SYSTEM PROVISIONING

AND

CONFIGURATION MANAGEMENT

ASSIGNMENT - 1

School of Computer Science and Engineering

University of Petroleum & Energy Studies

Dehradun - 248001

2017 - 2020

Submitted by:

Name	Roll No	Branch	Sap Id
Saurabh Dimri	R171217052	CSE-DevOps	500061583

TASK DESCRIPTION

Write Terraform script to perform following tasks.

Step 1: Create two T2 Micro EC2 Instances.

Step 2: Create a VPN on AWS.

Step 3: Create and S3 bucket on AWS using terraform.

Step 4: Write the code for step 1,2 and 3 in a IaC terraform file and run terraform commands to execute these steps.

- ## Create 2 t2.micro EC2 instances:

T2.micro: T2 instances are a low-cost, general purpose instance type that provides a baseline level of CPU performance with the ability to burst above the baseline when needed. With On-Demand Instance prices starting at $0.0058 per hour, T2 instances are one of the lowest-cost Amazon EC2 instance options and are ideal for a variety of general-purpose applications like micro-services, low-latency interactive applications.

To create 2 t2.micro EC2 instance we will have to define several information to the terraform:

- AMI type
- Instance_Type (t2.micro)
- Key Name (AWS passkey created)
- Security_Groups

Terraform Script

```
//Defining Profile User Information
provider "aws"{
region     = "ap-south-1"
profile    = "saurabh"
}

//Defining the security groups for connection
resource "aws_security_group" "allow_connection" {
  name        = "allowConnection"
  description = "Allow SSH and HTTP over inbound traffic"
  vpc_id      = "vpc-d71807bf"

  ingress {
    description = "SSH from VPC"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
```

```
  }

  ingress {
    description = "HTTP from VPC"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "allowConnections"
  }
}

//Defining the EC2 instance
resource "aws_instance" "basicOs" {
 count          = "2"
 ami            = "ami-0447a12f28fddb066"
 instance_type = "t2.micro"
 key_name       = "basicKey_saurabh"
 security_groups = ["allowConnection"]
 tags = {
     Name = "firstOS -${ count.index + 1}"
 }
}
```

- **Create VPN on AWS:**

VPN: AWS Virtual Private Network solutions establish secure connections between your on-premises networks, remote offices, client devices, and the AWS global network. AWS VPN is comprised of two services: AWS Site-to-Site VPN and AWS Client VPN. Together, they deliver a highly-available, managed, and elastic cloud VPN solution to protect your network traffic.

AWS Site-to-Site VPN creates encrypted tunnels between your network and your Amazon Virtual Private Clouds or AWS Transit Gateways. For managing remote access, AWS Client VPN connects your users to AWS or on-premises resources using a VPN software client.

Terraform Script

```
provider "aws" {
  region = "ap-south-1"
  profile = "saurabh"
}

variable "vpc_private_subnets" {
  type    = list(string)
  default = ["10.10.11.0/24", "10.10.12.0/24", "10.10.13.0/24"]
}

module "vpn_gateway" {
  source = "../../"
  create_vpn_connection = false
  vpn_gateway_id      = module.vpc.vgw_id
  customer_gateway_id = aws_customer_gateway.main.id
  vpc_id                       = module.vpc.vpc_id
  vpc_subnet_route_table_ids   = module.vpc.private_route_table_ids
  vpc_subnet_route_table_count = length(var.vpc_private_subnets)
}

resource "aws_customer_gateway" "main" {
  bgp_asn    = 65000
  ip_address = "172.83.124.12"
  type       = "ipsec.1"
  tags = {
    Name = "Vpn-gateway"
  }
}

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "~> 2.0"
  name = "vpn-gateway"
  cidr = "10.10.0.0/16"
  azs             = ["ap-south-1a", "ap-south-1b", "ap-south-1a"]
  public_subnets  = ["10.10.1.0/24", "10.10.2.0/24", "10.10.3.0/24"]
  private_subnets = var.vpc_private_subnets
  enable_nat_gateway = false
  enable_vpn_gateway = true
  tags  = {
      Name = "Assignment-Vpn-Gateway"
  }
}
```

- ## Create a S3 bucket on AWS:

S3 Bucket: Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics.

Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% (11 9's) of durability, and stores data for millions of applications for companies all around the world.

Terraform Scripts

```
provider "aws" {
    region = "ap-south-1"
    profile = "saurabh"
}

resource "aws_s3_bucket" "mytestbucket125" {
  bucket = "mytestbucket456789"
  acl    = "public-read"
  tags = {
    Name = "testbucket456789"
  }
  versioning {
  enabled =true
  }
}

resource "aws_s3_bucket_object" "s3object" {
  bucket = "${aws_s3_bucket.mytestbucket125.id}"
  key    = "Image6.png"
  source = "C:/Users/Saurabh/Downloads/Image6.png"
}
```

Executing the Scripts:

Open terminal in this folder, and execute ‘terraform init’.

Terraform init will tell Terraform to scan the code, figure out which providers you’re using, and download the code for them. By default, the provider code will be downloaded into a .terraform folder, which is Terraform’s scratch directory.

```
Saurabh@DESKTOP-THLBIKL MINGW64 /e/terraform (master)
$ terraform init

Initializing the backend...

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.
```

Now that you have the provider code downloaded, run the terraform plan command. The plan command lets you see what Terraform will do before actually making any changes.

To actually create the Instance, run the terraform apply command. Adding –auto- approve will eliminate the requirement of entering “yes” after command execution.

After executing the terraform apply command, we will have a terminal popped up will all the planned changes that are to be done in our machine using terraform.

```
Saurabh@DESKTOP-THLBIKL MINGW64 /e/terraform/OrchastratingInfrastrucutre (master)
$ terraform apply

Warning: Interpolation-only expressions are deprecated

  on orchestration.tf line 68, in resource "null_resource" "null2":
  68:     host        = "${aws_instance.basicOs.public_ip}"

Terraform 0.11 and earlier required all non-constant expressions to be
provided via interpolation syntax, but this pattern is now deprecated. To
silence this warning, remove the "${ sequence from the start and the }"
sequence from the end of this expression, leaving just the inner expression.

Template interpolation syntax is still used to construct strings from
expressions when the template includes multiple interpolation sequences or a
mixture of literal strings and interpolations. This deprecation applies only
to templates that consist entirely of a single interpolation sequence.

(and 3 more similar warnings elsewhere)
```

Now as the terraform init and apply have been successfully created let’s move on to the Ubuntu Terminal and pass the credentials so that the terraform can interact with the AWS and can create the instances and machines required over the AWS.

```
  # aws_vpn_connection.vpn will be created
  + resource "aws_vpn_connection" "vpn" {
      + arn                              = (known after apply)
      + customer_gateway_configuration = (known after apply)
      + customer_gateway_id              = (known after apply)
      + id                               = (known after apply)
      + tunnel2_bgp_asn                  = (known after apply)
      + tunnel2_bgp_holdtime             = (known after apply)
      + tunnel2_cgw_inside_address       = (known after apply)
      + tunnel2_inside_cidr              = (known after apply)
      + tunnel2_preshared_key            = (sensitive value)
      + tunnel2_vgw_inside_address       = (known after apply)
      + type                             = "ipsec.1"
      + vgw_telemetry                    = (known after apply)
      + vpn_gateway_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.
```

Successful execution of terraform plan

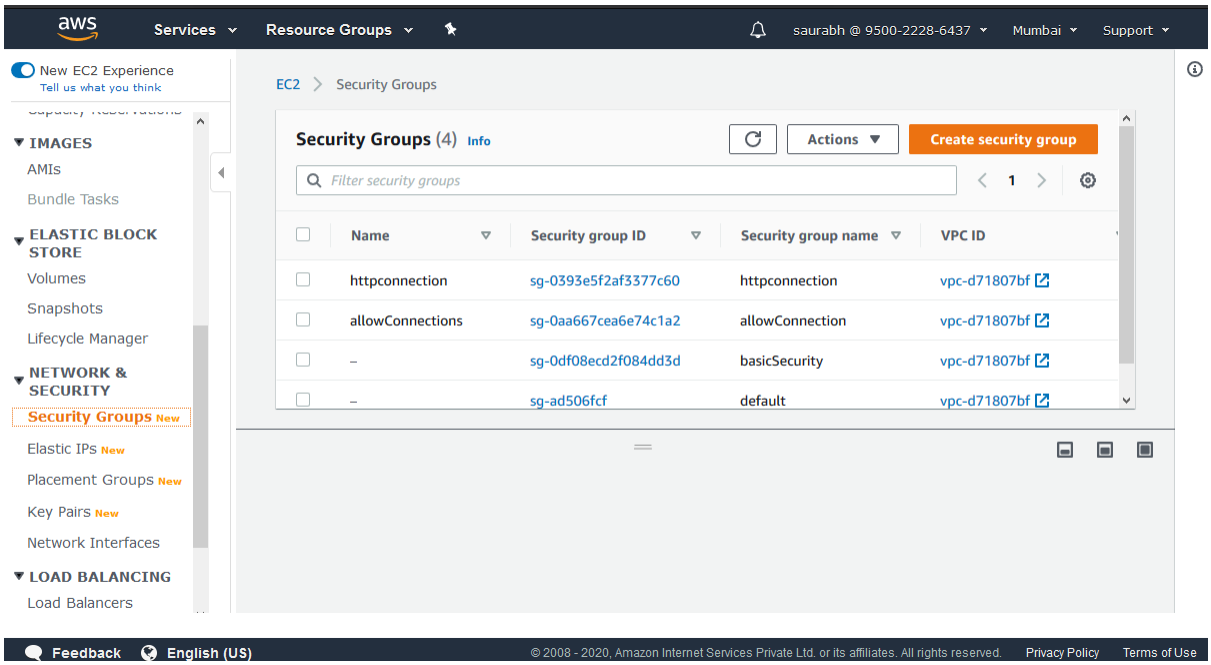
```
aws_customer_gateway.customer_gateway: Creating...
aws_vpc.vpc: Creating...
aws_instance.FirstEc2Instance: Creating...
aws_s3_bucket.bucket: Creating...
aws_instance.SecondEc2Instance: Creating...
aws_instance.FirstEc2Instance: Still creating... [10s elapsed]
aws_vpc.vpc: Still creating... [10s elapsed]
aws_customer_gateway.customer_gateway: Still creating... [10s elapsed]
aws_s3_bucket.bucket: Still creating... [10s elapsed]
aws_instance.SecondEc2Instance: Still creating... [10s elapsed]
aws_customer_gateway.customer_gateway: Creation complete after 17s [id=cgw-0c43b1c1aa4e3738f]
aws_vpc.vpc: Creation complete after 17s [id=vpc-0207a49b6da427839]
aws_vpn_gateway.vpn_gateway: Creating...
aws_instance.FirstEc2Instance: Still creating... [20s elapsed]
aws_s3_bucket.bucket: Still creating... [20s elapsed]
aws_instance.SecondEc2Instance: Still creating... [20s elapsed]
aws_s3_bucket.bucket: Creation complete after 22s [id=darshbucket7899]
aws_vpn_gateway.vpn_gateway: Still creating... [10s elapsed]
aws_instance.FirstEc2Instance: Still creating... [30s elapsed]
aws_instance.SecondEc2Instance: Still creating... [30s elapsed]
aws_vpn_gateway.vpn_gateway: Creation complete after 16s [id=vgw-06ad6ec93c15795dc]
aws_vpn_connection.vpn: Creating...
aws_instance.FirstEc2Instance: Creation complete after 40s [id=i-0877949be9a90cbd5]
aws_instance.SecondEc2Instance: Still creating... [40s elapsed]
aws_vpn_connection.vpn: Still creating... [10s elapsed]
aws_instance.SecondEc2Instance: Still creating... [50s elapsed]
aws_instance.SecondEc2Instance: Creation complete after 51s [id=i-0a2b156bb39f3adfc]
aws_vpn_connection.vpn: Still creating... [10s elapsed]
aws_instance.SecondEc2Instance: Still creating... [50s elapsed]
aws_instance.SecondEc2Instance: Creation complete after 51s [id=i-0a2b156bb39f3adfc]
aws_vpn_connection.vpn: Still creating... [20s elapsed]
aws_vpn_connection.vpn: Still creating... [30s elapsed]
aws_vpn_connection.vpn: Still creating... [40s elapsed]
aws_vpn_connection.vpn: Still creating... [50s elapsed]
aws_vpn_connection.vpn: Still creating... [1m0s elapsed]
aws_vpn_connection.vpn: Still creating... [1m10s elapsed]
aws_vpn_connection.vpn: Still creating... [1m20s elapsed]
aws_vpn_connection.vpn: Still creating... [1m30s elapsed]
aws_vpn_connection.vpn: Still creating... [1m40s elapsed]
aws_vpn_connection.vpn: Still creating... [1m50s elapsed]
aws_vpn_connection.vpn: Still creating... [2m0s elapsed]
aws_vpn_connection.vpn: Still creating... [2m10s elapsed]
aws_vpn_connection.vpn: Still creating... [2m20s elapsed]
aws_vpn_connection.vpn: Still creating... [2m30s elapsed]
aws_vpn_connection.vpn: Still creating... [2m40s elapsed]
aws_vpn_connection.vpn: Still creating... [2m50s elapsed]
aws_vpn_connection.vpn: Still creating... [3m0s elapsed]
aws_vpn_connection.vpn: Still creating... [3m10s elapsed]
aws_vpn_connection.vpn: Still creating... [3m20s elapsed]
aws_vpn_connection.vpn: Still creating... [3m30s elapsed]
aws_vpn_connection.vpn: Still creating... [3m40s elapsed]
aws_vpn_connection.vpn: Creation complete after 3m44s [id=vpn-088710edd122491ea]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
```

After Executing Terraform Apply

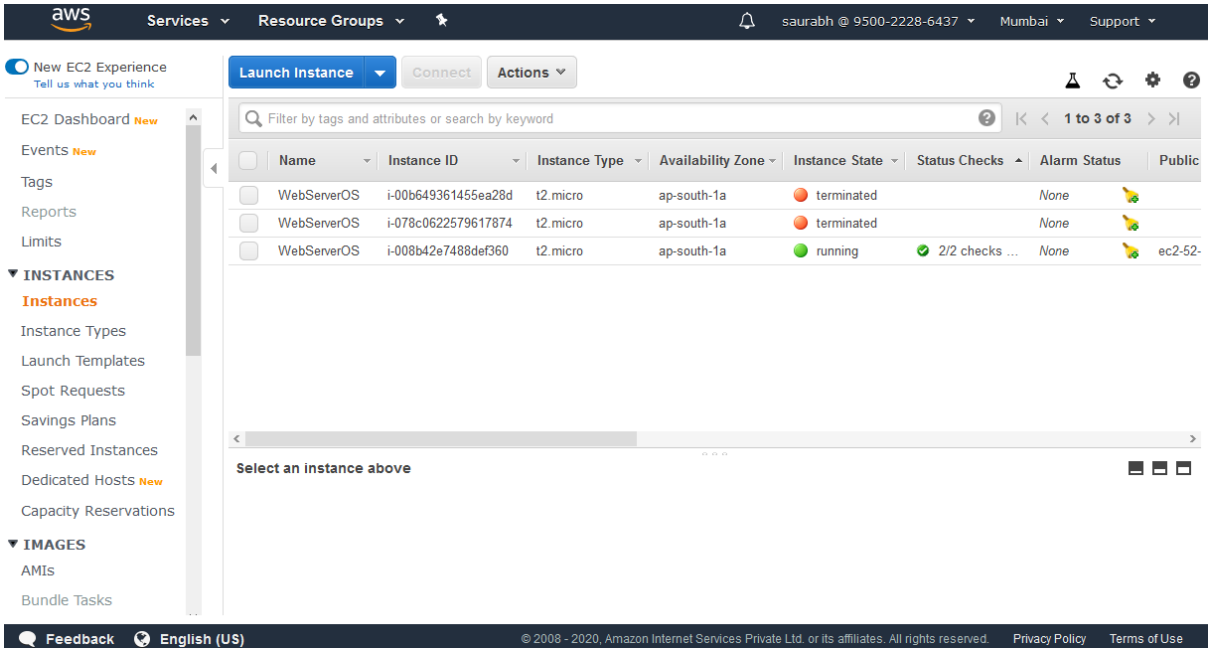
Final Outputs Generated:

Task 1:

To create 2 EC2 instance with t2.micro, we have to create the security groups too to allow access. The security group created was named allowConnections. The Security Group created on the AWS instance is:



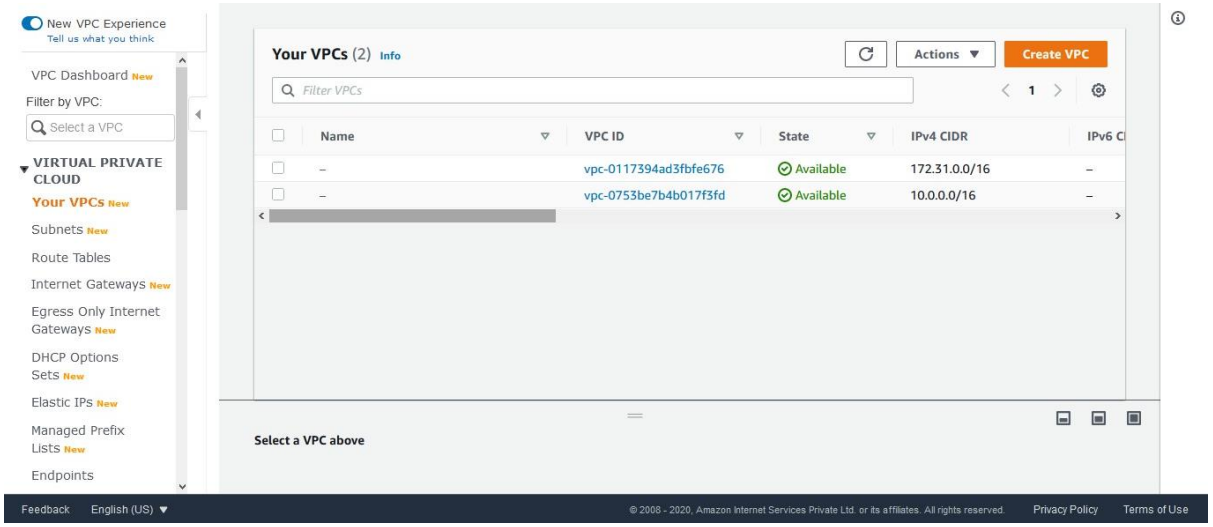
After creating the security group then we will create our EC2 instances.



As we can see we have 2 WebServerOS with t2.micro created in the location ap-south-1, which we defined while creating our terraform scripts.

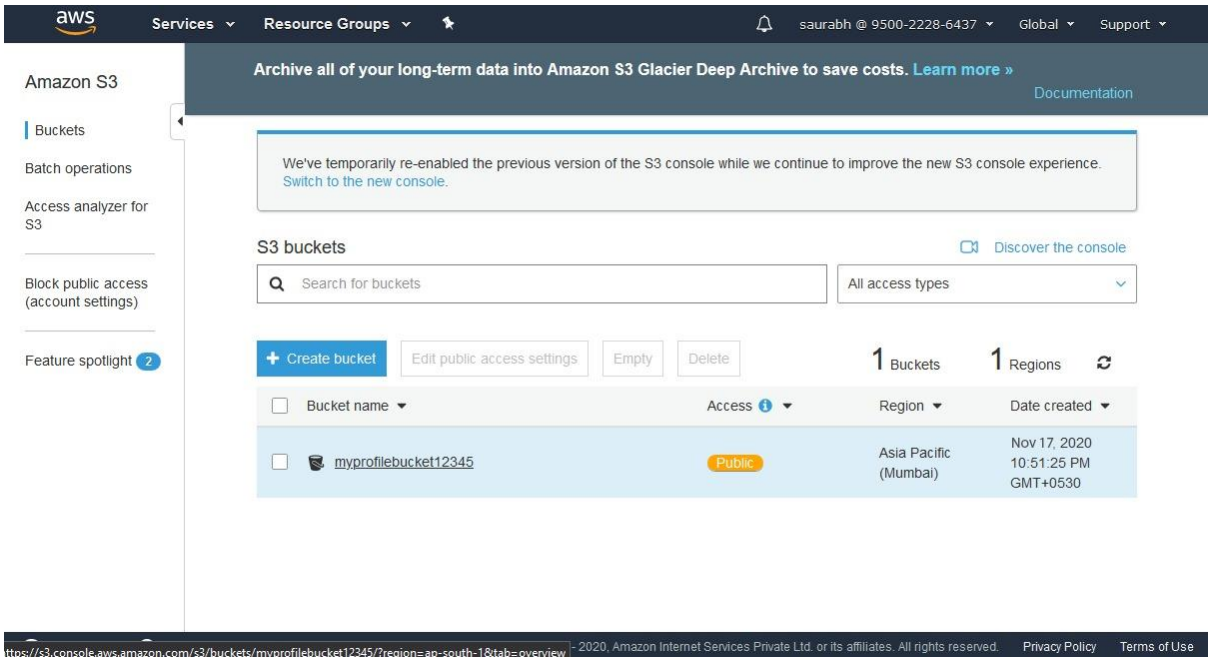
Task 2:

The VPC created will take the information again from the security groups which we created during the first task. The security's inbound and outbound rules will be used to create the VPC's on the AWS.



Task 3:

To create the S3 bucket, we will have to define for the types of rules which will be followed by the bucket. They are known as acl rules. The acl rule defined here is public-read so that we can push some files to the S3 bucket too.



Now to push the file to the S3 bucket, we will push the file from out machine to the S3 Bucket. Here I pushed an image from my machine to S3 Bucket which will be visible inside the bucket