Terraform:

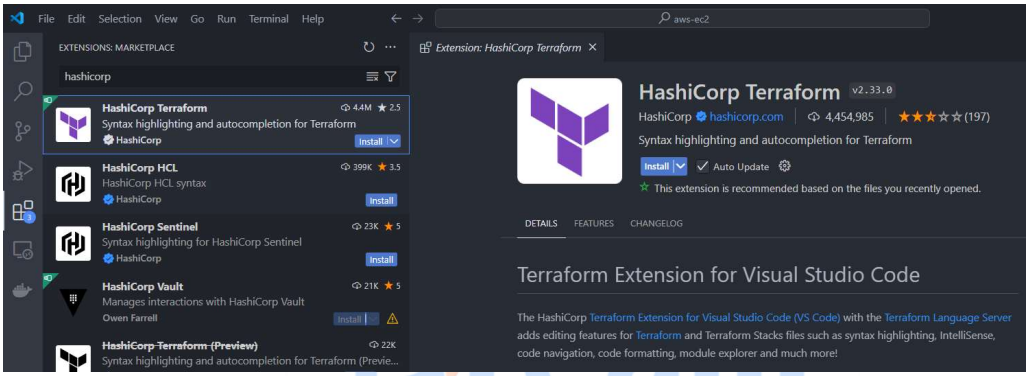
- Terraform is a leading IaC tool that allows developers and operations teams to define and manage infrastructure through code.
- Download and do the path set for Terraform. After path set, check Terraform Version using **terraform version.**

```
C:\Users\sony>terraform version
Terraform v1.9.8
on windows_386

C:\Users\sony>
```

Launch AWS EC2 instance with Terraform.

1. Install Terraform extension on VS Code.



2. Create a main.tf file.
3. Open google.com and search for terraform.
4. Open the link https://www.terraform.io/.
5. Click on **Registry**.
6. Click on **Browse Providers**.
7. Click on **AWS**.
8. Click on **Use Provider**.
9. Copy the code and paste in main.tf file.

Sample main.tf file:

```
terraform {
 required_providers {
   aws = {
    source = "hashicorp/aws"
    version = "5.74.0"
   }
 }
}
provider "aws" {
   region = "ap-south-1"
}
resource "aws_instance" "ec2-server" {
   ami = "ami-04a37924ffe27da53"
   instance_type = "t2.micro"
```

```
  tags = {
    Name = "vm-1"
  }
}
```

terraform init:

- Run the command **terraform init** on the folder where **main.tf** file is present.
- The terraform init command initializes a working directory for Terraform.
- It's the first command that should be run after writing or cloning a Terraform configuration.
- You can run terraform init multiple times.
- If you change modules or backend configuration, you should rerun the command to reinitialize your working directory.
- After running terraform init, you can run other commands like terraform plan and terraform apply.

Here are some things the terraform init command does:

- Backend initialization: Consults the root configuration directory to initialize the chosen backend.
- Child module installation: Searches for module blocks and retrieves source code for referenced modules.
- Provider plugin download: Downloads the necessary provider plugins and ensures they are available for use.
- State management: Keeps track of the resources that are being managed.
- Dependency resolution: Resolves dependencies on external modules or remote sources.

```
D:\DevOps\Terraform\TF-AWS\aws-ec2>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.74.0"...
- Installing hashicorp/aws v5.74.0...
- Installed hashicorp/aws v5.74.0 (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.

D:\DevOps\Terraform\TF-AWS\aws-ec2>
```

terraform plan:

The terraform plan command creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure. By default, when Terraform creates a plan it:

- Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.
- Compares the current configuration to the prior state and noting any differences.
- Proposes a set of change actions that should, if applied, make the remote objects match the configuration.

```
D:\DevOps\Terraform\TF-AWS\aws-ec2>terraform plan

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.ec2-server will be created
  + resource "aws_instance" "ec2-server" {
      + ami                                  = "ami-04a37924ffe27da53"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
```

terraform apply:

- The terraform apply command in Terraform carries out the actions proposed in a Terraform plan.
- This includes creating, updating, or deleting infrastructure resources to match the new state outlined in your Infrastructure-as-Code (IaC) configuration.
- By default, terraform apply asks for confirmation from the user before making any changes.

```
D:\DevOps\Terraform\TF-AWS\aws-ec2>terraform apply

Terraform used the selected providers to generate the following execution
 plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.ec2-server will be created
  + resource "aws_instance" "ec2-server" {
      + ami                                  = "ami-04a37924ffe27da53"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
```

Resource Change:

1. Change the instance type from t2.micro to t2.nano. It will modify the existing instance.

```
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.ec2-server: Modifying... [id=i-0b9d7539fb3c84ae3]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 10s elapsed]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 20s elapsed]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 30s elapsed]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 40s elapsed]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 50s elapsed]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 1m0s elapsed]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 1m10s elapsed]
aws_instance.ec2-server: Still modifying... [id=i-0b9d7539fb3c84ae3, 1m20s elapsed]
aws_instance.ec2-server: Modifications complete after 1m23s [id=i-0b9d7539fb3c84ae3]

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

D:\DevOps\Terraform\TF-AWS\aws-ec2>
```

2. Change the AMI ID to ami-0dee22c13ea7a9a67. It will destroy the current instance first, then it will launch a new instance.
   The terraform.tfstate file maintains a detailed record of the current state of managed resources.

```
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.ec2-server: Destroying... [id=i-0b9d7539fb3c84ae3]
aws_instance.ec2-server: Still destroying... [id=i-0b9d7539fb3c84ae3, 10s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0b9d7539fb3c84ae3, 20s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0b9d7539fb3c84ae3, 30s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0b9d7539fb3c84ae3, 40s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0b9d7539fb3c84ae3, 50s elapsed]
aws_instance.ec2-server: Destruction complete after 50s
aws_instance.ec2-server: Creating...
aws_instance.ec2-server: Still creating... [10s elapsed]
aws_instance.ec2-server: Creation complete after 13s [id=i-0a7392bb5b9409668]

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

D:\DevOps\Terraform\TF-AWS\aws-ec2>
```

terraform apply -auto-approve:
We can use the auto approve flag to skip the interactive approval process. It will not ask any intermittent input from user.

Resource Destroy:
The **terraform destroy** command is a convenient way to destroy all remote objects managed by a particular Terraform configuration.

```
aws_instance.ec2-server: Destroying... [id=i-0a7392bb5b9409668]
aws_instance.ec2-server: Still destroying... [id=i-0a7392bb5b9409668, 10s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0a7392bb5b9409668, 20s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0a7392bb5b9409668, 30s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0a7392bb5b9409668, 40s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0a7392bb5b9409668, 50s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0a7392bb5b9409668, 1m0s elapsed]
aws_instance.ec2-server: Still destroying... [id=i-0a7392bb5b9409668, 1m10s elapsed]
aws_instance.ec2-server: Destruction complete after 1m11s

Destroy complete! Resources: 1 destroyed.
```

terraform validate – This command validates the syntax written on the configuration file.

```
D:\DevOps\Terraform\TF-AWS\aws-ec2>terraform validate
Success! The configuration is valid.


D:\DevOps\Terraform\TF-AWS\aws-ec2>
```

Example#1 – Increase the volume size and Make multiple instances of same type:
When you attach a volume to your instance, you include a device name for the volume. This device name is used by Amazon EC2.
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html#available-ec2-device-names

```
terraform {
 required_providers {
   aws = {
     source = "hashicorp/aws"
     version = "5.74.0"
   }
 }
}
provider "aws" {
   region = "ap-south-1"
}
resource "aws_instance" "ec2-server" {
   ami = "ami-08bf489a05e916bbd"
   instance_type = "t2.micro"
   tags = {
      Name = "tf-vm-1"
   }
   ebs_block_device {
     device_name = "/dev/xvda"
     volume_size = 10
     volume_type = "gp3"
   }
   count = 2
}
```

Example #2 – Select a particular instance to delete:
Fetch the list of instances by using:
terraform state list

```
aws_instance.app-server[0]
aws_instance.web-server[0]
```

Delete a selected instance by using:
terraform destroy -target aws_instance.app-server[0] -auto-approve

Example #3: Create multiple instances of different instance type:

```
terraform {
 required_providers {
   aws = {
     source = "hashicorp/aws"
     version = "5.74.0"
   }
 }
}
```

```
provider "aws" {
   region = "ap-south-1"
}
resource "aws_instance" "web-server" {
   ami = "ami-08bf489a05e916bbd"
   instance_type = "t2.micro"
   tags = {
      Name = "web"
   }
   ebs_block_device {
    device_name = "/dev/xvda"
    volume_size = 9
    volume_type = "gp3"
   }
   count = 1
}

resource "aws_instance" "app-server" {
   ami = "ami-08bf489a05e916bbd"
   instance_type = "t2.nano"
   tags = {
      Name = "app"
   }
   ebs_block_device {
    device_name = "/dev/xvda"
    volume_size = 9
    volume_type = "gp3"
   }
   count = 1
}
```

Example#4: assign security group

```
terraform {
  required_providers {
   aws = {
    source = "hashicorp/aws"
    version = "5.74.0"
   }
  }
}
provider "aws" {
   region = "ap-south-1"
}
resource "aws_instance" "web-server" {
   ami = "ami-08bf489a05e916bbd"
   instance_type = "t2.micro"
   tags = {
      Name = "web"
```

```
  }
  ebs_block_device {
   device_name = "/dev/xvda"
   volume_size = 8
   volume_type = "gp3"
  }
  count = 1
  vpc_security_group_ids = [ "sg-0b0e46ba953101b16" ]
}
```