

# Deploy a Kubernetes Application With Terraform and AWS EKS

## 1. Introduction

### Case Study Overview:

The chosen case study focuses on building and managing multi-cloud infrastructure using Terraform and Kubernetes on AWS. The objective is to create and manage resources, such as an AWS S3 bucket and a Kubernetes cluster, using Terraform scripts, followed by deploying a sample application on the Kubernetes cluster to ensure functionality.

### Key Feature and Application:

The unique feature of this experiment is the seamless integration of multi-cloud infrastructure management using Terraform. The practical use of this feature is seen in the automation of provisioning and managing both storage and compute resources (AWS S3 and Kubernetes) across different cloud services. It demonstrates how Terraform simplifies infrastructure management by treating it as code, making it easier to deploy applications consistently across multiple cloud environments.

### Third-Year Project Integration (Optional):

In my third-year project, MentorLink, where mentees connect with mentors for guidance, deploying the application on a cloud infrastructure like Kubernetes would enhance its scalability and reliability. Using Terraform for infrastructure management could help MentorLink scale as the number of users grows, providing a robust and flexible platform.

## 2. Step-by-Step Explanation

### 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 an EKS cluster using Terraform CLI on the AWS cloud.

- AWS CLI is installed and configured with your temporary credentials from AWS Academy.
- Terraform is installed on your machine.
- kubectl is installed to manage the Kubernetes cluster.

## 1. Set Up Your Environment

- **Install Terraform:** Make sure you have Terraform installed. You can download it from the Terraform website.

Create a directory for experiment and cd into it and open using vs code

```
PS C:\Users\ADITYA DUBEY> mkdir eks-setup

Directory: C:\Users\ADITYA DUBEY

Mode                LastWriteTime         Length Name
----                -
d-----          10/21/2024  10:49 PM             eks-setup

PS C:\Users\ADITYA DUBEY> cd .\eks-setup\
PS C:\Users\ADITYA DUBEY\eks-setup> code .
```

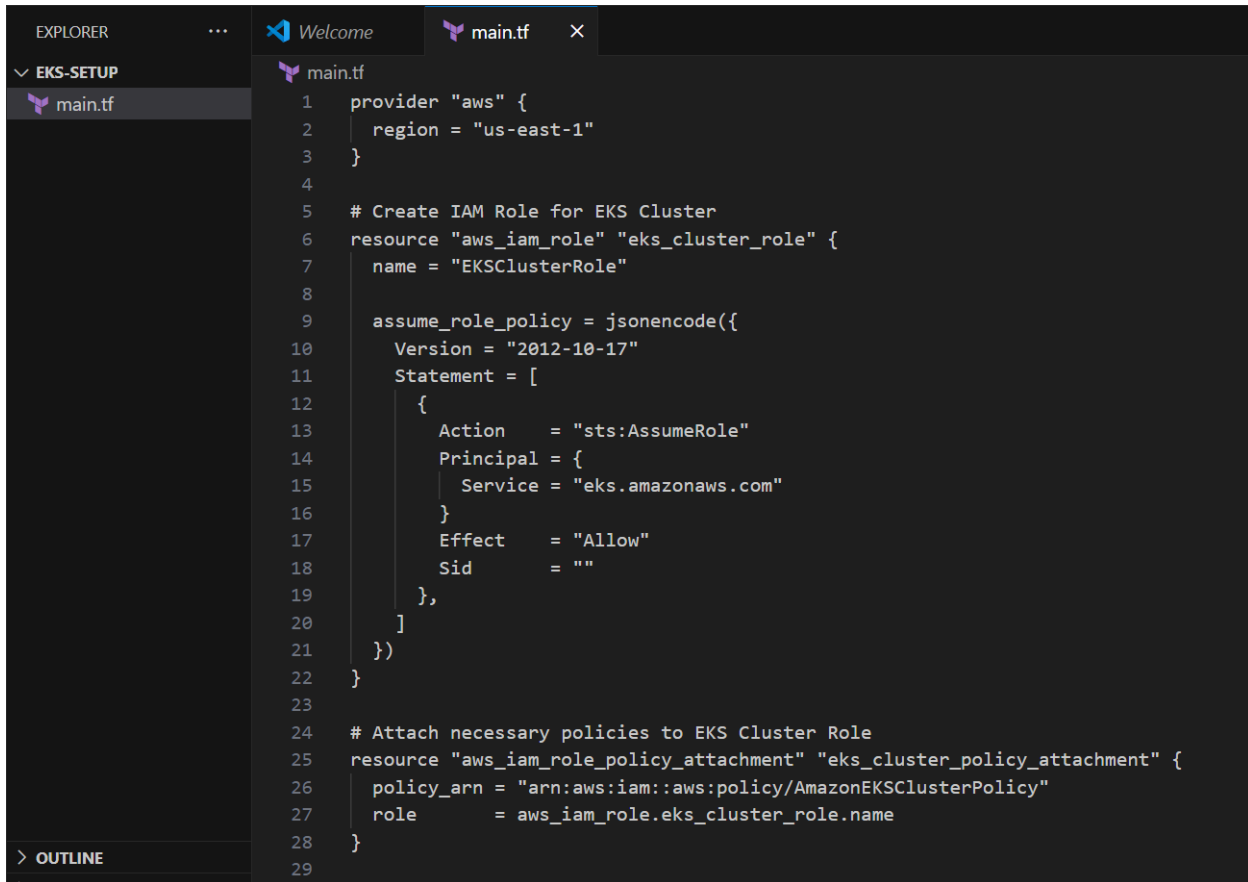
**AWS CLI:** Install and configure the AWS CLI with your credentials:

aws configure

Create an iam user and the configuration as needed

## 2. Create a Terraform Configuration File

Create a file named main.tf and paste the following code into it:



```
1 provider "aws" {
2   region = "us-east-1"
3 }
4
5 # Create IAM Role for EKS Cluster
6 resource "aws_iam_role" "eks_cluster_role" {
7   name = "EKSClusterRole"
8
9   assume_role_policy = jsonencode({
10     Version = "2012-10-17"
11     Statement = [
12       {
13         Action      = "sts:AssumeRole"
14         Principal = {
15           Service = "eks.amazonaws.com"
16         }
17         Effect      = "Allow"
18         Sid         = ""
19       },
20     ]
21   })
22 }
23
24 # Attach necessary policies to EKS Cluster Role
25 resource "aws_iam_role_policy_attachment" "eks_cluster_policy_attachment" {
26   policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
27   role       = aws_iam_role.eks_cluster_role.name
28 }
29
```

terraform init

```
PS C:\Users\ADITYA DUBEY\eks-setup> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
```

terraform plan

```
PS C:\Users\ADITYA DUBEY\eks-setup> terraform plan

Terraform used the selected providers to generate the following execution plan. R
following symbols:
+ create

Terraform will perform the following actions:

# aws_eks_cluster.my_cluster will be created
+ resource "aws_eks_cluster" "my_cluster" {
  + arn                               = (known after apply)
  + bootstrap_self_managed_addons    = true
  + certificate_authority             = (known after apply)
  + cluster_id                       = (known after apply)
  + created_at                       = (known after apply)
  + endpoint                         = (known after apply)
  + id                               = (known after apply)
  + identity                         = (known after apply)
  + name                             = "eks-cluster-14920251"
  + platform_version                 = (known after apply)
  + role_arn                         = (known after apply)
  + status                           = (known after apply)
  + tags_all                         = (known after apply)
  + version                         = "1.21"

  + access_config (known after apply)

  + kubernetes_network_config (known after apply)

  + upgrade_policy (known after apply)
```

terraform apply

- Type yes to confirm the deployment.

```
you run "terraform apply" now.
PS C:\Users\ADITYA DUBEY\eks-setup> terraform apply

Terraform used the selected providers to generate the following execution plan. Resou
following symbols:
+ create

Terraform will perform the following actions:

# aws_eks_cluster.my_cluster will be created
+ resource "aws_eks_cluster" "my_cluster" {
  + arn                               = (known after apply)
  + bootstrap_self_managed_addons    = true
  + certificate_authority             = (known after apply)
  + cluster_id                       = (known after apply)
  + created_at                       = (known after apply)
  + endpoint                         = (known after apply)
  + id                               = (known after apply)
  + identity                         = (known after apply)
  + name                             = "eks-cluster-14920251"
  + platform_version                 = (known after apply)
  + role_arn                         = (known after apply)
  + status                           = (known after apply)
  + tags_all                         = (known after apply)
  + version                         = "1.21"

  + access_config (known after apply)

  + kubernetes_network_config (known after apply)
```

```
aws_subnet.public[0]: Creation complete after 13s [Id=subnet-07f97a86339f
aws_eks_cluster.my_cluster: Creating...
aws_eks_cluster.my_cluster: Still creating... [10s elapsed]
aws_eks_cluster.my_cluster: Still creating... [20s elapsed]
aws_eks_cluster.my_cluster: Still creating... [30s elapsed]
aws_eks_cluster.my_cluster: Still creating... [40s elapsed]
aws_eks_cluster.my_cluster: Still creating... [50s elapsed]
aws_eks_cluster.my_cluster: Still creating... [1m0s elapsed]
aws_eks_cluster.my_cluster: Still creating... [1m10s elapsed]
aws_eks_cluster.my_cluster: Still creating... [1m20s elapsed]
aws_eks_cluster.my_cluster: Still creating... [1m30s elapsed]
aws_eks_cluster.my_cluster: Still creating... [1m40s elapsed]
aws_eks_cluster.my_cluster: Still creating... [1m50s elapsed]
aws_eks_cluster.my_cluster: Still creating... [2m0s elapsed]
```

```
aws_eks_node_group.my_node_group: Still creating... [18m41s elapsed]
aws_eks_node_group.my_node_group: Still creating... [18m51s elapsed]
aws_eks_node_group.my_node_group: Still creating... [19m1s elapsed]
aws_eks_node_group.my_node_group: Still creating... [19m11s elapsed]
aws_eks_node_group.my_node_group: Still creating... [19m21s elapsed]
```

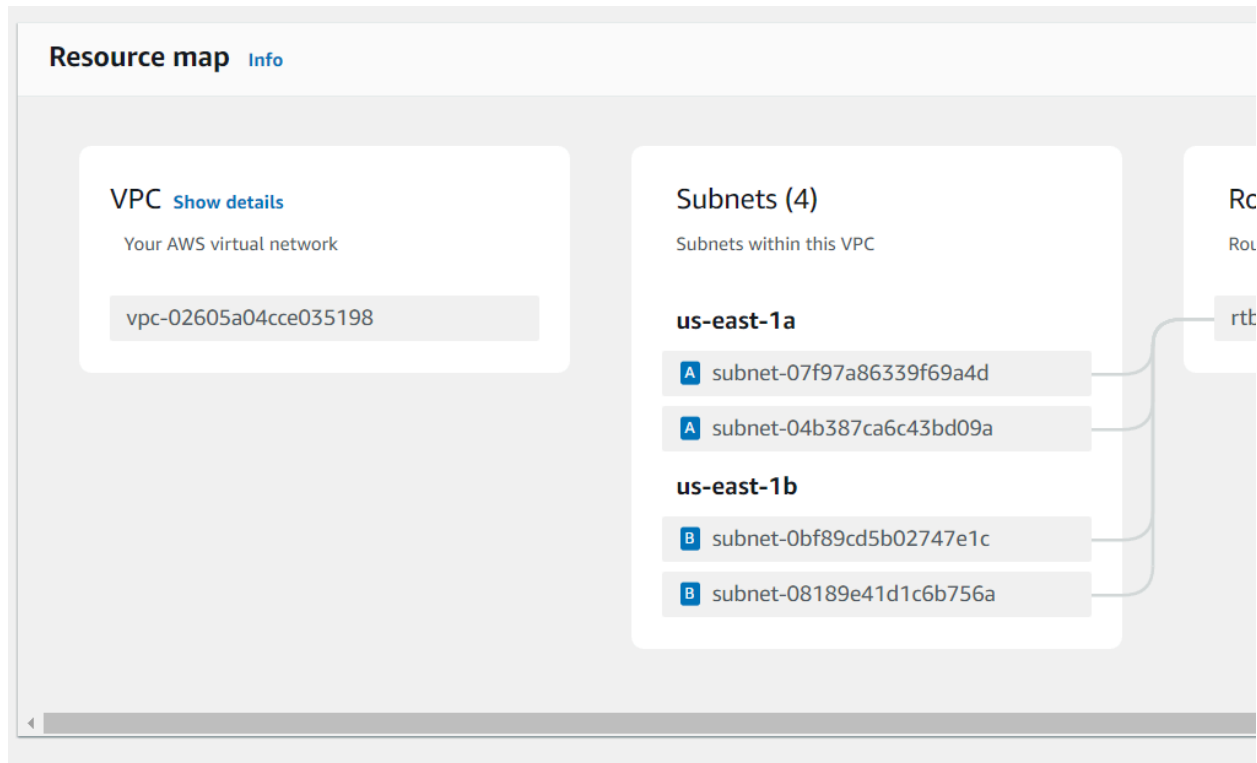
It will take long time 10-20 min

## 6. Verify the Deployment

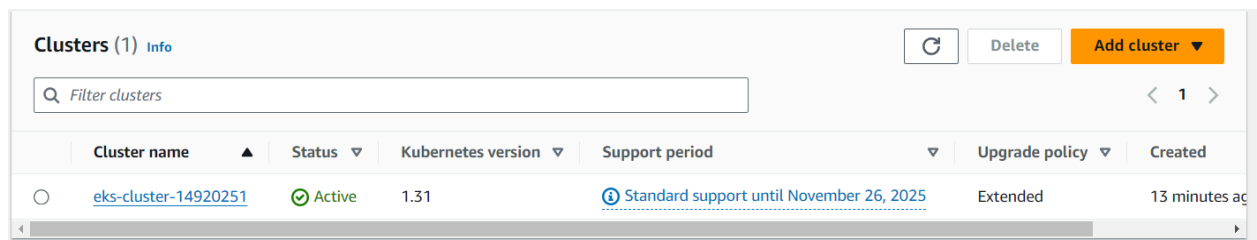
For S3 bucket:

| General purpose buckets (1) <a href="#">Info</a> <a href="#">All AWS Regions</a> |   |                                 |   |  |
|--|---|---------------------------------|---|--|
| Buckets are containers for data stored in S3.                                    |   |                                 |   |  |
| <input type="text" value="Find buckets by name"/>                                |   |                                 |   |  |
| <div>&lt; 1 &gt; ⚙</div>   |   |                                 |   |  |
|  | Name ▲  | AWS Region ▼                    | IAM Access Analyzer                         | Creation date ▼                        |
| <input type="radio"/>  | <a href="#">s3-bucket-14920251-646321345213</a> | US East (N. Virginia) us-east-1 | <a href="#">View analyzer for us-east-1</a> | October 22, 2024, 00:22:54 (UTC+05:30) |

For vpc and subnet:



### For EKS Cluster:



- **Using AWS Management Console:**
  - Go to the [EKS service](#).
  - Check that the cluster eks-cluster-14920251 is listed and its status is **ACTIVE**.

Connect to the EKS cluster with **kubectl**:

```
aws eks update-kubeconfig --name eks-cluster-14920251
```

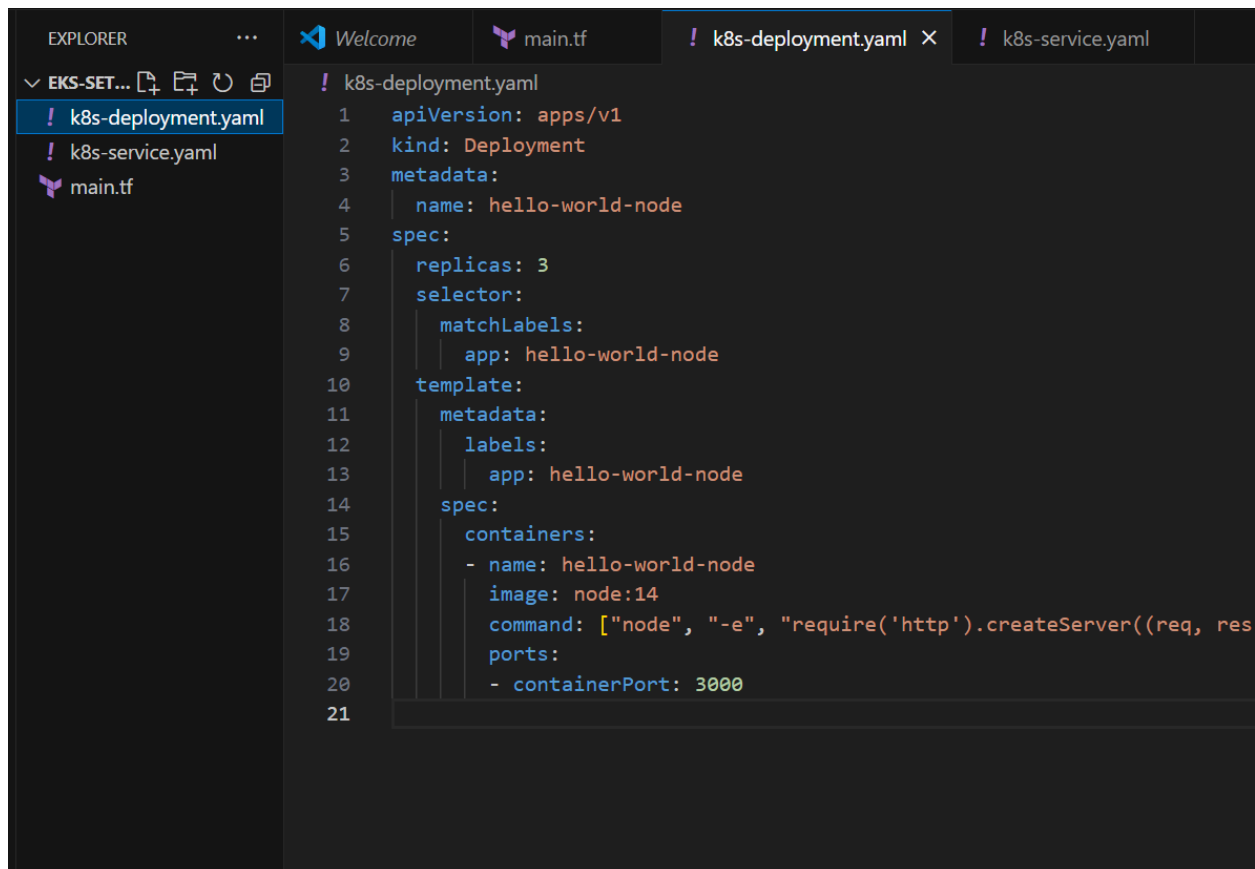
```
kubectl get nodes
```

## 7. Integrate Your Node.js Application

After verifying that your EKS cluster and S3 bucket are functioning, integrate your Node.js application with EKS. This usually involves deploying the application using Kubernetes manifests (YAML files) that define your deployments, services, and any other resources required by your application.

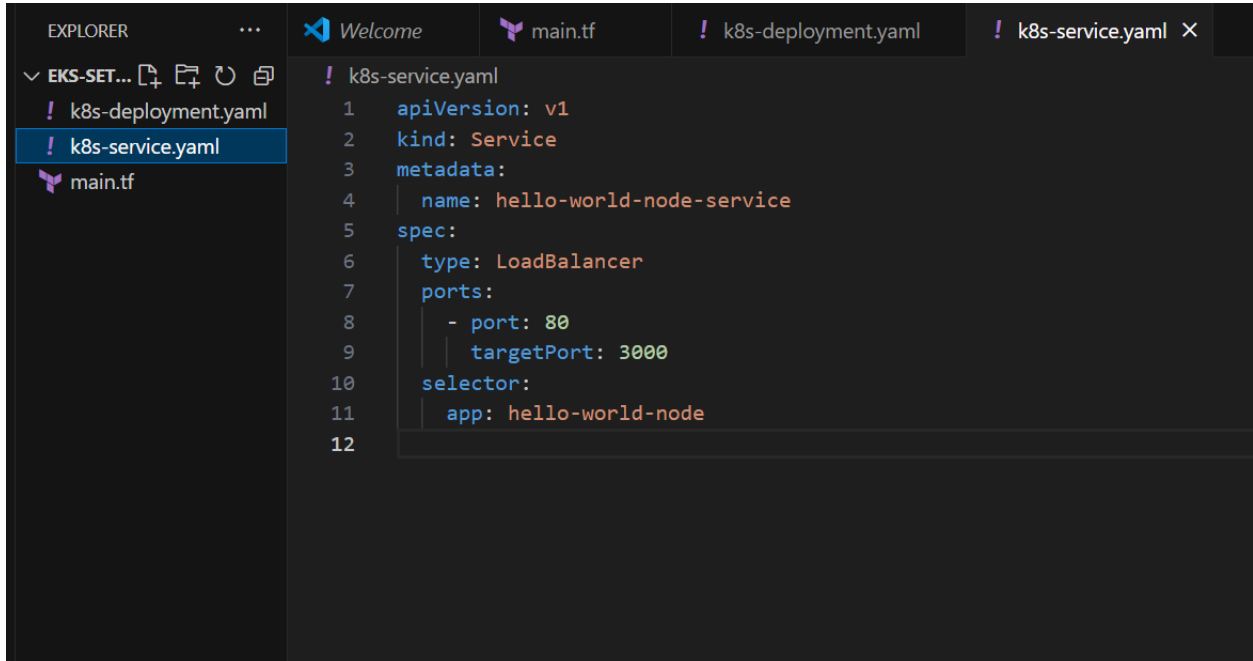
### Step 5: Set Up Kubernetes Deployment

**Create a Kubernetes YAML File:** Create a file named `k8s-deployment.yaml` in the same directory

A screenshot of a code editor interface. The Explorer panel on the left shows a directory named 'EKS-SET...' containing three files: 'k8s-deployment.yaml' (highlighted), 'k8s-service.yaml', and 'main.tf'. The main editor area displays the content of 'k8s-deployment.yaml' with line numbers 1 through 21. The YAML content is as follows:

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: hello-world-node
5  spec:
6    replicas: 3
7    selector:
8      matchLabels:
9        app: hello-world-node
10   template:
11     metadata:
12       labels:
13         app: hello-world-node
14     spec:
15       containers:
16       - name: hello-world-node
17         image: node:14
18         command: ["node", "-e", "require('http').createServer((req, res
19         ports:
20         - containerPort: 3000
21
```

**Create a Service YAML File:** Create a file named `k8s-service.yaml` in the same directory:



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays a file tree with 'EKS-SET...', 'k8s-deployment.yaml', 'k8s-service.yaml' (selected), and 'main.tf'. The main editor area shows the content of 'k8s-service.yaml' with the following YAML code:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: hello-world-node-service
5  spec:
6    type: LoadBalancer
7    ports:
8      - port: 80
9        targetPort: 3000
10   selector:
11     app: hello-world-node
12
```

## Step 6: Configure kubectl

**Install kubectl:** Follow the instructions on the Kubernetes website to install kubectl.

**Update kubeconfig:** Use the following command to update your kubeconfig to connect to the EKS cluster:

```
aws eks update-kubeconfig --name eks-cluster-14920251
```

1.

## Step 7: Deploy to Kubernetes

**Apply the Deployment and Service:** Run the following commands:

```
kubectl apply -f k8s-deployment.yaml
```

```
kubectl apply -f k8s-service.yaml
```

## Step 8: Verify Your Deployment



**Check the Status of Your Pods:**`bash``Copy code``kubectl get pods`**Get the External IP of Your Service:**`kubectl get svc hello-world-node-service`

1. Look for the EXTERNAL-IP field. It may take a few minutes for the external IP to be assigned.
2. **Access Your Application:** Open a web browser and go to `http://<EXTERNAL-IP>/. You should see the message "Hello, World!"`.

**Step 9: Clean Up Resources**

After testing, you may want to remove the resources to avoid incurring charges.

**1. Run Terraform Destroy:**`bash``Copy code``terraform destroy`

```
PS C:\Users\ADITYA DUBEY\eks-setup> terraform destroy
data.aws_availability_zones.available: Reading...
aws_iam_role.eks_cluster_role: Refreshing state... [id=EKSClusterRole]
aws_vpc.my_vpc: Refreshing state... [id=vpc-02605a04cce035198]
aws_iam_role.eks_node_role: Refreshing state... [id=EKSNodeRole]
aws_s3_bucket.my_bucket: Refreshing state... [id=s3-bucket-14920251-646321345213]
data.aws_availability_zones.available: Read complete after 1s [id=us-east-1]
aws_iam_role_policy_attachment.eks_cluster_policy_attachment: Refreshing state... [id=EKSClusterRole-2024102118520000001]
aws_iam_role_policy_attachment.eks_cni_policy_attachment: Refreshing state... [id=EKSClusterRole-202410211852002]
aws_iam_role_policy_attachment.eks_worker_policy_attachment: Refreshing state... [id=EKSNodeRole-202410211852004]
aws_iam_role_policy_attachment.eks_node_cni_policy_attachment: Refreshing state... [id=EKSNodeRole-20241021185200003]
aws_iam_role_policy_attachment.eks_ecr_read_only: Refreshing state... [id=EKSNodeRole-20241021185256253000000]
aws_subnet.public[1]: Refreshing state... [id=subnet-08189e41d1c6b756a]
aws_subnet.public[0]: Refreshing state... [id=subnet-07f97a86339f69a4d]
aws_subnet.private[1]: Refreshing state... [id=subnet-0bf89cd5b02747e1c]
aws_subnet.private[0]: Refreshing state... [id=subnet-04b387ca6c43bd09a]
aws_security_group.eks_security_group: Refreshing state... [id=sg-073d8f5a2524aa5bb]
aws_eks_cluster.my_cluster: Refreshing state... [id=eks-cluster-14920251]
aws_eks_node_group.my_node_group: Refreshing state... [id=eks-cluster-14920251:my-node-group]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated
following symbols:
- destroy

Terraform will perform the following actions:
```

Type yes for destruction and thus stop further charges

## Conclusion:

In this project, we utilized **Terraform** to automate the creation and management of multi-cloud resources, focusing on **AWS**. The solution involved setting up an **S3 bucket** for storage and provisioning a **Kubernetes cluster** on AWS for container orchestration. By deploying a sample application (such as a Node.js app) on the Kubernetes cluster, we demonstrated the seamless integration of infrastructure and applications across the cloud.

The key takeaways include:

1. **Terraform's flexibility:** It simplifies the process of infrastructure management by codifying resources, allowing us to deploy, manage, and scale services efficiently.
2. **Scalability with Kubernetes:** The Kubernetes cluster provided an environment for easy deployment and scaling of applications, ensuring smooth and reliable operations.
3. **Cloud-native storage with S3:** The AWS S3 bucket offers scalable and durable storage, supporting the data needs of the deployed application.

This project showcases the power of **infrastructure as code (IaC)** in building scalable, cloud-based solutions.