SYSTEM PROVISIONING AND CONFIGURATION MANAGEMENT

ASSIGNMENT 1

Anmol Ahuja

—

500061317 | R171217009

—

CSE-DEVOPS-VI

INSTALLATION

Use the following commands to install Terraform on your Ubuntu based Linux system, more information on other OS here- https://www.terraform.io/downloads.html

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -


sudo apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb_release -cs) main"


sudo apt install terraform
```

AUTHENTICATION

For AWS, the quickest and mostly secure way is to use AWS CLI tool which you can download from here https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html.
After installation run the following command to enter your user access key and secret access key.

$ aws configure
# Output
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json

STEPS

1. When you create a new configuration or make changes to the providers block in your .tf files; you need to initialize the directory with 'terraform init'.

2. Initializing a directory or sub folder downloads and installs providers used in the configuration files, which in this case is the aws provider. I have already installed aws provider files so my output looks like this.

```
[anmol@fedora terraform_files]$ terraform init

Initializing the backend...

Initializing provider plugins...
- Using previously-installed hashicorp/aws v2.70.0

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.
[anmol@fedora terraform_files]$
```

3. Now we have to write our .tf files in the same directory so that we can create our own infrastructure.

```
terraform { # give the version of aws provider to download
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 2.70"
    }
  }
}

provider "aws" { # it will take credentials from ~/.aws/credentials
  profile = "default"
  region  = "us-east-1"
}

resource "aws_instance" "example" {
  count         = 2                 # create 2 instances
  ami           = "ami-0885b1f6bd170450c" # Ubuntu based machine
```

```
  instance_type = "t2.micro"            # lowest and cheapest tier
  tags = {
    Name = "practice_${count.index + 1}" # name changed to practice 1 and 2
  }
}

resource "aws_s3_bucket" "buckety" { # create a S3 bucket
  bucket = "anmolahujabucket"     # name of the bucket
  acl    = "private"

  tags = {
    Name = "BIG BUCKET"
  }
}

# In AWS, to connect your network to a VPC securely you need a VPN
# A VPN requires a virtual private gateway on the VPC side
# and a customer gateway on your network side
# to create a virtual private gateway we need a VPC which we will create
# Then we will create a customer gateway
# Finally a VPN connection can be made using the above two resources

resource "aws_vpc" "vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_vpn_gateway" "vpn_gateway" {
  vpc_id = aws_vpc.vpc.id
}

resource "aws_customer_gateway" "customer_gateway" {
  bgp_asn    = 65000
  ip_address = "172.0.0.1"
  type       = "ipsec.1"
}

resource "aws_vpn_connection" "main" {
  vpn_gateway_id      = aws_vpn_gateway.vpn_gateway.id
  customer_gateway_id = aws_customer_gateway.customer_gateway.id
  type                = "ipsec.1"
  static_routes_only  = true
}
```

4. Now we will format and validate the configuration files using ‘terraform fmt’ and ‘terraform validate’. The terraform fmt command automatically updates configurations in the current directory for easy readability and consistency. Terraform will return the names of the files it formatted. To make sure your configuration files are syntactically correct the terraform validate command will check and report errors.

```
[anmol@fedora terraform_files]$ terraform fmt
main.tf
[anmol@fedora terraform_files]$ terraform validate
Success! The configuration is valid.

[anmol@fedora terraform_files]$
```

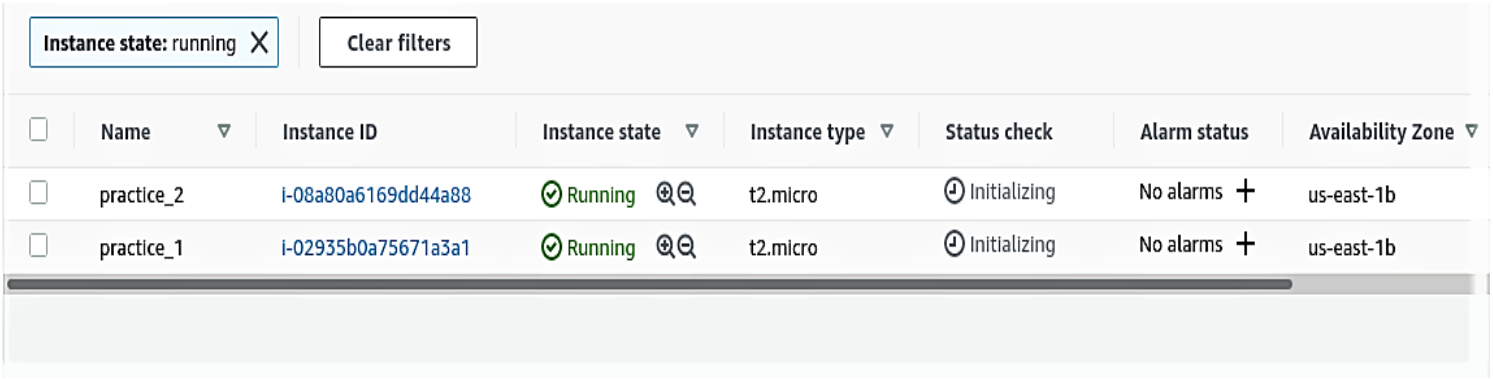
5. Then you need to just plan and apply the configuration, terraform plan command will show you what changes will be made to the infrastructure by checking with AWS what infrastructure is currently up so it does not create the same resources again. The terraform apply command will make the changes that plan showed and will prompt you for confirmation.

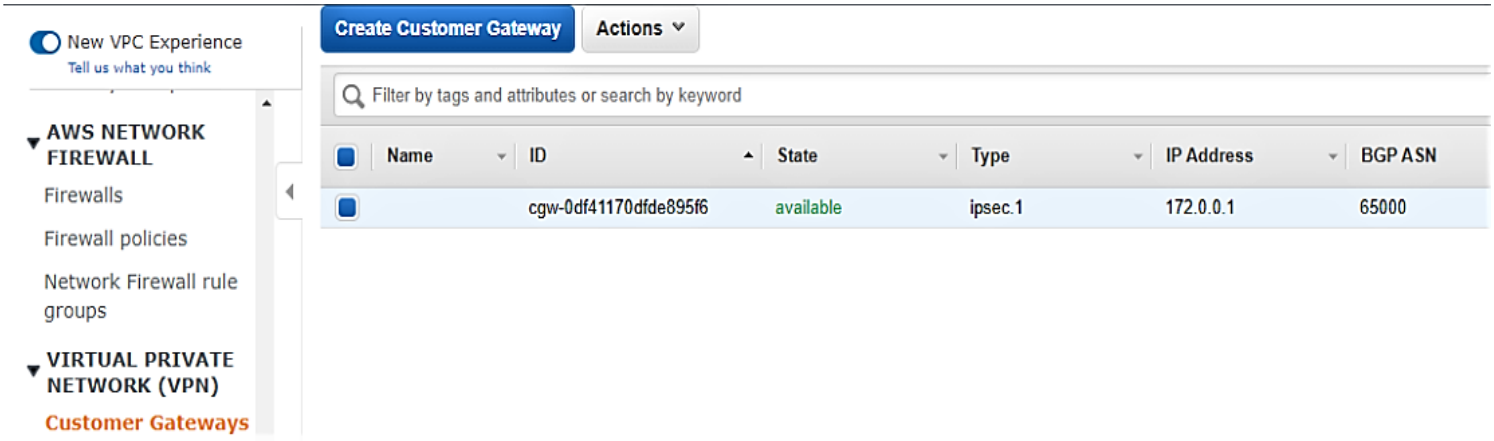
```
[anmol@fedora terraform_files]$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_customer_gateway.customer_gateway will be created
  + resource "aws_customer_gateway" "customer_gateway" {
      + arn        = (known after apply)
      + bgp_asn    = "65000"
      + id         = (known after apply)
      + ip_address = "172.0.0.1"
      + type       = "ipsec.1"
    }

  # aws_instance.example[0] will be created
  + resource "aws_instance" "example" {
      + ami                          = "ami-0885b1f6bd170450c"
      + 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)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)
      + key_name                     = (known after apply)
      + network_interface_id         = (known after apply)
      + outpost_arn                  = (known after apply)
      + password_data                = (known after apply)
      + placement_group              = (known after apply)
      + primary_network_interface_id = (known after apply)
```

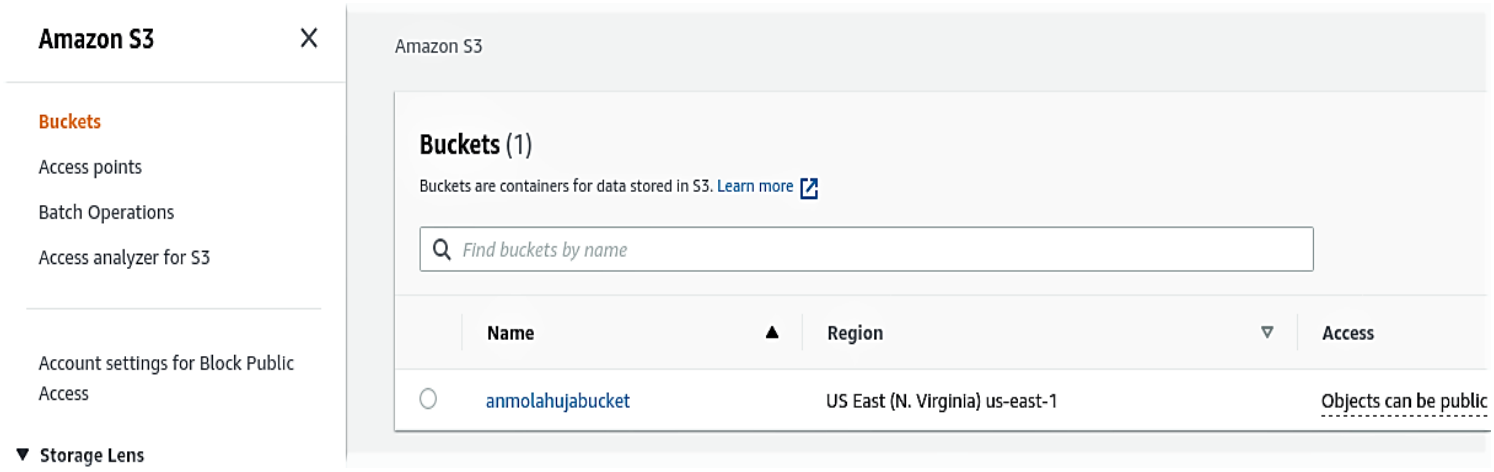
```
  # aws_vpn_gateway.vpn_gateway will be created
  + resource "aws_vpn_gateway" "vpn_gateway" {
      + amazon_side_asn = (known after apply)
      + arn             = (known after apply)
      + id              = (known after apply)
      + vpc_id          = (known after apply)
    }

Plan: 7 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.

[anmol@fedora terraform_files]$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_customer_gateway.customer_gateway will be created
  + resource "aws_customer_gateway" "customer_gateway" {
      + arn        = (known after apply)
      + bgp_asn    = "65000"
      + id         = (known after apply)
      + ip_address = "172.0.0.1"
      + type       = "ipsec.1"
    }
```

```
Plan: 7 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_s3_bucket.buckety: Creating...
aws_instance.example[0]: Creating...
aws_vpc.vpc: Creating...
aws_customer_gateway.customer_gateway: Creating...
aws_instance.example[1]: Creating...
aws_instance.example[0]: Still creating... [10s elapsed]
aws_vpc.vpc: Still creating... [10s elapsed]
aws_instance.example[1]: Still creating... [10s elapsed]
aws_instance.example[0]: Still creating... [20s elapsed]
aws_vpc.vpc: Still creating... [20s elapsed]
aws_instance.example[1]: Still creating... [20s elapsed]
aws_instance.example[0]: Still creating... [30s elapsed]
aws_vpc.vpc: Still creating... [30s elapsed]
aws_instance.example[1]: Still creating... [30s elapsed]
aws_instance.example[0]: Still creating... [40s elapsed]
aws_vpc.vpc: Still creating... [40s elapsed]
```

6. You can login to your console and see the created infrastructure.







7. You can destroy everything with terraform destroy command.

```
[anmol@fedora terraform_files]$ terraform destroy
aws_s3_bucket.buckety: Refreshing state... [id=anmolahujabucket]
aws_instance.example[1]: Refreshing state... [id=i-08a80a6169dd44a88]
aws_instance.example[0]: Refreshing state... [id=i-02935b0a75671a3a1]

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # aws_instance.example[0] will be destroyed
  - resource "aws_instance" "example" {
      - ami                          = "ami-0885b1f6bd170450c" -> null
      - arn                          = "arn:aws:ec2:us-east-1:883338338815:instance/i-02935b0a75671a3a1" -> null
      - associate_public_ip_address  = true -> null
      - availability_zone            = "us-east-1b" -> null
      - cpu_core_count               = 1 -> null
      - cpu_threads_per_core         = 1 -> null
      - disable_api_termination      = false -> null
      - ebs_optimized                = false -> null
      - get_password_data            = false -> null
      - hibernation                  = false -> null
      - id                           = "i-02935b0a75671a3a1" -> null
      - instance_state               = "running" -> null
      - instance_type                = "t2.micro" -> null
      - ipv6_address_count           = 0 -> null
      - ipv6_addresses               = [] -> null
      - monitoring                   = false -> null
      - primary_network_interface_id = "eni-070736707cc190377" -> null
      - private_dns                  = "ip-172-31-38-161.ec2.internal" -> null
      - private_ip                   = "172.31.38.161" -> null
      - public_dns                   = "ec2-18-232-106-2.compute-1.amazonaws.com" -> null
      - public_ip                    = "18.232.106.2" -> null
      - security_groups              = [
          - "default",
        ] -> null
      - source_dest_check            = true -> null
      - subnet_id                    = "subnet-15075e49" -> null
      - tags                         = {
```