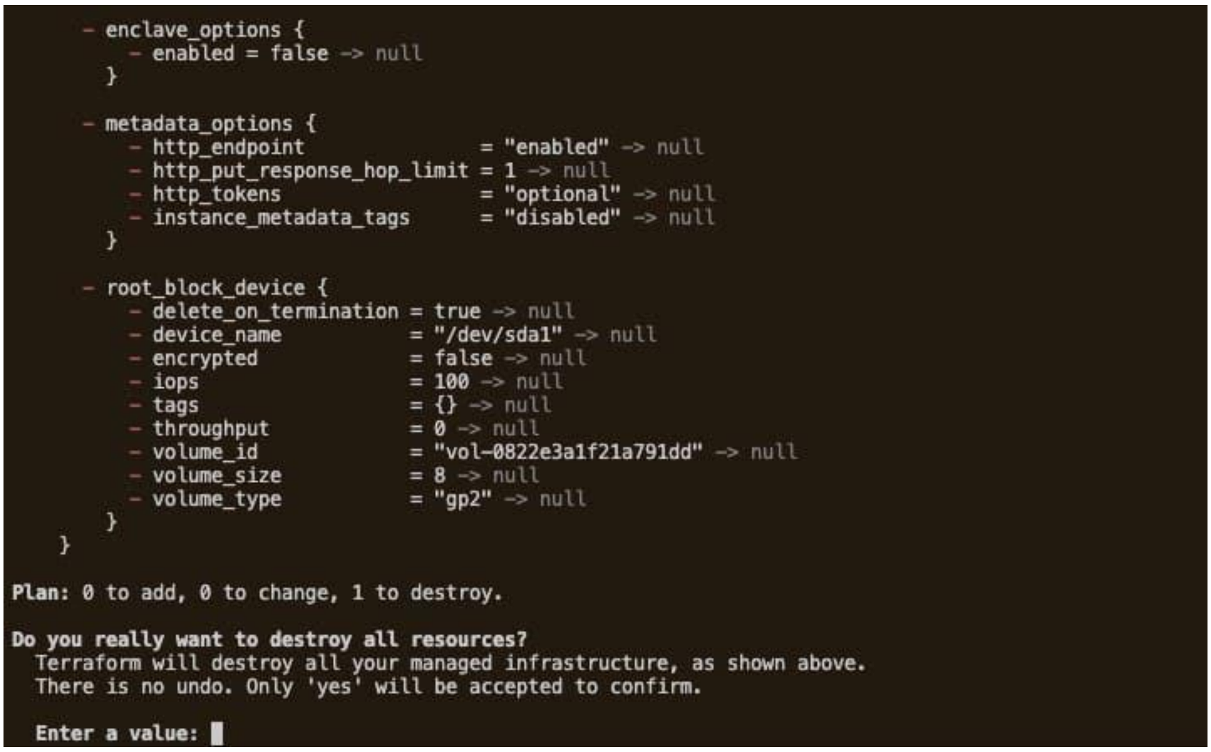
To be sure, log into your AWS account and confirm the deployment.



Name	Instance ID	Instance state	Instance type	Status check
My EC2 instance	i-06d55aadb6b68681e	Running	t2.micro	Initializing

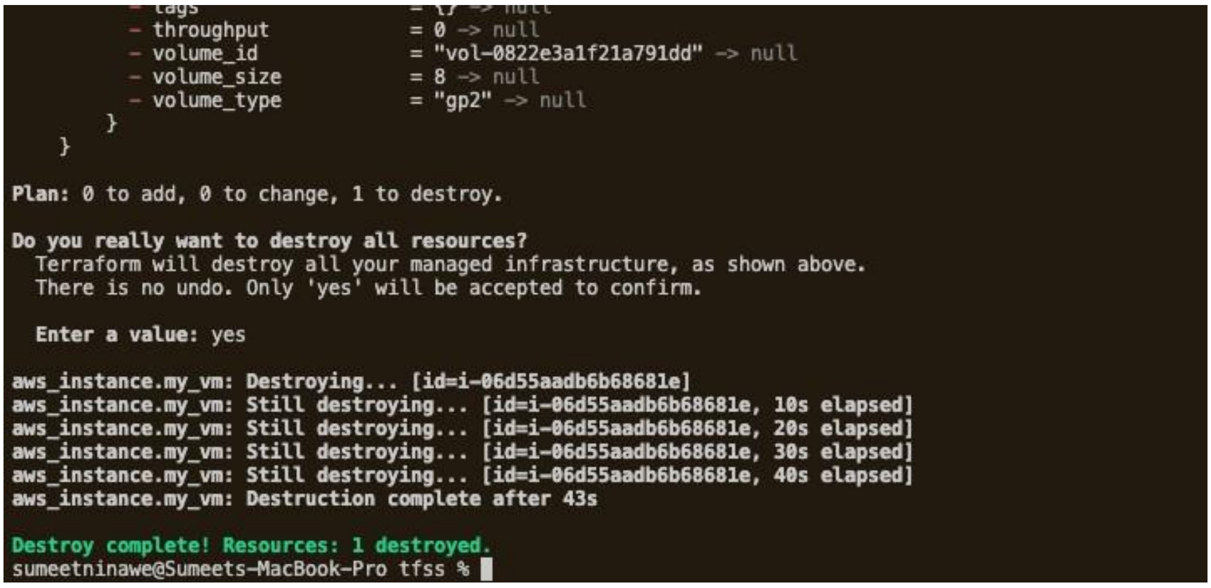
Destroying an EC2

Terraform is now managing our EC2 instance – and we didn’t even need to log into the AWS console! If we wish, we can also destroy the instance. To do so, just run terraform destroy.



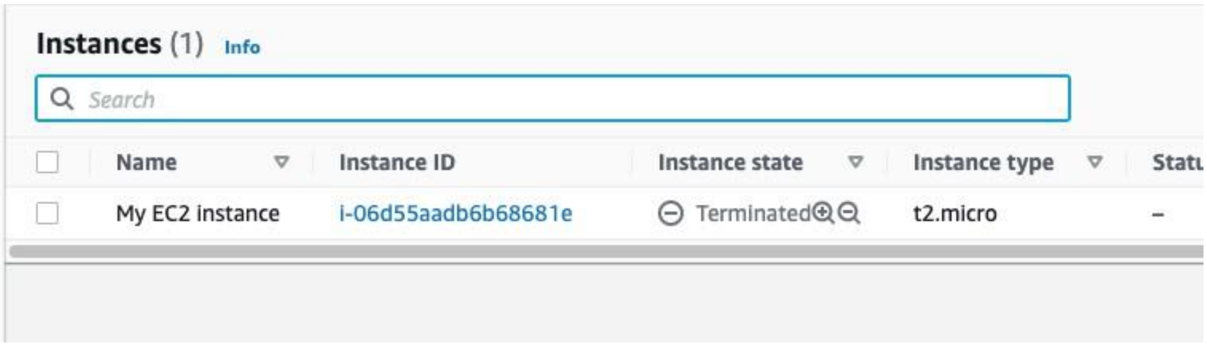
```
      - enclave_options {
          - enabled = false -> null
        }

      - metadata_options {
          - http_endpoint               = "enabled" -> null
          - http_put_response_hop_limit = 1 -> null
          - http_tokens                 = "optional" -> null
          - instance_metadata_tags      = "disabled" -> null
        }

      - root_block_device {
          - delete_on_termination = true -> null
          - device_name           = "/dev/sda1" -> null
          - encrypted             = false -> null
          - iops                  = 100 -> null
          - tags                  = {} -> null
          - throughput            = 0 -> null
          - volume_id             = "vol-0822e3a1f21a791dd" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp2" -> null
        }
    }

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: 
```

When we do this, Terraform asks for final confirmation and provides a summary of the resources it will delete.
Type **yes** and hit enter. As shown below, Terraform will run the destroy operation immediately. As with the creation process, it will keep us updated on its progress.



```
        - throughput              = 0 -> null
        - volume_id               = "vol-0822e3a1f21a791dd" -> null
        - volume_size             = 8 -> null
        - volume_type             = "gp2" -> null
      }
  }

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.my_vm: Destroying... [id=i-06d55aadb6b68681e]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 10s elapsed]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 20s elapsed]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 30s elapsed]
aws_instance.my_vm: Still destroying... [id=i-06d55aadb6b68681e, 40s elapsed]
aws_instance.my_vm: Destruction complete after 43s

Destroy complete! Resources: 1 destroyed.
sumeetninawe@Sumeets-MacBook-Pro tfss %
```

To confirm that the deletion happened, go back to your AWS console.



Instances (1) Info

Name	Instance ID	Instance state	Instance type	Statu
My EC2 instance	i-06d55aadb6b68681e	Terminated	t2.micro	–

Conclusion

This was a quick and practical introduction, allowing you to get started with Terraform. We covered a range of basic (and not so basic) concepts,