Department of Information Technology

Academic Year: 2024-25
Class / Branch: TE IT
Subject: Advance DevOps Lab

Semester: V
Name: Ritesh Deshmukh
Student id: 22104058

Assignment No 01

Lab Outcomes (LO):

LO1	Apply the fundamentals of Cloud Computing and fully proficient Cloud based DevOps solution deployment options to meet the business requirements.
LO2	Build single and multiple container applications and manage application deployments with rollouts in Kubernetes.
LO3	Make use of best practices for managing infrastructure as code environments to build cloud infrastructure using Terraform.
LO4	Develop the security in the development process using SAST Techniques to remediate application vulnerabilities.
LO5	Experiment with Continuous Monitoring Tools to resolve system errors and to reduce negative impact on the business productivity.
LO6	Make use of AWS Lambda nano services to setup functions with the Serverless Framework.

Q. How to configure Amazon S3 buckets for security, enable versioning, and manage ob classes and how create an isolated network environment in AWS to deploy, and resources (BL3, LO1)

Ans.- 1. Create an S3 Bucket

1. **Log in to the AWS Management Console**
2. **Navigate to S3**
   - In the AWS Console, search for "S3" and click on the S3 service.
3. **Create a Bucket**
   - Click **Create bucket**
   - Give your bucket a **unique name** (e.g., `my-unique-bucket-lab`).

Choose a region (e.g., `us-east-1`).

2. Configure Security (Permissions & Policies)

A. Bucket Policies:

1. Go to the **Permissions** tab of your newly created bucket.
2. Click on **Bucket policy**
    - Add a policy to allow read access for everyone: Uncheck **Block all public access** if you want public access for testing (though this isn't recommended for production).
    - Click **Create bucket**

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-unique-bucket-lab/*"
    }
  ]
}
```

1. Replace `my-unique-bucket-lab` with your bucket name.
2. Click **Save**

B. IAM User Policies:

1. Navigate to **IAM** in the AWS Console.
2. Create or select an IAM user.
3. Attach an S3 policy:
    - Choose the **AmazonS3FullAccess** policy for full access.
    - Alternatively, use a custom policy for more specific access:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::my-unique-bucket-lab/*"
    }
  ]
}
```

C. Access Control Lists (ACLs):

1. **Go back to your bucket in the S3 console.**
2. **Click on the Permissions tab.**
3. **In the Access control list (ACL) section, configure permissions for specific users**

3. Enable Versioning

1. **In the S3 Console, navigate to the Properties tab of your bucket.**
2. **Scroll down to the Bucket Versioning section.**
3. **Click Enable versioning.**
4. **Now, when you upload a file with the same name, older versions of the file will**

4. Use S3 Storage Classes

A. Storage Classes Overview:

- **Standard: General-purpose, frequently accessed data.**
- **Intelligent-Tiering: Automatically**

To create an isolated network environment in AWS to deploy your resources safely steps-

Step 1: Create a VPC

1. **Navigate to VPC Dashboard:**
   - Log in to the AWS Management Console.
   - Go to the **VPC** service by typing "VPC" in the search bar.
2. **Create a new VPC**
   - Click on **Create VPC**.
   - Enter the following details:
     - **Name Tag:** `MyCustomVPC`.
     - **IPv4 CIDR Block:** `10.0.0.0/16` (You can use any valid CIDR block for your VPC).
     - Leave other settings as default.
   - Click **Create**.

Step 2: Create Subnets

You need to create public and private subnets to host your resources in different zones.

1. **Create a Public Subnet**
   - In the VPC Dashboard, click on **Subnets**.
   - Click **Create Subnet**.
   - Enter the following details:
     - **Name Tag:** `PublicSubnet`.
     - **VPC ID:** Select the VPC you created (`MyCustomVPC`).
     - **Availability Zone:** Choose an Availability Zone (e.g., `us-east-1a`).
     - **IPv4 CIDR Block:** `10.0.1.0/24`.
   - Click **Create**.
2. **Create a Private Subnet**
   - Repeat the above steps to create a private subnet with the following values:
     - **Name Tag:** `PrivateSubnet`.
     - **VPC ID:** `MyCustomVPC`.
     - **Availability Zone:** Use a different zone (e.g., `us-east-1b`).
     - **IPv4 CIDR Block:** `10.0.2.0/24`.
   - Click **Create**.

Step 3: Create an Internet Gateway (IGW)

An Internet Gateway allows communication between instances in the public subnet and the internet.

1. **Create an IGW**
   - Go to the **Internet Gateways** section.
   - Click **Create Internet Gateway**.
   - Name it `MyInternetGateway`.

- Click **Create Internet Gateway**

2. **Attach the IGW to the VPC**
   - Once created, select the newly created IGW.
   - Click on **Actions** and choose **Attach to VPC**.
   - Select `MyCustomVPC` from the dropdown and click **Attach**.

Step 4: Configure Route Tables

1. **Create a Route Table for the Public Subnet**
   - In the VPC Dashboard, click on **Route Tables**.
   - Click **Create Route Table**.
   - Enter the following details:
     - **Name Tag**: `PublicRouteTable`.
     - **VPC ID**: Select `MyCustomVPC`.
   - Click **Create**.
2. **Edit Routes to Allow Internet Traffic**
   - Select the newly created route table (`PublicRouteTable`).
   - Click on the **Routes** tab, then click **Edit Routes**.
   - Add a route with the following details:
     - **Destination**: `0.0.0.0/0` (This allows all traffic).
     - **Target**: Select `MyInternetGateway`.
   - Save the route.
3. **Associate the Public Subnet with the Public Route Table**
   - In the **Subnet Associations** tab, click **Edit Subnet Associations**.
   - Select the checkbox next to `PublicSubnet` and click **Save**.
4. **Create and Configure a Private Route Table**
   - Repeat the same process for a private route table (name it `PrivateRouteTable`), but do **not** add any internet routes, since private subnets are not internet-facing.
   - Associate this route table with the `PrivateSubnet`.

Step 5: Create Security Groups

1. **Create a Security Group for the Public Subnet**
   - Navigate to **Security Groups**.
   - Click **Create Security Group**.
   - Enter the following details:
     - **Name**: `PublicSecurityGroup`.
     - **VPC**: Select `MyCustomVPC`.
   - Click **Create**.
2. **Configure Inbound Rules for the Public Security Group**
   - Select the `PublicSecurityGroup`.
   - In the **Inbound Rules** tab, click **Edit Inbound Rules**.
   - Add the following rule:
     - **Type**: HTTP
     - **Protocol**: TCP
     - **Port Range**: 80
     - **Source**: Anywhere (0.0.0.0/0).
   - Also add a rule for **SSH** (Port 22) from anywhere so you can connect to your instance.

- Click **Save**.

3. **Create and Configure Private Security Group**
   - Repeat the process to create a `PrivateSecurityGroup`, but configure rules to allow access only from instances in the public subnet or specific IP addresses, as needed.

Step 6: Launch EC2 Instances

1. **Launch a Public Instance**
   - Go to the **EC2 Dashboard** and click **Launch Instance**.
   - Choose an AMI (e.g., Amazon Linux 2).
   - Select an instance type (e.g., t2.micro).
   - Configure the instance with the following details:
     - **Network**: Select `MyCustomVPC`.
     - **Subnet**: Select `PublicSubnet`.
     - **Auto-assign Public IP**: Enable (for public internet access).
     - **Security Group**: Select `PublicSecurityGroup`.
   - Launch the instance and connect using SSH.
2. **Launch a Private Instance**
   - Repeat the process for a private instance, but place it in the `PrivateSubnet` and assign the `PrivateSecurityGroup`. This instance won't have direct internet access.

Q. Apply the concept of cloud computing to create Kubernetes Cluster using Terra deploy it with Terraform.

(**BL3, LO1, LO2, LO3**)

Answer:

Why deploy with Terraform?

While you could use kubectl or similar CLI-based tools to manage your Kubernetes resources, using Terraform has the following benefits:

- **Unified Workflow**: If you are already provisioning Kubernetes clusters with Terraform, use the same configuration language to deploy your applications into your cluster.

- **Full Lifecycle Management** - Terraform doesn't only create resources, it updates, and deletes tracked resources without requiring you to inspect the API to identify those resources.

- **Graph of Relationships** - Terraform understands dependency relationships between resources. For example, if a Persistent Volume Claim claims space from a particular Persistent Volume, Terraform won't attempt to create the claim if it fails to create the volume.

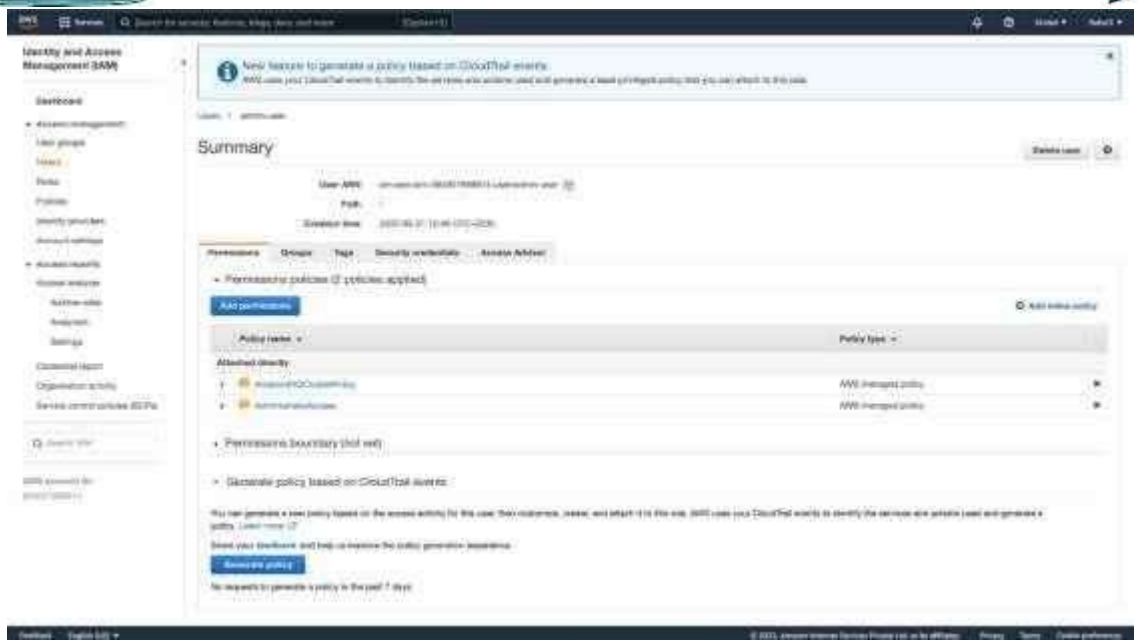
Prerequisites for Terraform Kubernetes Deployment:

Before we proceed and provision EKS Cluster using Terraform, there are a few commands or tools you need to have in mind and on hand. First off, you must have an AWS Account, and Terraform must be installed on your host machine, seeing as we are going to create EKS Cluster using Terraform CLI on the AWS cloud.

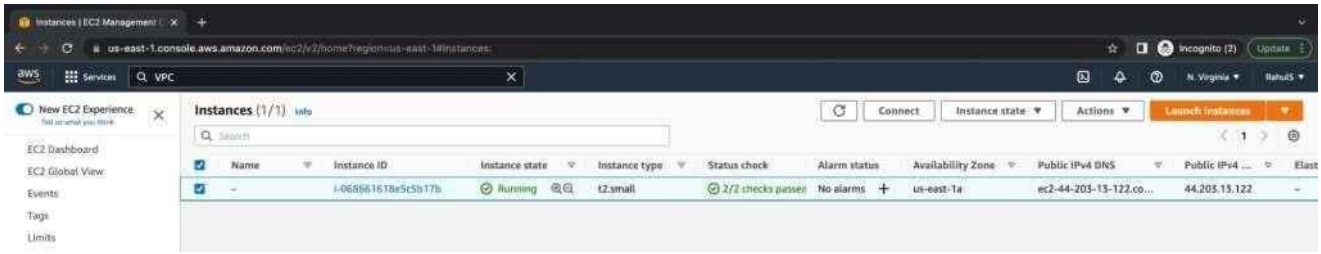
Now, let's take a look at the prerequisites for this setup and help you install them.

1. **AWS Account:** If you don't have an AWS account, you can register for a Free Tier Account and use it for test purposes. Click here to learn more about the Free Tier AWS Account or create one if you don't already have one.

2. **IAM Admin User:** You must have an IAM user with AmazonEKSClusterPolicy and AdministratorAccess permissions as well as its secret and access keys. We will be using the IAM user credentials to provision EKS Cluster using Terraform. Click here to learn more about the AWS IAM Service. The keys that you create for this user will be used to connect to the AWS account from the CLI (Command Line Interface). When working on production clusters, only provide the required access and avoid providing admin privileges.



3. **EC2 Instance:** We will be using Ubuntu 18.04 EC2 Instance as a host machine to execute our Terraform code. You may use another machine; however, you will need to verify which commands are compatible with your host machine in order to install the required packages. Click here to learn more about the AWS EC2 service. The first step is installing the required packages on your machine. You can also use your personal computer to install the required tools. This step is optional.



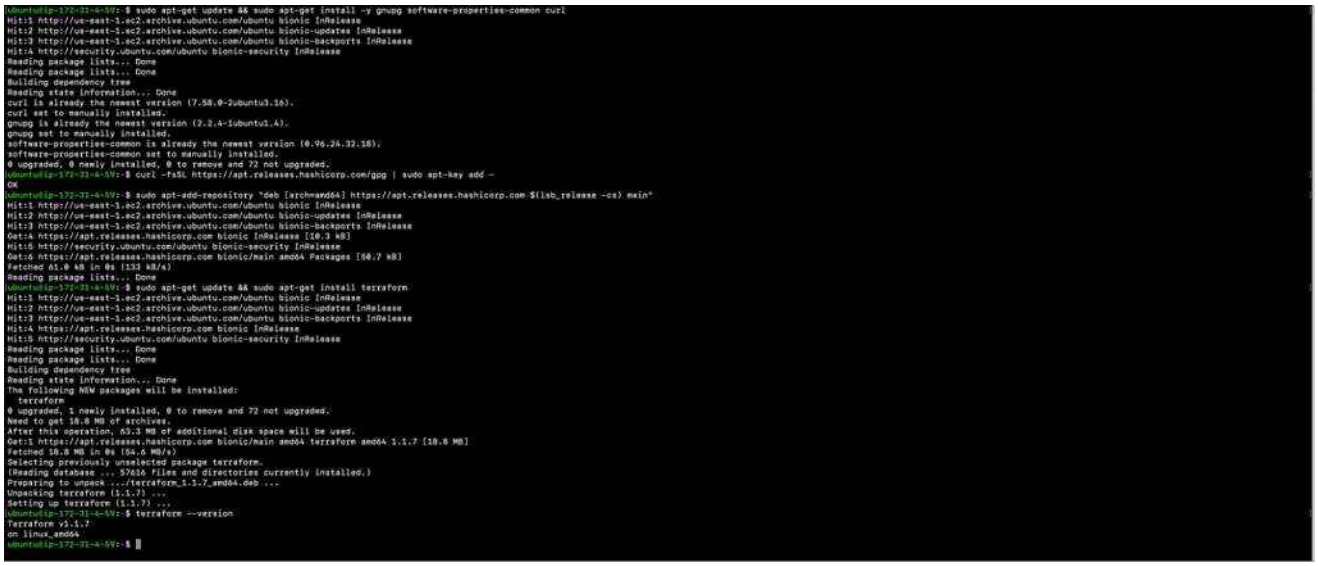
Access to the Host Machine: Connect to the EC2 Instance and Install the Unzip package.

a. ssh -i "<key-name.pem>" ubuntu@<public-ip-of-the–ec2-instance>
If you are using your personal computer, you may not need to connect to the EC2 instance, however, in this case, the installation command will differ.
b. sudo apt-get update -y
c. sudo apt-get install unzip -y

5. **Terraform:** To create EKS Cluster using Terraform, you need to have Terraform on your Host machine. Use the following commands to install Terraform on an Ubuntu 18.04 EC2 machine.

a. sudo apt-get update && sudo apt-get install -y gnupg software-properties-common curl
b. curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add –
c. sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"

d. sudo apt-get update && sudo apt-get install terraform
e. terraform –version



AWS CLI: There is not much to do with aws-cli, however, we need to use it to check the details about the IAM user whose credentials will be used from the terminal. To install it, use the commands below.

a. curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
b. unzip awscliv2.zip
c. sudo ./aws/install

Kubectl: We will be using the kubectl command against the Kubernetes Cluster to view the resources in the EKS Cluster that we want to create. Install kubectl on Ubuntu 18.04 EC2 machine using the commands below.

a. curl -LO "https://dl.k8s.io/release/$(curl -L

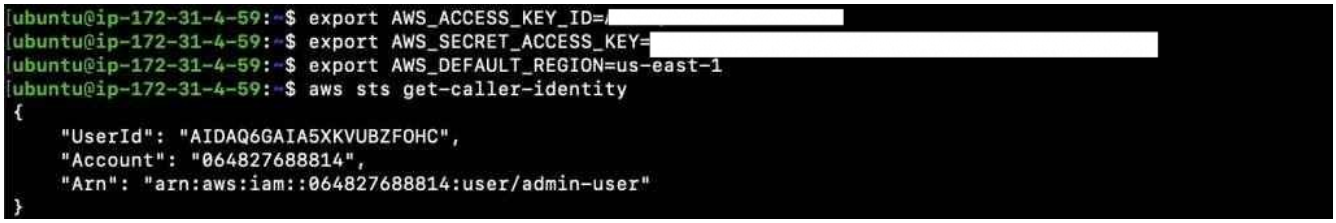
-s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
b. curl -LO "https://dl.k8s.io/$(curl -L

-s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256″
c. echo "$(cat kubectl.sha256) kubectl" | sha256sum –check
d. sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
e. kubectl version –client

Export your AWS Access and Secret Keys for the current session. If the session expires, you will need to export the keys again on the terminal. There are other ways to use your keys that allow aws-cli to interact with AWS.

a. export AWS_ACCESS_KEY_ID=**<YOUR_AWS_ACCESS_KEY_ID>**
b. export AWS_SECRET_ACCESS_KEY=**<YOUR_AWS_SECRET_ACCESS_KEY>**
c. export AWS_DEFAULT_REGION=**<YOUR_AWS_DEFAULT_REGION>**
Here, replace <YOUR_AWS_ACCESS_KEY_ID> with your access key, <YOUR_AWS_SECRET_ACCESS_KEY> with your secret key and <YOUR_AWS_DEFAULT_REGION> with the default region for your aws-cli.

10. Check the details of the IAM user whose credentials are being used. Basically, this will display the details of the user whose keys you used to configure the CLI in the above step.
a. aws sts get-caller-identity



Architecture:

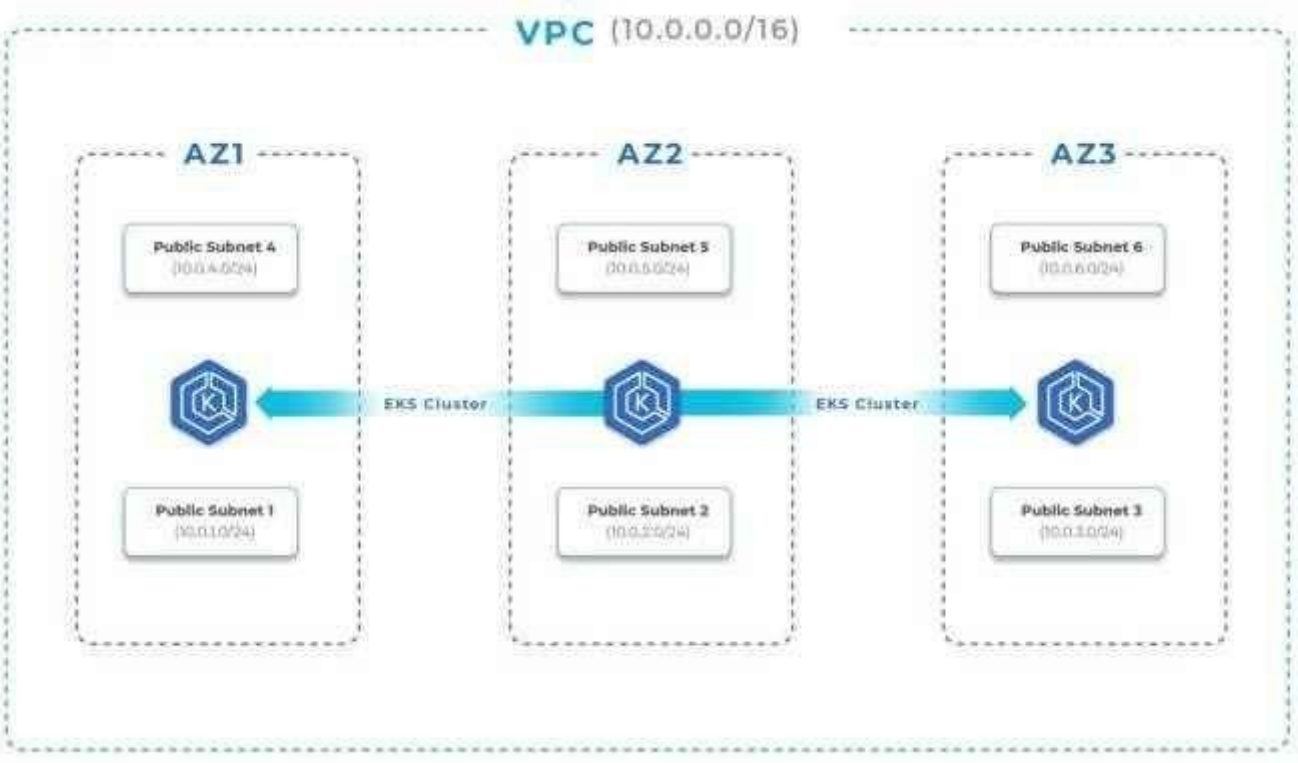
The architecture should appear as follows.

A VPC will be created with three Public Subnets and three Private Subnets. Traffic from Private Subnets will route through the NAT Gateway and traffic from Public Subnets will route through the Internet Gateway.

Kubernetes Cluster Nodes will be created as part of Auto-Scaling groups and will reside in Private Subnets. Public Subnets can be used to create Bastion Servers that can be used to connect to Private Nodes.

Three Public Subnets and three Private Subnets will be created in three different Availability Zones.

You can change the VPC CIDR in the Terraform configuration files if you wish. If you are just getting started, we recommend following the blog without making any unfamiliar changes to the configuration in order to avoid human errors.



How to Create Kubernetes Cluster Using Terraform?

1. Clone the Github Repository in your home directory.
    1. cd ~/
    2. pwd
    3. git clone https://github.com/shivalkarrahul/DevOps.git

```
ubuntu@ip-172-31-4-59:~$ cd ~/
ubuntu@ip-172-31-4-59:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-4-59:~$ git clone https://github.com/shivalkarrahul/DevOps.git
Cloning into 'DevOps'...
remote: Enumerating objects: 559, done.
remote: Counting objects: 100% (431/431), done.
remote: Compressing objects: 100% (341/341), done.
remote: Total 559 (delta 191), reused 301 (delta 74), pack-reused 128
Receiving objects: 100% (559/559), 39.01 MiB | 29.57 MiB/s, done.
Resolving deltas: 100% (215/215), done.
ubuntu@ip-172-31-4-59:~$ ls -l
total 91260
drwxrwxr-x 8 ubuntu ubuntu     4096 Apr  9 14:08 DevOps
```

2. After you clone the repo, change your “Present Working Directory” to ~/DevOps/aws/terraform/terraform-kubernetes-deployment/
    1. cd ~/DevOps/aws/terraform/terraform-kubernetes-deployment/

```
ubuntu@ip-172-31-4-59:~$ cd ~/DevOps/aws/terraform/terraform-kubernetes-deployment/
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment$ pwd
/home/ubuntu/DevOps/aws/terraform/terraform-kubernetes-deployment
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment$ ls -l
total 8
-rw-rw-r-- 1 ubuntu ubuntu    0 Apr  9 14:08 README.md
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr  9 14:08 eks-cluster
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr  9 14:08 nodejs-application
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment$
```

The Terraform files for creating an EKS Cluster must be in one folder and the Terraform files for deploying a sample NodeJs application must be in another.

1. eks-cluster
   This folder contains all of the .tf files required for creating the EKS Cluster.
2. nodejs-application
   This folder contains a .tf file required for deploying the sample NodeJs application.

Now, let's proceed with the creation of an **EKS Cluster using Terraform.**

1. Verify your "Present Working Directory", it should be ~/DevOps/aws/terraform/terraform-kubernetes-deployment/ as you have already cloned my Github Repository in the previous step.
   1. cd ~/DevOps/aws/terraform/terraform-kubernetes-deployment/

2. Next, change your "Present Working Directory" to ~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster
   1. cd ~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster

3. You should now have all the required files.

```
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster$ pwd
/home/ubuntu/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster$ ls -l
total 1152
-rw-rw-r-- 1 ubuntu ubuntu    1062 Apr  9 14:08 eks-cluster.tf
-rw-rw-r-- 1 ubuntu ubuntu 1151316 Apr  9 14:08 graph.svg
-rw-rw-r-- 1 ubuntu ubuntu    1210 Apr  9 14:08 kubernetes.tf
-rw-rw-r-- 1 ubuntu ubuntu     862 Apr  9 14:08 outputs.tf
-rw-rw-r-- 1 ubuntu ubuntu     816 Apr  9 14:08 security-groups.tf
-rw-rw-r-- 1 ubuntu ubuntu     481 Apr  9 14:08 versions.tf
-rw-rw-r-- 1 ubuntu ubuntu    1148 Apr  9 14:08 vpc.tf
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster$
```

4. If you haven't cloned the repo then you are free to create the required .tf files in a new folder.

5. Create your eks-cluster.tf file with the content below. In this case, we are using the module from terraform-aws-modules/eks/aws to create our EKS Cluster.

```
module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  version         = "17.24.0"
  cluster_name    = local.cluster_name
  cluster_version = "1.20"
  subnets         = module.vpc.private_subnets

  vpc_id = module.vpc.vpc_id

  workers_group_defaults = {
    root_volume_type = "gp2"
  }

  worker_groups = [
```

```
    {
      name                          = “worker-group-1”
      instance_type                 = “t2.small”
      additional_userdata           = “echo nothing”
      additional_security_group_ids = [aws_security_group.worker_group_mgmt_one.id]
      asg_desired_capacity          = 2
    },
    {
      name                          = “worker-group-2”
      instance_type                 = “t2.medium”
      additional_userdata           = “echo nothing”
      additional_security_group_ids = [aws_security_group.worker_group_mgmt_two.id]
      asg_desired_capacity          = 1
    },
  ]
}

data “aws_eks_cluster” “cluster” {
  name = module.eks.cluster_id
}

data “aws_eks_cluster_auth” “cluster” {
  name = module.eks.cluster_id
}
```

6.Create a kubernetes.tf file with the following content. Here, we are using Terraform Kubernetes Provider in order to create Kubernetes objects such as a Namespace, Deployment, and Service using Terraform. We are creating these resources for testing purposes only. In the following steps, we will also be deploying a sample application using Terraform.

```
provider “kubernetes” {
  host                   = data.aws_eks_cluster.cluster.endpoint
  token                  = data.aws_eks_cluster_auth.cluster.token
  cluster_ca_certificate = base64decode(data.aws_eks_cluster.cluster.certificate_authority.0.data)
}


resource “kubernetes_namespace” “test” {
  metadata {
    name = “nginx”
  }
}
resource “kubernetes_deployment” “test” {
  metadata {
    name      = “nginx”
    namespace = kubernetes_namespace.test.metadata.0.name
  }
```

```
  spec {
    replicas = 2
    selector {
      match_labels = {
        app = “MyTestApp”
      }
    }
    template {
      metadata {
        labels = {
          app = “MyTestApp”
        }
      }
      spec {
        container {
          image = “nginx”
          name  = “nginx-container”
          port {
            container_port = 80
          }
        }
      }
    }
  }
}
resource “kubernetes_service” “test” {
  metadata {
    name      = “nginx”
    namespace = kubernetes_namespace.test.metadata.0.name
  }
  spec {
    selector = {
      app = kubernetes_deployment.test.spec.0.template.0.metadata.0.labels.app
    }
    type = “LoadBalancer”
    port {
      port        = 80
      target_port = 80
    }
  }
}
```

7. Create an outputs.tf file with the following content. This is done to export the structured data related to our resources. It will provide information about our EKS infrastructure to other Terraform setups. Output values are equivalent to return values in other programming languages.

```
output “cluster_id” {
  description = “EKS cluster ID.”
  value       = module.eks.cluster_id
}

output “cluster_endpoint” {
  description = “Endpoint for EKS control plane.”
  value       = module.eks.cluster_endpoint
}

output “cluster_security_group_id” {
  description = “Security group ids attached to the cluster control plane.”
  value       = module.eks.cluster_security_group_id
}

output “kubectl_config” {
  description = “kubectl config as generated by the module.”
  value       = module.eks.kubeconfig
}

output “config_map_aws_auth” {
  description = “A kubernetes configuration to authenticate to this EKS cluster.”
  value       = module.eks.config_map_aws_auth
}

output “region” {
  description = “AWS region”
  value       = var.region
}

output “cluster_name” {
  description = “Kubernetes Cluster Name”
  value       = local.cluster_name
}
```

8. Create a security-groups.tf file with the following content. In this case, we are defining the security groups which will be used by our worker nodes.

```
resource “aws_security_group” “worker_group_mgmt_one” {
  name_prefix = “worker_group_mgmt_one”
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = “tcp”

    cidr_blocks = [
```

```
    "10.0.0.0/8",
  ]
 }
}

resource "aws_security_group" "worker_group_mgmt_two" {
 name_prefix = "worker_group_mgmt_two"
 vpc_id      = module.vpc.vpc_id

 ingress {
  from_port = 22
  to_port  = 22
  protocol  = "tcp"

  cidr_blocks = [
   "192.168.0.0/16",
  ]
 }
}

resource "aws_security_group" "all_worker_mgmt" {
 name_prefix = "all_worker_management"
 vpc_id      = module.vpc.vpc_id

 ingress {
  from_port = 22
  to_port  = 22
  protocol  = "tcp"

  cidr_blocks = [
   "10.0.0.0/8",
   "172.16.0.0/12",
   "192.168.0.0/16",
  ]
 }
}
```

9. Create a versions.tf file with the following content. Here, we are defining version constraints along with a provider as AWS.

```
terraform {
 required_providers {
  aws = {
   source  = "hashicorp/aws"
   version = ">= 3.20.0"
  }

  random = {
```

```
    source  = “hashicorp/random”
    version = “3.1.0”
  }

  local = {
    source  = “hashicorp/local”
    version = “2.1.0”
  }

  null = {
    source  = “hashicorp/null”
    version = “3.1.0”
  }

  kubernetes = {
    source  = “hashicorp/kubernetes”
    version = “>= 2.0.1”
  }
 }

 required_version = “>= 0.14”
}
```

10.Create a vpc.tf file with the following content. For the purposes of our example, we have defined a 10.0.0.0/16 CIDR for a new VPC that will be created and used by the EKS Cluster. We will also be creating Public and Private Subnets. To create a VPC, we will be using the terraform-aws-modules/vpc/aws module. All of our resources will be created in the us-east-1 region. If you want to use any other region for creating the EKS Cluster, all you need to do is change the value assigned to the “region” variable.

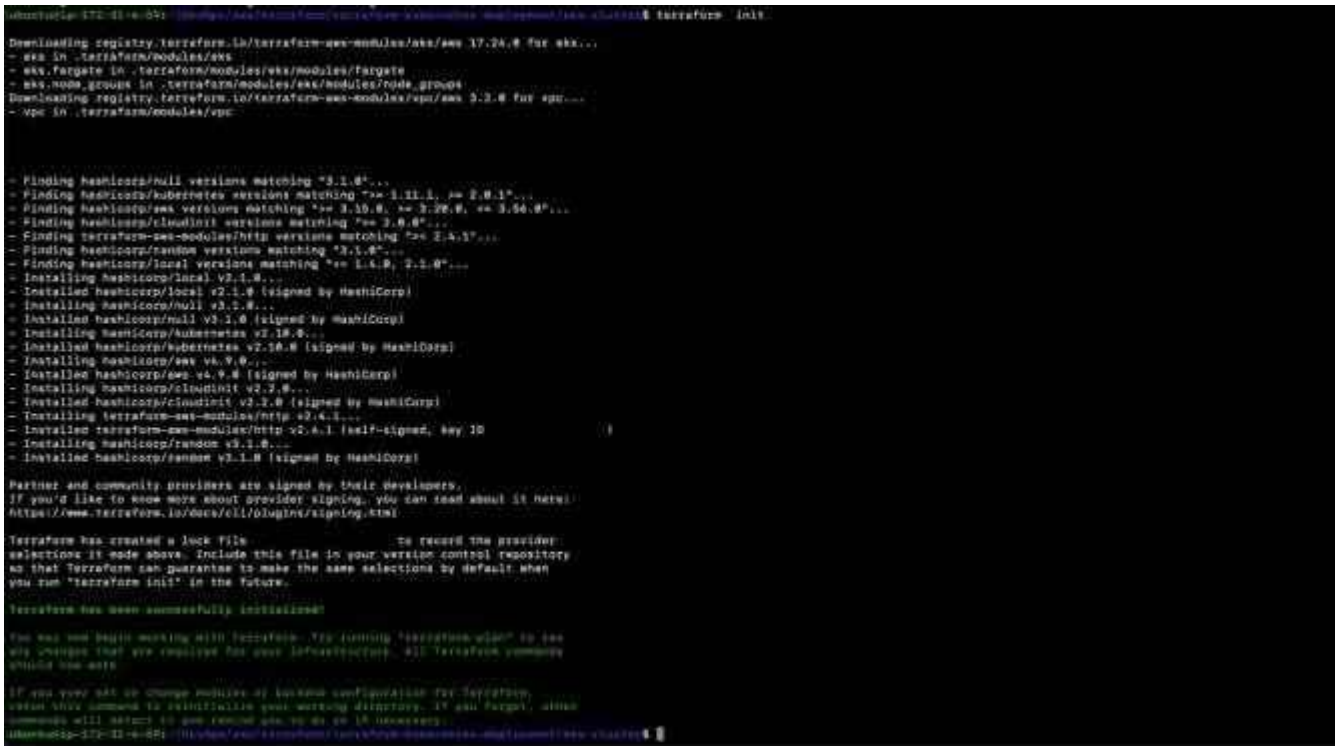
```
variable “region” {
 default     = “us-east-1”
 description = “AWS region”
}

provider “aws” {
 region = var.region
}

data “aws_availability_zones” “available” {}

locals {
 cluster_name = “test-eks-cluster-${random_string.suffix.result}”
}

resource “random_string” “suffix” {
 length  = 8
 special = false
```

```
}

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "3.2.0"

  name                 = "test-vpc"
  cidr                 = "10.0.0.0/16"
  azs                  = data.aws_availability_zones.available.names
  private_subnets      = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets       = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true

  tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }

  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                      = "1"
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = "1"
  }
}
```

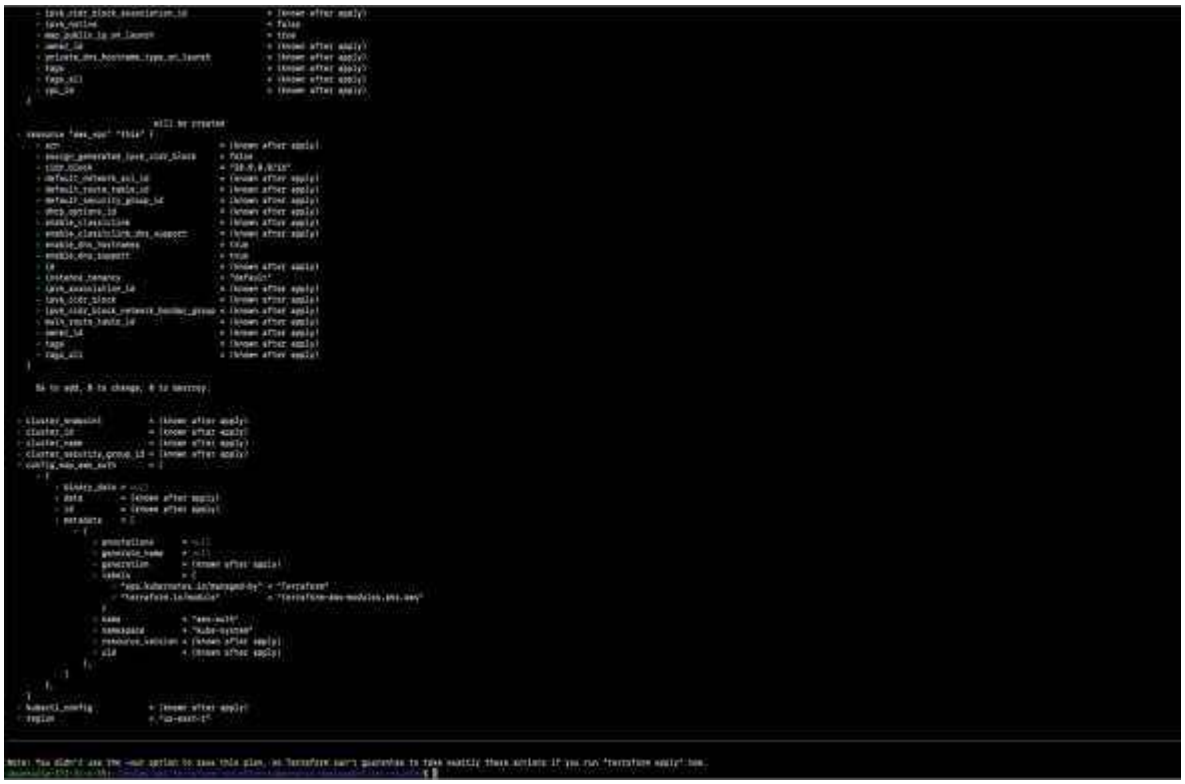
11. At this point, you should have six files in your current directory (as can be seen below). All of these files should be in one directory. In the next step, you will learn to deploy a sample NodeJs application and to do so, you will need to create a new folder. That said, if you have cloned the repo herein, you won't need to worry.
    1. eks-cluster.tf
    2. kubernetes.tf
    3. outputs.tf
    4. security-groups.tf
    5. versions.tf
    6. vpc.tf

```
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster$ ls -l
total 24
-rw-rw-r-- 1 ubuntu ubuntu 1062 Mar 31 09:37 eks-cluster.tf
-rw-rw-r-- 1 ubuntu ubuntu 1210 Mar 31 10:40 kubernetes.tf
-rw-rw-r-- 1 ubuntu ubuntu  862 Mar 31 08:15 outputs.tf
-rw-rw-r-- 1 ubuntu ubuntu  816 Mar 31 08:15 security-groups.tf
-rw-rw-r-- 1 ubuntu ubuntu  481 Mar 31 08:15 versions.tf
-rw-rw-r-- 1 ubuntu ubuntu 1148 Mar 31 09:37 vpc.tf
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster$
```

12. To create an EKS Cluster, execute the following command to initialize the current working directory containing the Terraform configuration files (.tf files).
    1. terraform init

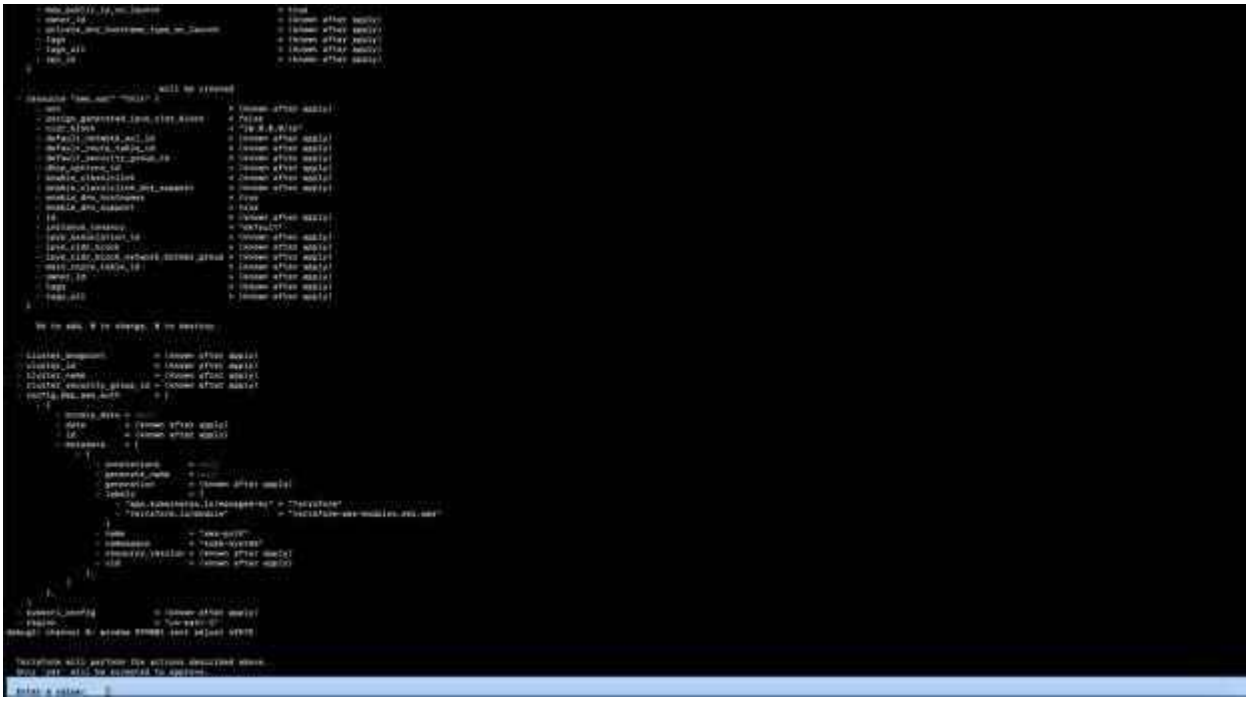


13. Execute the following command to determine the desired state of all the resources defined in the above .tf files.
    1. terraform plan

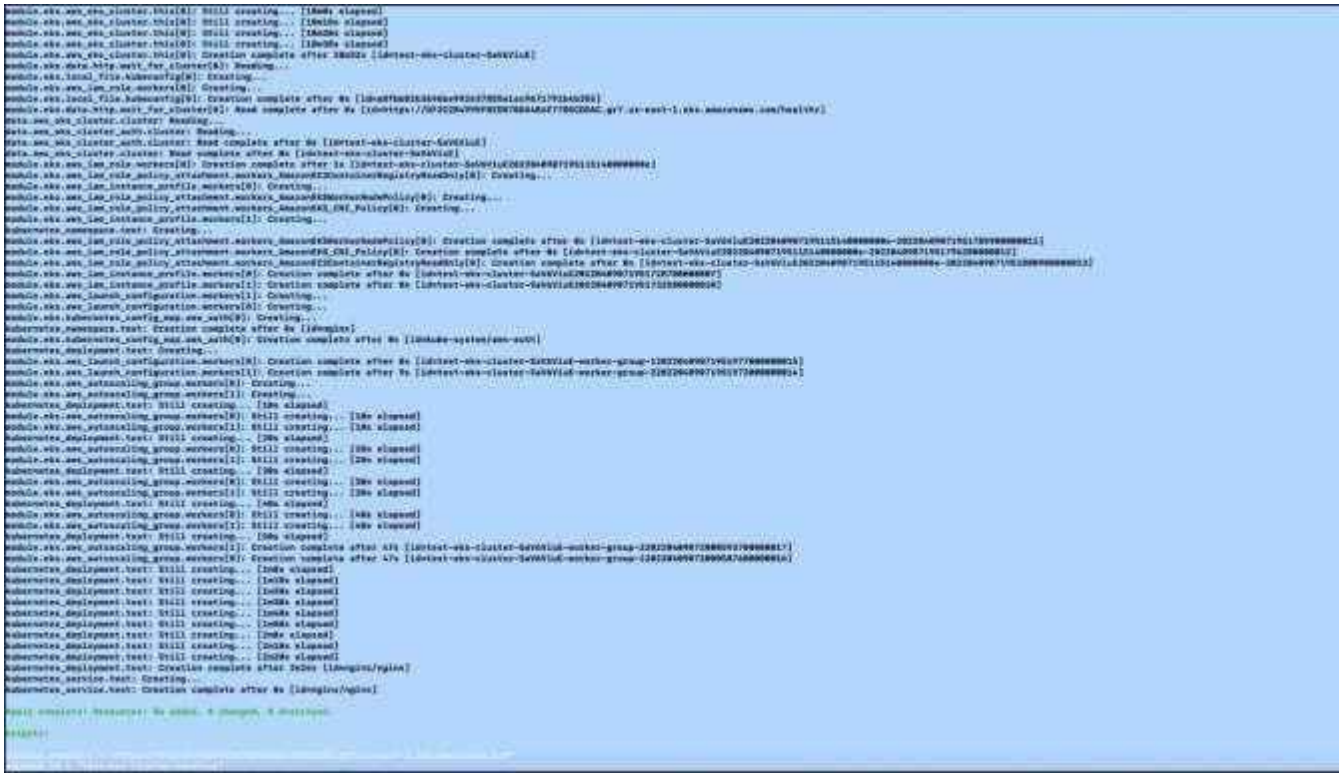


14. Before we go ahead and create our EKS Cluster, let's take a look at the "terraform graph" command. This is an optional step that you can skip if you wish. Basically, we are simply generating a visual representation of our execution plan. Once you execute the following command, you will get the output in a graph.svg file. You can try to open the file on your personal computer or online with SVG Viewer.
    1. terraform graph -type plan | dot -Tsvg > graph.svg

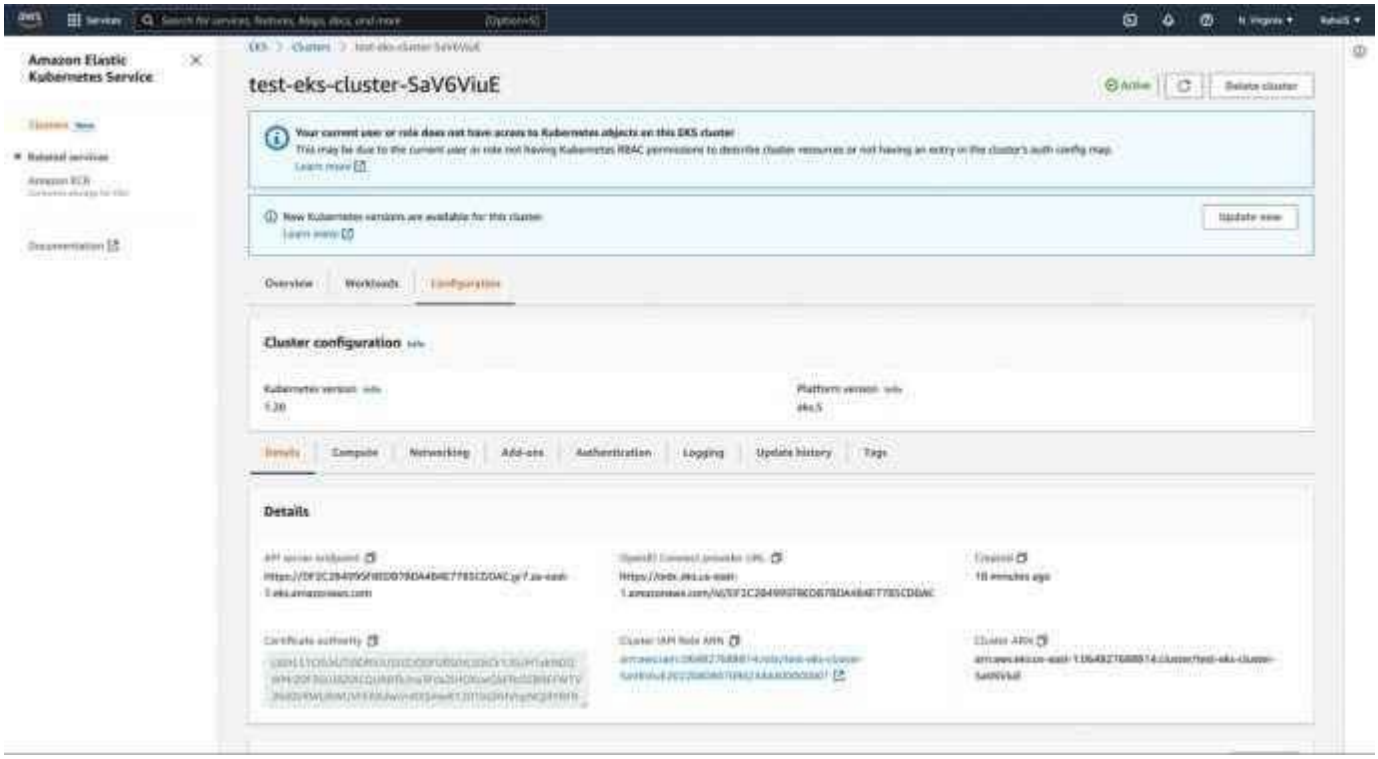
15. We must now create the resources using the .tf files in our current working directory. Execute the following command. This will take around 10 minutes to complete, therefore feel free to go have a cup of coffee. Once the command is successfully completed, the EKS Cluster will be ready.
    1. terraform apply



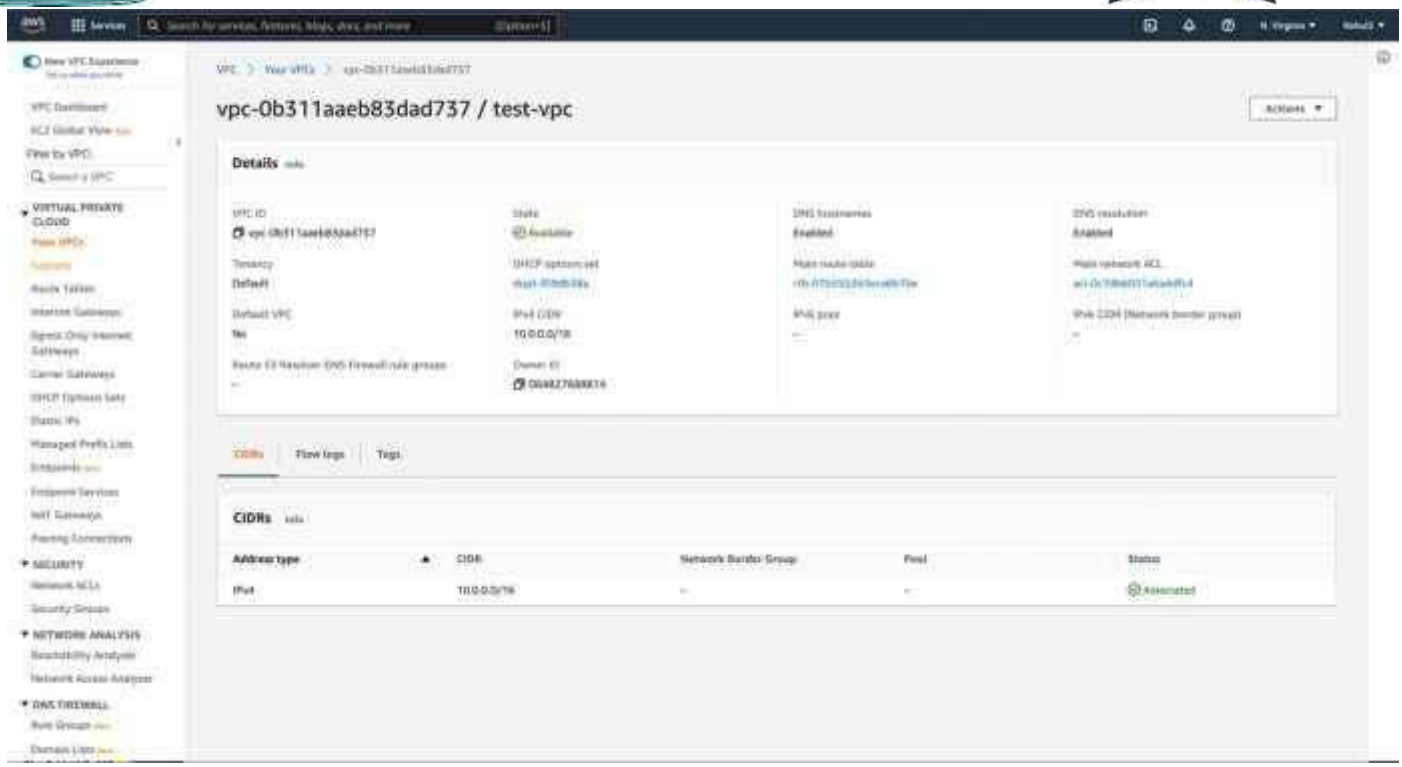
After the “terraform apply” command is successfully completed, you should see the output as depicted below.



16. You can now go to the AWS Console and verify the resources created as part of the EKS Cluster.
    1. EKS Cluster



    2. VPC

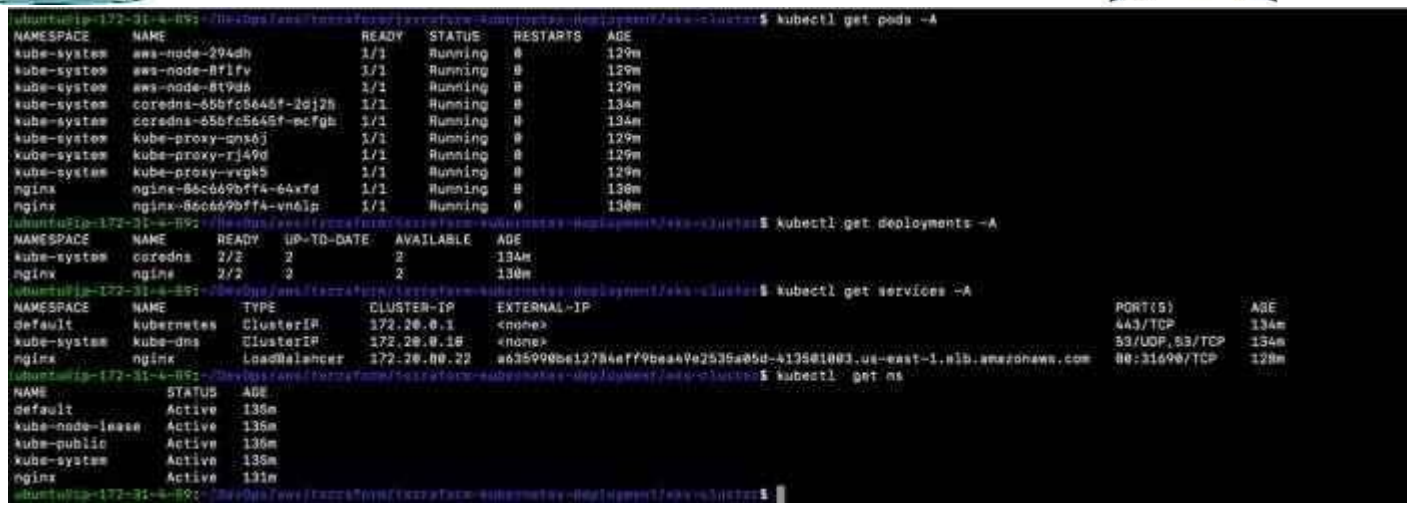


3. EC2 Instance

17. Now, you are ready to connect to your EKS Cluster and check the nodes in the Kubernetes Cluster using the following command:
    1. kubectl get nodes

```
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster$ kubectl get nodes
NAME                           STATUS   ROLES    AGE    VERSION
ip-10-0-1-146.ec2.internal     Ready    <none>   128m   v1.20.11-eks-f17b81
ip-10-0-1-238.ec2.internal     Ready    <none>   128m   v1.20.11-eks-f17b81
ip-10-0-2-151.ec2.internal     Ready    <none>   128m   v1.20.11-eks-f17b81
ubuntu@ip-172-31-4-59:~/DevOps/aws/terraform/terraform-kubernetes-deployment/eks-cluster$
```

18. Check the resources such as pods, deployments and services that are available in the Kubernetes Cluster across all namespaces.
    1. kubectl get pods -A
    2. kubectl get deployments -A
    3. kubectl get services -A
    4. kubectl get ns


$ kubectl get pods -A
NAMESPACE NAME READY STATUS RESTARTS AGE
kube-system aws-node-294dh 1/1 Running 0 129m
kube-system aws-node-8flfv 1/1 Running 0 129m
kube-system aws-node-8t9d8 1/1 Running 0 129m
kube-system coredns-65bfc5645f-2dj2h 1/1 Running 0 134m
kube-system coredns-65bfc5645f-mcfgb 1/1 Running 0 134m
kube-system kube-proxy-gnx6j 1/1 Running 0 129m
kube-system kube-proxy-rj49d 1/1 Running 0 129m
kube-system kube-proxy-vvgk5 1/1 Running 0 129m
nginx nginx-86c669bff4-64xfd 1/1 Running 0 130m
nginx nginx-86c669bff4-vn6lp 1/1 Running 0 130m
$ kubectl get deployments -A
NAMESPACE NAME READY UP-TO-DATE AVAILABLE AGE
kube-system coredns 2/2 2 2 134m
nginx nginx 2/2 2 2 130m
$ kubectl get services -A
NAMESPACE NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
default kubernetes ClusterIP 172.20.0.1 <none> 443/TCP 134m
kube-system kube-dns ClusterIP 172.20.0.10 <none> 53/UDP,53/TCP 134m
nginx nginx LoadBalancer 172.20.80.22 a635990be127b4eff9bea49e2535a05d-413501003.us-east-1.elb.amazonaws.com 80:31690/TCP 128m
$ kubectl get ns
NAME STATUS AGE
default Active 135m
kube-node-lease Active 135m
kube-public Active 135m
kube-system Active 135m
nginx Active 131m
$