NAME: MULLA AFRAH AKKAS ALI

ROLL NO.: 612038

BRANCH: T.E. – I.T.

SEMESTER: ODD SEMESTER 5

COURSE: Advance DevOPs (ITL504)

DATE: 11-10-2022

EXPERIMENT 13

1. What is Terraform?

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle.

Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.

2. What is Infrastructure as a Code (IaC)?

Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code instead of through manual processes.

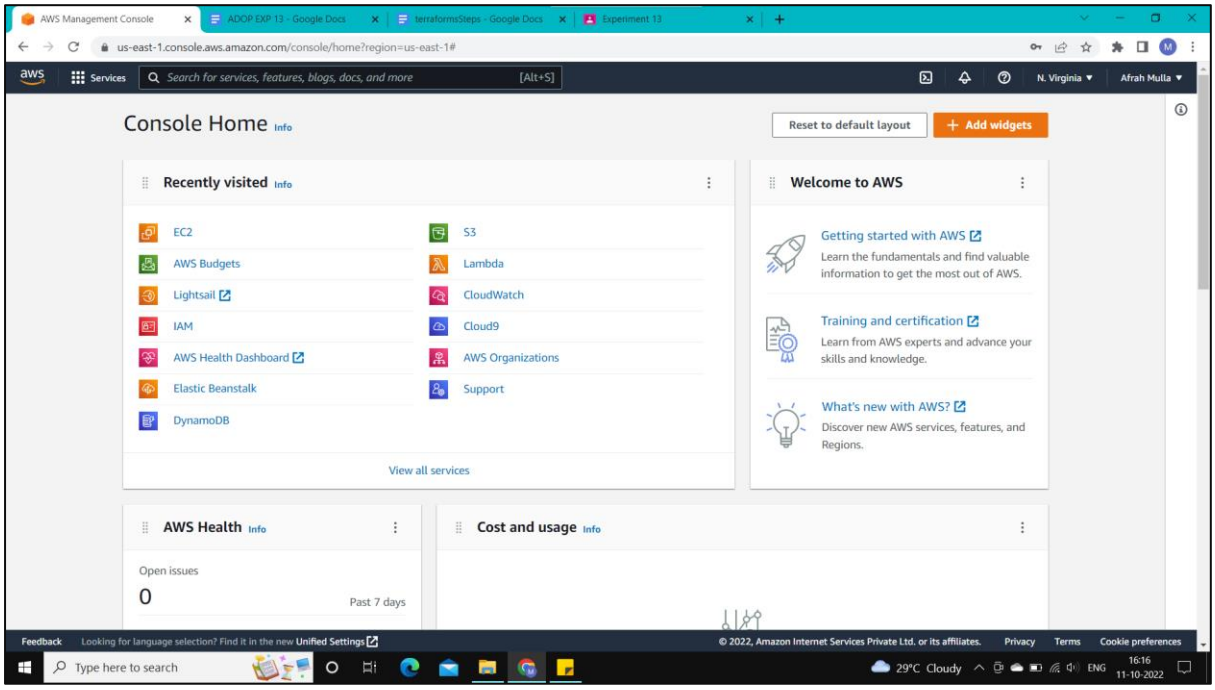
With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations. It also ensures that you provision the same environment every time.

Version control is an important part of IaC, and your configuration files should be under source control just like any other software source code file. Deploying your infrastructure as code also means that you can divide your infrastructure into modular components that can then be combined in different ways through automation.

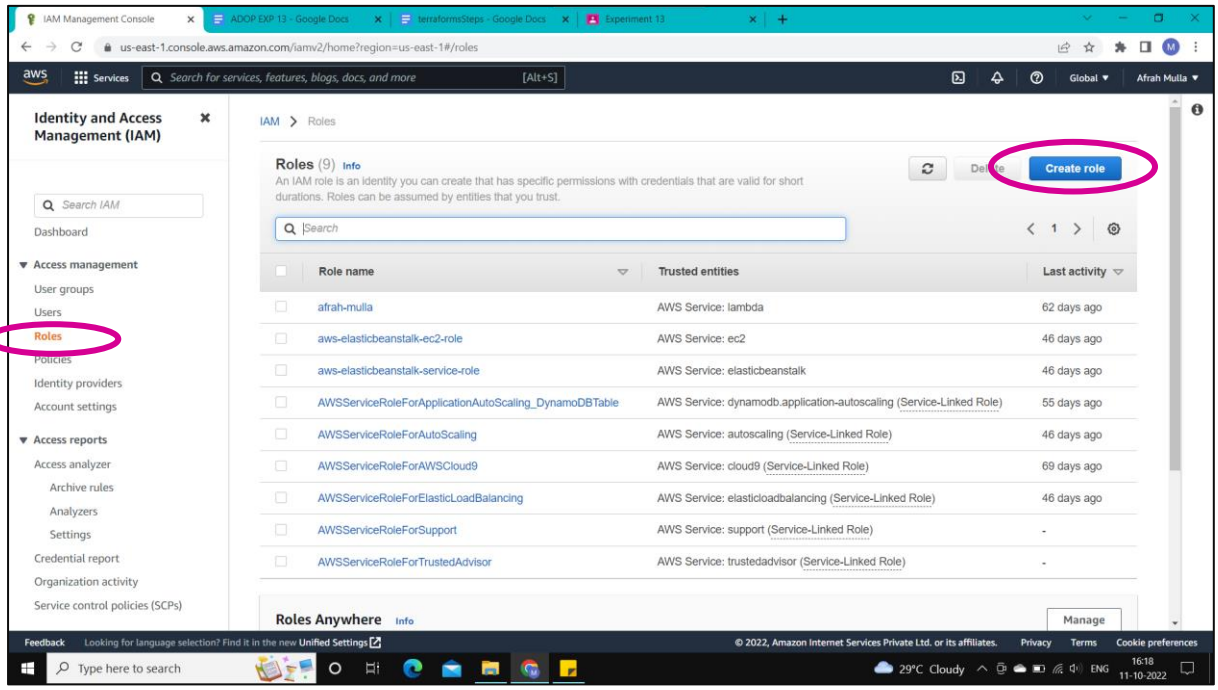
3. Perform an experiment, to understand Terraform lifecycle, core concepts/terminologies and install it on a Linux Machine.

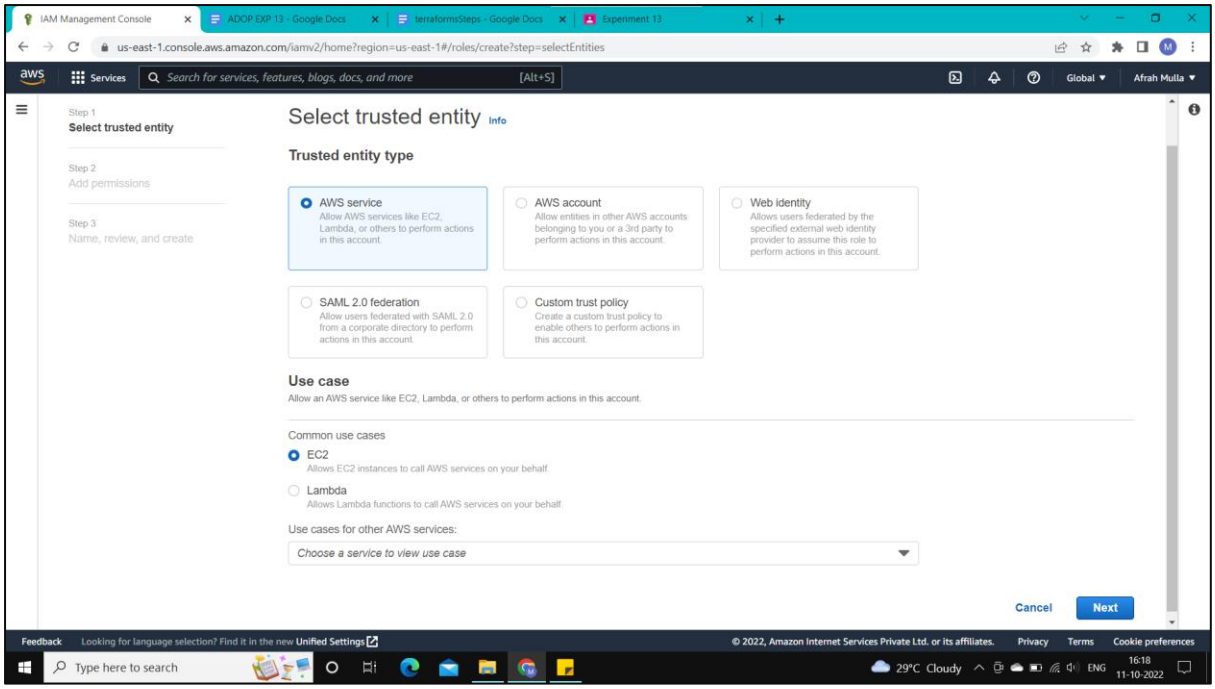
4. Using Terraform, create an EC2 instance on AWS cloud.

Step1: AWS Management Console Dashboard

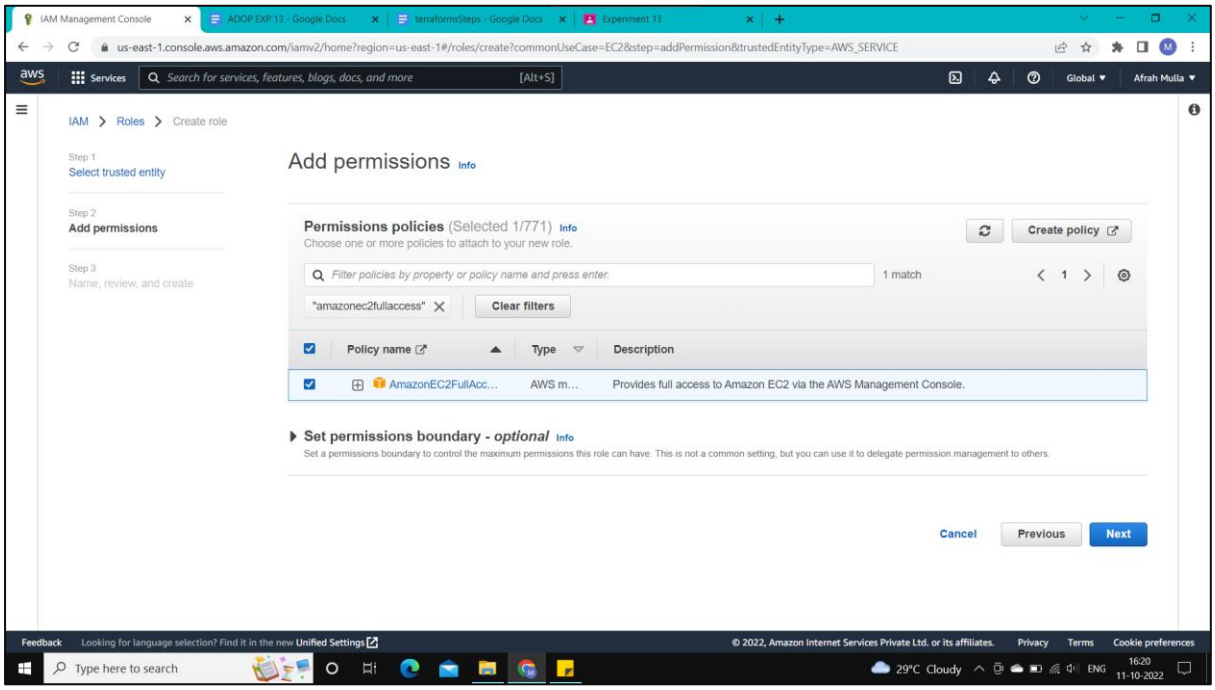


Step 2: IAM -> Roles -> Create role -> Set Trusted entity type to AWS service, use case to EC2

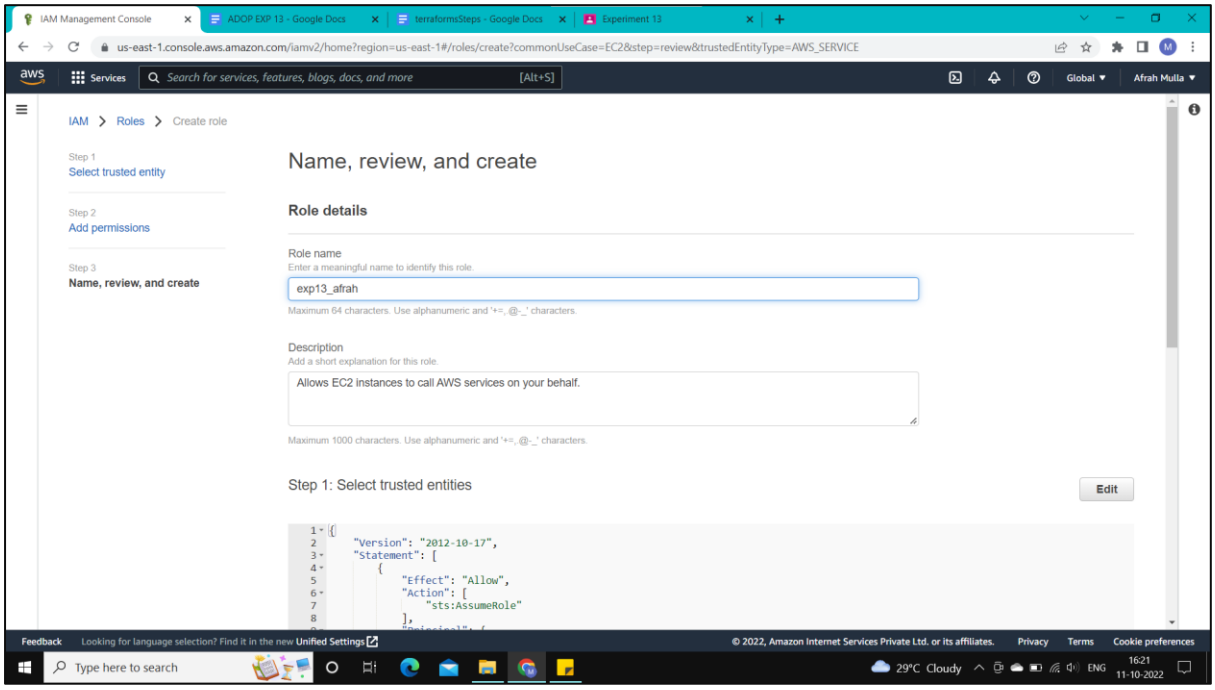


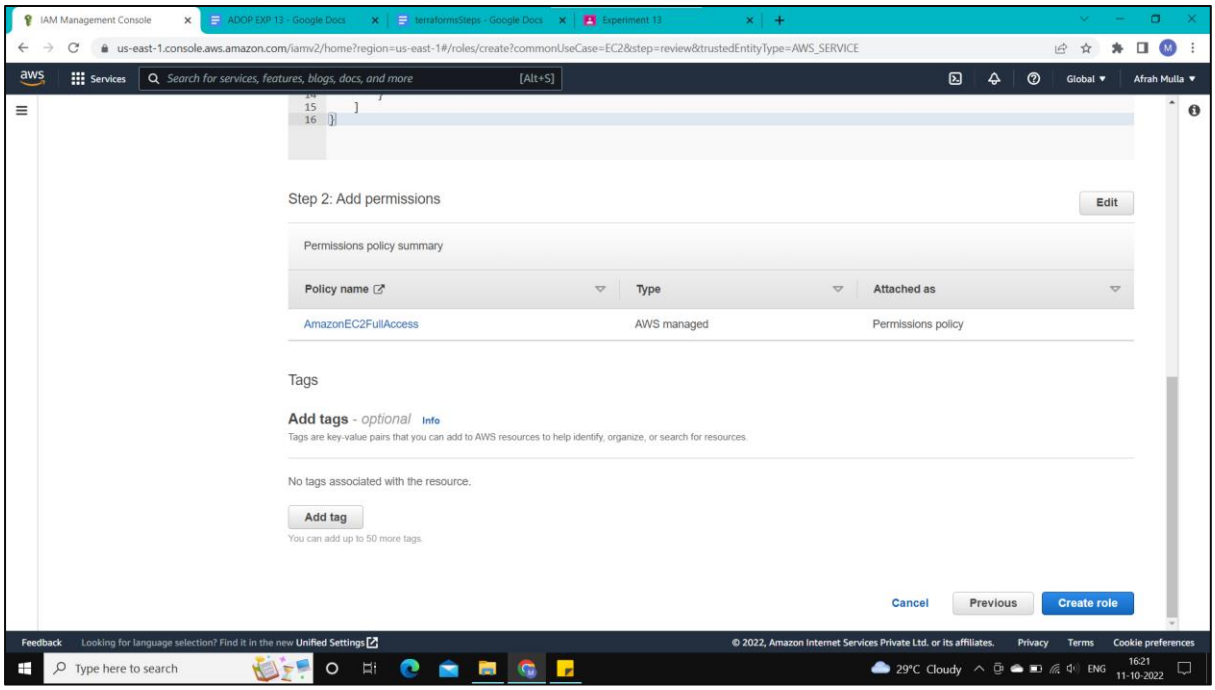


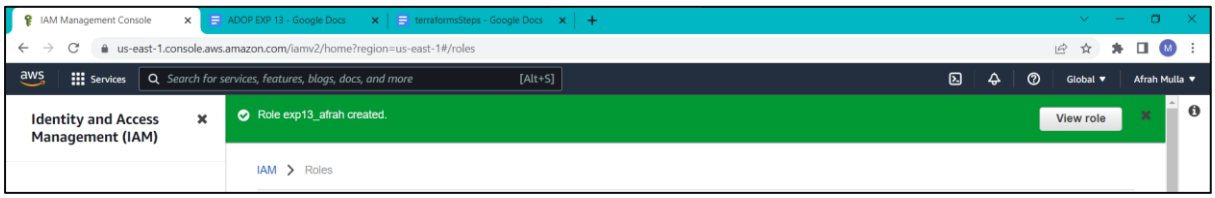
Step 3: Search for AmazonEC2FullAccess in Permission policies -> Select the policy -> Next



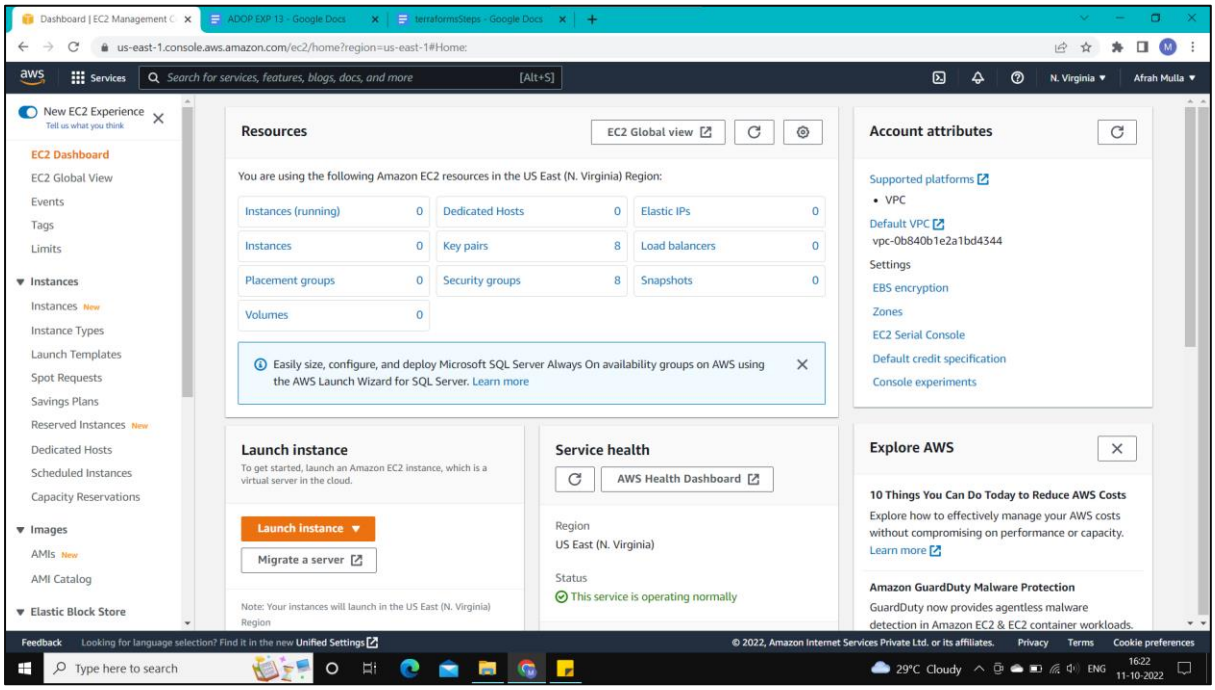
Step 4: Assign a name to your role -> Create role

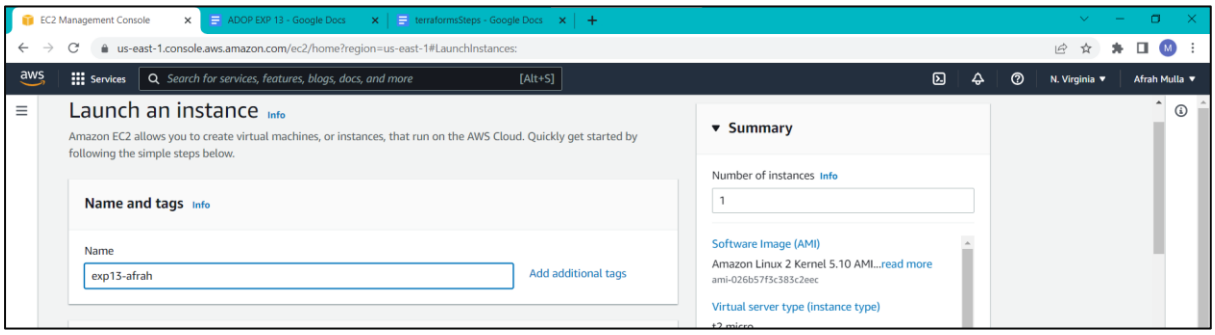


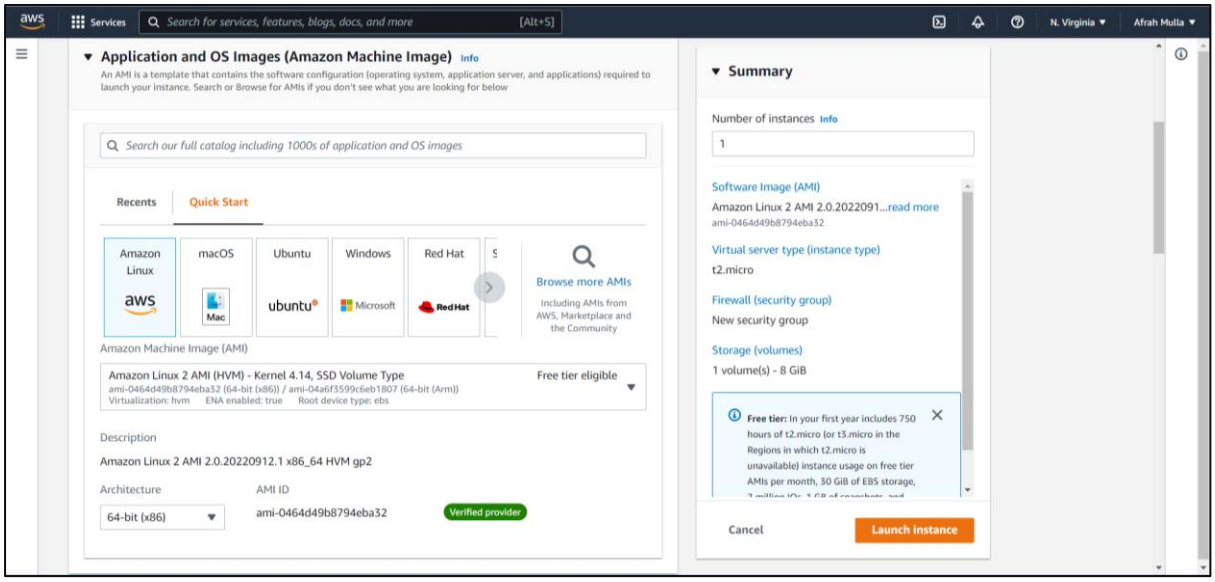




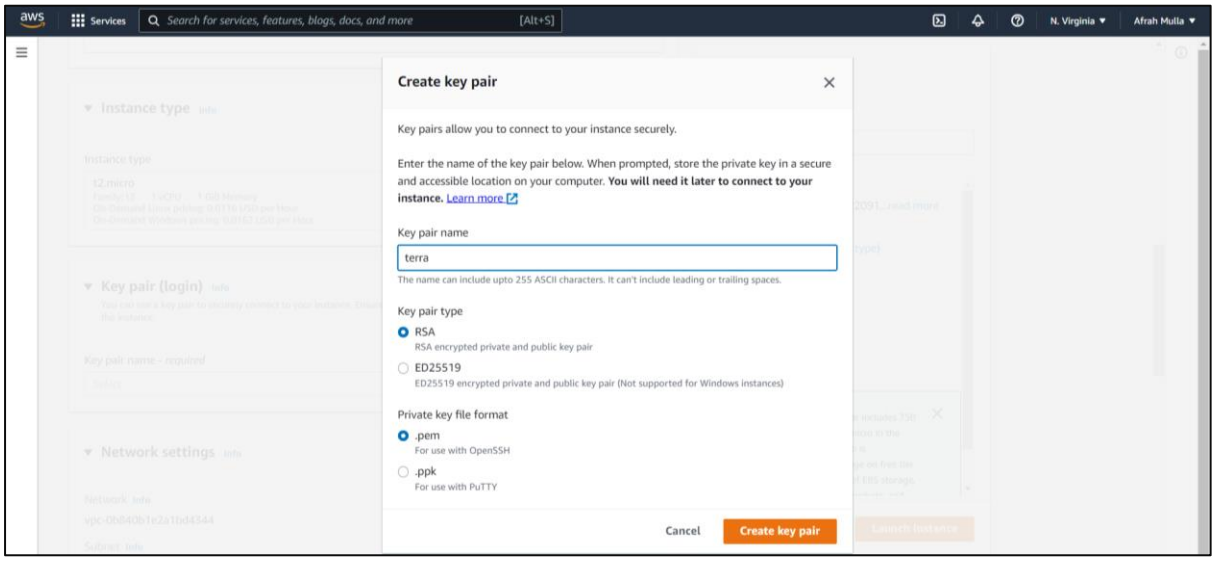
Step 5: Search for EC2 -> Select ‘Launch an instance’. Assign a name to you instance -> Choose the ‘Amazon Linux’ machine -> Select the ‘Amazon Linux 2 AMI (HVM) - Kernel 4.14, SSD Volume Type’ machine image

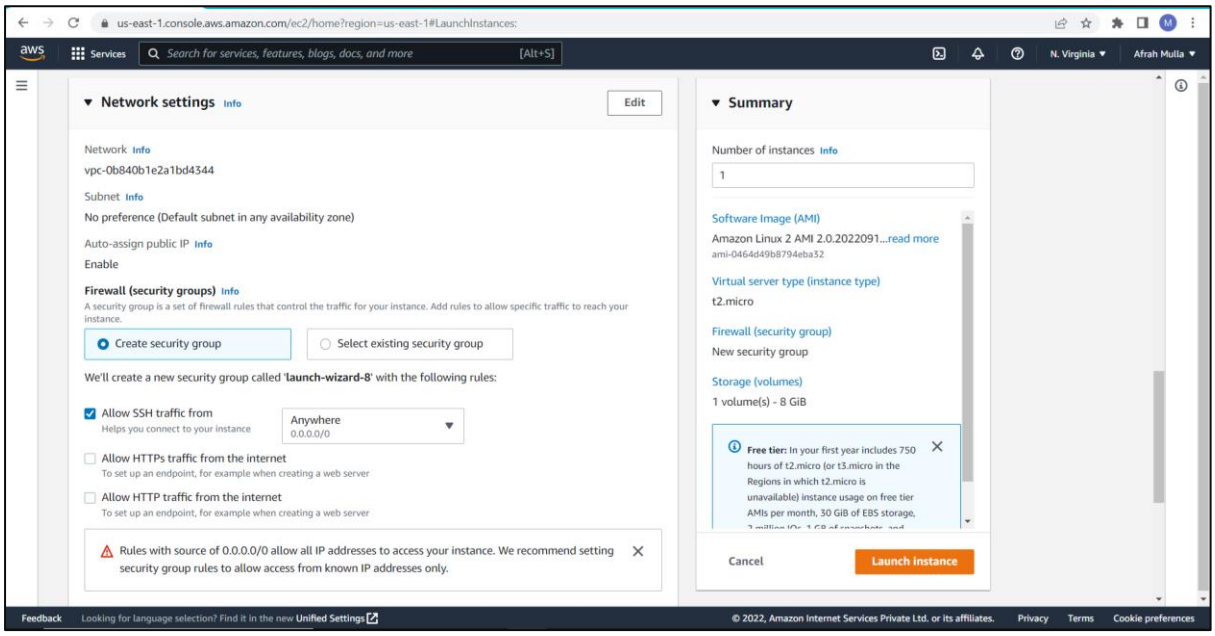


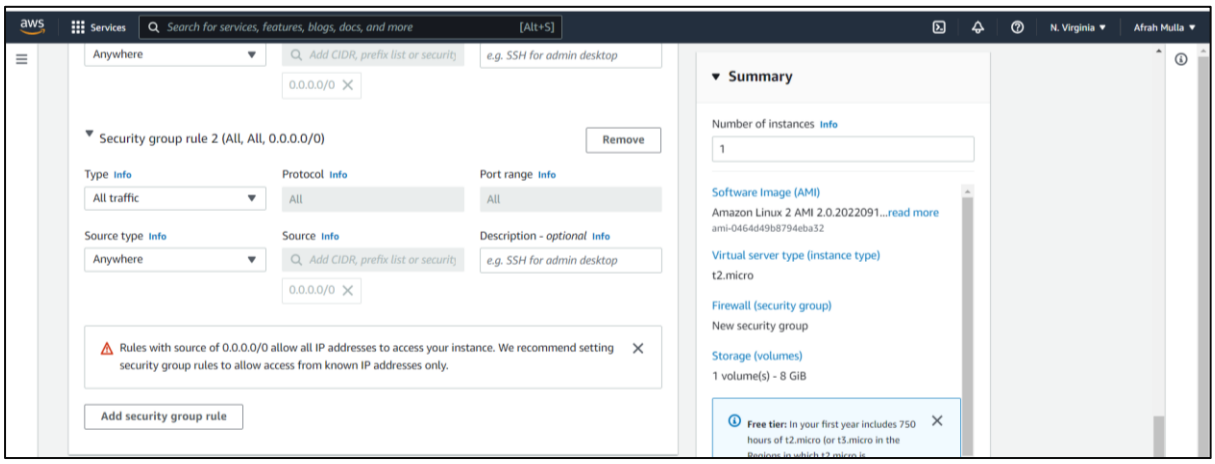




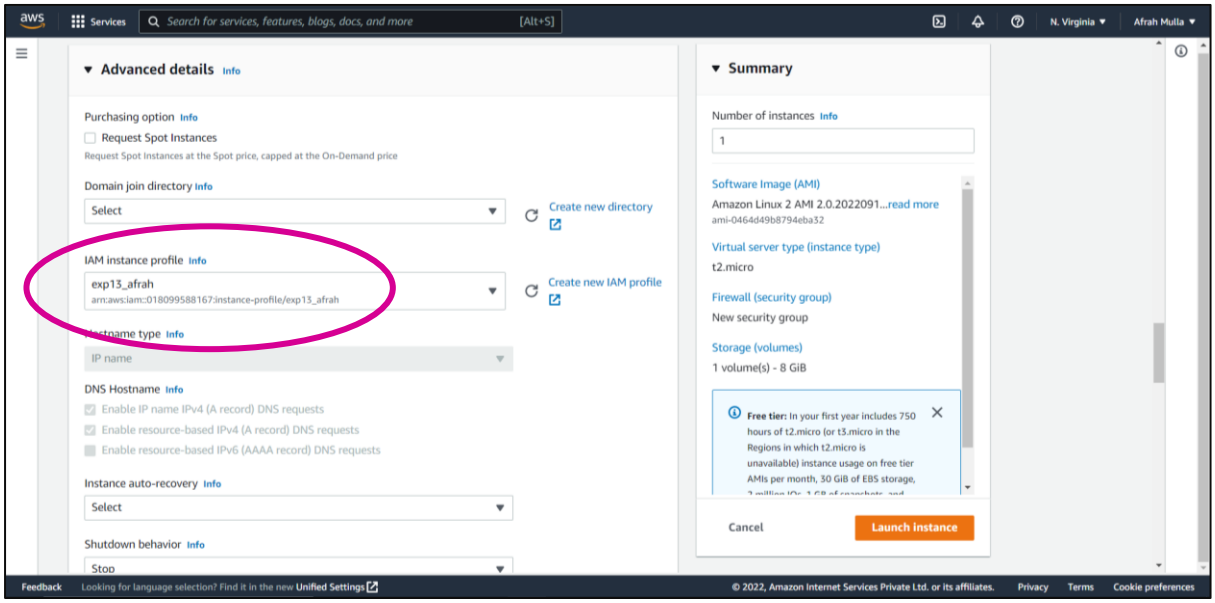
Step 6: Create a key pair -> In the network settings, allow the SSH traffic from anywhere -> Click on edit -> Add a security group for ‘All traffic’ and select the source type as ‘anywhere’

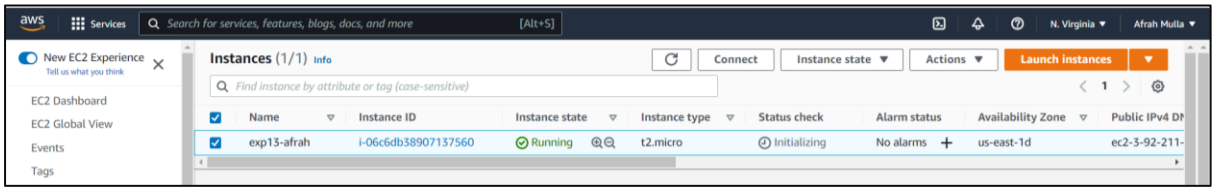


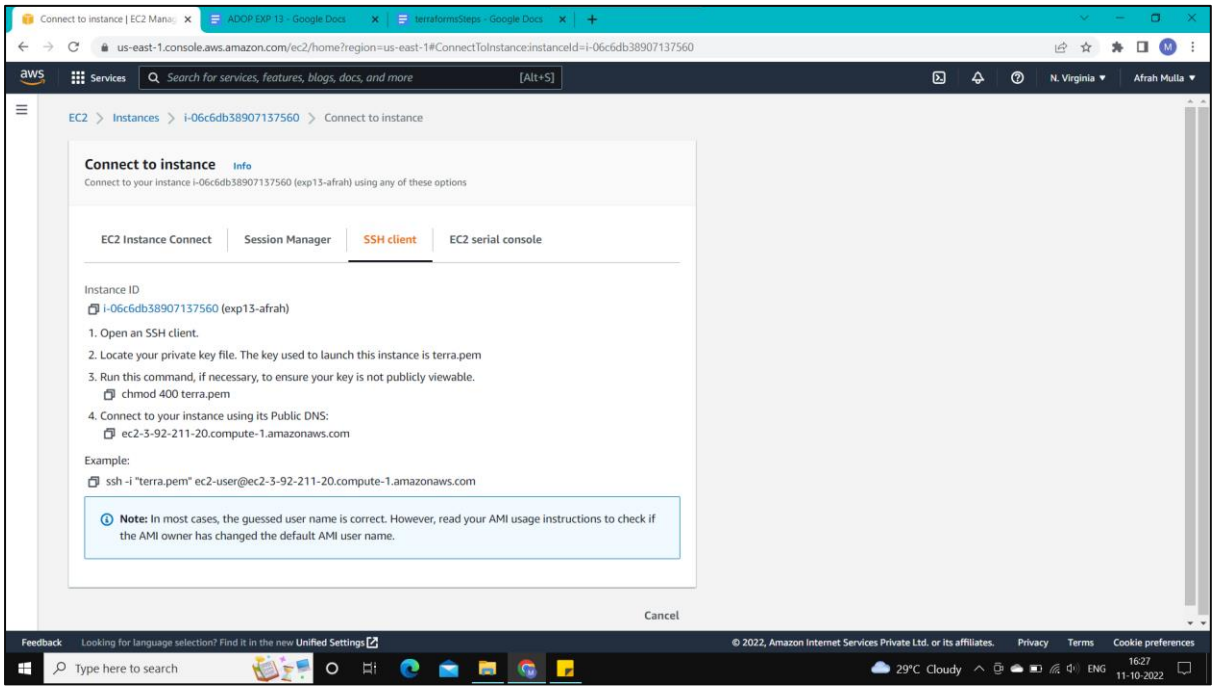




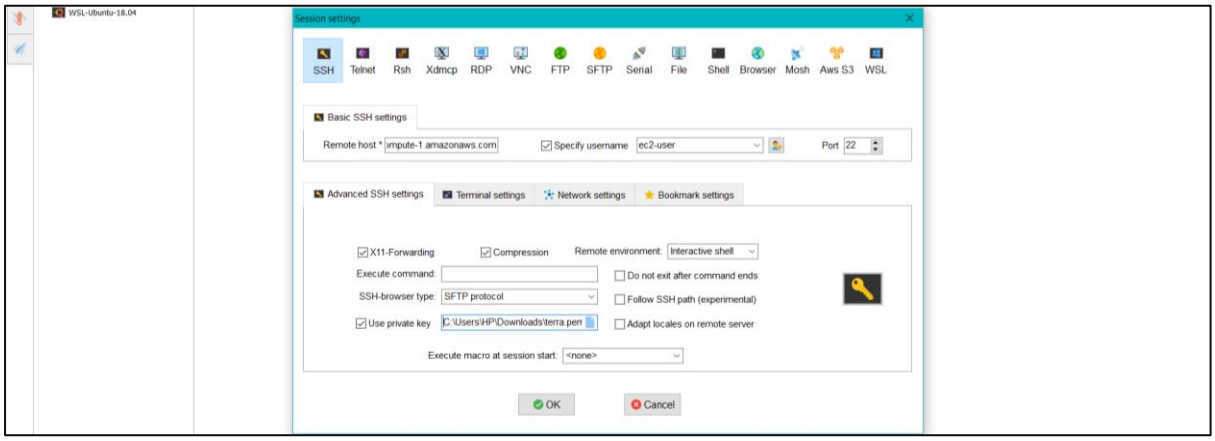
Step 7: In the advanced details, select the IAM role created. Finally, launch the instance







Step 8: Launch MobaXterm -> Select SSH session -> Copy the public DNS of your instance and paste it into the remote host. Use the downloaded key pair as the private key



Step 9: Create Terraform files by executing the following commands –

- sudo su
- mkdir exp13_afrah
- cd exp13_afrah
- nano variables.tf
- nano main.tf

variables.tf

```
variable "aws_region" {
      description = "The AWS region to create things in."
      default     = "us-east-1"
}

variable "key_name" {
   description = " SSH keys to connect to ec2 instance"
   default     =  "terra"
}

variable "instance_type" {
   description = "instance type for ec2"
   default     =  "t2.micro"
}
```

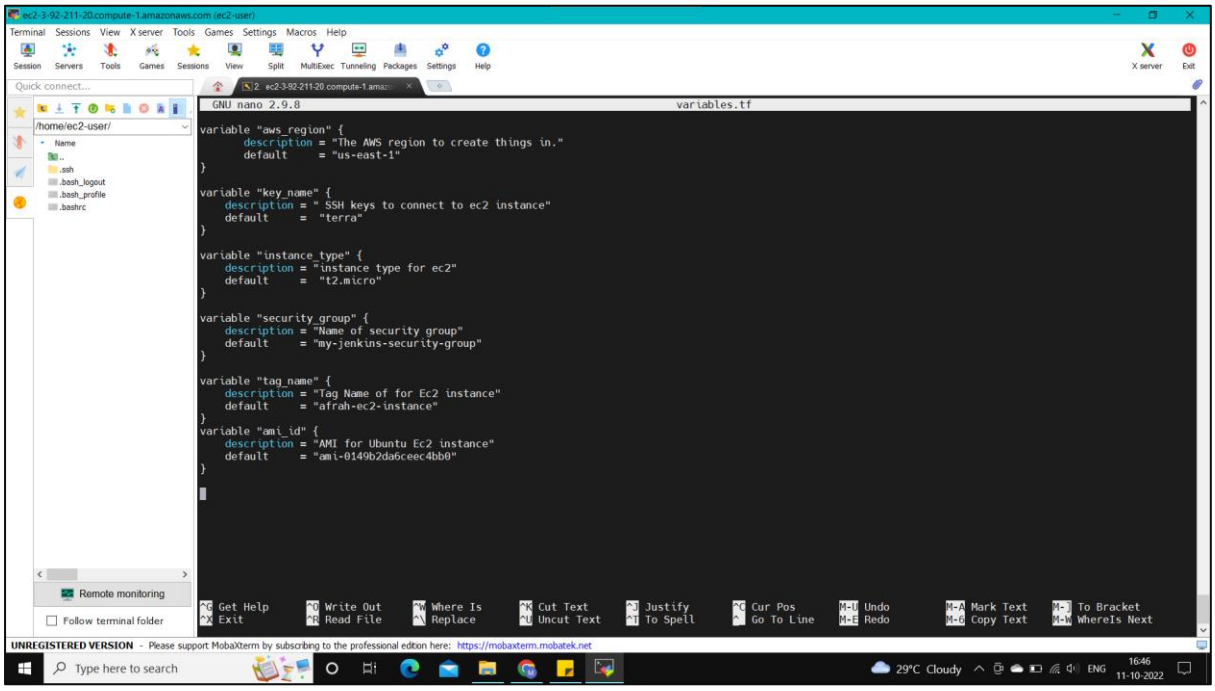
```
variable "security_group" {
    description = "Name of security group"
    default     = "my-jenkins-security-group"
}

variable "tag_name" {
    description = "Tag Name of for Ec2 instance"
    default     = "afrah-ec2-instance"
}
variable "ami_id" {
    description = "AMI for Ubuntu Ec2 instance"
    default     = "ami-0149b2da6ceec4bb0"
}
```

main.tf

```
provider "aws" {
  region = var.aws_region
}

#Create security group with firewall rules
resource "aws_security_group" "security_jenkins_grp" {
  name        = var.security_group
  description = "security group for jenkins"

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
```

```
  }

 ingress {
   from_port   = 22
   to_port     = 22
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
  }

 # outbound from jenkis server
  egress {
   from_port   = 0
   to_port     = 65535
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
  }

  tags= {
   Name = var.security_group
  }
}

resource "aws_instance" "myFirstInstance" {
  ami           = var.ami_id
  key_name = var.key_name
  instance_type = var.instance_type
  security_groups= [var.security_group]
  tags= {
   Name = var.tag_name
  }
```

```
}

# Create Elastic IP address
resource "aws_eip" "myFirstInstance" {
  vpc      = true
  instance = aws_instance.myFirstInstance.id
tags= {
    Name = "jenkins_elastic_ip"
  }
}
```



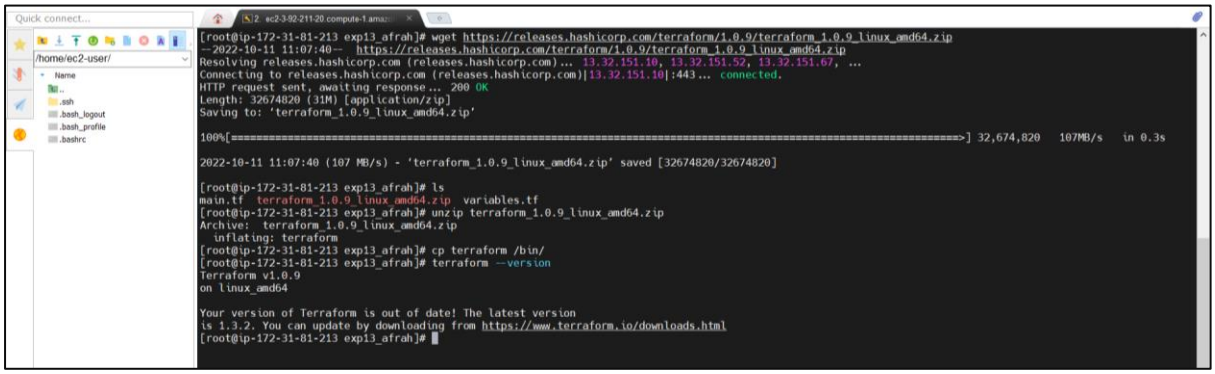
```
GNU nano 2.9.8                                  main.tf

provider "aws" {
  region = var.aws_region
}

#Create security group with firewall rules
resource "aws_security_group" "security_jenkins_grp" {
  name        = var.security_group
  description = "security group for jenkins"

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

 ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

 # outbound from jenkis server
  egress {
    from_port   = 0
    to_port     = 65535
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags= {
    Name = var.security_group
  }
}

resource "aws_instance" "myFirstInstance" {
                                          [ Wrote 55 lines ]
^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify   ^C Cur Pos   M-U Undo   M-A Mark Text   M-] To Bracket
^X Exit       ^R Read File   ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo   M-6 Copy Text   M-W WhereIs Next
```

```
GNU nano 2.9.8                                  main.tf

  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
  }

 # outbound from jenkis server
  egress {
    from_port   = 0
    to_port     = 65535
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags= {
    Name = var.security_group
  }
}

resource "aws_instance" "myFirstInstance" {
  ami           = var.ami_id
  key_name = var.key_name
  instance_type = var.instance_type
  security_groups= [var.security_group]
  tags= {
    Name = var.tag_name
  }
}

# Create Elastic IP address
resource "aws_eip" "myFirstInstance" {
  vpc      = true
  instance = aws_instance.myFirstInstance.id
tags= {
    Name = "jenkins_elastic_ip"
  }
}

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify   ^C Cur Pos   M-U Undo   M-A Mark Text   M-] To Bracket
^X Exit       ^R Read File   ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo   M-6 Copy Text   M-W WhereIs Next
```

Step 10: Install Terraform by executing the commands given below –

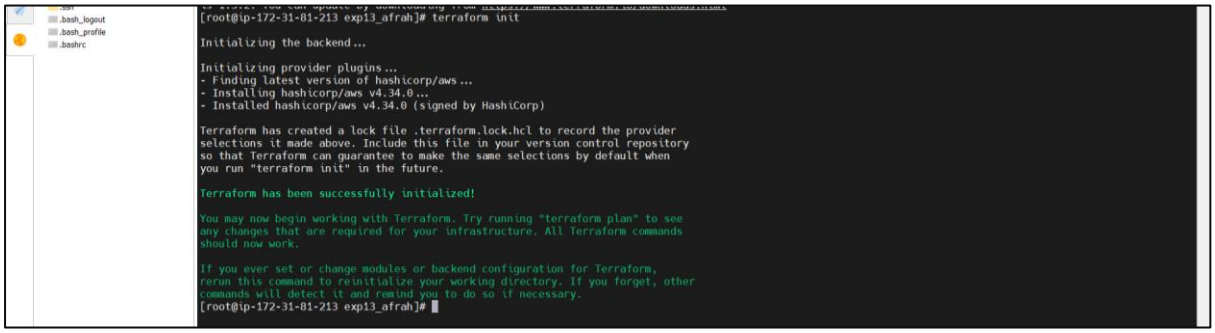
- wget https://releases.hashicorp.com/terraform/1.0.9/terraform_1.0.9_linux_+amd64.zip
- ls
- unzip terraform_1.0.9_linux_amd64.zip
- cp terraform /bin/

Now, check the Terraform version installed by using – terraform --version



Step 11: Execute the following Terraform commands to create an EC2 instance using Terraform –

- terraform init
- terraform plan
- terraform apply
  Enter 'yes' to complete the creation of EC2 instance using Terraform



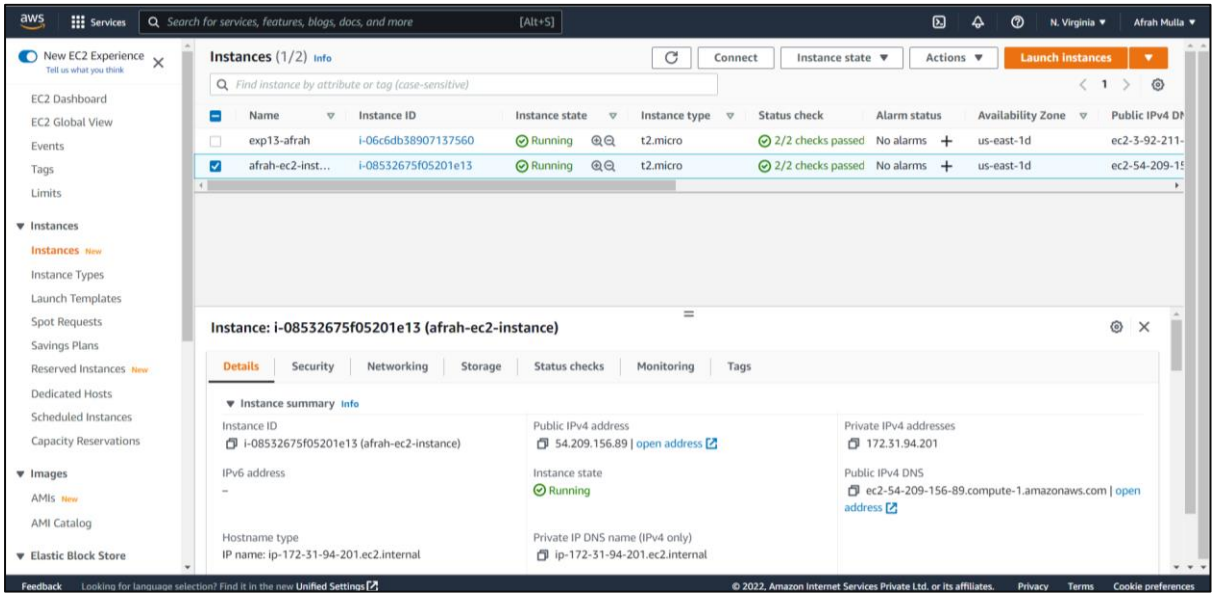
```
[root@ip-172-31-81-213 exp13_afrah]# terraform plan

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_eip.myFirstInstance will be created
  + resource "aws_eip" "myFirstInstance" {
      + allocation_id        = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = (known after apply)
      + id                   = (known after apply)
      + instance             = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + public_dns           = (known after apply)
      + public_ip            = (known after apply)
      + public_ipv4_pool     = (known after apply)
      + tags                 = {
          + "Name" = "jenkins_elastic_ip"
        }
      + tags_all             = {
          + "Name" = "jenkins_elastic_ip"
        }
      + vpc                  = true
    }

  # aws_instance.myFirstInstance will be created
  + resource "aws_instance" "myFirstInstance" {
      + ami                                  = "ami-0149b2da6ceec4bb0"
      + 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)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
```

```
            + description      = ""
            + from_port        = 22
            + ipv6_cidr_blocks = []
            + prefix_list_ids  = []
            + protocol         = "tcp"
            + security_groups  = []
            + self             = false
            + to_port          = 22
          },
        + {
            + cidr_blocks      = [
                + "0.0.0.0/0",
              ]
            + description      = ""
            + from_port        = 8080
            + ipv6_cidr_blocks = []
            + prefix_list_ids  = []
            + protocol         = "tcp"
            + security_groups  = []
            + self             = false
            + to_port          = 8080
          },
      ]
      + name                   = "my-jenkins-security-group"
      + name_prefix            = (known after apply)
      + owner_id               = (known after apply)
      + revoke_rules_on_delete = false
      + tags                   = {
          + "Name" = "my-jenkins-security-group"
        }
      + tags_all               = {
          + "Name" = "my-jenkins-security-group"
        }
      + vpc_id                 = (known after apply)
    }

Plan: 3 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.
[root@ip-172-31-81-213 exp13_afrah]# D
```

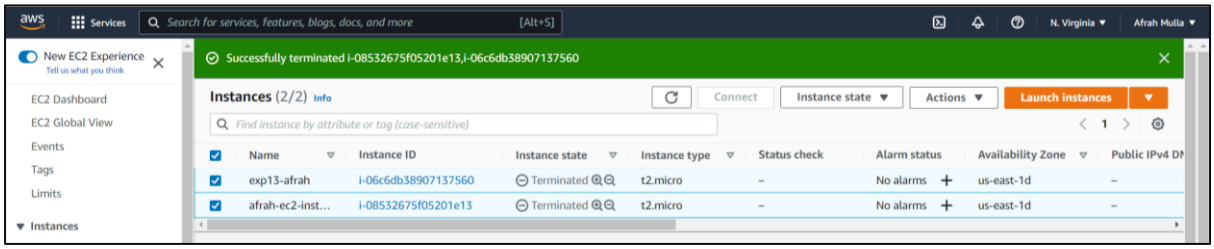
```
[root@ip-172-31-81-213 exp13_afrah]# D
[root@ip-172-31-81-213 exp13_afrah]# terraform apply

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_eip.myFirstInstance will be created
  + resource "aws_eip" "myFirstInstance" {
      + allocation_id        = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = (known after apply)
      + id                   = (known after apply)
      + instance             = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + public_dns           = (known after apply)
      + public_ip            = (known after apply)
      + public_ipv4_pool     = (known after apply)
      + tags                 = {
          + "Name" = "jenkins_elastic_ip"
        }
      + tags_all             = {
          + "Name" = "jenkins_elastic_ip"
        }
      + vpc                  = true
    }

  # aws_instance.myFirstInstance will be created
  + resource "aws_instance" "myFirstInstance" {
      + ami                                  = "ami-0149b2da6ceec4bb0"
      + 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)
      + disable_api_stop                     = (known after apply)
```

```
+ from_port        = 8080
+ ipv6_cidr_blocks = []
+ prefix_list_ids  = []
+ protocol         = "tcp"
+ security_groups  = []
+ self             = false
+ to_port          = 8080
},
]
+ name                   = "my-jenkins-security-group"
+ name_prefix            = (known after apply)
+ owner_id               = (known after apply)
+ revoke_rules_on_delete = false
+ tags                   = {
  + "Name" = "my-jenkins-security-group"
}
+ tags_all               = {
  + "Name" = "my-jenkins-security-group"
}
+ vpc_id                 = (known after apply)
}

Plan: 3 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_security_group.security_jenkins_grp: Creating...
aws_instance.myFirstInstance: Creating...
aws_security_group.security_jenkins_grp: Creation complete after 3s [id=sg-056ab23da922f1a98]
aws_instance.myFirstInstance: Still creating... [10s elapsed]
aws_instance.myFirstInstance: Still creating... [20s elapsed]
aws_instance.myFirstInstance: Still creating... [30s elapsed]
aws_instance.myFirstInstance: Creation complete after 32s [id=i-08532675f05201e13]
aws_eip.myFirstInstance: Creating...
aws_eip.myFirstInstance: Creation complete after 2s [id=eipalloc-0f27458c0a3c53e27]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
[root@ip-172-31-81-213 exp13_afrah]#
```

Step 12: Now go to EC2 console, to see the new instances up and running



Finally, delete the IAM role and terminate both the instances



5. Explain following Terraform commands in one line

(i) terraform init

The terraform init command initializes a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

(ii) terraform validate

The terraform validate command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

(iii) terraform plan

The terraform plan command creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure.

(iv) terraform apply

The terraform apply command executes the actions proposed in a Terraform plan.

(v) terraform destroy

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