

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.

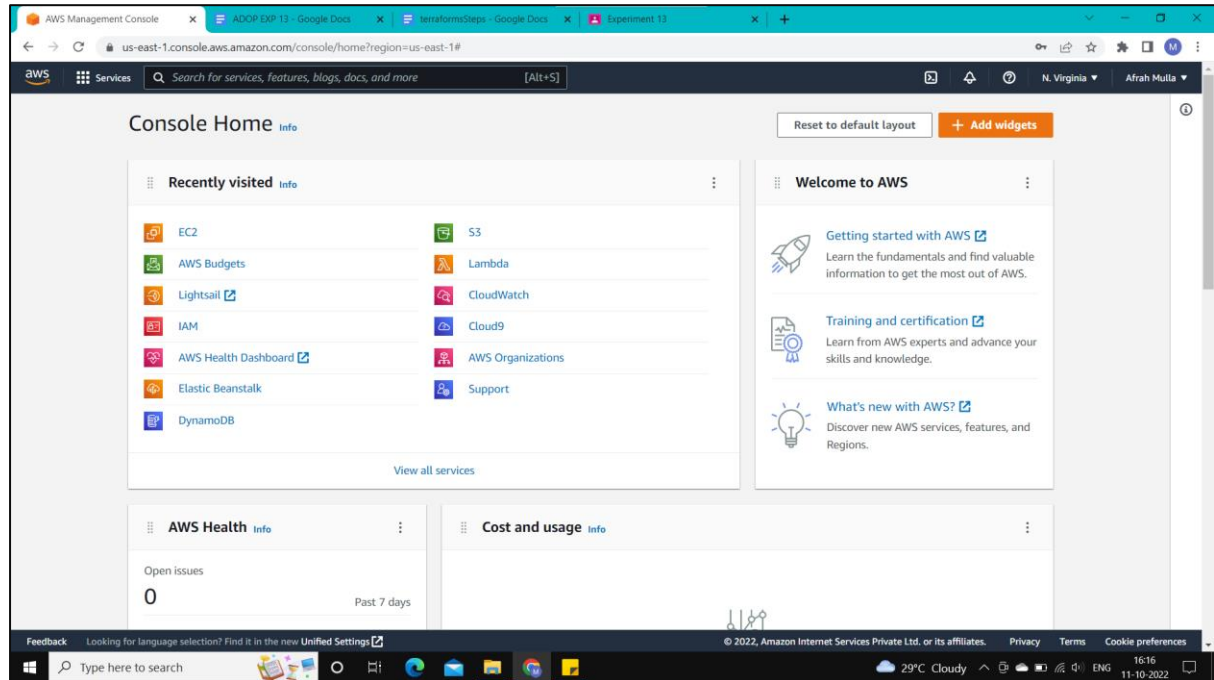
With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations. It also ensures that you provision the same environment every time.

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.

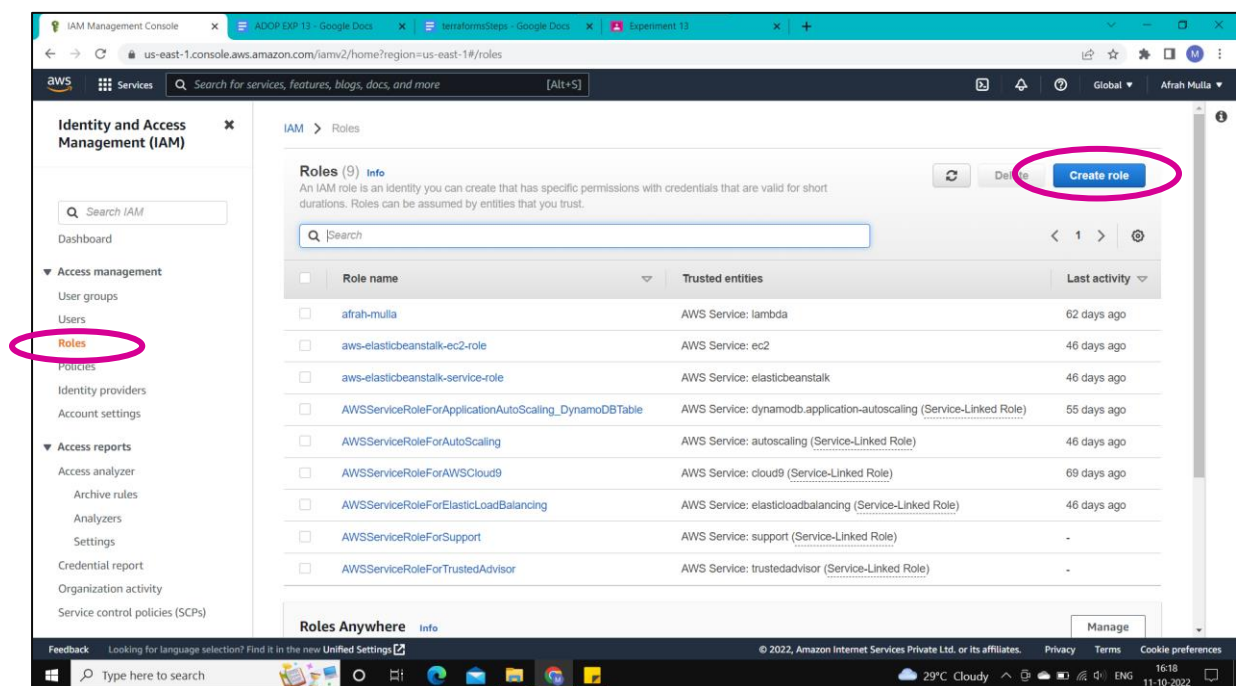
3. Perform an experiment, to understand Terraform lifecycle, core concepts/terminologies and install it on a Linux Machine.

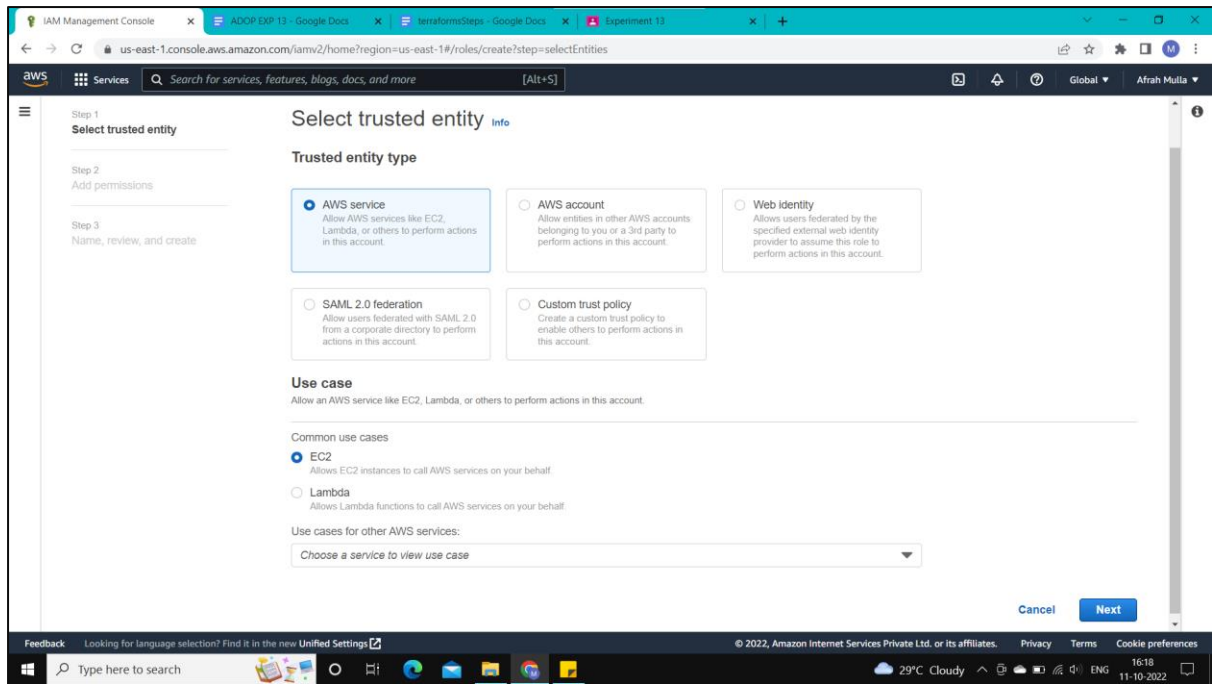
4. Using Terraform, create an EC2 instance on AWS cloud.

Step1: AWS Management Console Dashboard

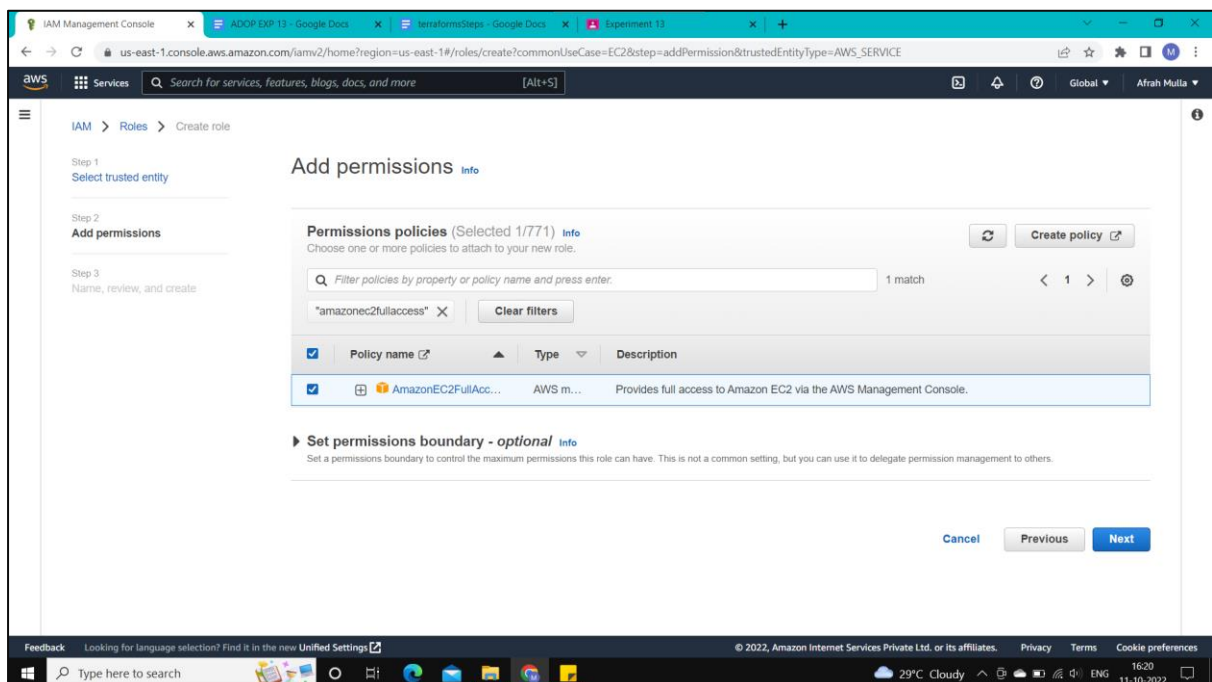


Step 2: IAM -> Roles -> Create role -> Set Trusted entity type to AWS service, use case to EC2





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



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

The screenshot shows the AWS IAM console 'Create role' page. The left sidebar indicates the current step is 'Step 1: Name, review, and create'. The main content area is titled 'Name, review, and create' and contains the 'Role details' section. The 'Role name' field is filled with 'exp13_afrah'. The 'Description' field contains 'Allows EC2 instances to call AWS services on your behalf.' Below the role details, the 'Step 1: Select trusted entities' section shows a JSON policy snippet for assuming an EC2 role.

Role details

Role name
Enter a meaningful name to identify this role.

Maximum 64 characters. Use alphanumeric and "+", "@", "-" characters.

Description
Add a short explanation for this role.

Maximum 1000 characters. Use alphanumeric and "+", "@", "-" characters.

Step 1: Select trusted entities

```
1- {  
2-   "Version": "2012-10-17",  
3-   "Statement": [  
4-     {  
5-       "Effect": "Allow",  
6-       "Action": [  
7-         "sts:AssumeRole"  
8-       ],  
9-       "Resource": "arn:aws:iam::123456789012:role/EC2Role"
```

The screenshot shows the AWS IAM console 'Create role' page, Step 2: Add permissions. The 'Permissions policy summary' table lists the 'AmazonEC2FullAccess' policy, which is AWS managed and attached as a permissions policy. Below this, the 'Tags' section indicates that no tags are currently associated with the resource.

Step 2: Add permissions

Permissions policy summary

Policy name	Type	Attached as
AmazonEC2FullAccess	AWS managed	Permissions policy

Tags

Add tags - optional [Info](#)

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

[Add tag](#)

You can add up to 50 more tags.

[Cancel](#) [Previous](#) [Create role](#)

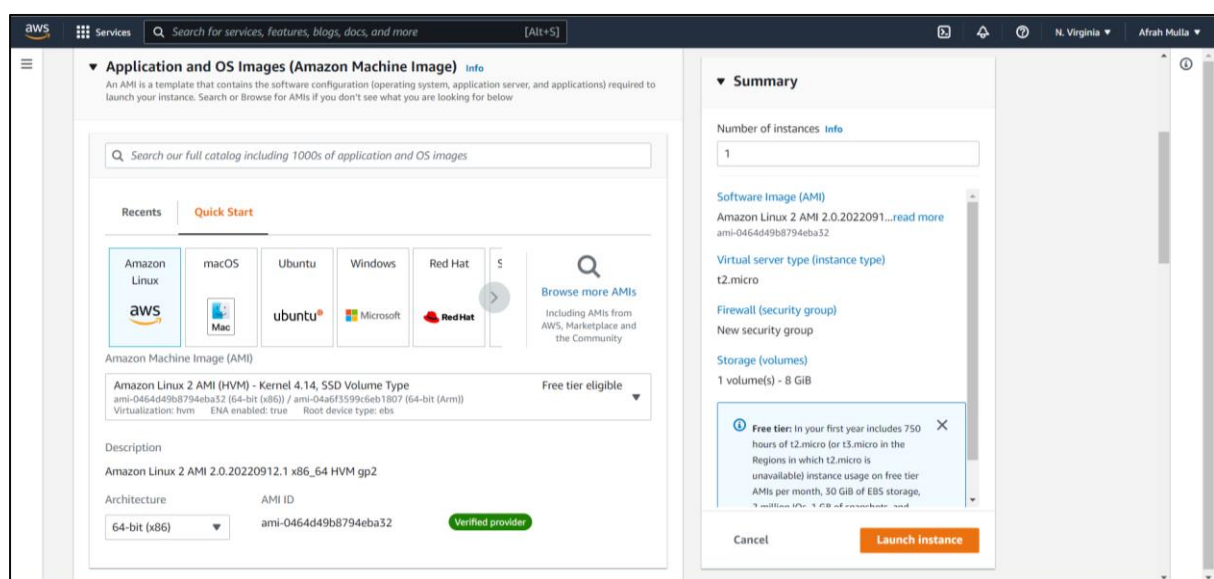
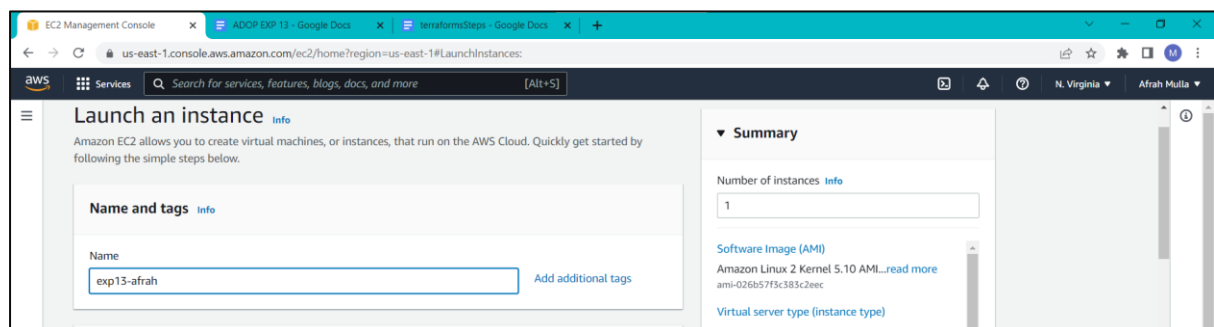
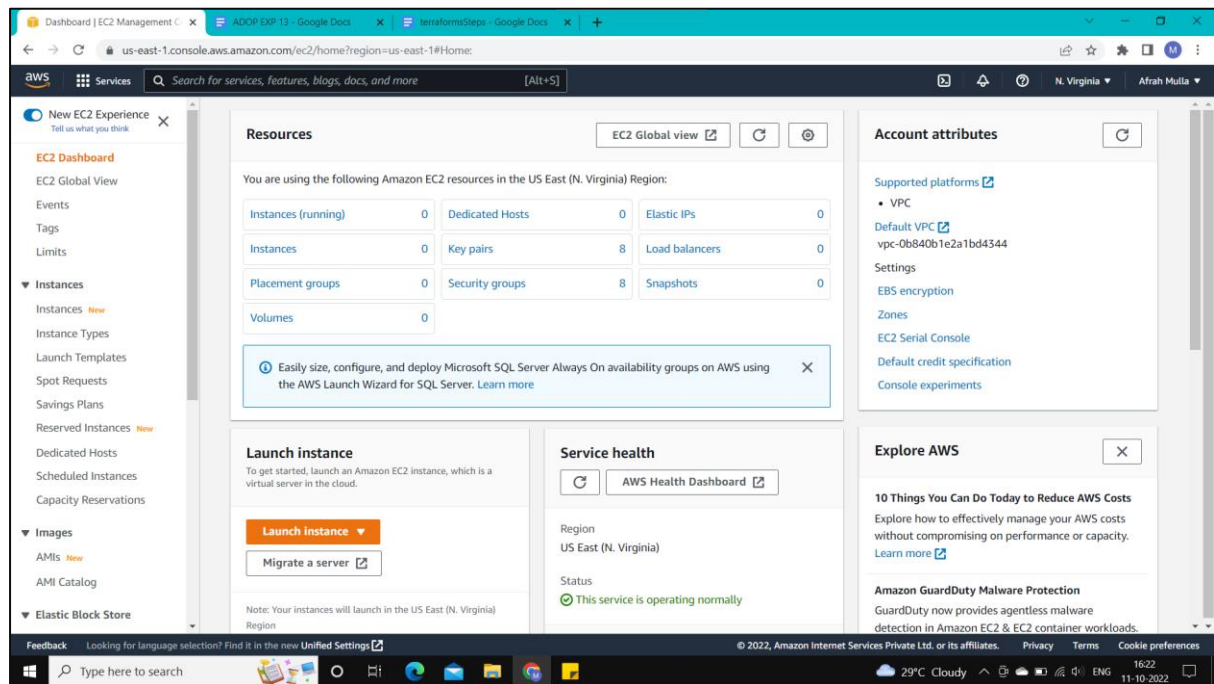
The screenshot shows the AWS IAM console after the role has been created. A green success message banner at the top states 'Role exp13_afrah created.' with a 'View role' button. The left sidebar shows the 'Identity and Access Management (IAM)' section, and the main content area displays the 'IAM > Roles' path.

Identity and Access Management (IAM)

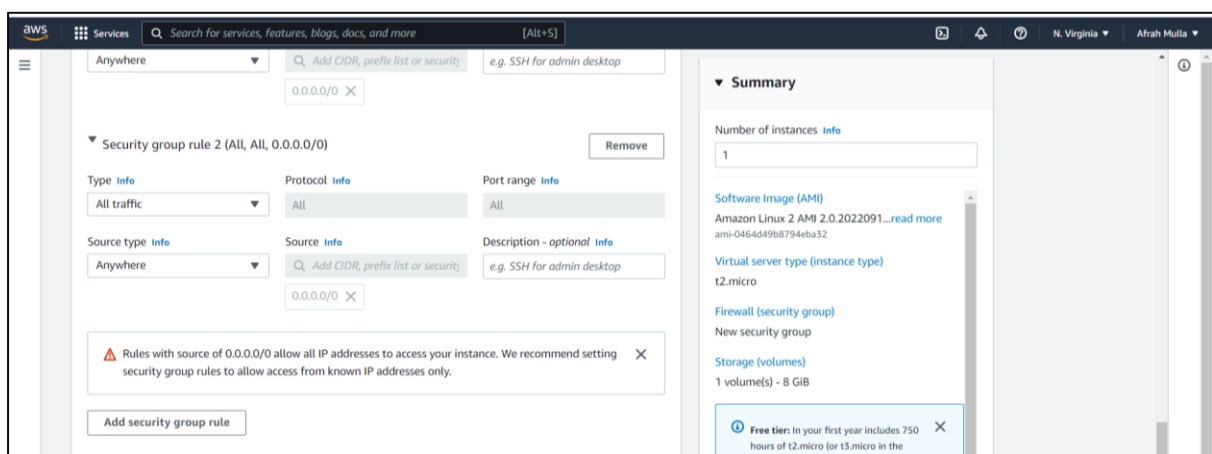
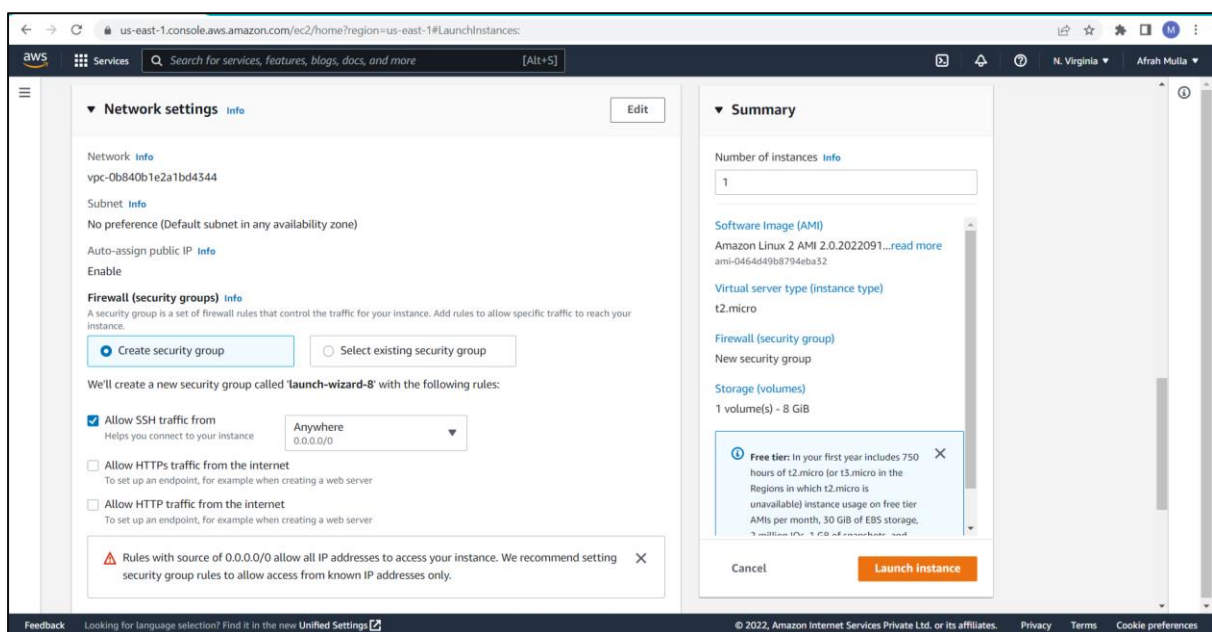
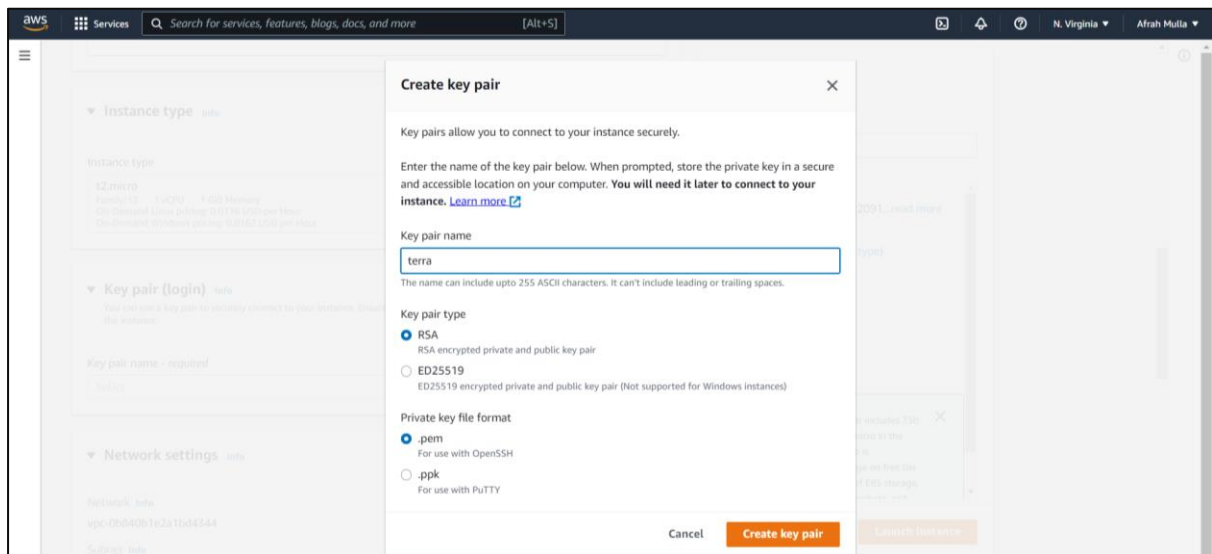
Role exp13_afrah created. [View role](#)

[IAM > Roles](#)

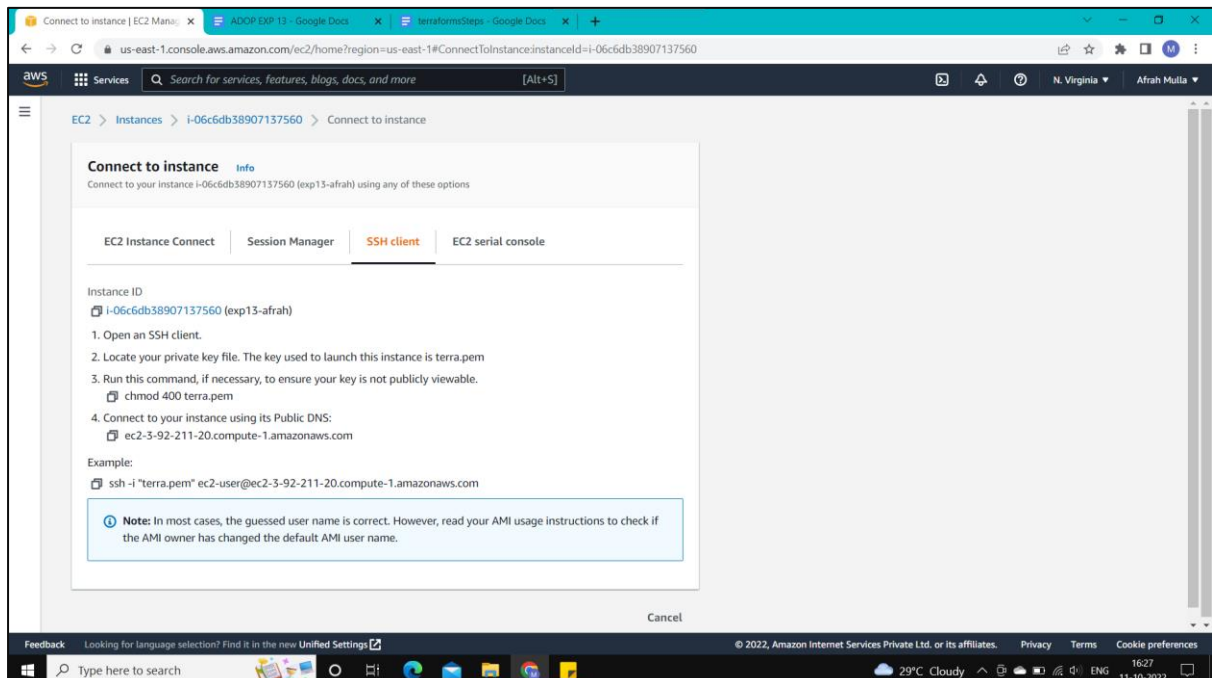
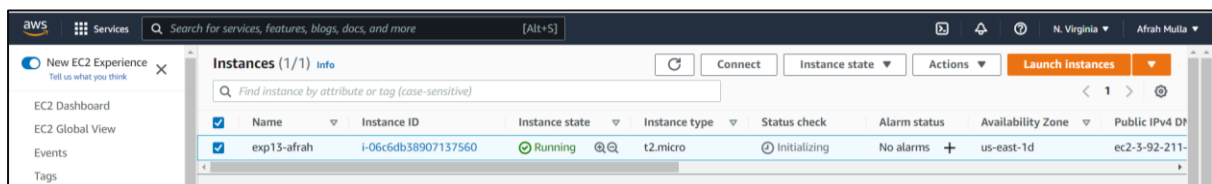
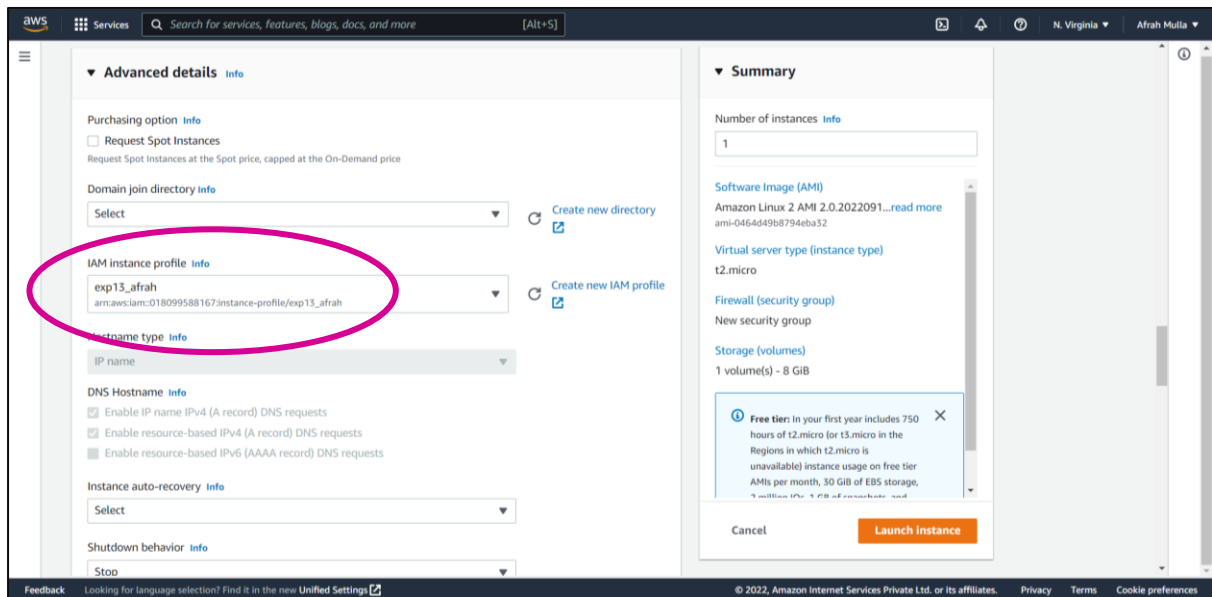
Step 5: Search for EC2 -> Select 'Launch an instance'. Assign a name to you instance -> Choose the 'Amazon Linux' machine -> Select the 'Amazon Linux 2 AMI (HVM) - Kernel 4.14, SSD Volume Type' machine image



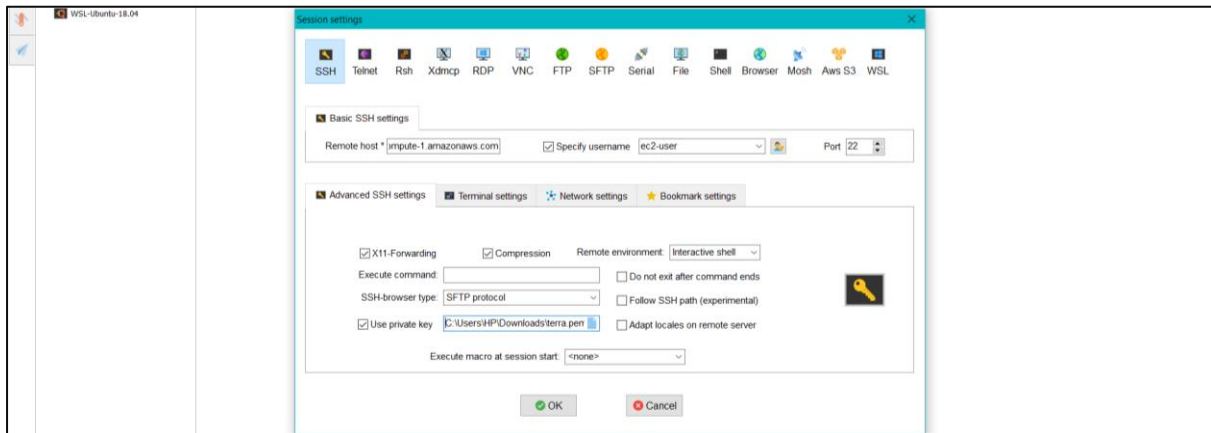
Step 6: Create a key pair -> In the network settings, allow the SSH traffic from anywhere -> Click on edit -> Add a security group for 'All traffic' and select the source type as 'anywhere'



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 Jenkins server
```

```
egress {  
    from_port = 0  
    to_port   = 65535  
    protocol  = "tcp"  
    cidr_blocks = ["0.0.0.0/0"]  
}
```

```
tags= {  
    Name = var.security_group  
}  
}
```

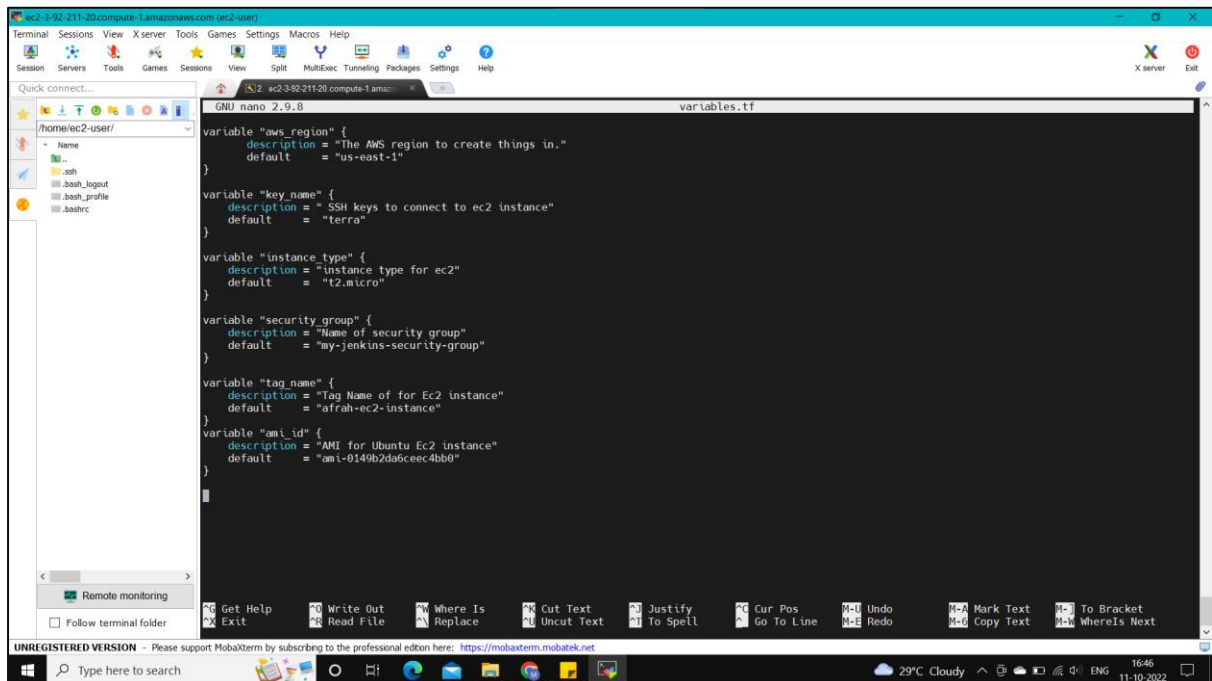
```
resource "aws_instance" "myFirstInstance" {  
    ami          = var.ami_id  
    key_name     = var.key_name  
    instance_type = var.instance_type  
    security_groups = [var.security_group]  
    tags= {  
        Name = var.tag_name  
    }  
}
```

```
}
```

Create Elastic IP address

```
resource "aws_eip" "myFirstInstance" {  
  
    vpc      = true  
  
    instance = aws_instance.myFirstInstance.id  
  
    tags = {  
  
        Name = "jenkins_elastic_ip"  
  
    }  
}
```

```
https://aws.amazon.com/amazon-linux-2/  
[ec2-user@ip-172-31-81-213 ~]$ sudo su  
[root@ip-172-31-81-213 ec2-user]# mkdir exp13_afrah  
[root@ip-172-31-81-213 ec2-user]# cd exp13_afrah  
[root@ip-172-31-81-213 exp13_afrah]# nano variables.tf  
[root@ip-172-31-81-213 exp13_afrah]# nano main.tf
```



The screenshot shows a MobaXterm terminal window with a file explorer on the left displaying the directory structure of a remote host. The terminal window is running GNU nano 2.9.8 and shows the following Terraform configuration in 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 jenkins 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" {
```

The terminal window includes a status bar at the bottom with system information: 29°C Cloudy, 1636, and 11-10-2022.

The screenshot shows the continuation of the Terraform configuration in main.tf from the previous window:

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

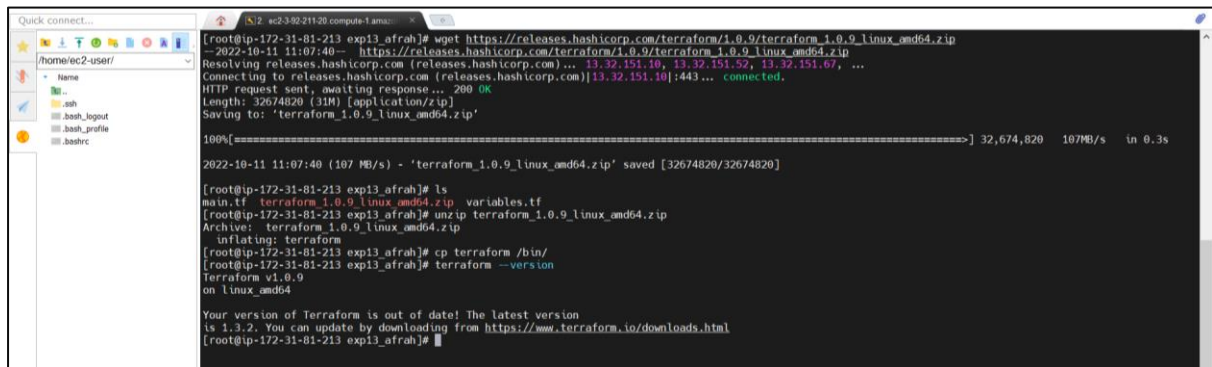
# Create Elastic IP address
resource "aws_eip" "myFirstInstance" {
  vpc      = true
  instance = aws_instance.myFirstInstance.id
  tags = {
    Name = "jenkins_elastic_ip"
  }
}
```

The terminal window includes a status bar at the bottom with system information: 29°C Cloudy, 1636, and 11-10-2022.

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`



```
[root@ip-172-31-81-213 exp13_afrah]# wget https://releases.hashicorp.com/terraform/1.0.9/terraform_1.0.9_linux_amd64.zip
--2022-10-11 11:07:40-- https://releases.hashicorp.com/terraform/1.0.9/terraform_1.0.9_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 13.32.151.10, 13.32.151.52, 13.32.151.67, ...
Connecting to releases.hashicorp.com (releases.hashicorp.com)|13.32.151.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 32674820 (31M) [application/zip]
Saving to: 'terraform_1.0.9_linux_amd64.zip'

100%[=====] 32,674,820  107MB/s  in 0.3s

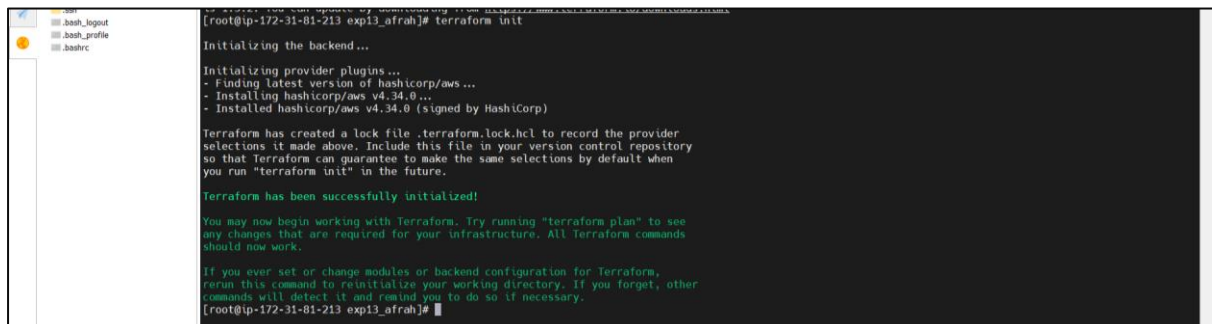
2022-10-11 11:07:40 (107 MB/s) - 'terraform_1.0.9_linux_amd64.zip' saved [32674820/32674820]

[root@ip-172-31-81-213 exp13_afrah]# ls
main.tf  terraform_1.0.9_linux_amd64.zip  variables.tf
[root@ip-172-31-81-213 exp13_afrah]# unzip terraform_1.0.9_linux_amd64.zip
Archive:  terraform_1.0.9_linux_amd64.zip
  inflating: terraform
[root@ip-172-31-81-213 exp13_afrah]# cp terraform /bin/
[root@ip-172-31-81-213 exp13_afrah]# terraform --version
Terraform v1.0.9
on linux_amd64

Your version of Terraform is out of date! The latest version
ls 1.3.2. You can update by downloading from https://www.terraform.io/downloads.html
[root@ip-172-31-81-213 exp13_afrah]#
```

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 init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v4.34.0...
- Installed hashicorp/aws v4.34.0 (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.
[root@ip-172-31-81-213 exp13_afrah]#
```

```
Quick connect...
/home/ec2-user/
+ Name
+ .ssh
+ .ssh_logout
+ .ssh_profile
+ .sshrc

[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)
}
```

```
Quick connect...
/home/ec2-user/
+ Name
+ .ssh
+ .ssh_logout
+ .ssh_profile
+ .sshrc

+ 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 it takes exactly these actions if you run "terraform apply" now.
[root@ip-172-31-81-213 exp13_afrah]#
```

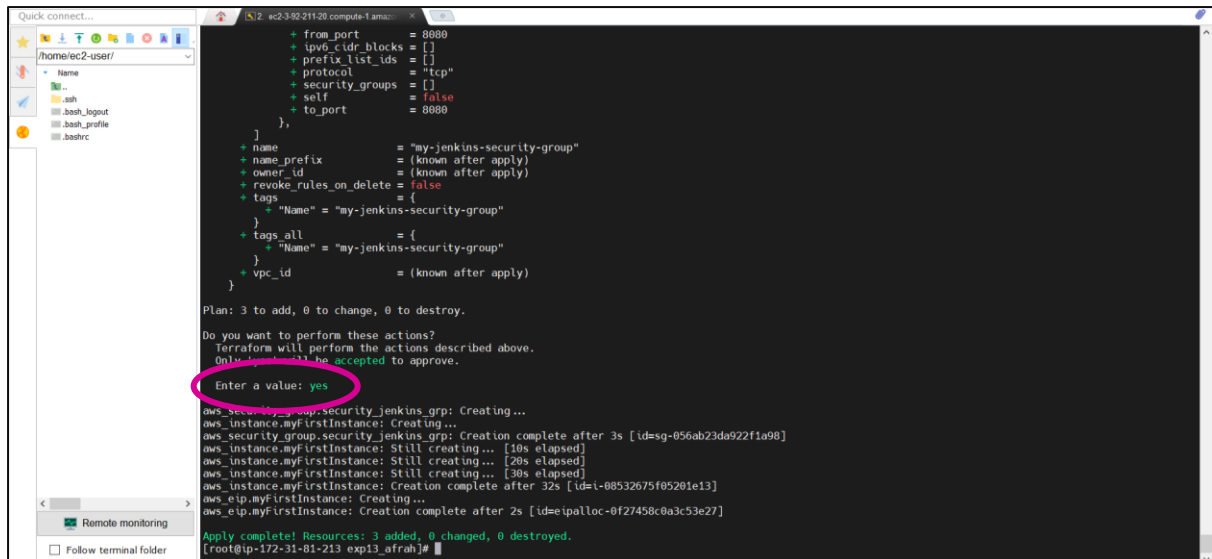
```
Quick connect...
/home/ec2-user/
+ Name
+ .ssh
+ .ssh_logout
+ .ssh_profile
+ .sshrc

[root@ip-172-31-81-213 exp13_afrah]# 0
[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

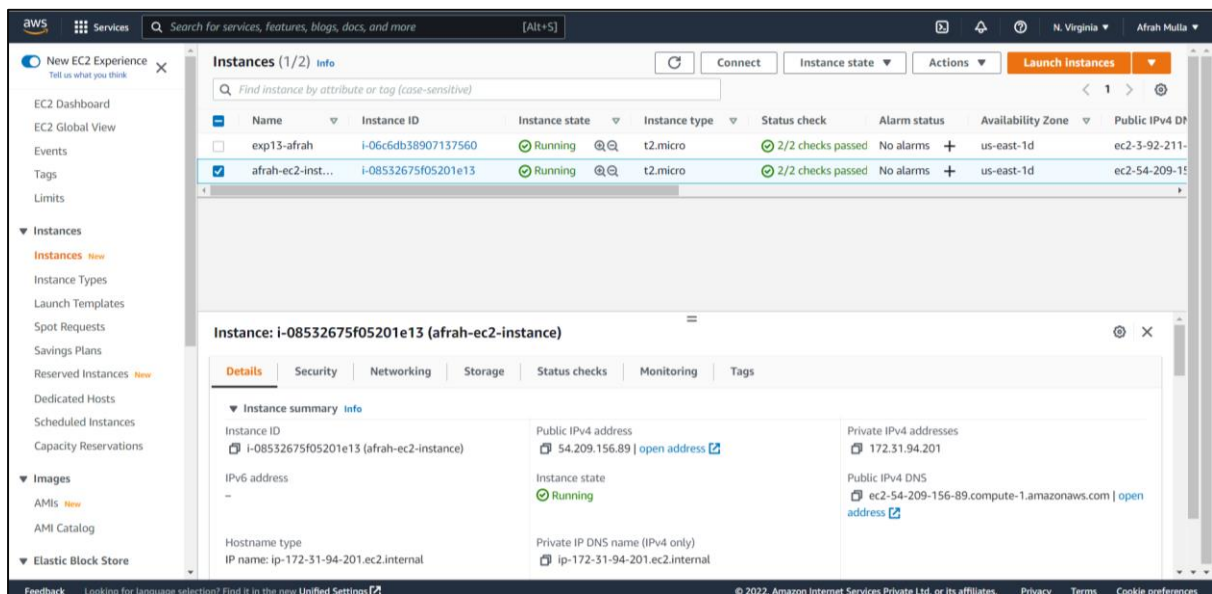
Terraform will perform the following actions:

# 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
}

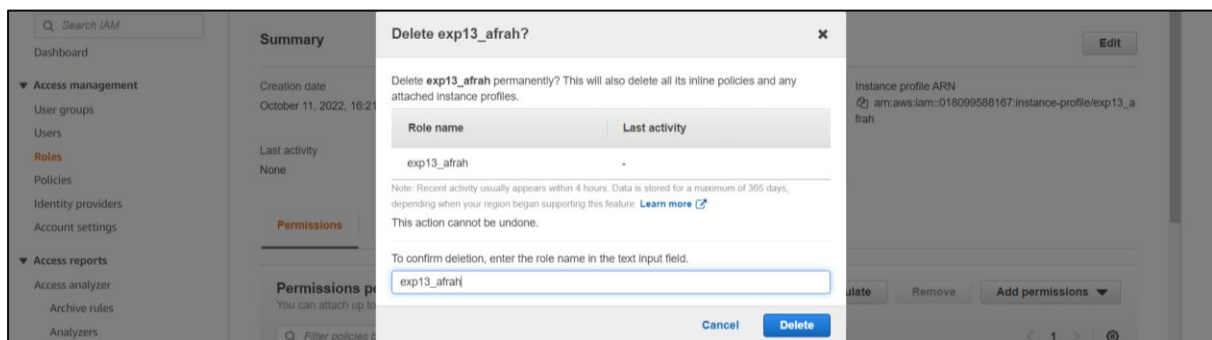
# aws_instance.myFirstInstance will be created
+ resource "aws_instance" "myFirstInstance" {
+   ami              = "ami-0149b2da6ceec4bb0"
+   arn              = (known after apply)
+   associate_public_ip_address = (known after apply)
+   availability_zone = (known after apply)
+   cpu_core_count   = (known after apply)
+   cpu_threads_per_core = (known after apply)
+   disable_api_stop  = (known after apply)
}
```

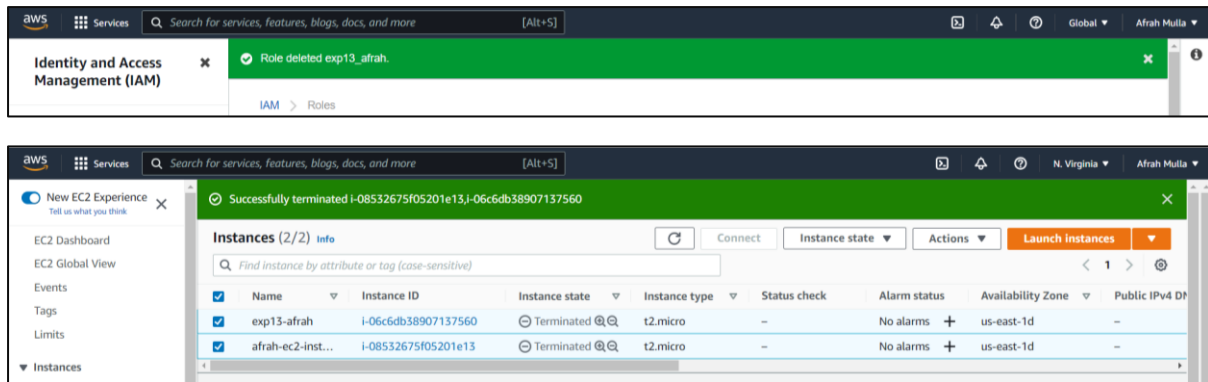



Step 12: Now go to EC2 console, to see the new instances up and running



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.