**Terraform Assignment -1**

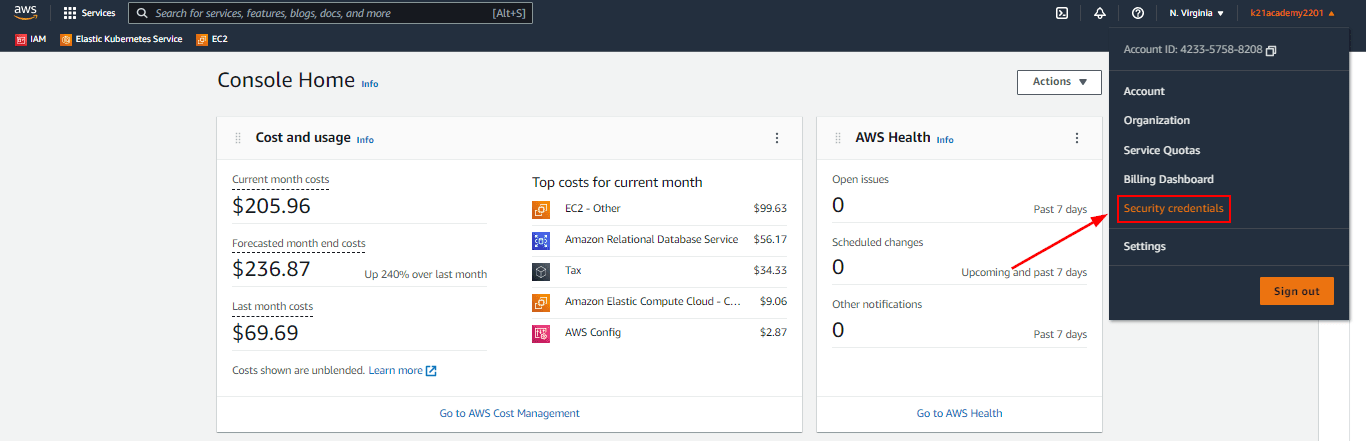
**Tasks To Be Performed:**

**1. Create an EC2 service in the default subnet in the Ohio region**

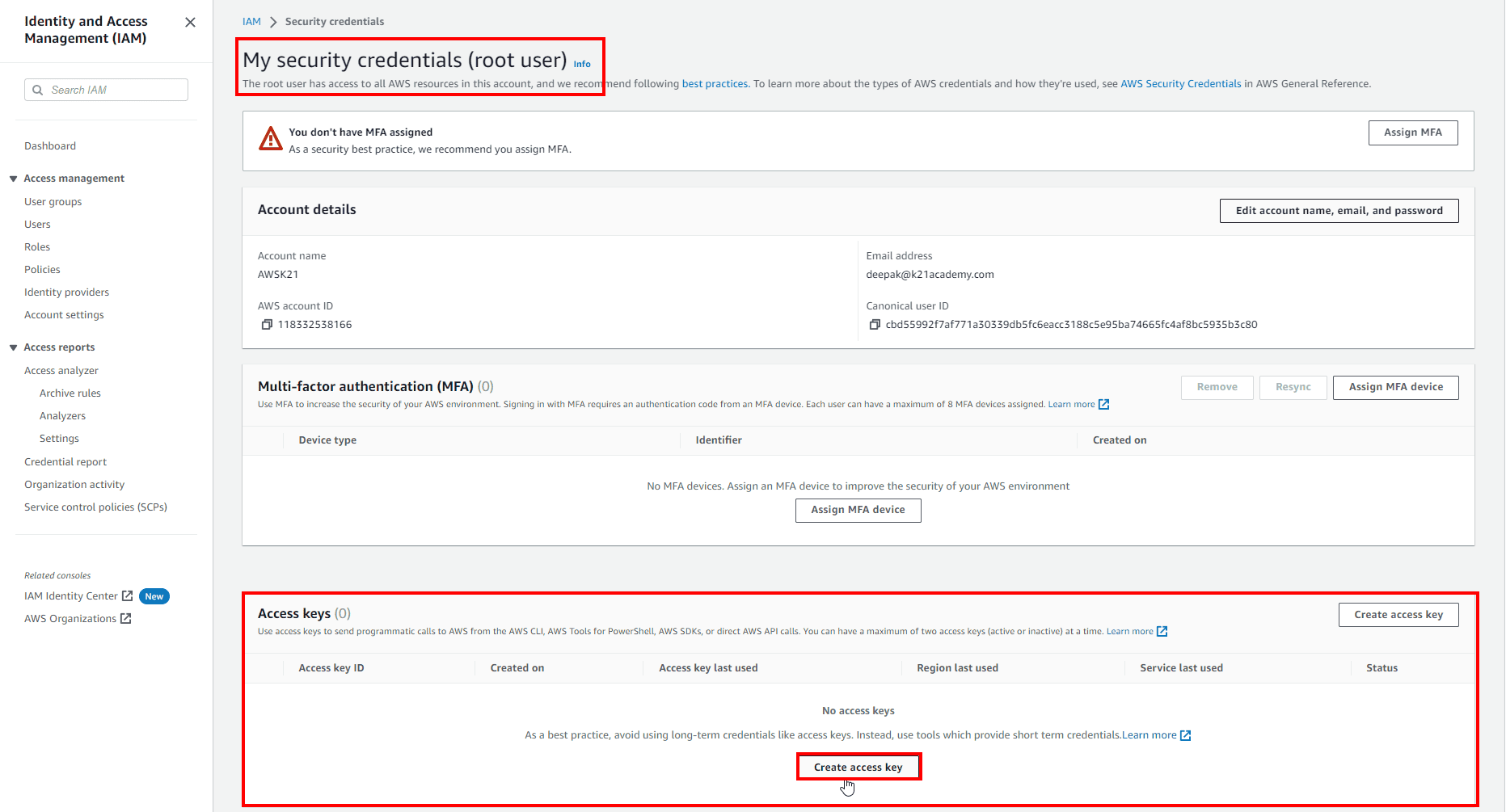
**Step 1** – First we need to create access id and security key for perform any terraform task

**Steps to create AWS Access key**

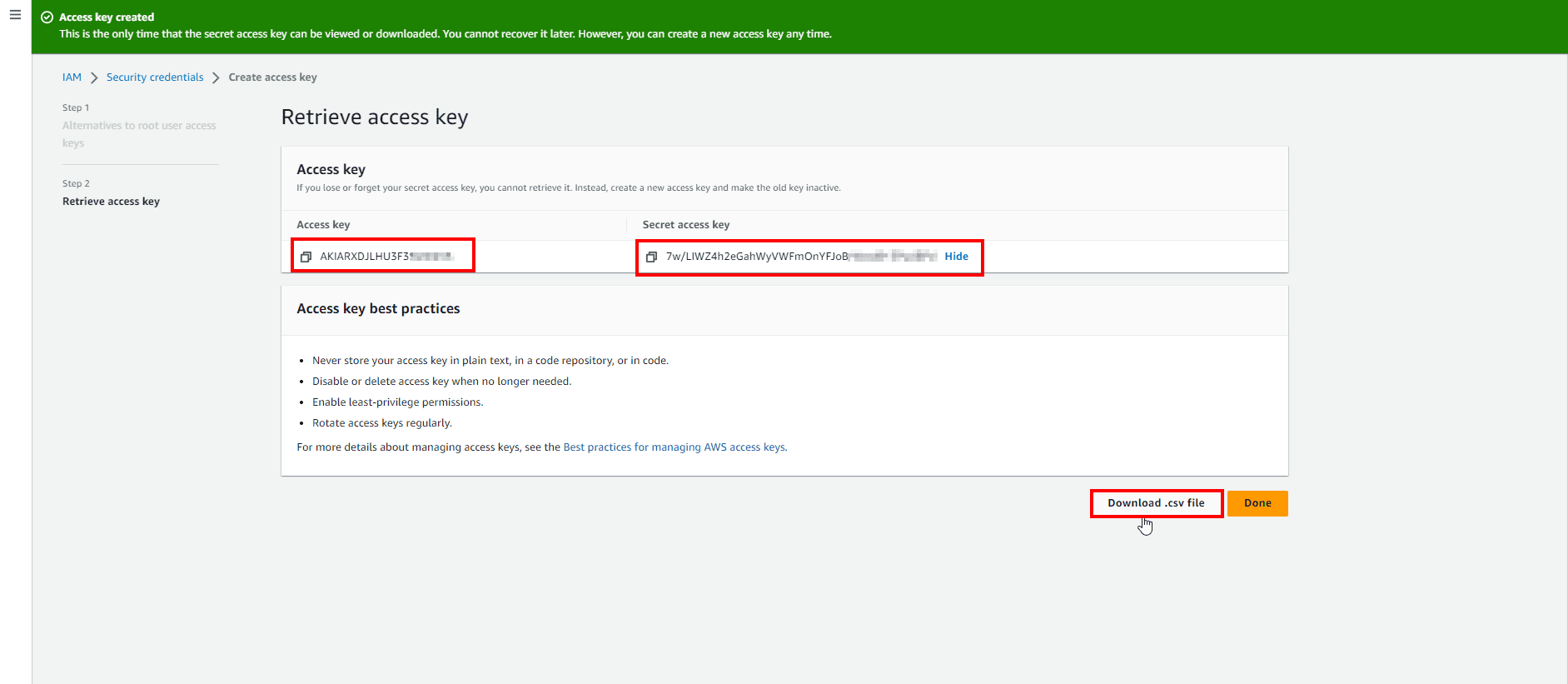
1. Go to the **AWS management console**, click on your Profile name, and then click on **My Security Credentials**.



1. Go to Access Keys and select Create New Access Key.



1. Click on **Show Access Key** and **save/download** the access key and secret access key.



**Note :-Best Practices For Managing AWS Access Keys**

Here are some of the best practices that you must follow while managing AWS access keys:

* **Never generate an account access key.:** One of the best ways to protect your account is to not create access keys for your AWS account root user unless required. Instead, the recommended best practice is to create one or more AWS Identity and Access Management (IAM) users and grant those IAM users the necessary permissions and use them for everyday interaction with AWS.
* **Use temporary security credentials instead of long-term access keys:** Long-term access keys that never expire are not often required. Instead, you can create IAM roles and generate temporary security credentials. Temporary security credentials consist of an access key ID and a secret access key, but they also include a security token that indicates when the credentials expire.
* **Manage IAM user access keys properly:** If you must create access keys for programmatic access to AWS, create them for IAM users, granting the users only the permissions they require.
* **Access keys should not be directly embedded in the code:**Put access keys in either the **AWS Credentials file** or **Environment Variables**.

**Step 2** After perform this task install terraform in your local machine here we have using windows version of terraform. Please follow the documentation

<https://spacelift.io/blog/how-to-install-terraform>

Now we need to our task and the task is Create an EC2 service in the default subnet in the Ohio region.

Before initialize the terraform we need the below configuration file here a sample code

## Providers (AWS/Azure/GCP)

# provider block

provider "aws" {

access\_key = "access key"

secret\_key = "secret key"

region = "ap-south-1"

}

# resource block

resource "aws\_instance" "MyWebServer" {

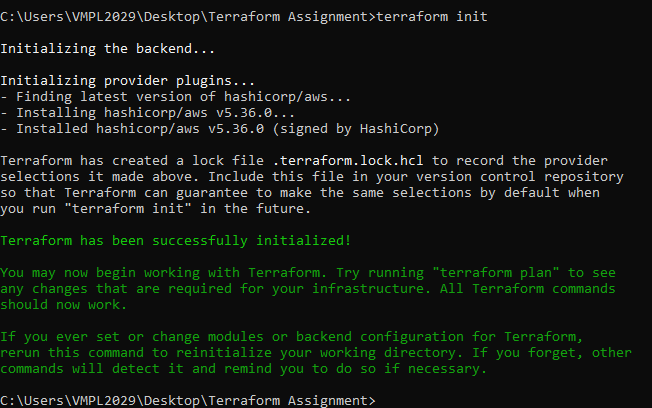
# resource arguments

ami = "ami-0f5ee92e2d63afc18" # replace with ami-id in your account

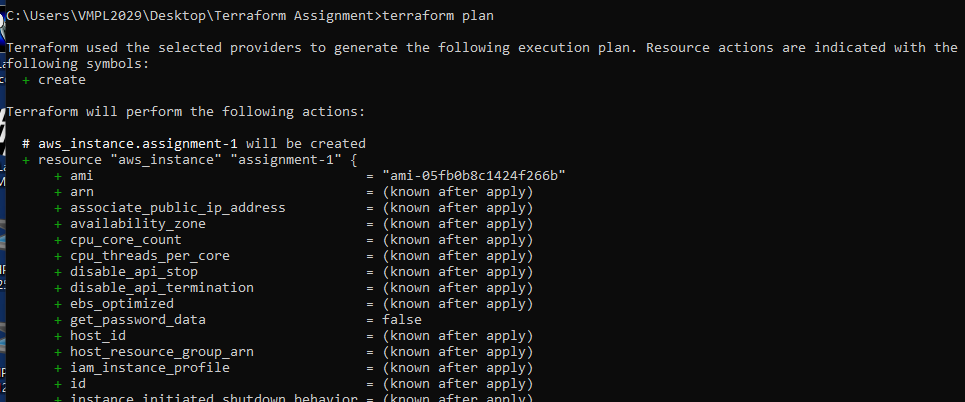
instance\_type = "t2.micro"

}

Where we have created configuration .tf file Initialize terraform



After terraform init just check the what plan of the terraform using $ terraform plan

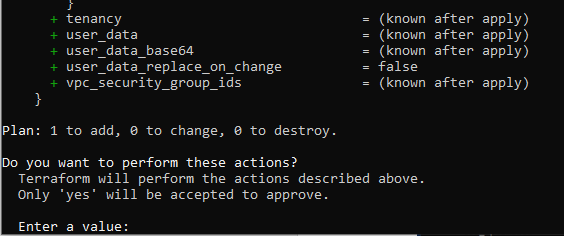


And here everything looking good so

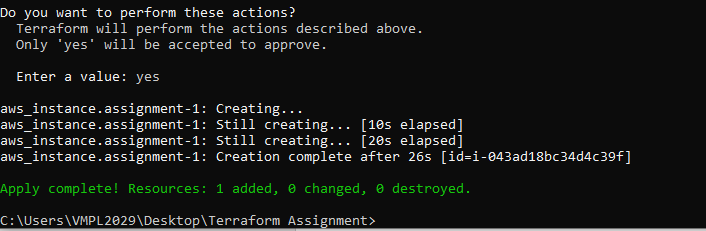
Now apply your code.

$ terraform apply

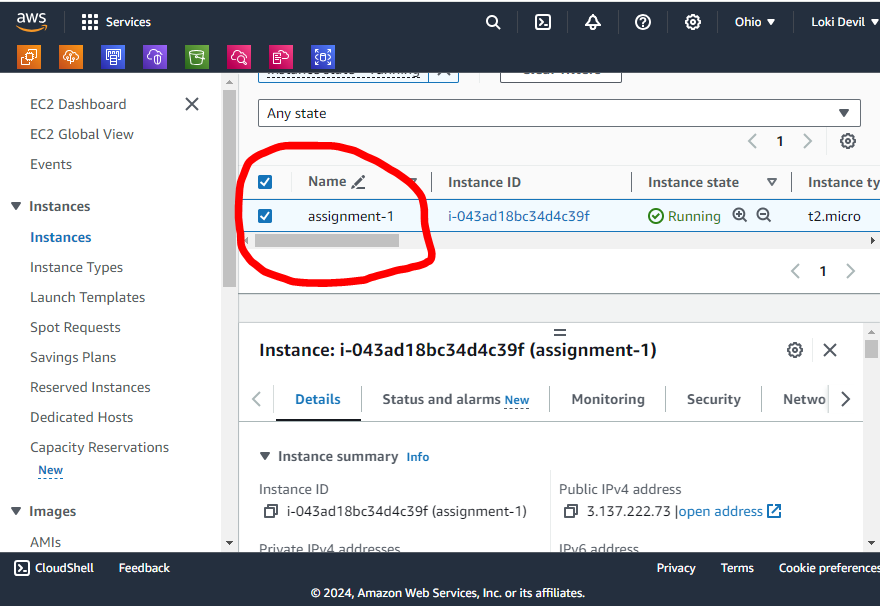
Here ask for the final permission just type yes and hit enter



There we go we have complete our first assignment



Lets verify in our AWS console



We have completed assignment 1 successfully.

**Terraform Assignment -2**

**Tasks To Be Performed:**

1. **Destroy the previous deployment**

To destroy the previous deployment in Terraform, you can use the **terraform destroy** command. Navigate to the directory containing your Terraform configuration files and execute the following command:

1. **Create a new EC2 instance with an Elastic IP**

**Step 1 :-** To create a new EC2 instance with an Elastic IP using Terraform, you can use the following configuration:

resource "aws\_instance" "assignment-2" {

    ami = "ami-05fb0b8c1424f266b"

    instance\_type = "t2.micro"

    key\_name = "terraform-kp"

    tags = {

    Name = "assignment-2"

    }

}

resource "aws\_eip" "eip" {

    vpc = true

}

resource "aws\_eip\_association" "eip\_assoc" {

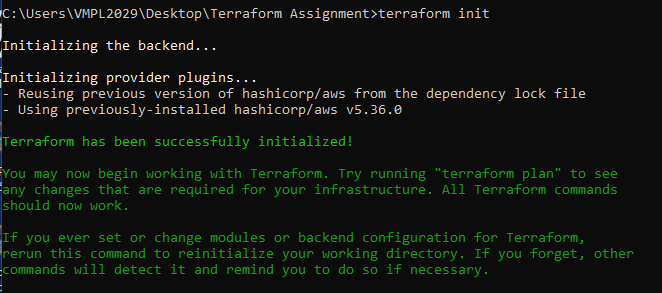
    instance\_id   = aws\_instance.assignment-2.id

    allocation\_id = aws\_eip.eip.id

}

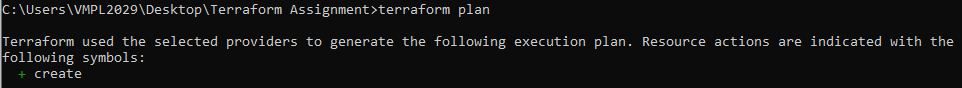
Make sure to replace **"ami-12345678"** with your desired AMI ID and modify other parameters as per your requirements. This configuration will create an EC2 instance and associate an Elastic IP address with it.

**Step 2:-** As the previous initialised the terraform using the command $ terraform init



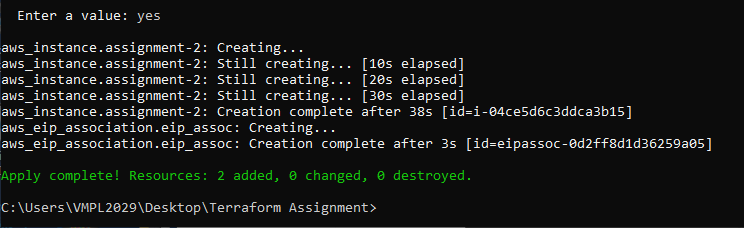
After applying this configuration, you can use the **public\_ip** output to get the Elastic IP address assigned to the EC2 instance.

**And then** $ terraform plan



Finally apply the configuration

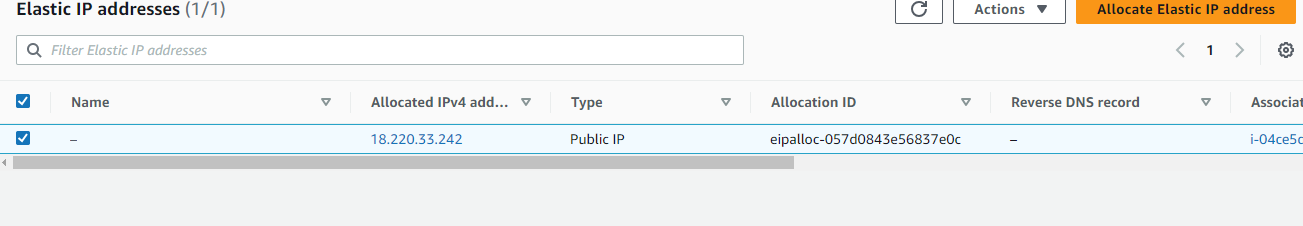
Using $ terraform apply



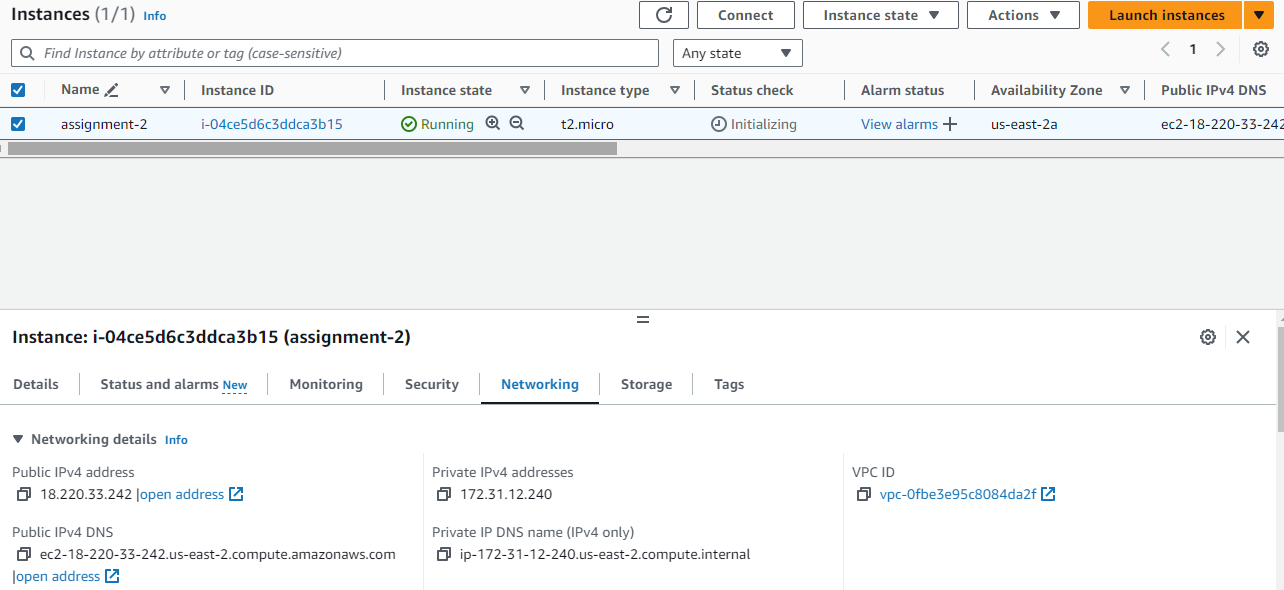
Finnaly we have complte our 2nd assgnment.

Lets verify

This is the EIP



And this is the EC2 associate with the EIP

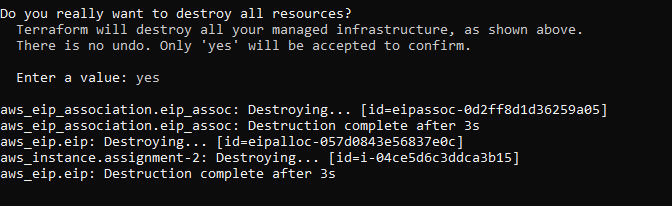


**Terraform Assignment -3**

**Tasks To Be Performed:**

1. Destroy the previous deployment

To destroy the previous deployment in Terraform, you can use the **terraform destroy** command. Navigate to the directory containing your Terraform configuration files and execute the following command:



1. Create 2 EC2 instances in Ohio and N.Virginia respectively
2. 3. Rename Ohio’s instance to ‘hello-ohio’ and Virginia’s instance to ‘hello-virginia

Here we taking N.Virginia EC2 name ansible and Ohio named as

**Step 1:-**Below is a Terraform configuration to create two EC2 instances, each in a different region (Ohio and N. Virginia):

**Step 2:-** To rename the instances to 'hello-ohio' and 'hello-virginia' respectively, you can use the **tags** parameter in Terraform. Below is the updated Terraform configuration:

In this configuration, I've added tags with the key Name for each instance resource. This will rename the instances accordingly. Ensure you replace **"ami-12345678"** with your desired AMI ID for each region and modify other parameters as needed.

provider "aws" {

    alias = "Ohio"

    region = "us-east-2"

    access\_key = "AKIA5XB3OVIEMFVRVNJ6"

    secret\_key = "eMFMn/3t8ipDKwWV0WEKLXOWa9Q8E2dBDz5dqHfr"

}

resource "aws\_instance" "assignment-3-1" {

    provider = aws.NV

    ami = "ami-0c7217cdde317cfec"

    instance\_type = "t2.micro"

    key\_name = "ansiblekp"

    tags = {

    Name = "hello-virginia"

    }

}

resource "aws\_instance" "assignment-3-2" {

    provider = aws.Ohio

    ami = "ami-05fb0b8c1424f266b"

    instance\_type = "t2.micro"

    key\_name = "terraform-kp"

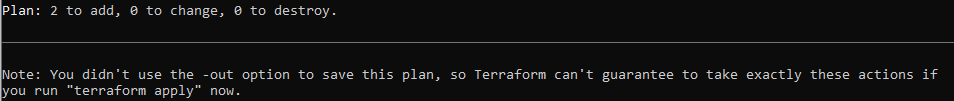
    tags = {

    Name = "hello-ohio"

    }

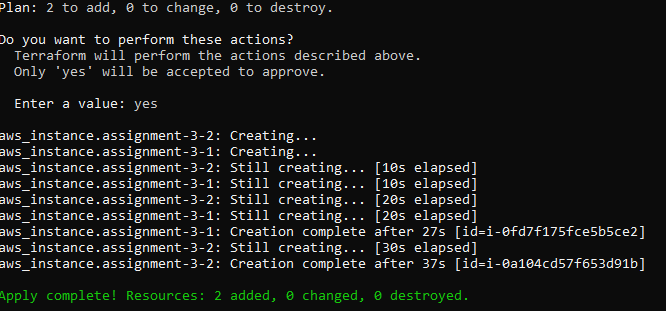
}

And first run $ terraform plan



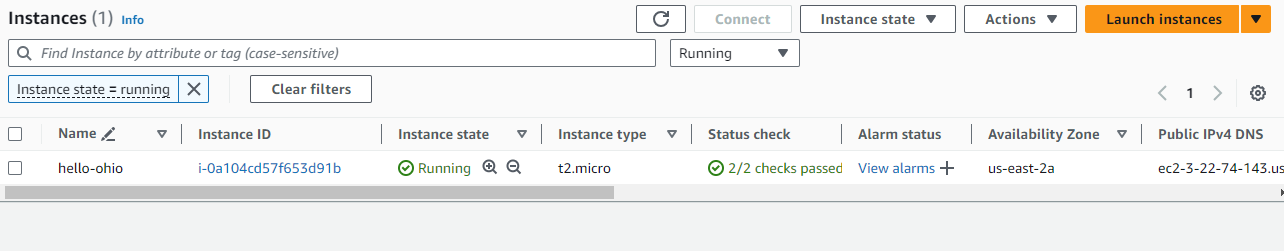
And here everything looks good

Now apply these configuration using the command $ terraform apply

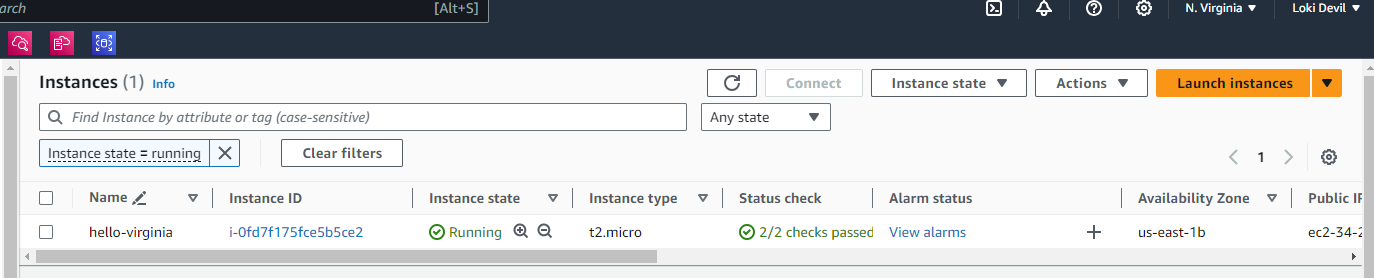


Lets verify the result that we have applied

Here in the ohio region 1 EC2



And in the N. Virginia Region



Now we have successfully completed our assignment 3.

**Terraform Assignment -4**

**Tasks To Be Performed:**

1. Destroy the previous deployments

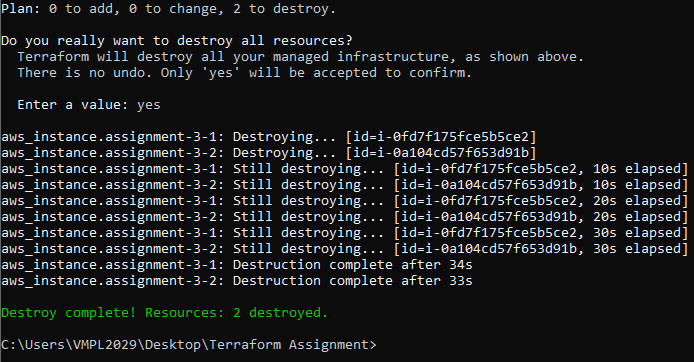
2. Create a VPC with the required components using Terraform

3. Deploy an EC2 instance inside the VPC

**Steps:-** To accomplish the tasks you've outlined, we'll break it down into three parts:

**Task 1: Destroy Previous Deployments**

To destroy previous deployments, you can run the **$ terraform destroy** command in the directory where your Terraform configuration files are located.



**Task 2: Create a VPC with Required Components**

Below is a Terraform configuration to create a VPC with required components:

resource "aws\_instance" "assignment-4" {

    ami = "ami-05fb0b8c1424f266b"

    instance\_type = "t2.micro"

    subnet\_id = aws\_subnet.assignment-4-subnet.id

    key\_name = "terraform-kp"

    tags = {

    Name = "assignment-4"

    }

}

resource "aws\_vpc" "assignment-4-vpc" {

    cidr\_block = "10.0.0.0/16"

    tags = {

    Name = "assignment-4-vpc"

    }

}

**Task 3: Deploy an EC2 Instance Inside the VPC**

Below is a Terraform configuration to deploy an EC2 instance inside the VPC:

 resource "aws\_subnet" "assignment-4-subnet" {

    vpc\_id = aws\_vpc.assignment-4-vpc.id

    cidr\_block = "10.0.1.0/24"

    availability\_zone = "us-east-2a"

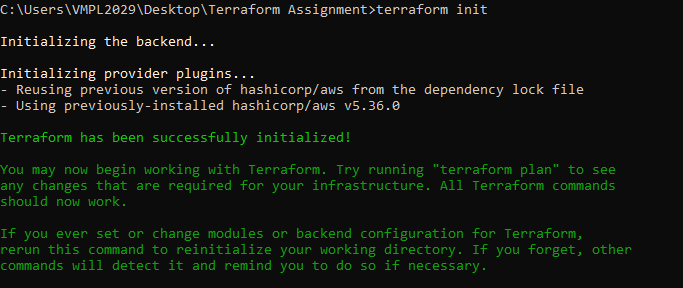
    tags = {

    Name = "assignment-4-subnet"

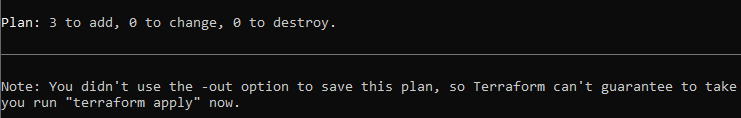
    }

}

After creating these Terraform configurations, you can initialize Terraform in your directory (**terraform init**), apply the changes (**terraform apply**), and then destroy them when needed (**terraform destroy**). Make sure to review and adapt the configurations to your specific requirements before applying them.

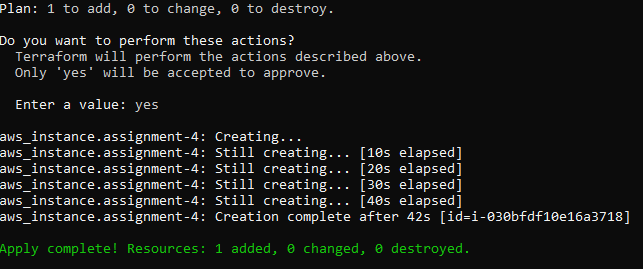


After that check the plan of this configuration

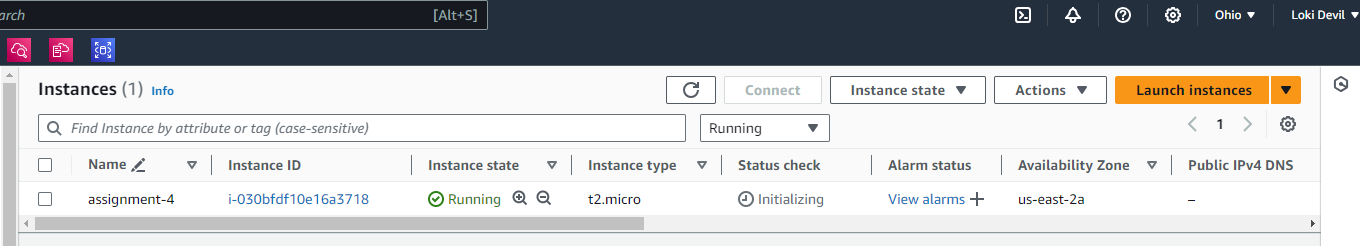


Looks everything good

Apply using the command $ terraform apply



Lets verify



Now we have performed assignment 4 successfully.

**Terraform Assignment – 5**

**You have been asked to:**

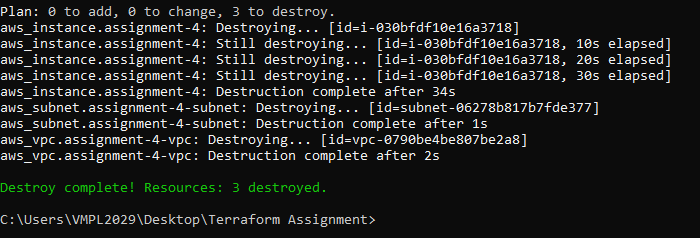
1. Destroy the previous deployments
2. Create a script to install apache2
3. Run this script on a newly created EC2 instance
4. Print the IP address of the instance in a file on the local, once deployed

**To full fill the tasks you've outlined, we'll break it down into steps**:

**Step 1: Destroy Previous Deployments**

Ensure you are in the directory containing your Terraform configuration files and execute the following command to destroy previous deployments:

$ terraform destroy -auto-approve



**Step 2: Create a Script to Install Apache2**

Create a bash script named install\_apache.sh with the following content:

#!/bin/bash

sudo apt update -y

sudo apt install apache2 -y

sudo su

echo "assignmnet 5 is done" > /var/www/html/index.html

**Step 3: Run the Script on a Newly Created EC2 Instance**

Modify your Terraform configuration to include user data that runs the script on the EC2 instance:

**Step 4: Print the IP Address of the Instance to a File Locally**

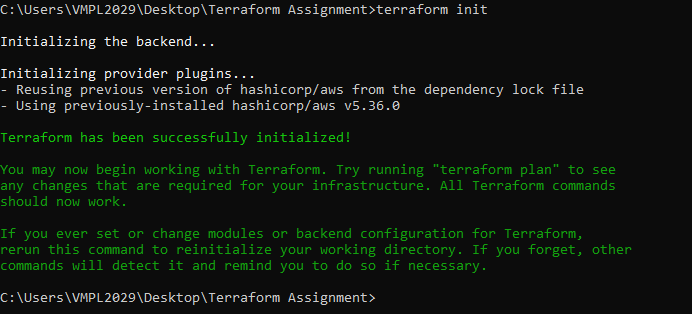
After applying the Terraform configuration, you can use Terraform output to print the IP address of the instance to a file locally. Add an output block to your Terraform configuration:

output "IPv4" {

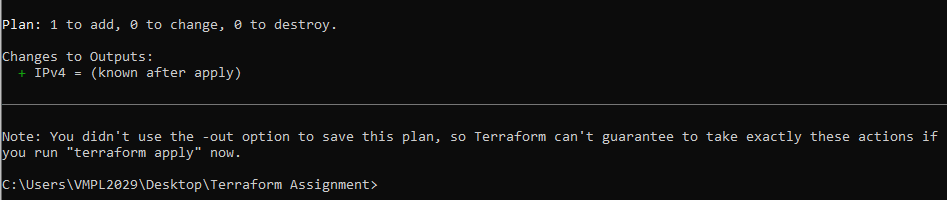
    value = aws\_instance.a5-instance.public\_ip

}

And initialize the configuration using $ terraform init command

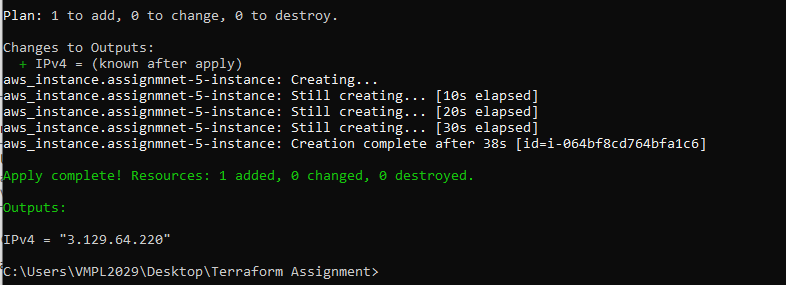


Check terraform plan

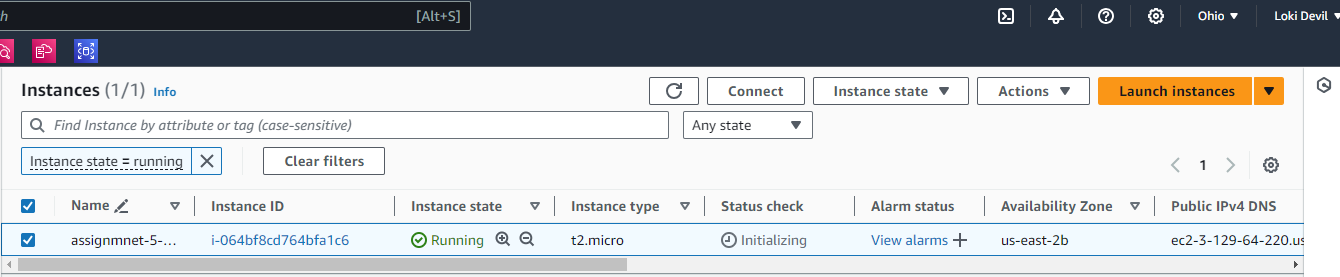


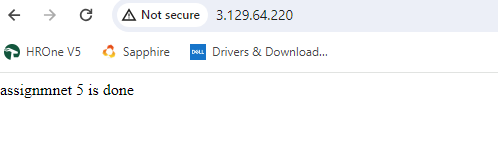
Everything looks good in the our configuration lets apply it.

Using command $ terraform apply -auto-approve



Lets verify it





Now we have successfully performed our assignment 5

**Summary:**

Destroy the previous deployments with terraform destroy -auto-approve.

Create the install\_apache.sh script to install Apache2.

Modify your Terraform configuration to include user data to run the script on the EC2 instance.

Add an output block to print the IP address of the instance locally.

Run terraform apply to create the instance.

Capture the IP address with terraform output instance\_ip > instance\_ip.txt.