**Name: Baloch Muzammil Hafeez**

**Roll No.: 612012**

**Branch: T.E. / I.T.**

**Subject: ADOP LAB**

**Experiment No. 13**

**Q1. What is Terraform**

**Ans:**

1. Terraform is an open-source, infrastructure as code, software tool created by HashiCorp. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON.
2. Terraform manages external resources with "providers".
3. HashiCorp maintains an extensive list of official providers, and can also integrate with community-developed providers.
4. Users can interact with Terraform providers by declaring resources or by calling data sources. Rather than using imperative commands to provision resources, Terraform uses declarative configuration to describe the desired final state.
5. Once a user invokes Terraform on a given resource, Terraform will perform CRUD actions on the user's behalf to accomplish the desired state.
6. The infrastructure as code can be written as modules, promoting reusability and maintainability.
7. Terraform supports a number of cloud infrastructure providers such as Amazon Web Services, Microsoft Azure, IBM Cloud, Serverspace, Google Cloud Platform, DigitalOcean, Oracle Cloud Infrastructure, Yandex.Cloud, VMware vSphere, and OpenStack.

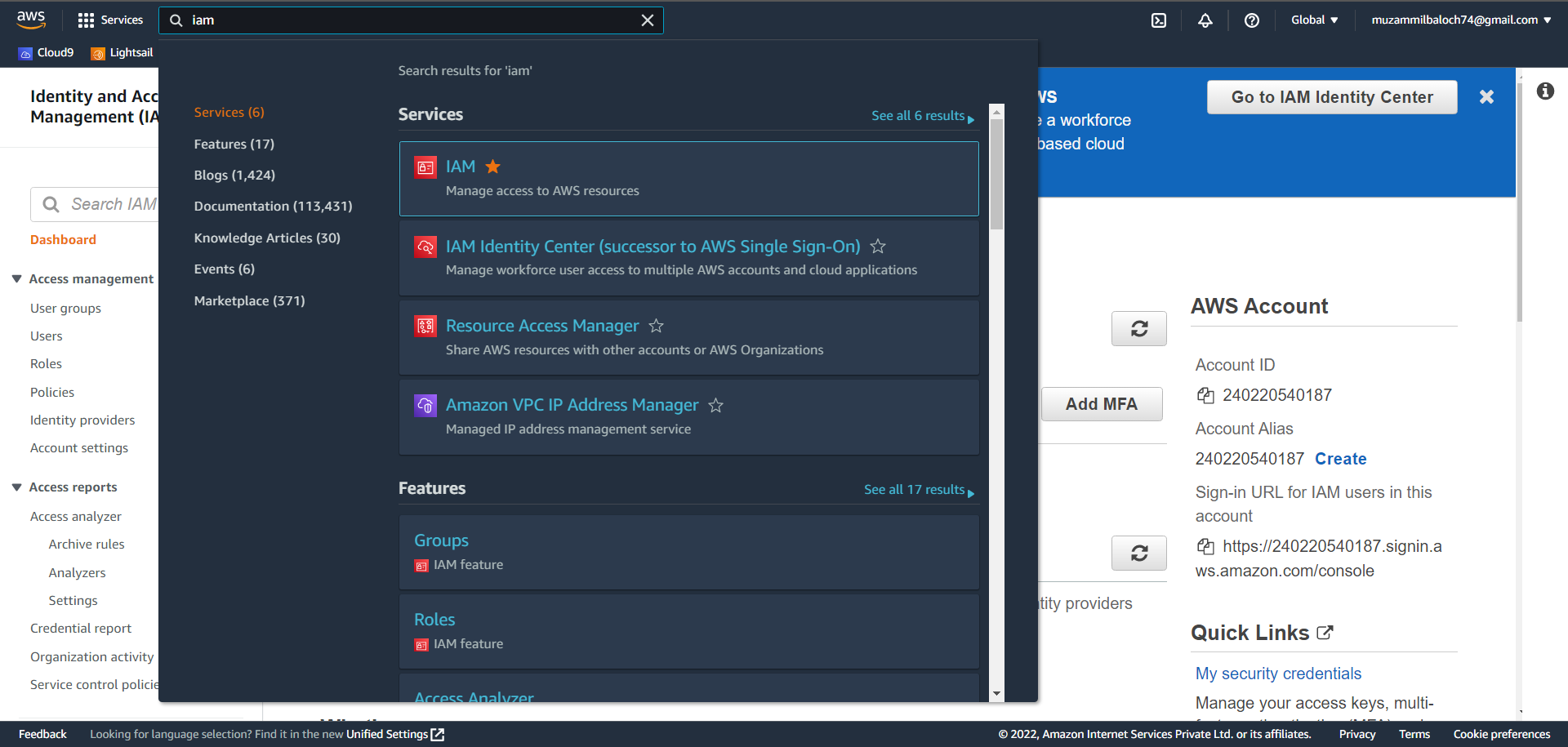
**Q2. What is Infrastructure as a Code (IaC)**

**Ans:**

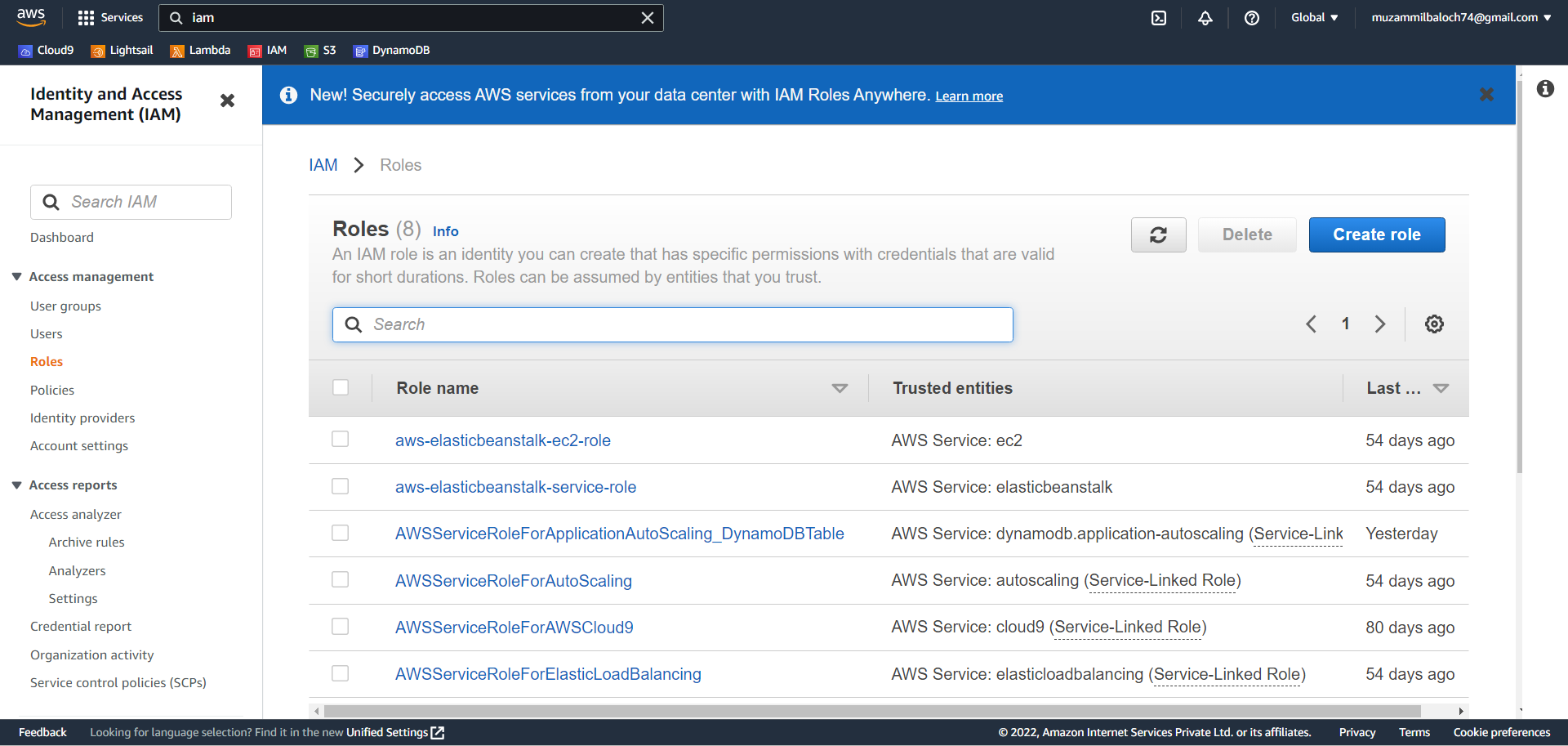
1. Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.
2. The IT infrastructure managed by this process comprises both physical equipment, such as bare-metal servers, as well as virtual machines, and associated configuration resources.
3. The definitions may be in a version control system.
4. The code in the definition files may use either scripts or declarative definitions, rather than maintaining the code through manual processes, but IaC more often employs declarative approaches.
5. There are two methods of IaC: push and pull.
6. The main difference is the manner in which the servers are told how to be configured.
7. In the pull method, the server to be configured will pull its configuration from the controlling server.
8. In the push method, the controlling server pushes the configuration to the destination system.

**Q3. Perform an experiment, to understand Terraform lifecycle, core concepts/terminologies and install it on a Linux Machine Ans:**

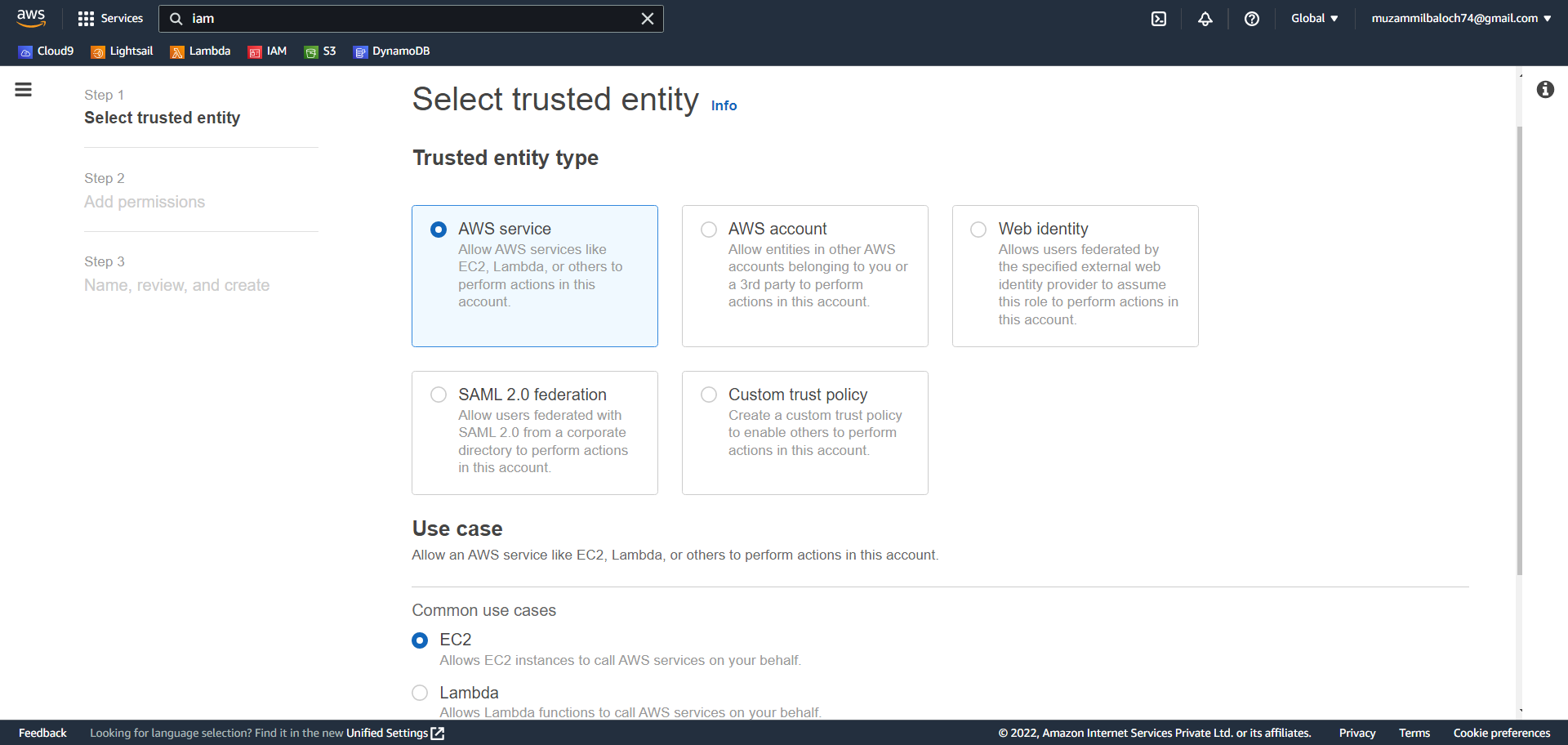
1. Login in you aws account
2. In your console management search for i am role



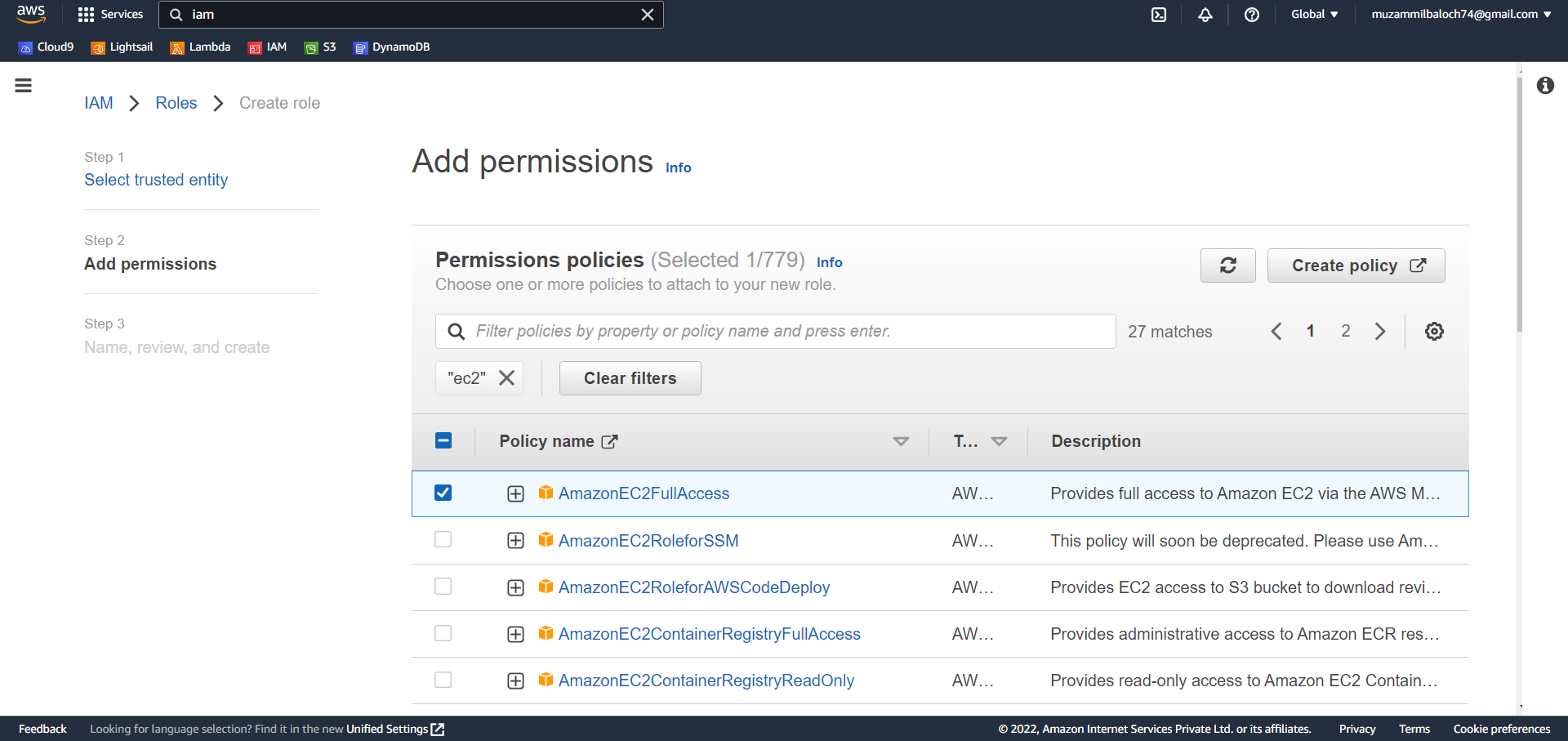
1. Click on role and then click create role



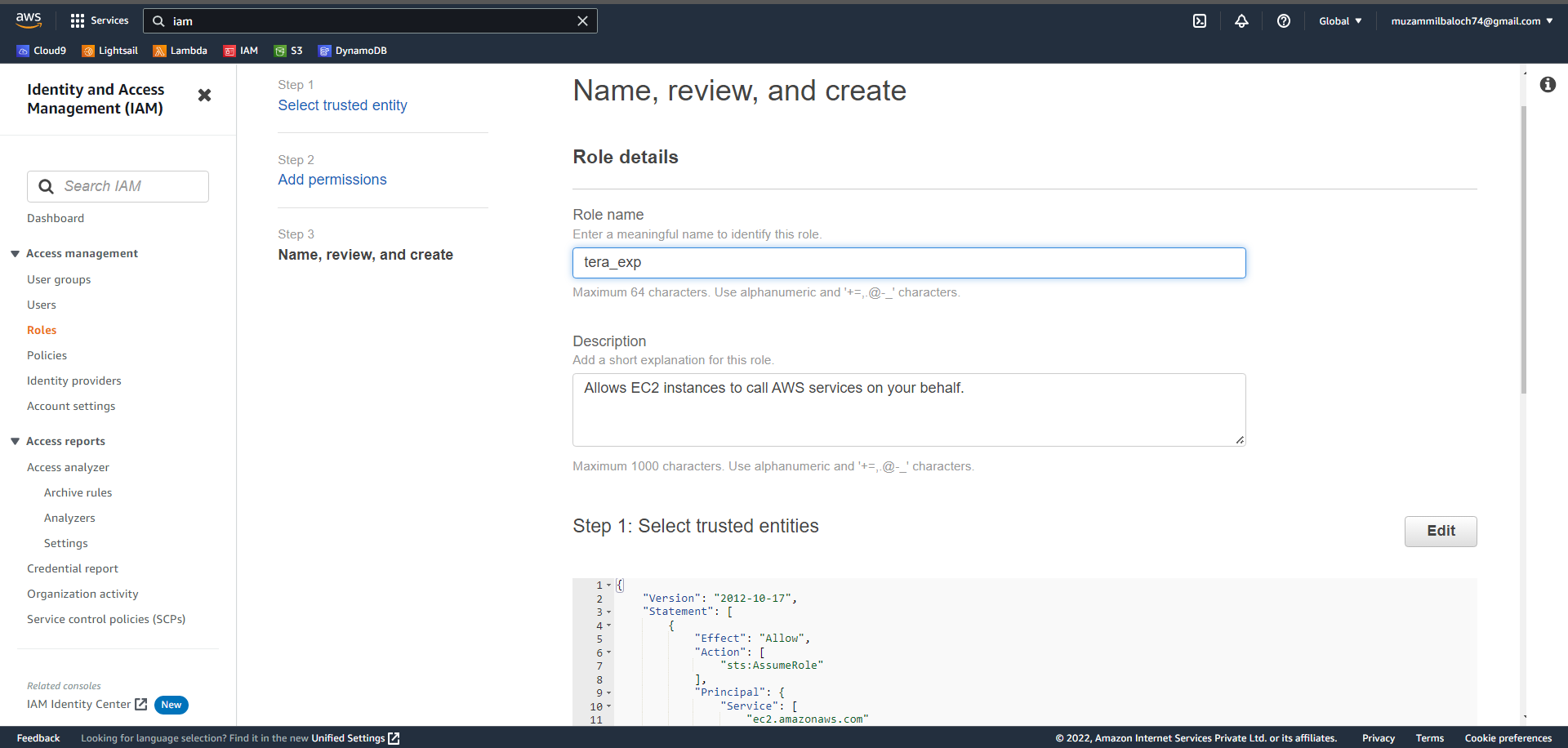
1. Select aws service and select EC2 in use case



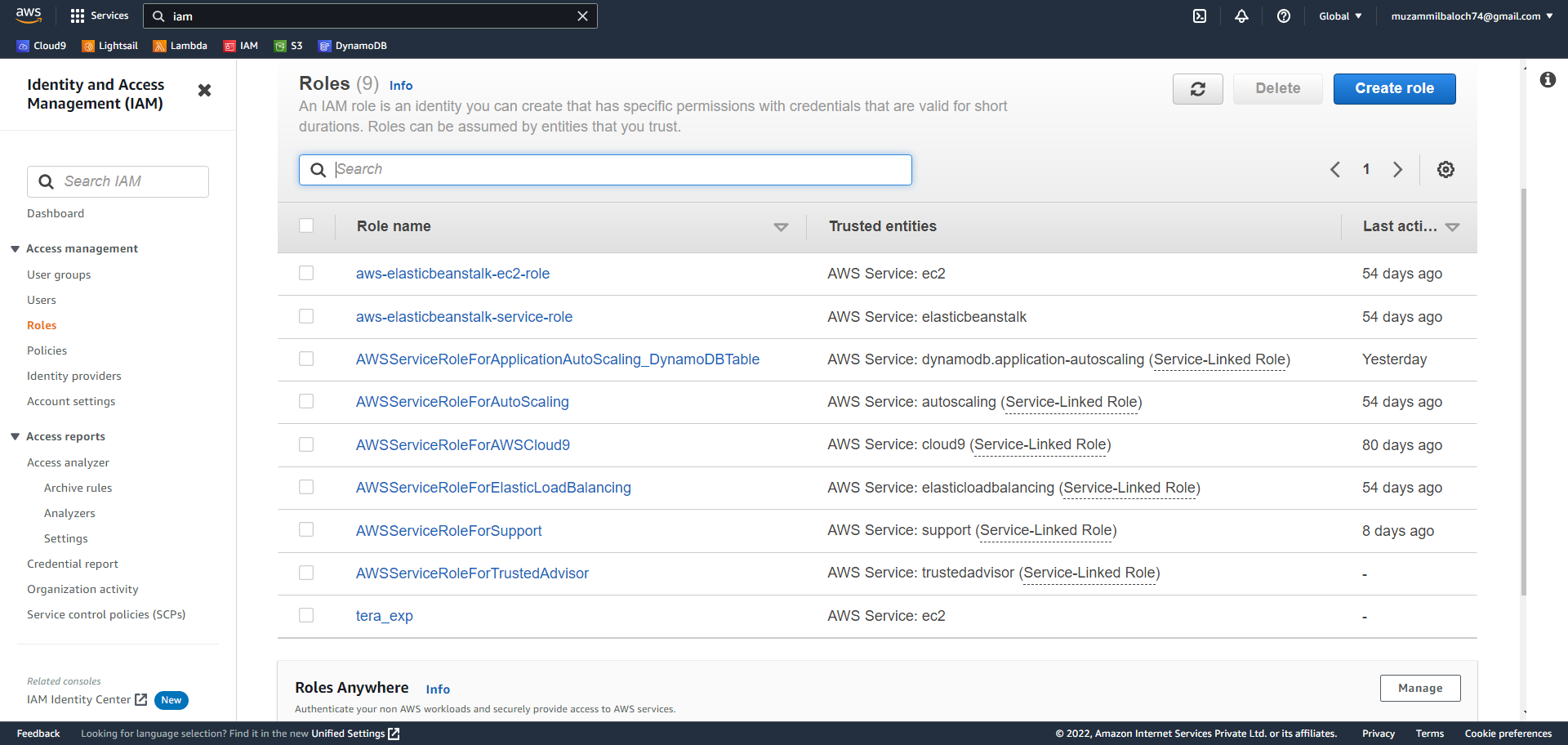
1. Search for AmazonEC2 and select Amazon EC2 via the AWS Management Console. And click on next



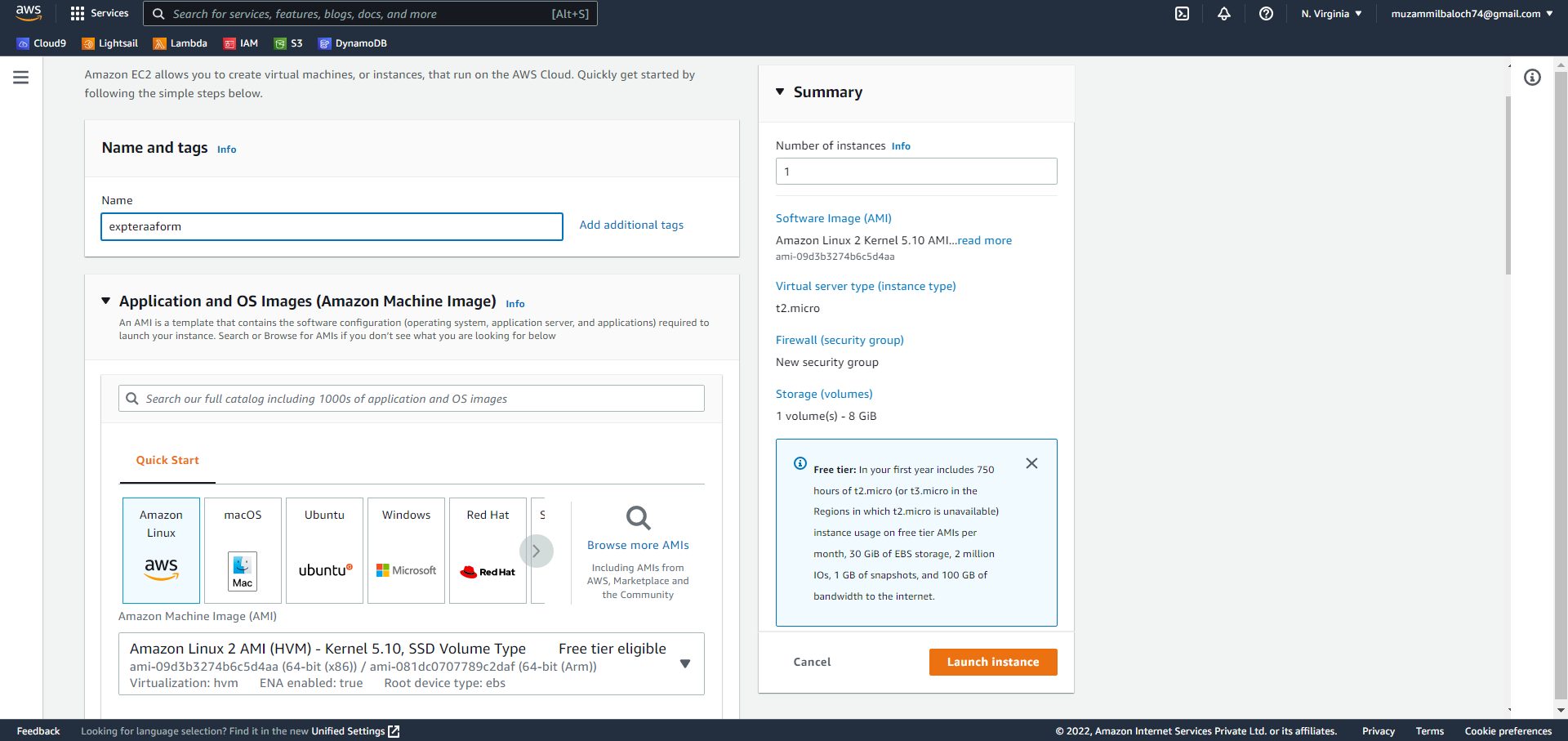
1. Select a name for your role and click on create role



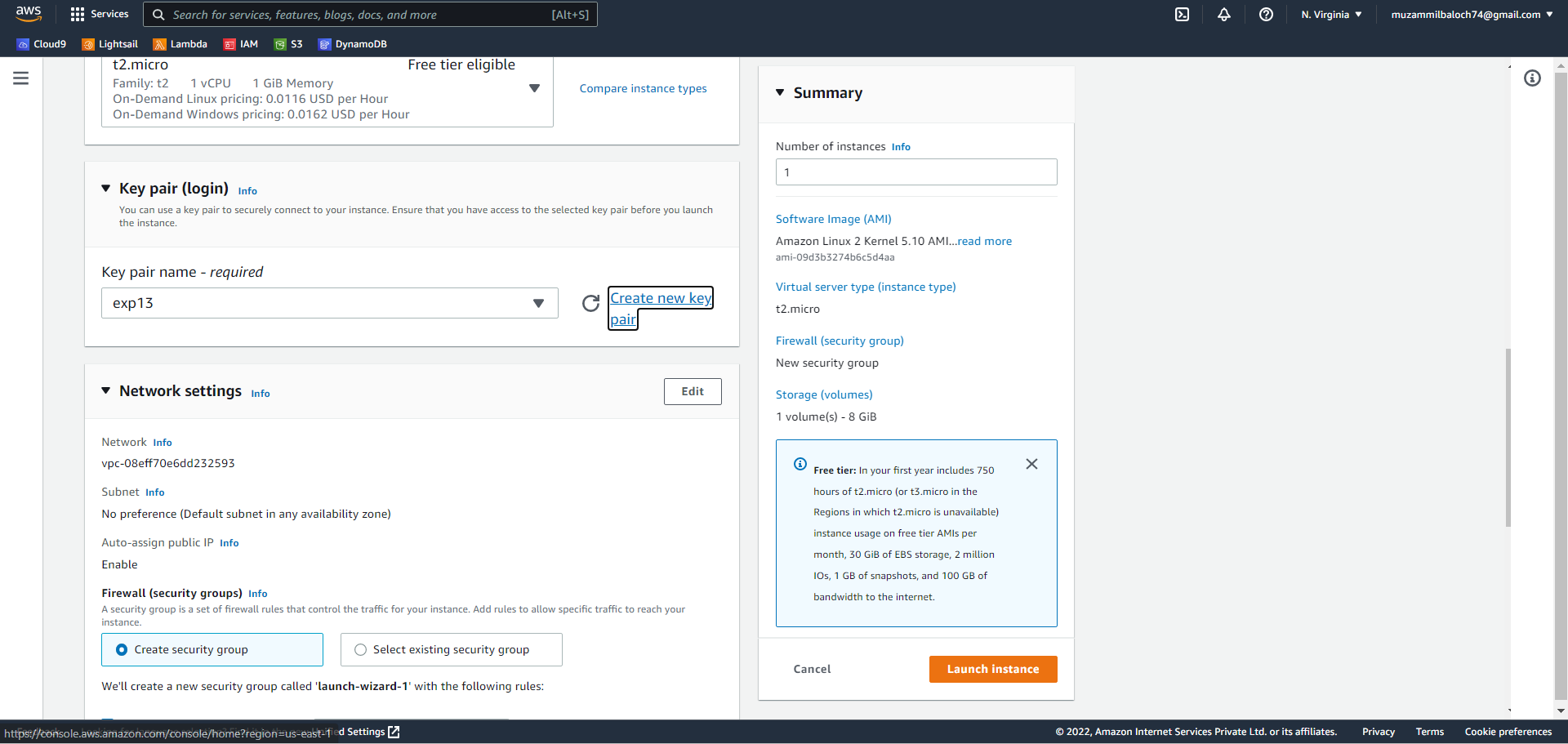
g. Done role created



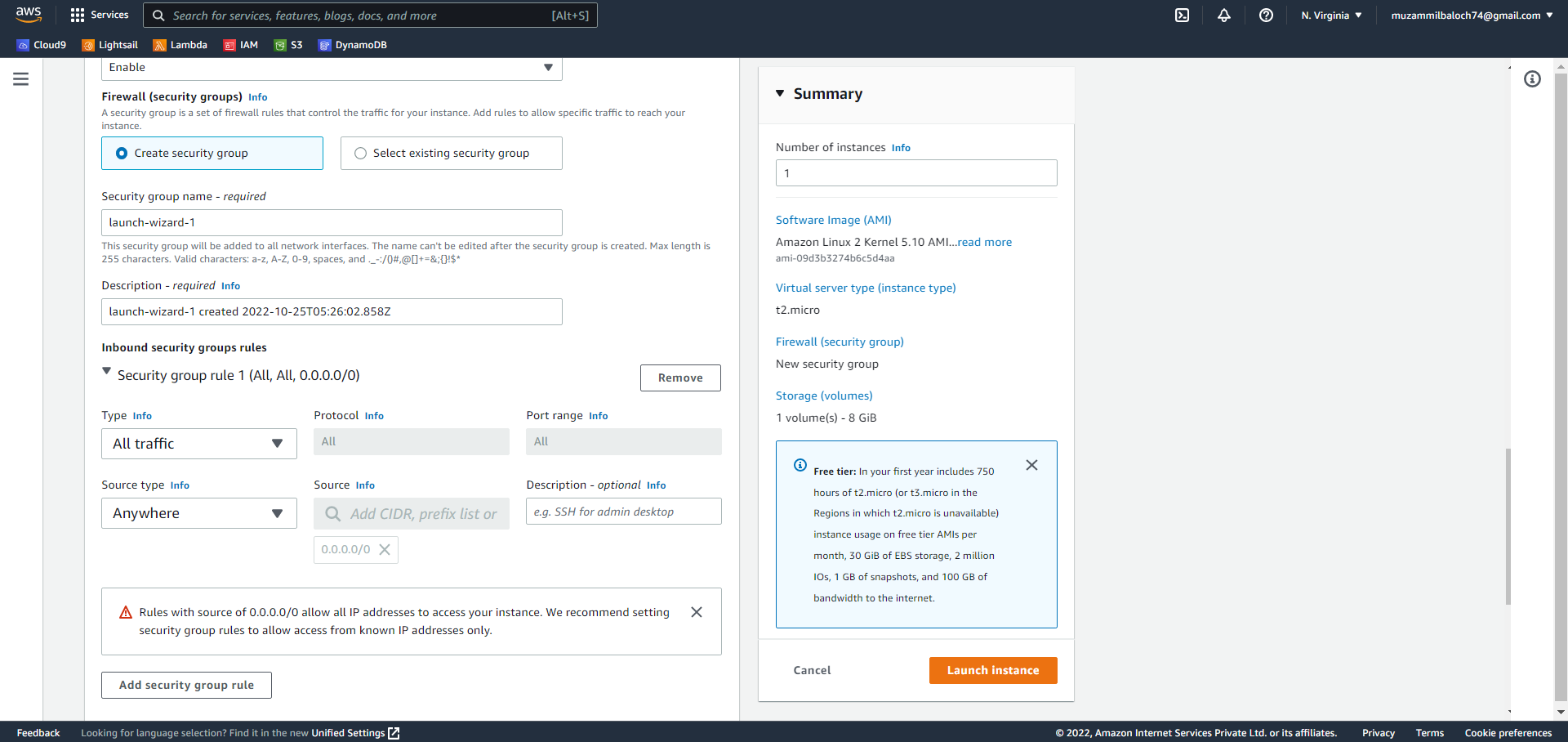
1. Now goto EC2 machine and click on launch instance then select a name for your machine and select amazon linux as you OS



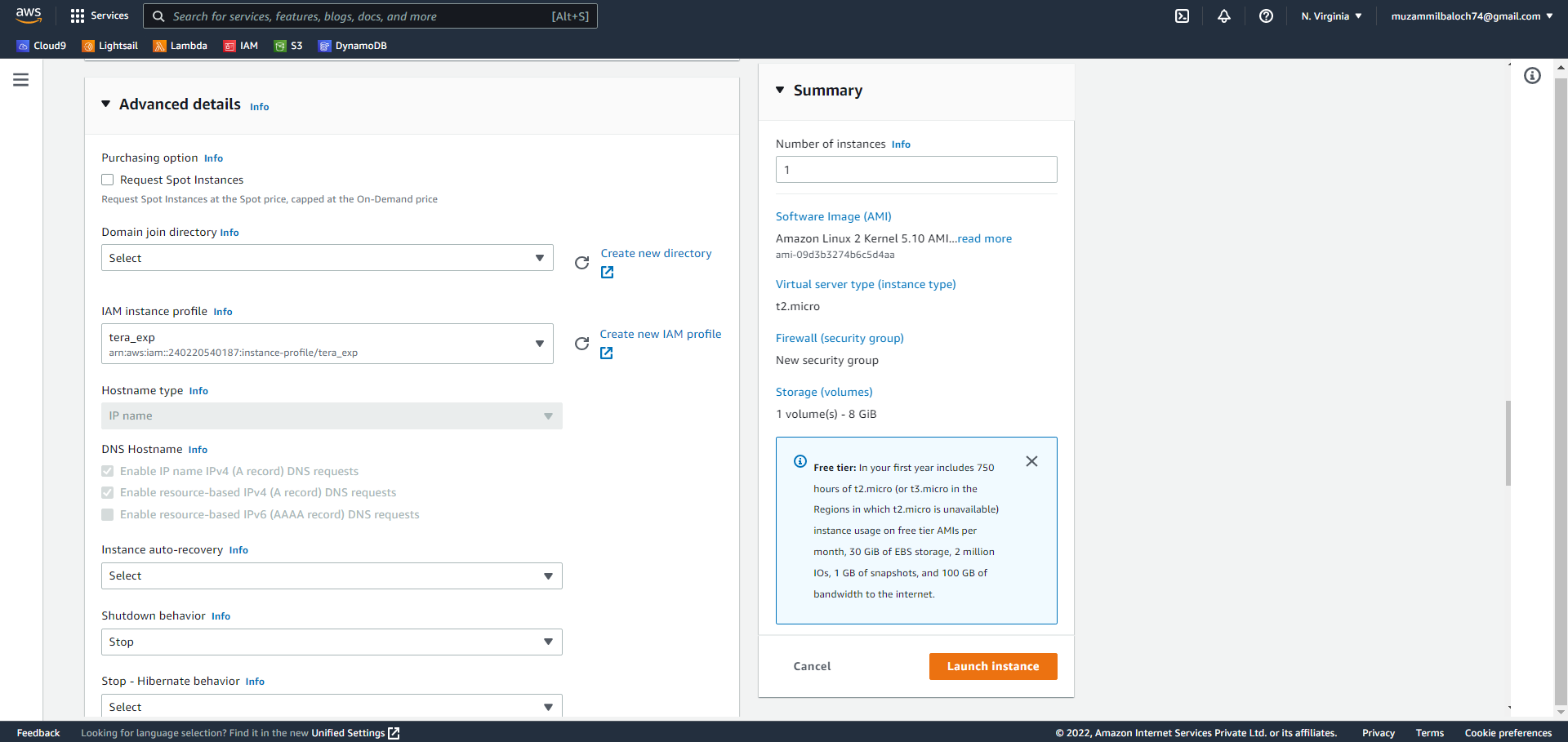
1. Create a key pair for your linux machine



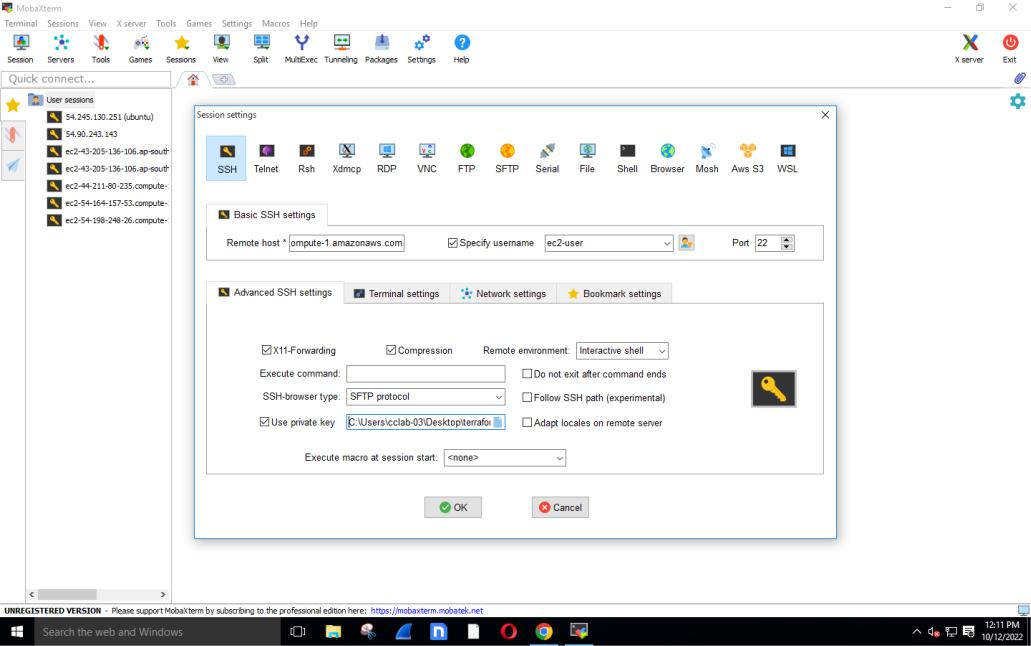
1. In network setting click on edit and add security group role and select type as all traffic and source type as anywhere



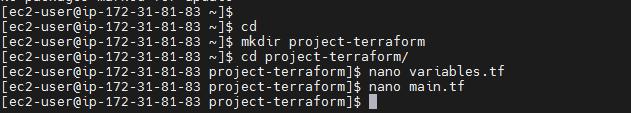
1. Now click on advance setting and select the role we have created early and click on launch instance



1. After the machine is create click on connect and goto ssh client and copy public dns and copy username to connect on moboxterm



1. After connecting, first update and upgrade your machine and then create a dir using command “ mkdir project-terraform ” and change dir to project-terraform. Create a file variables.tf and main.tf
2. Paste the code in your variables.tf file



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 = "terraformkey"

}

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 = "my-ec2-instance"

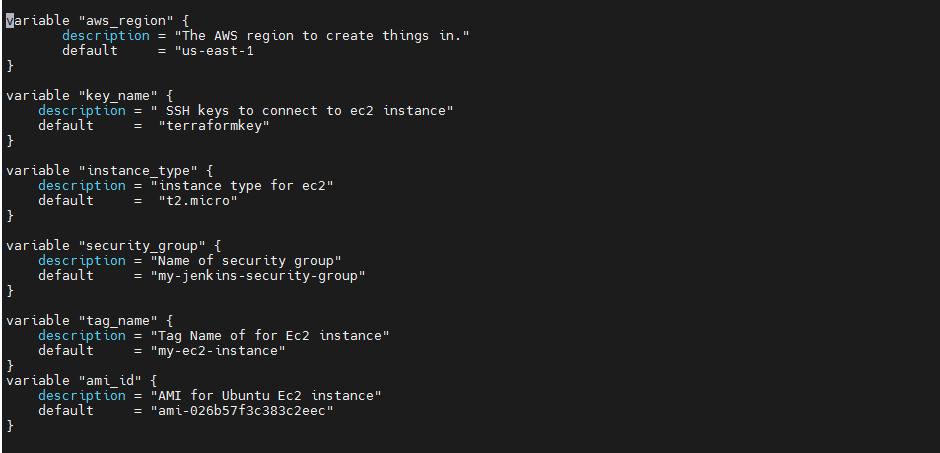
}

variable "ami\_id" {

description = "AMI for Ubuntu Ec2 instance"

default = "ami-026b57f3c383c2eec"

}



1. Create main.tf and paste the code : provider "aws" {

region = var.aws\_region

}

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"]

}

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

}

}

resource "aws\_eip" "myFirstInstance" {

vpc = true

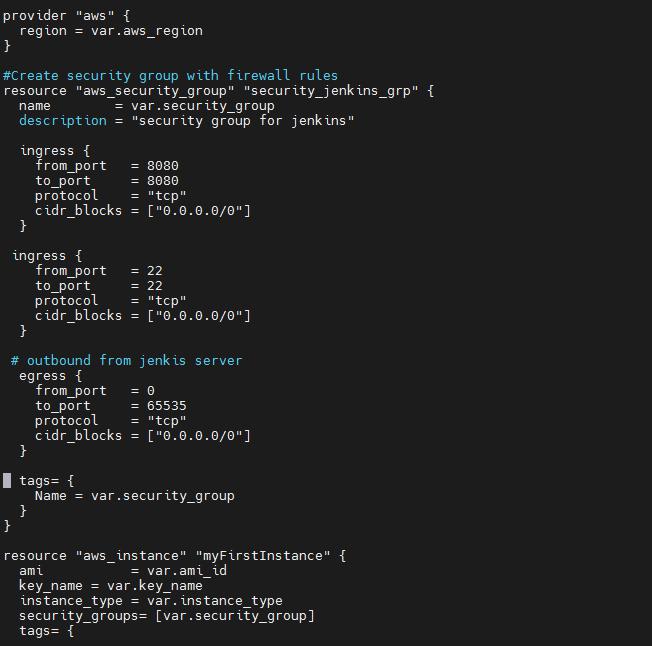
instance = aws\_instance.myFirstInstance.id

tags= {

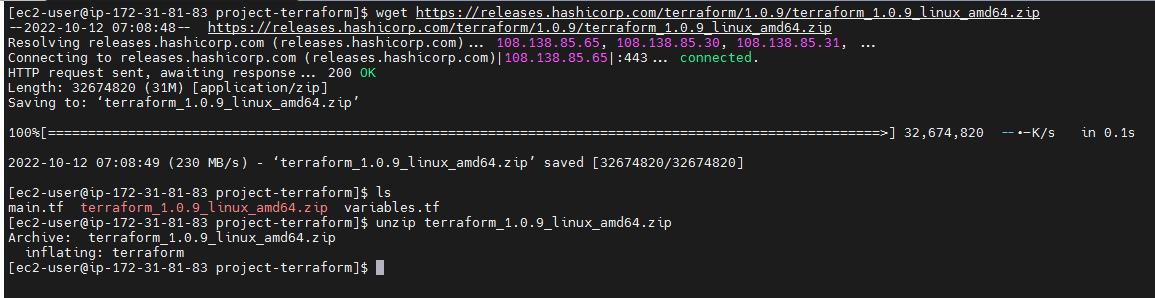
Name = "jenkins\_elastic\_ip"

}

}



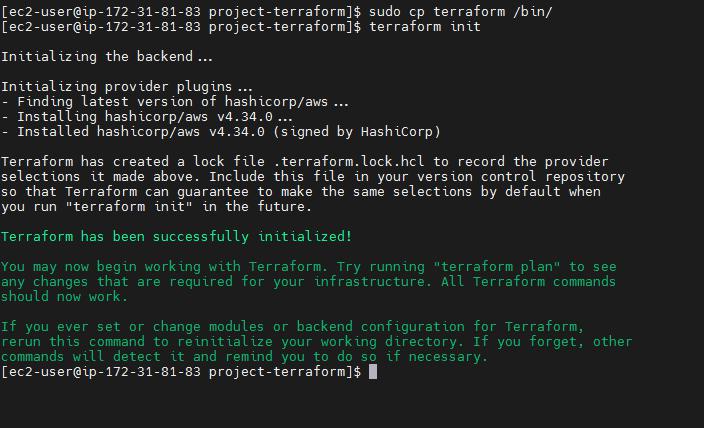
1. Now get the terraform by the following command “ wget https://releases.hashicorp.com/terraform/1.0.9/terraform\_1.0.9\_linux\_amd64.zip” and unzip using “ unzip terraform\_1.0.9\_linux\_amd64.zip”



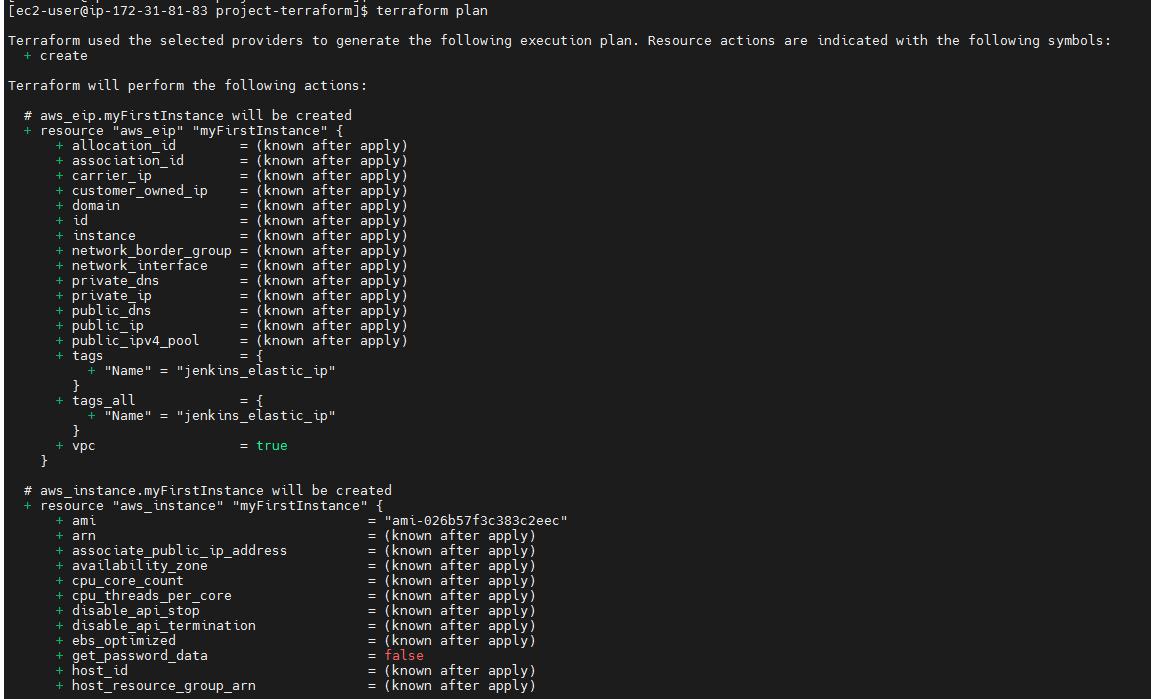
**Q4. Using Terraform , create an EC2 instance on AWS cloud**

**Ans:**

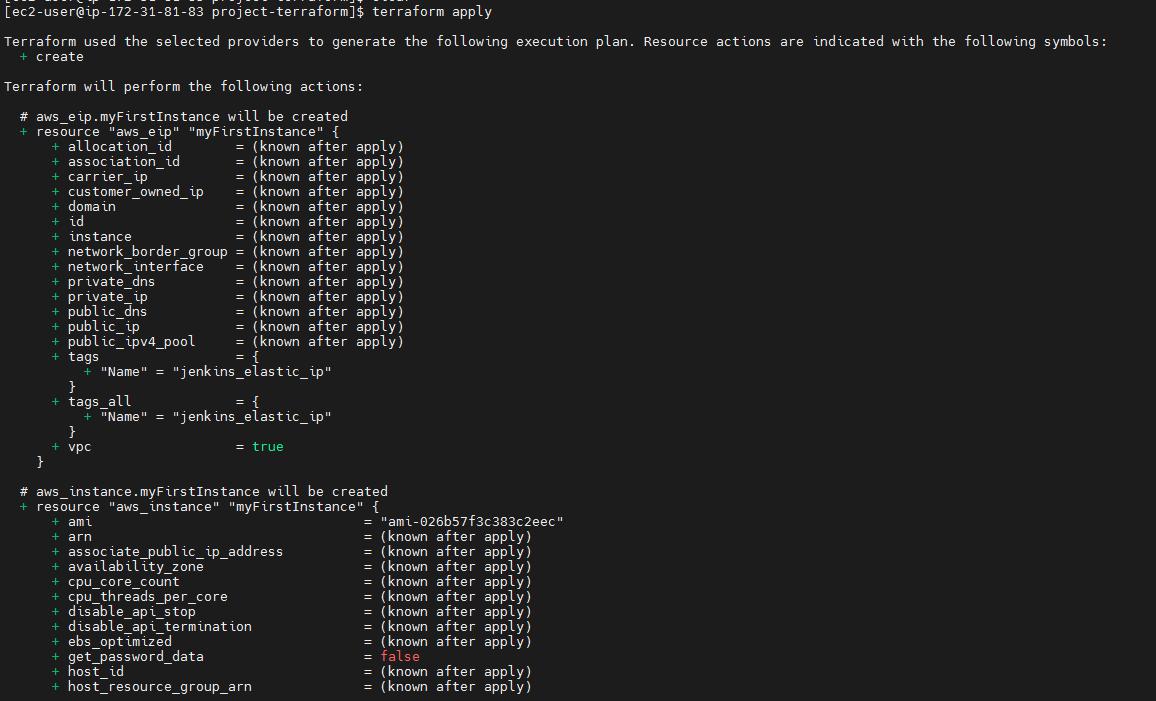
1. Now copy the terraform dir to /bin/ using command “ **cp Terraform /bin/** ” and change to your project-terraform dir and type terraform init

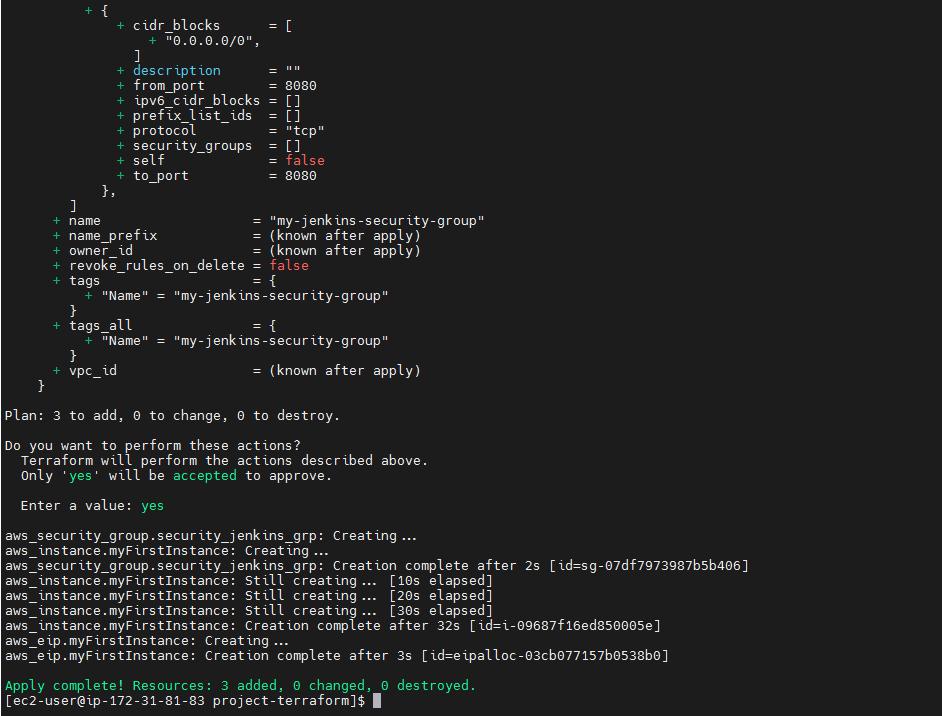


b. Now run terraform plan

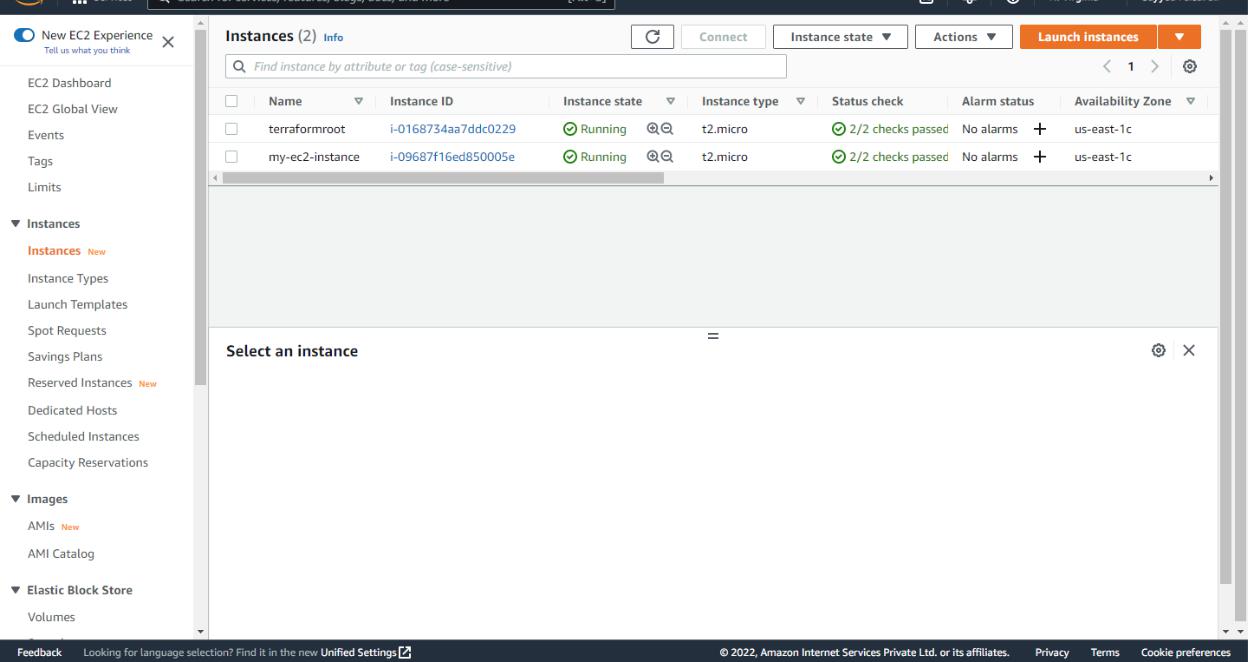


c. then run terraform apply to apply the change





1. Now go to your aws console and check whether the ec2 machine is created or not and well we successfully created the ec2 machine



e. Now terminate the machine we have created

**Q5 Explain following Terraform commands in one line**

* **terraform init:**
* **terraform validate:**
* **terraform plan:**
* **terraform apply:**
* **terraform destroy: Ans:**

a. **terraform init**: The terraform init command initializes a working directory containing Terraform configuration files.

b. **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.

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

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

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