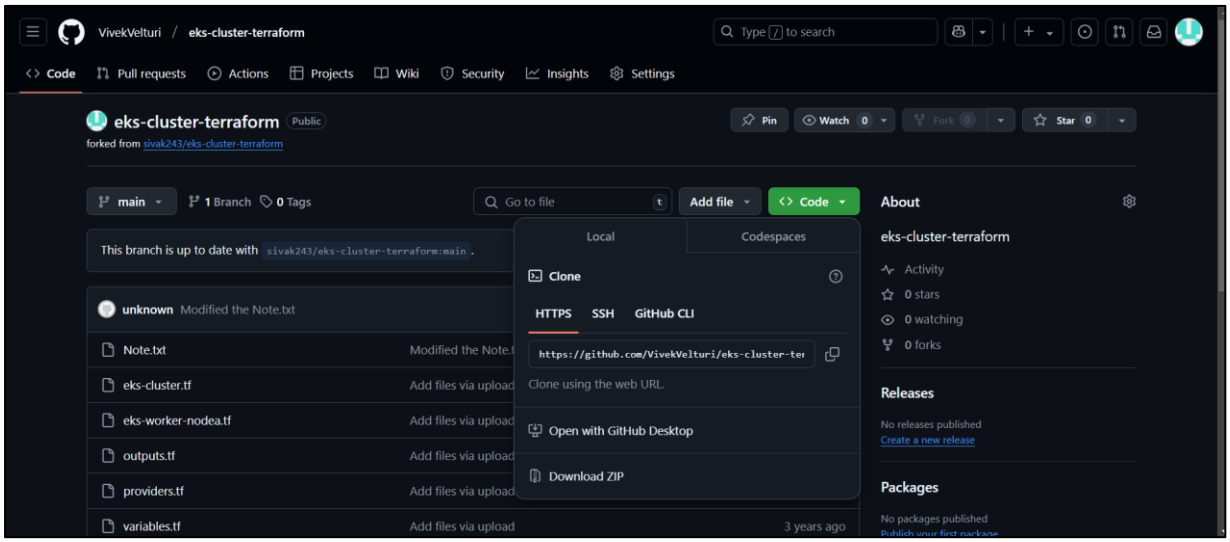
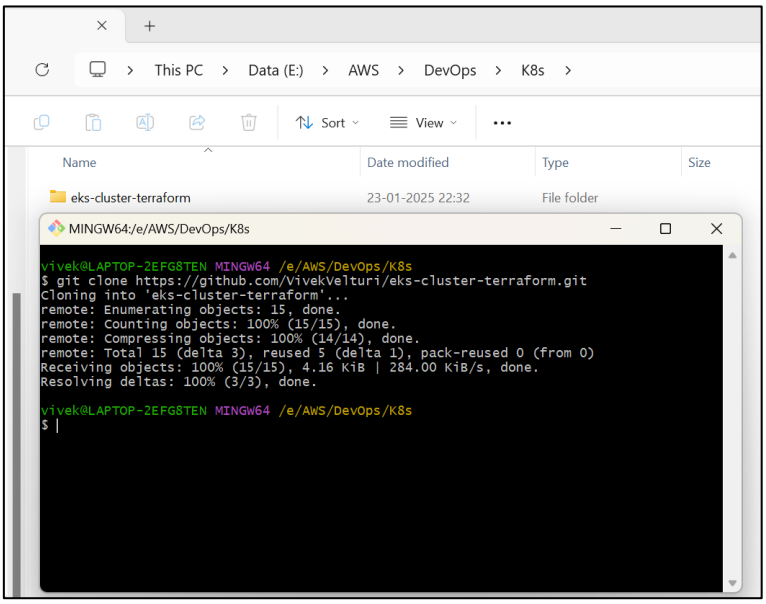
EKS Cluster Deployment with Terraform and Advanced Kubernetes Configurations

In this Project, we will set up an EKS cluster using Terraform, deploy Pods, and explore NodePort and LoadBalancer services. Additionally, we will cover key concepts such as liveness probes, Persistent Volumes (PV) and Persistent Volume Claims (PVC), namespaces, and taints and tolerations.

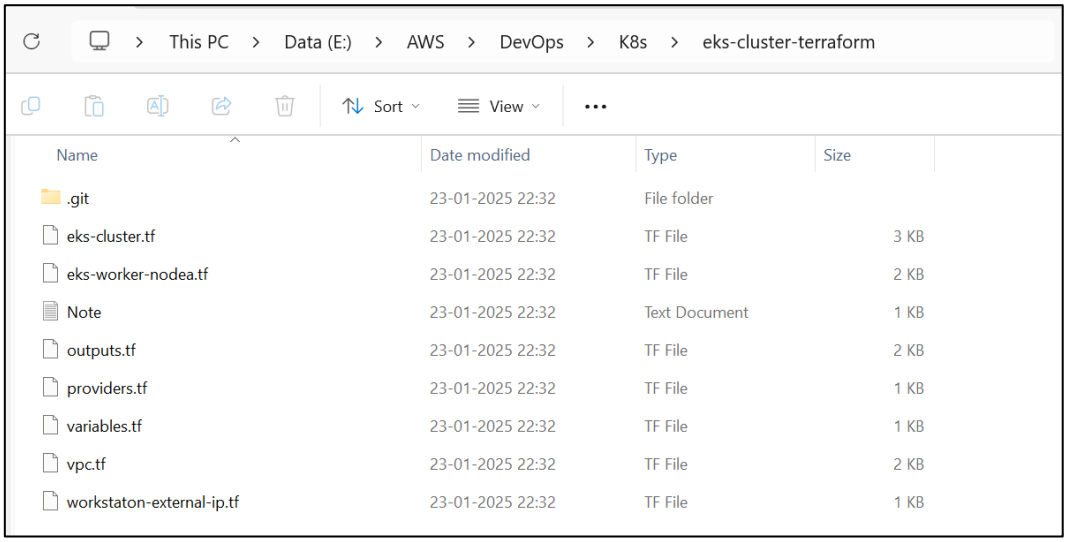
CLONE TERRAFORM SCRIPTS FROM GitHub:

- Copy the URL for the Terraform script and clone them into folder in your local computer using the command git clone <URL of the scripts repository> in my case: git clone https://github.com/VivekVelturi/eks-cluster-terraform.git





The copied terraform script is as follows:



eks-cluster.tf

```
#
# EKS Cluster Resources
#  * IAM Role to allow EKS service to manage other AWS services
#  * EC2 Security Group to allow networking traffic with EKS cluster
#  * EKS Cluster
#

resource "aws_iam_role" "demo-cluster" { # Creates an IAM role for the EKS cluster
  name = "terraform-eks-demo-cluster" # Name of the IAM role

  assume_role_policy = <<POLICY
{ # Policy allowing EKS service to assume this IAM role
  "Version": "2012-10-17", # Defines the policy version
  "Statement": [ # List of statements granting permissions
    {
      "Effect": "Allow", # Allows the action
      "Principal": { # The entity that can assume the role
        "Service": "eks.amazonaws.com" # Specifies the EKS service
      },
      "Action": "sts:AssumeRole" # Grants permission to assume this role
    }
  ]
}
POLICY
}

resource "aws_iam_role_policy_attachment" "demo-cluster-
AmazonEKSClusterPolicy" {
```

```
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy" # Attaches the EKS Cluster Policy
  role       = aws_iam_role.demo-cluster.name # Associates the policy with the demo-cluster IAM role
}

resource "aws_iam_role_policy_attachment" "demo-cluster-AmazonEKSVPCResourceController" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController" # Attaches the VPC Resource Controller policy
  role       = aws_iam_role.demo-cluster.name # Associates the policy with the IAM role
}

resource "aws_security_group" "demo-cluster" {
  name        = "terraform-eks-demo-cluster" # Security group name
  description = "Cluster communication with worker nodes" # Description of the security group
  vpc_id      = aws_vpc.demo.id # Associates the security group with a specific VPC

  egress { # Egress rule to allow all outbound traffic
    from_port   = 0 # Start of port range (0 means all)
    to_port     = 0 # Start of port range (0 means all)
    protocol    = "-1" # Protocol "-1" allows all protocols
    cidr_blocks = ["0.0.0.0/0"] # CIDR block for all IPv4 addresses
  }

  tags = { # Tags for identifying the resource
    Name = "terraform-eks-demo" # Name tag for the security group
  }
}

resource "aws_security_group_rule" "demo-cluster-ingress-workstation-https" {
  cidr_blocks       = [local.workstation-external-cidr] # Allows access from the workstation's external IP range
  description       = "Allow workstation to communicate with the cluster API Server" # Description of the rule
  from_port         = 443 # Start of port range (443 is used for HTTPS)
  protocol          = "tcp" # Restricts to TCP protocol
  security_group_id = aws_security_group.demo-cluster.id # Associates the rule with the demo-cluster security group
  to_port           = 443 # End of port range (443 is used for HTTPS).
  type              = "ingress" # Ingress type means traffic coming into the cluster
}

resource "aws_eks_cluster" "demo" {
  name     = var.cluster-name # Name of the cluster, passed as a variable
  role_arn = aws_iam_role.demo-cluster.arn # IAM role to be used by the EKS cluster
```

```
  vpc_config { # VPC networking configuration for the cluster
    security_group_ids = [aws_security_group.demo-cluster.id] # VPC networking configuration for the cluster
    subnet_ids         = aws_subnet.demo[*].id # Subnets for the cluster (referenced from another resource)
  }

  depends_on = [ # Ensures these policies are attached before creating the cluster
    aws_iam_role_policy_attachment.demo-cluster-AmazonEKSClusterPolicy,
    aws_iam_role_policy_attachment.demo-cluster-AmazonEKSVPCResourceController,
  ]
}
```

This Terraform configuration:

1. Creates an IAM role with necessary permissions for EKS.
2. Sets up a security group for secure communication.
3. Deploys an EKS cluster in a VPC with the above configurations.

eks-worker-nodea.tf

```
#
# EKS Worker Nodes Resources
#  * IAM role allowing Kubernetes actions to access other AWS services
#  * EKS Node Group to launch worker nodes
#

resource "aws_iam_role" "demo-node" { # Creates an IAM role for EKS worker nodes
  name = "terraform-eks-demo-node" # Assigns a name to the IAM role

  assume_role_policy = <<POLICY # Policy that allows the EC2 service to assume this IAM role
{
  "Version": "2012-10-17", # Policy version
  "Statement": [ # List of policy statements
    {
      "Effect": "Allow", # Allows the specified action
      "Principal": { # The entity that can assume this role
        "Service": "ec2.amazonaws.com" # Specifies EC2 instances
      },
      "Action": "sts:AssumeRole" # Grants permission to assume this role
    }
  ]
}
POLICY
}
```

```
resource "aws_iam_role_policy_attachment" "demo-node-
AmazonEKSWorkerNodePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy" # Attaches the
Worker Node Policy to the IAM role
  role       = aws_iam_role.demo-node.name # Associates the policy with the demo-node
IAM role
}

resource "aws_iam_role_policy_attachment" "demo-node-AmazonEKS_CNI_Policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy" # Attaches the
Amazon EKS CNI Policy
  role       = aws_iam_role.demo-node.name # Associates the policy with the demo-node
IAM role
}

resource "aws_iam_role_policy_attachment" "demo-node-
AmazonEC2ContainerRegistryReadOnly" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly" #
Attaches a policy to allow read-only access to ECR
  role       = aws_iam_role.demo-node.name # Associates the policy with the demo-node
IAM role
}

resource "aws_eks_node_group" "demo" { # Creates an EKS Node Group for worker nodes
  cluster_name    = aws_eks_cluster.demo.name # Specifies the name of the EKS cluster to
associate with the node group
  node_group_name = "demo" # Assigns a name to the node group
  node_role_arn   = aws_iam_role.demo-node.arn # Specifies the IAM role to be assumed
by the worker nodes
  subnet_ids      = aws_subnet.demo[*].id # Associates the node group with the specified
subnets

  scaling_config { # Configures scaling for the node group
    desired_size = 1 # Sets the desired number of nodes in the group
    max_size     = 1 # Sets the maximum number of nodes
    min_size     = 1 # Sets the minimum number of nodes
  }

  depends_on = [ # Ensures the IAM policies are attached before creating the node group
    aws_iam_role_policy_attachment.demo-node-AmazonEKSWorkerNodePolicy,
    aws_iam_role_policy_attachment.demo-node-AmazonEKS_CNI_Policy,
    aws_iam_role_policy_attachment.demo-node-
AmazonEC2ContainerRegistryReadOnly,
  ]
}
```

This configuration sets up the resources needed for EKS worker nodes:

1. An IAM role (`aws_iam_role.demo-node`) is created, allowing EC2 instances (worker nodes) to assume the role and interact with AWS services.

2. Three managed policies are attached to the role:
    - `AmazonEKSWorkerNodePolicy`: Grants worker nodes the necessary permissions to interact with the EKS cluster.
    - `AmazonEKS_CNI_Policy`: Enables the nodes to manage network interfaces required for Kubernetes pods.
    - `AmazonEC2ContainerRegistryReadOnly`: Provides read-only access to the Amazon Elastic Container Registry (ECR) for pulling container images.
3. An EKS Node Group (`aws_eks_node_group.demo`) is created to launch and manage worker nodes. It uses the IAM role and is associated with specific subnets. The scaling configuration is set to maintain exactly one worker node in the group. The `depends_on` block ensures that the IAM role and its policies are attached before creating the node group, avoiding dependency issues.

outputs.tf

```
#
# Outputs
#

locals {  # Defines reusable local variables for the Terraform configuration
  config_map_aws_auth = <<CONFIGMAPAWSAUTH # Creates a multi-line string for the "aws-auth" ConfigMap in Kubernetes


apiVersion: v1 # Specifies the Kubernetes API version used for the ConfigMap
kind: ConfigMap # Declares the resource type as ConfigMap
metadata: # Metadata section for identifying the resource
  name: aws-auth # Assigns the name "aws-auth" to the ConfigMap
  namespace: kube-system # Places the ConfigMap in the "kube-system" namespace
data: # Contains the actual data for the ConfigMap
  mapRoles: | # Maps IAM roles to Kubernetes users and groups
    - rolearn: ${aws_iam_role.demo-node.arn} # References the ARN of the IAM role for worker nodes
      username: system:node:{{EC2PrivateDNSName}} # Sets the username format for the worker nodes
      groups: # Assigns worker nodes to Kubernetes groups
        - system:bootstrappers # Group responsible for bootstrapping nodes
        - system:nodes # Group for nodes in the Kubernetes cluster
CONFIGMAPAWSAUTH # End of the multi-line string for the ConfigMap

  kubeconfig = <<KUBECONFIG # Creates a multi-line string for the Kubernetes kubeconfig file


apiVersion: v1 # Specifies the Kubernetes API version used for the kubeconfig
clusters: # List of Kubernetes clusters the kubeconfig connects to
- cluster: # Cluster definition block
    server: ${aws_eks_cluster.demo.endpoint} # References the API server endpoint for the EKS cluster
```

```
    certificate-authority-data:
${aws_eks_cluster.demo.certificate_authority[0].data} # Specifies the certificate for
secure communication.
  name: Kubernetes # Assigns a name to the cluster
contexts: # List of contexts defining how to connect to the cluster
- context: # Context definition block
    cluster: Kubernetes # Specifies the cluster name for the context
    user: aws # Specifies the user for authentication
  name: aws # Assigns a name to the context
current-context: aws # Sets the current context to "aws"
kind: Config # Declares the resource type as Config
preferences: {} # Placeholder for user preferences (empty in this case)
users: # List of users for the kubeconfig
- name: aws # Assigns the name "aws" to the user
  user: # User authentication method
    exec: # Executes an external command for authentication
      apiVersion: client.authentication.k8s.io/v1beta1 # API version for the
authentication command
      command: aws-iam-authenticator # Specifies the authentication tool
      args: # List of arguments for the authenticator command
        - "token" # Generates a token for authentication
        - "-i" # Specifies the cluster name as an input parameter
        - "${var.cluster-name}" # References the cluster name variable
KUBECONFIG # End of the multi-line string for the kubeconfig
}

output "config_map_aws_auth" { # Declares an output for the "aws-auth" ConfigMap
  value = local.config_map_aws_auth # Outputs the content of the aws-auth ConfigMap
template
}

output "kubeconfig" { # Declares an output for the kubeconfig file
  value = local.kubeconfig # Outputs the content of the kubeconfig template
}
```

This configuration generates and outputs two essential resources for the EKS cluster:

1. **`config_map_aws_auth`**:
   - A Kubernetes ConfigMap named `aws-auth` is generated. It maps the worker node IAM role (`demo-node`) to Kubernetes users and groups, allowing the nodes to join the cluster and perform their functions.
   - The `system:bootstrappers` and `system:nodes` groups give worker nodes the required permissions in Kubernetes.
2. **`kubeconfig`**:
   - A kubeconfig file is generated, which contains the necessary configuration to connect to the EKS cluster. It includes the API server endpoint, the certificate for secure communication, and the authentication method using `aws-iam-authenticator`.

By outputting these configurations, users can easily deploy them to their Kubernetes cluster and access the cluster securely from their local machine or other tools.

vpc.tf

```
#
# VPC Resources
#  * VPC
#  * Subnets
#  * Internet Gateway
#  * Route Table
#

resource "aws_vpc" "demo" {  # Defines an AWS Virtual Private Cloud (VPC) resource
  cidr_block = "10.0.0.0/16"  # Specifies the CIDR block for the VPC, which provides a large address space (`10.0.0.0/16`)

  tags = tomap({  # Tags for the VPC, for organizational and identification purposes
    "Name"                                      = "terraform-eks-demo-node",
    "kubernetes.io/cluster/${var.cluster-name}" = "shared",
  })
}

resource "aws_subnet" "demo" {  # Defines an AWS Subnet resource
  count = 2  # Creates two subnets (subnet1 and subnet2)

  availability_zone       = data.aws_availability_zones.available.names[count.index]  # Fetches the availability zone names in a loop using count.index.
  cidr_block              = "10.0.${count.index}.0/24"  # Generates CIDR blocks `10.0.0.0/24` and `10.0.1.0/24`
  map_public_ip_on_launch = true  # Maps public IPs to the subnet upon launch (enables internet access)
  vpc_id                  = aws_vpc.demo.id  # Associates the subnet with the previously created VPC (id)

  tags = tomap({  # Tags applied to each subnet for organization and identification
    "Name"                                      = "terraform-eks-demo-node",
    "kubernetes.io/cluster/${var.cluster-name}" = "shared",
  })
}

resource "aws_internet_gateway" "demo" {  # Defines an Internet Gateway (IGW) for the VPC
  vpc_id = aws_vpc.demo.id  # Attaches the internet gateway to the VPC (id)

  tags = {  # Tags to help identify the internet gateway
    Name = "terraform-eks-demo"
  }
}

resource "aws_route_table" "demo" {  # Defines an AWS Route Table
```

```
  vpc_id = aws_vpc.demo.id # The route table is associated with the VPC created earlier

  route { # Creates a route within the route table
    cidr_block = "0.0.0.0/0" # Default route for all traffic
    gateway_id = aws_internet_gateway.demo.id # Route traffic to the internet via the internet gateway
  }
}

resource "aws_route_table_association" "demo" { # Associates the created route table with subnets
  count = 2 # Associates the route table with two subnets (subnet1 and subnet2)

  subnet_id      = aws_subnet.demo.*.id[count.index] # Iterates over both subnets
  route_table_id = aws_route_table.demo.id # Associates the subnets with the route table created earlier
}
```

This set of Terraform resources builds the foundational networking setup for an AWS cluster:

1. **`aws_vpc.demo`**: Creates a VPC with a large CIDR block (`10.0.0.0/16`) and applies organizational tags.
2. **`aws_subnet.demo`**: Creates two subnets (`10.0.0.0/24` and `10.0.1.0/24`), each in separate availability zones, with public IPs mapped at launch.
3. **`aws_internet_gateway.demo`**: Attaches an internet gateway to the VPC to allow internet access.
4. **`aws_route_table.demo`**: Creates a route table for the VPC, routing all traffic to the internet via the internet gateway.
5. **`aws_route_table_association.demo`**: Associates the created route table with both subnets to ensure proper internet connectivity.

This configuration ensures that instances deployed within the VPC have reliable connectivity to the internet, enabling communication with other AWS services or external systems.

providers.tf

```
terraform { # The block to define Terraform settings
  required_version = ">= 0.12" # Ensures that the Terraform version used is 0.12 or higher
}

provider "aws" { # Specifies the AWS provider block for interacting with AWS services
  region = var.aws_region # Sets the AWS region using a variable (`var.aws_region`) for flexibility
}

data "aws_availability_zones" "available" {} # Retrieves the list of available AWS availability zones in the chosen region

# Not required: currently used in conjunction with using
# icanhazip.com to determine local workstation external IP
# to open EC2 Security Group access to the Kubernetes cluster.
```

```
# See workstation-external-ip.tf for additional information.
provider "http" {} # Enables the HTTP provider for making HTTP requests
```

This configuration sets up the foundational elements for Terraform to interact with AWS and other services:

1. **Terraform Block**: Ensures the Terraform version is 0.12 or higher, ensuring compatibility with modern features.
2. **AWS Provider**: Configures the AWS provider to operate in a specified region, which is dynamically set using the `var.aws_region` variable.
3. **Data Source (`aws_availability_zones`)**: Retrieves all available AWS availability zones in the specified region, useful for resource placement and high availability.
4. **HTTP Provider**: Although not directly used in this code, it enables the ability to perform HTTP requests, which can be leveraged by other scripts (e.g., to fetch the local IP for dynamic security configurations).

This setup provides the groundwork for deploying infrastructure on AWS while also enabling advanced configurations such as dynamic Security Group rules based on external IP addresses.

variables.tf

```
variable "aws_region" { # Declares a variable named `aws_region`
  default = "us-west-2" # Sets the default value of the AWS region to `us-west-2`
}

variable "cluster-name" { # Declares a variable named `cluster-name`
  default = "terraform-eks-demo" # Sets the default name for the EKS cluster to `terraform-eks-demo`
  type    = string # Explicitly defines the type of the variable as a string
}
```

This code defines two variables to make the Terraform configuration flexible and reusable:

1. **`aws_region`**: Sets the AWS region for the infrastructure deployment. The default region is `us-west-2` (Oregon), but it can be overridden as needed.
2. **`cluster-name`**: Specifies the name of the EKS cluster to be deployed. The default value is `terraform-eks-demo`, but users can customize it.

By using these variables, the Terraform code becomes dynamic, allowing changes to key configurations (region and cluster name) without modifying the main code. This approach promotes better maintainability and reuse of the Terraform scripts.

workstaton-external-ip.tf

```
#
# Workstation External IP
#
# This configuration is not required and is
# only provided as an example to easily fetch
# the external IP of your local workstation to
# configure inbound EC2 Security Group access
```

```
# to the Kubernetes cluster.
#

data "http" "workstation-external-ip" {  # Defines a data source to make an HTTP GET request
  url = "http://ipv4.icanhazip.com"  # The URL returns the external IP address of the caller
}

# Override with variable or hardcoded value if necessary
locals {
  workstation-external-cidr = "${chomp(data.http.workstation-external-ip.body)}/32"
}
```

This code dynamically fetches the external IP address of your local workstation using the public API `http://ipv4.icanhazip.com` and formats it in CIDR notation (`<IP>/32`). The local variable `workstation-external-cidr` can be used elsewhere in your Terraform configuration, such as defining inbound Security Group rules to allow access only from your workstation.

Key points:

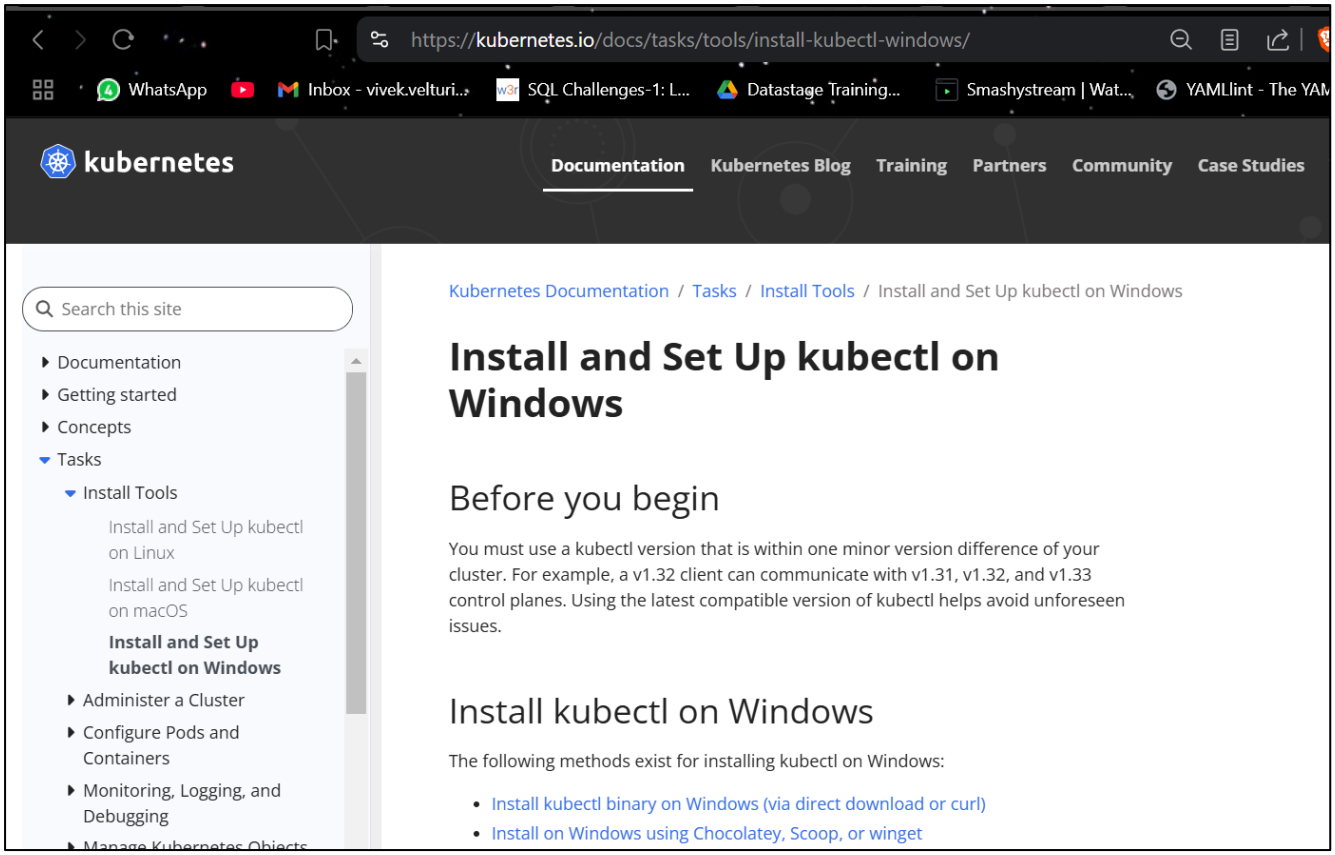
1. **`data.http.workstation-external-ip`**: Makes an HTTP request to fetch the workstation's public IP.
2. **`local.workstation-external-cidr`**: Formats the IP with `/32` to define a network allowing access for only that single IP.

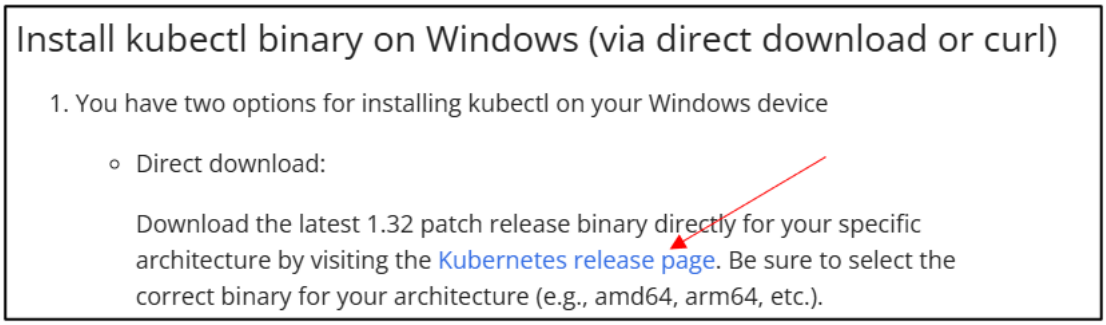
This configuration is optional and useful for securely restricting access to resources from your local machine.

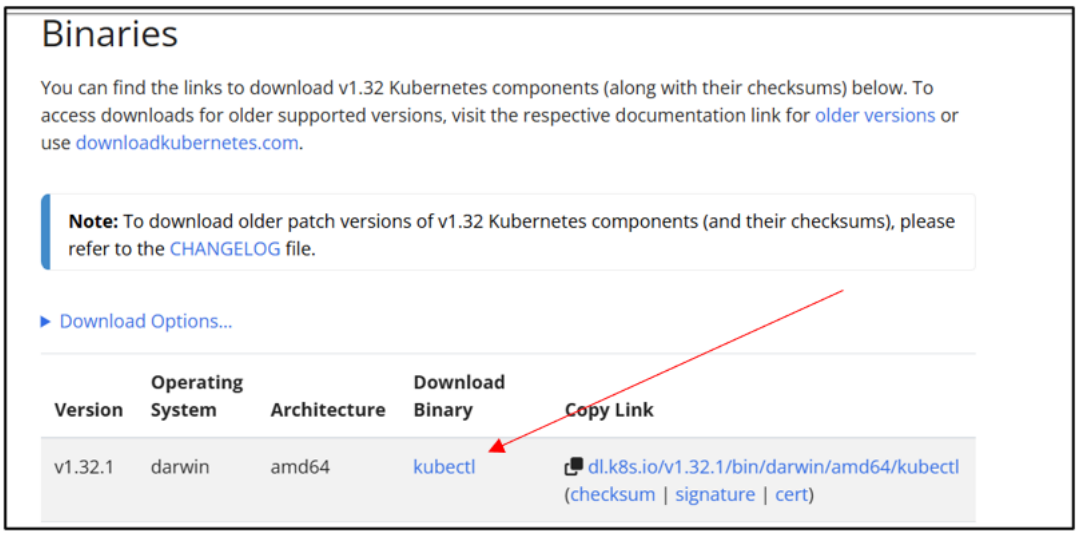
KUBECTL INSTALLATION:

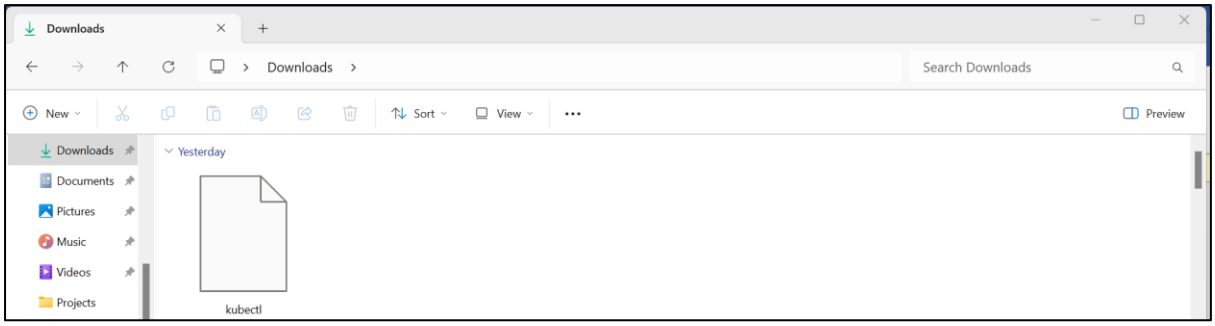
Before we can start working with the EKS cluster created using the Terraform script, we need a tool called kubectl installed on our local computer. Kubectl is a command-line utility that helps us manage and interact with Kubernetes clusters, like the EKS cluster. Since your computer runs on Windows, you'll first need to install kubectl on it. Once installed, you can use it to run commands and communicate with the cluster easily.

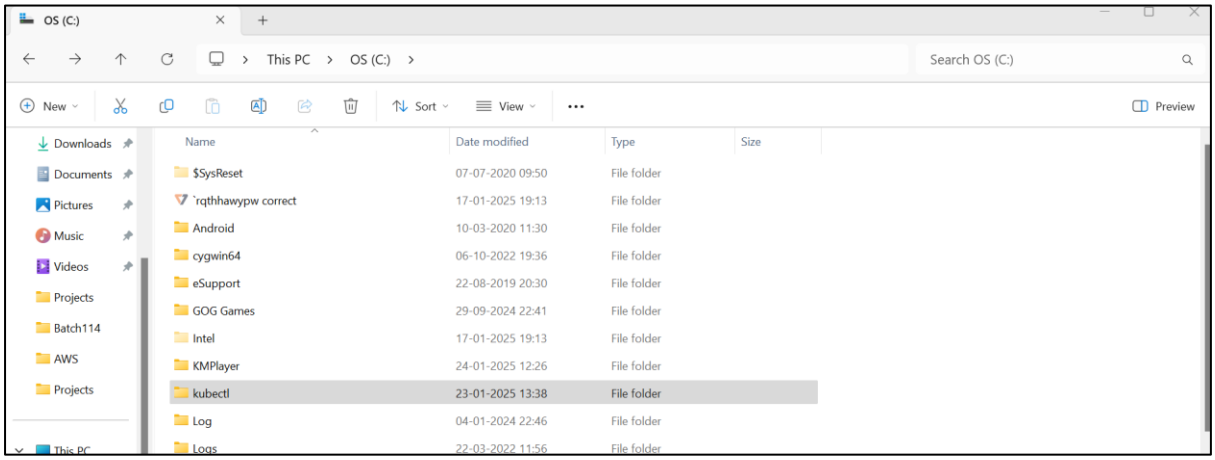
- To install kubectl on a Windows system, visit the official Kubernetes website at: **https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/**
- Under the "Install kubectl on Windows" section, click on "Install kubectl binary on Windows (via direct download or curl)".
- Select the latest version of kubectl for Windows (amd64) and click to download it.
- After downloading the kubectl binary, locate the file in the Downloads folder on your local computer. Copy the file, navigate to the C: Drive, and create a new folder named "kubectl." Paste the copied file into this folder.
- Now go to **Computer Properties**, then click on **Advanced system settings**. In the System Properties window, select the **Environment Variables** button. In the Environment Variables window, find the **Path** variable under System variables, **click Edit, and add C:\kubectl** to the list.
- Now, we have successfully configured kubectl on our local Windows OS computer.

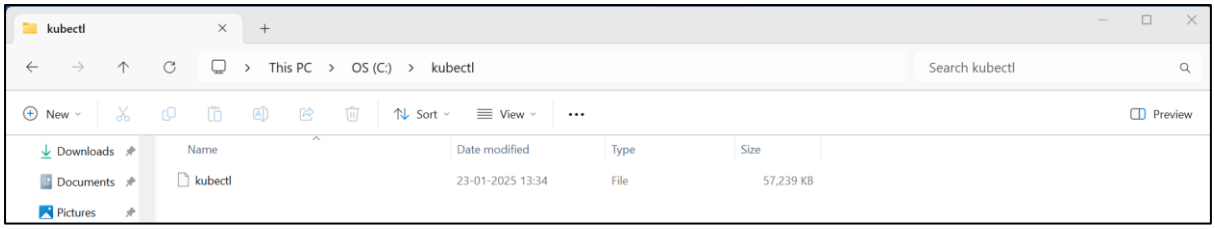

https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/
WhatsApp
Inbox - vivek.velturi...
SQL Challenges-1: L...
Datastage Training...
Smashystream | Wat...
YAMLlint - The YAM
kubernetes
Documentation
Kubernetes Blog
Training
Partners
Community
Case Studies
Search this site
Documentation
Getting started
Concepts
Tasks
Install Tools
Install and Set Up kubectl on Linux
Install and Set Up kubectl on macOS
Install and Set Up kubectl on Windows
Administer a Cluster
Configure Pods and Containers
Monitoring, Logging, and Debugging
Kubernetes Documentation / Tasks / Install Tools / Install and Set Up kubectl on Windows
Install and Set Up kubectl on Windows
Before you begin
You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.32 client can communicate with v1.31, v1.32, and v1.33 control planes. Using the latest compatible version of kubectl helps avoid unforeseen issues.
Install kubectl on Windows
The following methods exist for installing kubectl on Windows:
Install kubectl binary on Windows (via direct download or curl)
Install on Windows using Chocolatey, Scoop, or winget

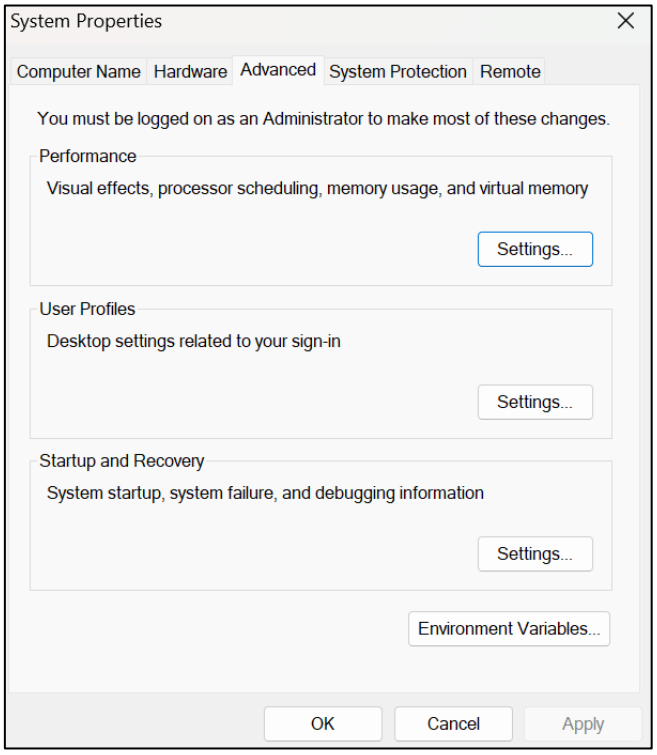

Install kubectl binary on Windows (via direct download or curl)
1. You have two options for installing kubectl on your Windows device
Direct download:
Download the latest 1.32 patch release binary directly for your specific architecture by visiting the Kubernetes release page. Be sure to select the correct binary for your architecture (e.g., amd64, arm64, etc.).

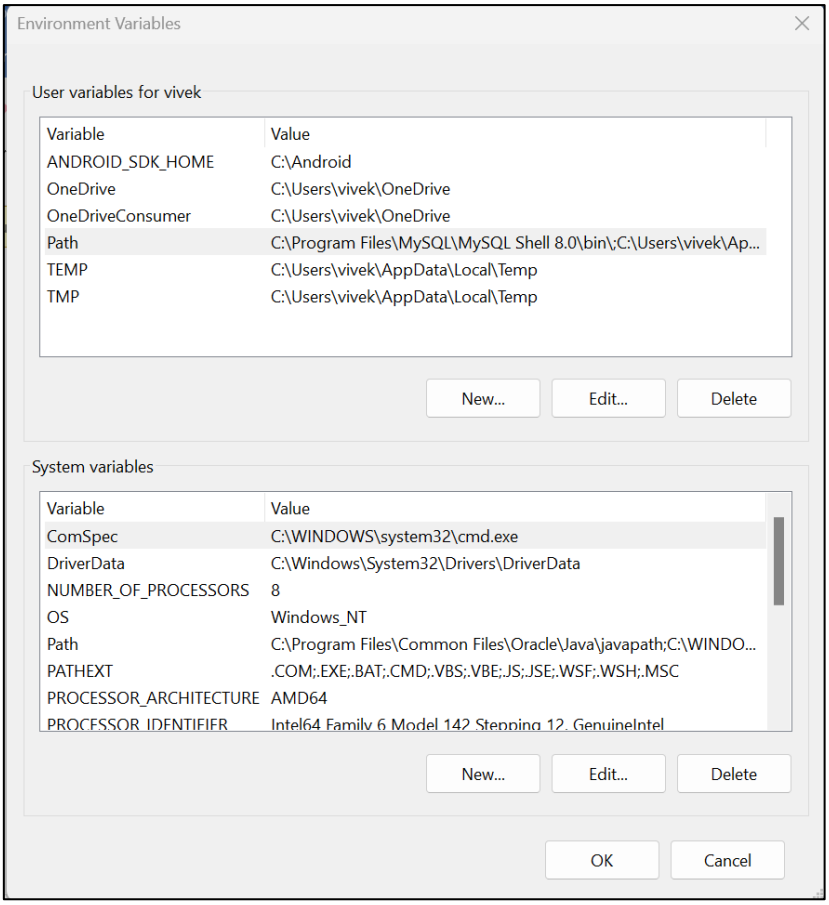

Binaries
You can find the links to download v1.32 Kubernetes components (along with their checksums) below. To access downloads for older supported versions, visit the respective documentation link for older versions or use downloadkubernetes.com.
Note: To download older patch versions of v1.32 Kubernetes components (and their checksums), please refer to the CHANGELOG file.
Download Options...
Version
Operating System
Architecture
Download Binary
Copy Link
v1.32.1
darwin
amd64
kubectl
dl.k8s.io/v1.32.1/bin/darwin/amd64/kubectl
(checksum | signature | cert)

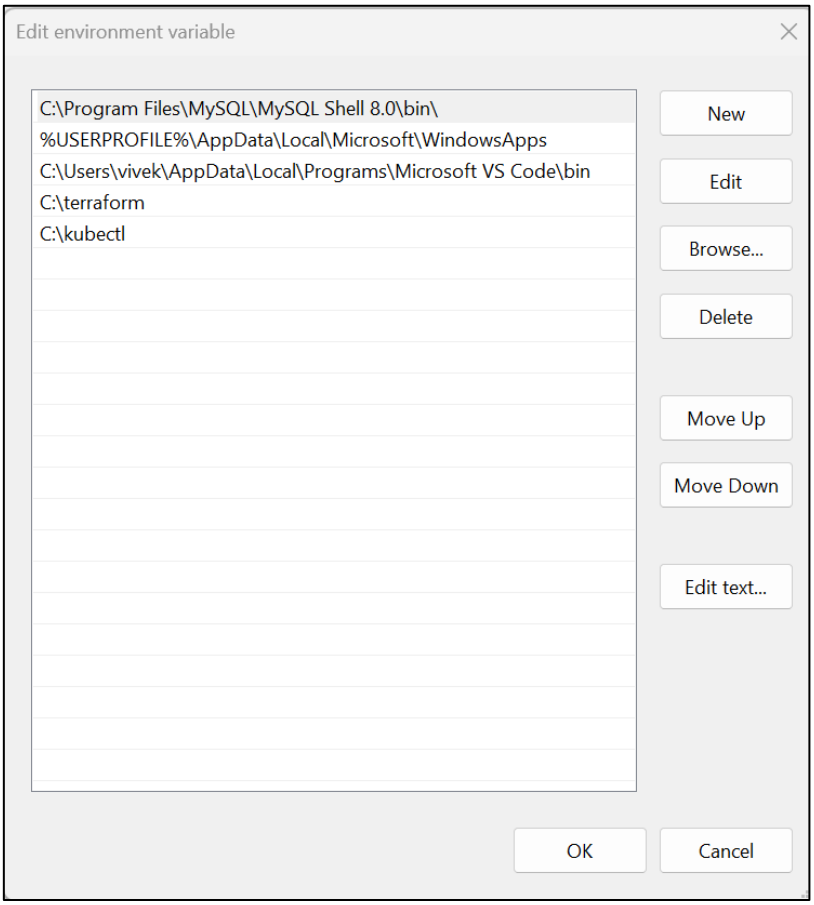

Downloads
Search Downloads
New
Sort
View
Preview
Yesterday
kubectl


OS (C:)
This PC
OS (C:)
Search OS (C:)
Name
Date modified
Type
Size
$SysReset
Android
cygwin64
eSupport
GOG Games
Intel
KMPlayer
kubectl
Log
Logs
Batch114
AWS
Projects


kubectl
This PC
OS (C:)
kubectl
Search kubectl
Name
Date modified
Type
Size
kubectl
23-01-2025 13:34
File
57,239 KB

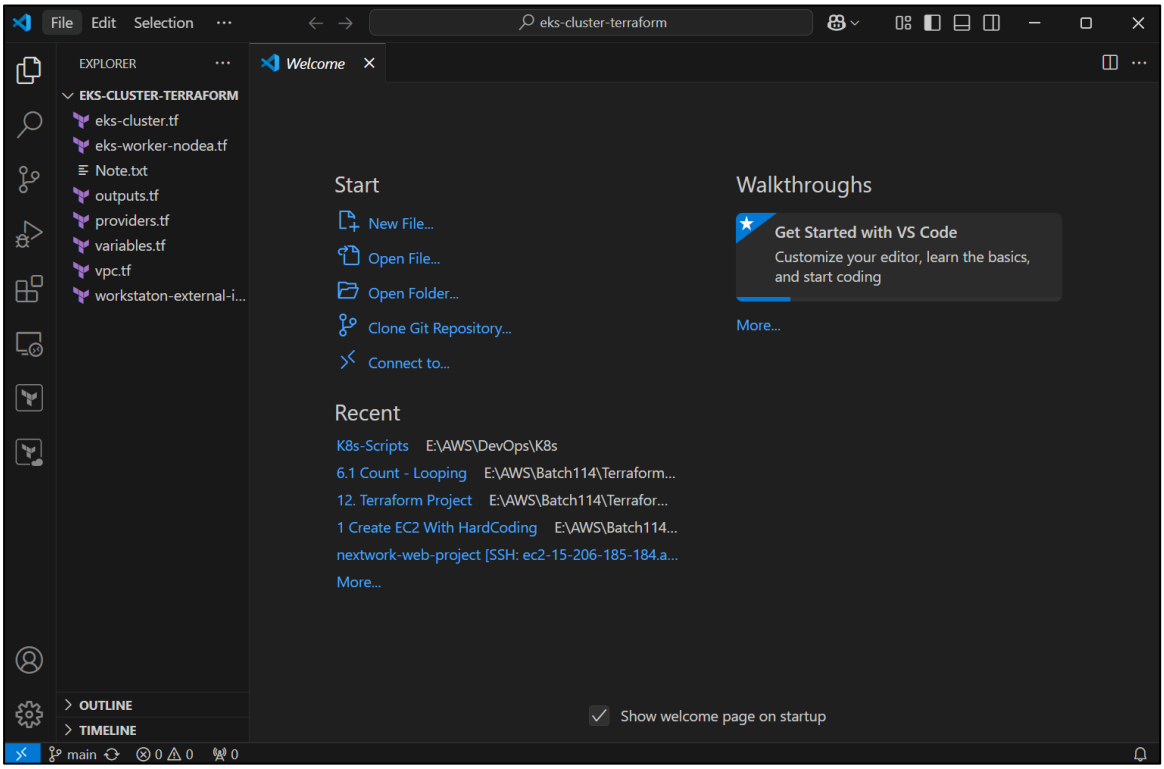

System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply


Environment Variables
User variables for vivek
Variable
Value
ANDROID_SDK_HOME
C:\Android
OneDrive
C:\Users\vivek\OneDrive
OneDriveConsumer
C:\Users\vivek\OneDrive
Path
C:\Program Files\MySQL\MySQL Shell 8.0\bin\;C:\Users\vivek\Ap...
TEMP
C:\Users\vivek\AppData\Local\Temp
TMP
C:\Users\vivek\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ComSpec
C:\WINDOWS\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS
8
OS
Windows_NT
Path
C:\Program Files\Common Files\Oracle\Java\javapath;C:\WINDO...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
New...
Edit...
Delete
OK
Cancel

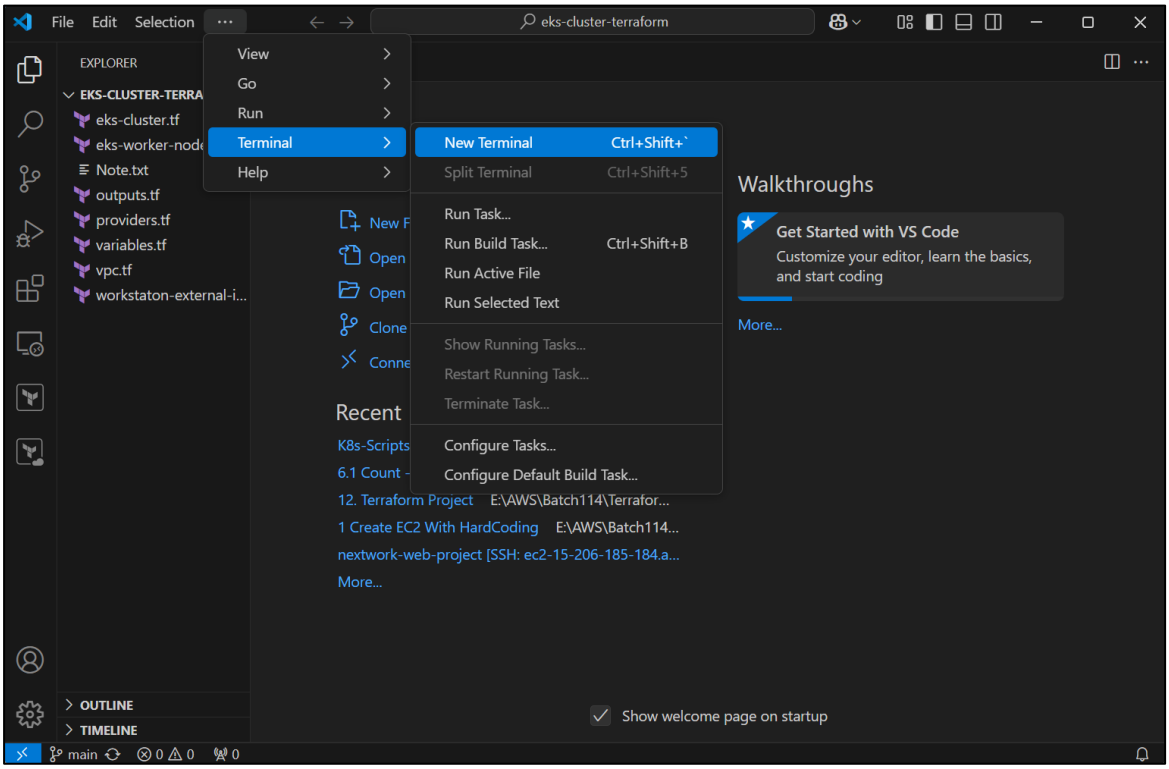

Edit environment variable
C:\Program Files\MySQL\MySQL Shell 8.0\bin\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\vivek\AppData\Local\Programs\Microsoft VS Code\bin
C:\terraform
C:\kubectl
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel

ACCESSING AND RUNNING TERRAFORM SCRIPTS IN VS CODE

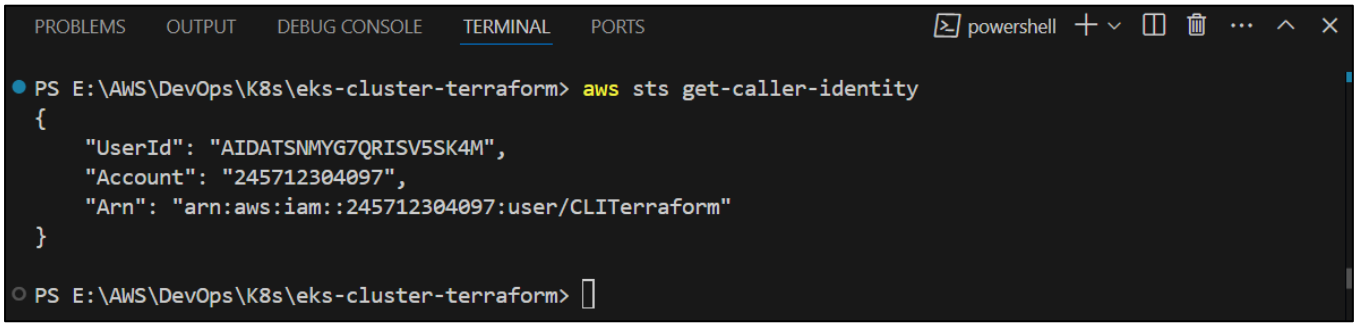
- Now that kubectl is set up on our computer, let's open Visual Studio Code and go to the folder where we cloned the Terraform scripts from GitHub.



- Open a new terminal.



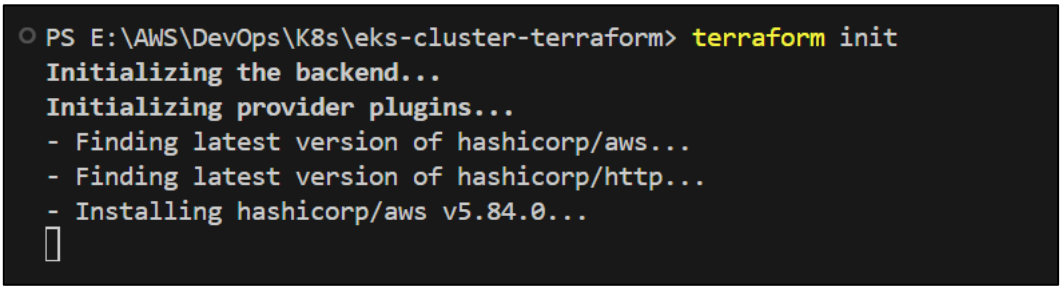
- To check if Visual Studio is connected to our AWS account, run the command: `aws sts get-caller-identity`.

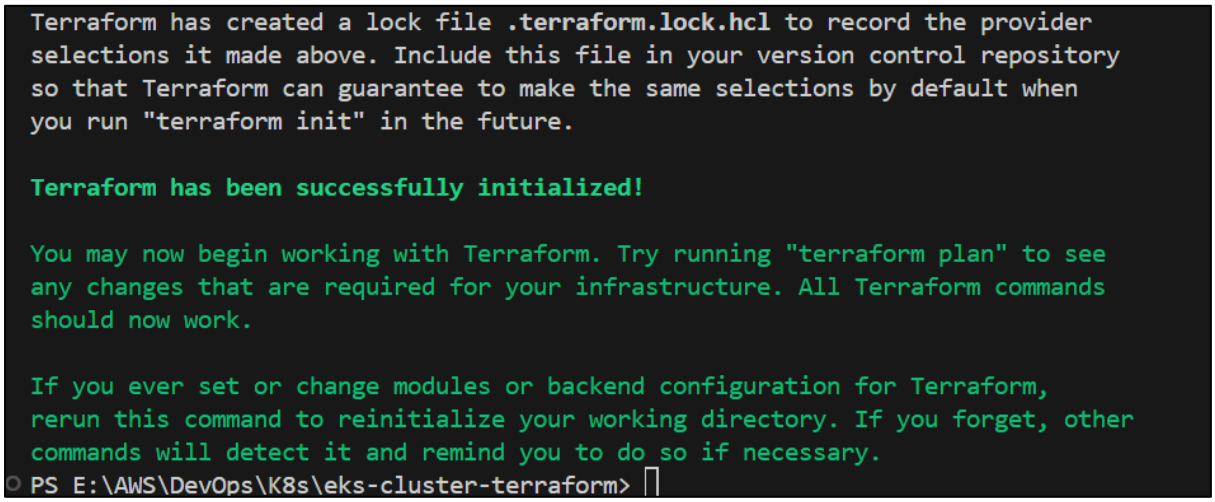


Note: If Visual Studio is not connected to AWS, follow these steps:

Verify AWS CLI Configuration: Ensure the AWS CLI is installed and configured properly. You can configure it by running the command `aws configure` in your command prompt or terminal, and then **enter**ing **your AWS Access Key, Secret Key, region, and output format.**

- Once your AWS is configured, initialize the Terraform script by executing the command: `terraform init`. This will set up the necessary dependencies and prepare the environment for running Terraform.

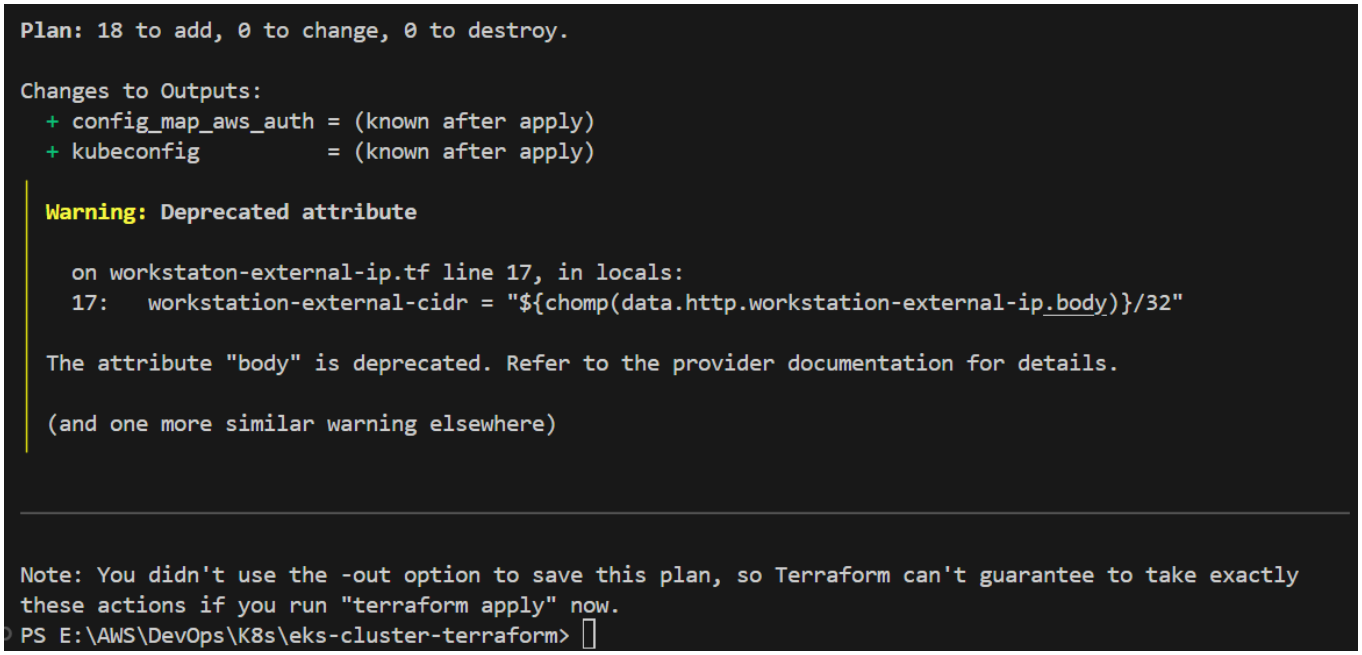




- After running `terraform init`, Terraform creates a lock file named **.terraform.lock.hcl** to record the provider selections it made. This file should be

included in your version control system to ensure consistent provider versions are used in future runs. Terraform initialization is now complete, and you can begin working with your infrastructure. If you modify modules or backend configurations, you should rerun `terraform init` to reinitialize your working directory.

- Now Execute `terraform plan` command. The `terraform plan` command is used to create an execution plan for your infrastructure. It compares the current state of your infrastructure (as defined in your configuration files) with the existing state in your cloud provider (like AWS). The command shows what actions Terraform will take to align the infrastructure with your configuration, such as creating, modifying, or deleting resources. It doesn't make any changes to your infrastructure; it simply provides a preview of what will happen when you run `terraform apply`.



The `terraform plan` output **shows that Terraform will create a total of 18 resources** in your AWS environment, including an **EKS cluster** (`aws_eks_cluster.demo`), **node group** (`aws_eks_node_group.demo`), **IAM roles, security groups, a VPC with subnets, route tables, and an internet gateway.** It also lists 0 resources to be modified or destroyed. The plan outlines the exact resources Terraform will add, along with any associated configurations, such as IAM role policy attachments and subnet associations. There are warnings about deprecated attributes, indicating that some of the configuration elements are outdated and may need to be updated in the future. This plan provides a detailed preview of what will be created when you apply the configuration, ensuring you can review the changes before proceeding.

- Next, we need to use the `terraform apply` command to create the resources outlined in the `terraform plan` output.

```
data.http.workstation-external-ip: Reading...
data.http.workstation-external-ip: Read complete after 0s [id=http://ipv4.icanhazip.com]
data.aws_availability_zones.available: Reading...
data.aws_availability_zones.available: Read complete after 1s [id=us-west-2]

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_eks_cluster.demo will be created
  + resource "aws_eks_cluster" "demo" {
      + 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                          = "terraform-eks-demo"
      + platform_version              = (known after apply)
      + role_arn                      = (known after apply)
```

```
      + role_arn                      = (known after apply)
      + status                        = (known after apply)
      + tags_all                      = (known after apply)
      + version                       = (known after apply)

      + access_config (known after apply)

      + kubernetes_network_config (known after apply)

      + upgrade_policy (known after apply)

      + vpc_config {
          + cluster_security_group_id = (known after apply)
          + endpoint_private_access   = false
          + endpoint_public_access    = true
          + public_access_cidrs       = (known after apply)
          + security_group_ids        = (known after apply)
          + subnet_ids                = (known after apply)
          + vpc_id                    = (known after apply)
        }
    }

  # aws_eks_node_group.demo will be created
```

```
  # aws_eks_node_group.demo will be created
  + resource "aws_eks_node_group" "demo" {
      + ami_type               = (known after apply)
      + arn                    = (known after apply)
      + capacity_type          = (known after apply)
      + cluster_name           = "terraform-eks-demo"
      + disk_size              = (known after apply)
      + id                     = (known after apply)
      + instance_types         = (known after apply)
      + node_group_name        = "demo"
      + node_group_name_prefix = (known after apply)
      + node_role_arn          = (known after apply)
      + release_version        = (known after apply)
      + resources              = (known after apply)
      + status                 = (known after apply)
      + subnet_ids             = (known after apply)
      + tags_all               = (known after apply)
      + version                = (known after apply)

      + node_repair_config (known after apply)

      + scaling_config {
```

```
      + scaling_config {
          + desired_size = 1
          + max_size     = 1
          + min_size     = 1
        }

      + update_config (known after apply)
    }

  # aws_iam_role.demo-cluster will be created
  + resource "aws_iam_role" "demo-cluster" {
      + arn                   = (known after apply)
      + assume_role_policy    = jsonencode(
            {
              + Statement = [
                  + {
                      + Action    = "sts:AssumeRole"
                      + Effect    = "Allow"
                      + Principal = {
                          + Service = "eks.amazonaws.com"
                        }
                    },
                ]
              + Version   = "2012-10-17"
            }
        )
      + create_date           = (known after apply)
      + force_detach_policies = false
      + id                    = (known after apply)
      + managed_policy_arns   = (known after apply)
      + max_session_duration  = 3600
      + name                  = "terraform-eks-demo-cluster"
      + name_prefix           = (known after apply)
      + path                  = "/"
      + tags_all              = (known after apply)
      + unique_id             = (known after apply)

      + inline_policy (known after apply)
    }

  # aws_iam_role.demo-node will be created
  + resource "aws_iam_role" "demo-node" {
      + arn                   = (known after apply)
      + assume_role_policy    = jsonencode(
            {
              + Statement = [
                  + {
                      + Action    = "sts:AssumeRole"
                      + Effect    = "Allow"
                      + Principal = {
                          + Service = "ec2.amazonaws.com"
                        }
                    },
                ]
              + Version   = "2012-10-17"
            }
        )
      + create_date           = (known after apply)
      + force_detach_policies = false
      + id                    = (known after apply)
      + managed_policy_arns   = (known after apply)
      + max_session_duration  = 3600
      + name                  = "terraform-eks-demo-node"
      + name_prefix           = (known after apply)
      + path                  = "/"
      + tags_all              = (known after apply)
      + unique_id             = (known after apply)

      + inline_policy (known after apply)
    }
```

```
  }

# aws_iam_role_policy_attachment.demo-cluster-AmazonEKSClusterPolicy will be created
+ resource "aws_iam_role_policy_attachment" "demo-cluster-AmazonEKSClusterPolicy" {
    + id         = (known after apply)
    + policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
    + role       = "terraform-eks-demo-cluster"
  }

# aws_iam_role_policy_attachment.demo-cluster-AmazonEKSVPCResourceController will be created
+ resource "aws_iam_role_policy_attachment" "demo-cluster-AmazonEKSVPCResourceController" {
    + id         = (known after apply)
    + policy_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
    + role       = "terraform-eks-demo-cluster"
  }

# aws_iam_role_policy_attachment.demo-node-AmazonEC2ContainerRegistryReadOnly will be created
+ resource "aws_iam_role_policy_attachment" "demo-node-AmazonEC2ContainerRegistryReadOnly" {
    + id         = (known after apply)
    + policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
    + role       = "terraform-eks-demo-node"
  }

# aws_iam_role_policy_attachment.demo-node-AmazonEKSWorkerNodePolicy will be created
```

```
  # aws_iam_role_policy_attachment.demo-node-AmazonEKSWorkerNodePolicy will be created
  + resource "aws_iam_role_policy_attachment" "demo-node-AmazonEKSWorkerNodePolicy" {
      + id         = (known after apply)
      + policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
      + role       = "terraform-eks-demo-node"
    }

  # aws_iam_role_policy_attachment.demo-node-AmazonEKS_CNI_Policy will be created
  + resource "aws_iam_role_policy_attachment" "demo-node-AmazonEKS_CNI_Policy" {
      + id         = (known after apply)
      + policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
      + role       = "terraform-eks-demo-node"
    }

  # aws_internet_gateway.demo will be created
  + resource "aws_internet_gateway" "demo" {
      + arn      = (known after apply)
      + id       = (known after apply)
      + owner_id = (known after apply)
      + tags     = {
          + "Name" = "terraform-eks-demo"
        }
      + tags_all = {
          + "Name" = "terraform-eks-demo"
```

```
          + "Name" = "terraform-eks-demo"
        }
      + vpc_id   = (known after apply)
    }

  # aws_route_table.demo will be created
  + resource "aws_route_table" "demo" {
      + arn              = (known after apply)
      + id               = (known after apply)
      + owner_id         = (known after apply)
      + propagating_vgws = (known after apply)
      + route            = [
          + {
              + cidr_block                = "0.0.0.0/0"
              + gateway_id                = (known after apply)
                # (11 unchanged attributes hidden)
            },
        ]
      + tags_all         = (known after apply)
      + vpc_id           = (known after apply)
    }

  # aws_route_table_association.demo[0] will be created
  + resource "aws_route_table_association" "demo" {
```

```
  + resource "aws_route_table_association" "demo" {
      + id             = (known after apply)
      + route_table_id = (known after apply)
      + subnet_id      = (known after apply)
    }

  # aws_route_table_association.demo[1] will be created
  + resource "aws_route_table_association" "demo" {
      + id             = (known after apply)
      + route_table_id = (known after apply)
      + subnet_id      = (known after apply)
    }

  # aws_security_group.demo-cluster will be created
  + resource "aws_security_group" "demo-cluster" {
      + arn                    = (known after apply)
      + description            = "Cluster communication with worker nodes"
      + egress                 = [
          + {
              + cidr_blocks      = [
                  + "0.0.0.0/0",
                ]
              + from_port        = 0
              + ipv6_cidr_blocks = []
```

```
              + ipv6_cidr_blocks = []
              + prefix_list_ids  = []
              + protocol         = "-1"
              + security_groups  = []
              + self             = false
              + to_port          = 0
                # (1 unchanged attribute hidden)
            },
        ]
      + id                     = (known after apply)
      + ingress                = (known after apply)
      + name                   = "terraform-eks-demo-cluster"
      + name_prefix            = (known after apply)
      + owner_id               = (known after apply)
      + revoke_rules_on_delete = false
      + tags                   = {
          + "Name" = "terraform-eks-demo"
        }
      + tags_all               = {
          + "Name" = "terraform-eks-demo"
        }
      + vpc_id                 = (known after apply)
    }
```

```
      + vpc_id                 = (known after apply)
    }

  # aws_security_group_rule.demo-cluster-ingress-workstation-https will be created
  + resource "aws_security_group_rule" "demo-cluster-ingress-workstation-https" {
      + cidr_blocks              = [
          + "49.204.238.212/32",
        ]
      + description              = "Allow workstation to communicate with the cluster API Server"
      + from_port                = 443
      + id                       = (known after apply)
      + protocol                 = "tcp"
      + security_group_id        = (known after apply)
      + security_group_rule_id   = (known after apply)
      + self                     = false
      + source_security_group_id = (known after apply)
      + to_port                  = 443
      + type                     = "ingress"
    }

  # aws_subnet.demo[0] will be created
  + resource "aws_subnet" "demo" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
```

```
    + assign_ipv6_address_on_creation               = false
    + availability_zone                             = "us-west-2a"
    + availability_zone_id                          = (known after apply)
    + cidr_block                                    = "10.0.0.0/24"
    + enable_dns64                                  = false
    + enable_resource_name_dns_a_record_on_launch   = false
    + enable_resource_name_dns_aaaa_record_on_launch = false
    + id                                            = (known after apply)
    + ipv6_cidr_block_association_id                = (known after apply)
    + ipv6_native                                   = false
    + map_public_ip_on_launch                       = true
    + owner_id                                      = (known after apply)
    + private_dns_hostname_type_on_launch           = (known after apply)
    + tags                                          = {
        + "Name"                                    = "terraform-eks-demo-node"
        + "kubernetes.io/cluster/terraform-eks-demo" = "shared"
      }
    + tags_all                                      = {
        + "Name"                                    = "terraform-eks-demo-node"
        + "kubernetes.io/cluster/terraform-eks-demo" = "shared"
      }
    + vpc_id                                        = (known after apply)
  }
```

```
  # aws_subnet.demo[1] will be created
  + resource "aws_subnet" "demo" {
      + arn                                         = (known after apply)
      + assign_ipv6_address_on_creation             = false
      + availability_zone                           = "us-west-2b"
      + availability_zone_id                        = (known after apply)
      + cidr_block                                  = "10.0.1.0/24"
      + enable_dns64                                = false
      + enable_resource_name_dns_a_record_on_launch = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                          = (known after apply)
      + ipv6_cidr_block_association_id              = (known after apply)
      + ipv6_native                                 = false
      + map_public_ip_on_launch                     = true
      + owner_id                                    = (known after apply)
      + private_dns_hostname_type_on_launch         = (known after apply)
      + tags                                        = {
          + "Name"                                  = "terraform-eks-demo-node"
          + "kubernetes.io/cluster/terraform-eks-demo" = "shared"
        }
      + tags_all                                    = {
          + "Name"                                  = "terraform-eks-demo-node"
          + "kubernetes.io/cluster/terraform-eks-demo" = "shared"
```

```
          + "kubernetes.io/cluster/terraform-eks-demo" = "shared"
        }
      + vpc_id                                      = (known after apply)
    }

  # aws_vpc.demo will be created
  + resource "aws_vpc" "demo" {
      + arn                                = (known after apply)
      + cidr_block                         = "10.0.0.0/16"
      + default_network_acl_id             = (known after apply)
      + default_route_table_id             = (known after apply)
      + default_security_group_id          = (known after apply)
      + dhcp_options_id                    = (known after apply)
      + enable_dns_hostnames               = (known after apply)
      + enable_dns_support                 = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                 = (known after apply)
      + instance_tenancy                   = "default"
      + ipv6_association_id                = (known after apply)
      + ipv6_cidr_block                    = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                = (known after apply)
      + owner_id                           = (known after apply)
      + tags                               = {
```

```
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name"                                    = "terraform-eks-demo-node"
          + "kubernetes.io/cluster/terraform-eks-demo" = "shared"
        }
      + tags_all                             = {
          + "Name"                                    = "terraform-eks-demo-node"
          + "kubernetes.io/cluster/terraform-eks-demo" = "shared"
        }
    }

Plan: 18 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + config_map_aws_auth = (known after apply)
  + kubeconfig          = (known after apply)
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

- **Enter yes** to confirm the creation of cluster in your AWS.

```
aws_iam_role.demo-node: Creating...
aws_iam_role.demo-cluster: Creating...
aws_vpc.demo: Creating...
aws_iam_role.demo-cluster: Creation complete after 2s [id=terraform-eks-demo-cluster]
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSVPCResourceController: Creating...
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSClusterPolicy: Creating...
aws_iam_role.demo-node: Creation complete after 2s [id=terraform-eks-demo-node]
aws_iam_role_policy_attachment.demo-node-AmazonEC2ContainerRegistryReadOnly: Creating...
aws_iam_role_policy_attachment.demo-node-AmazonEKS_CNI_Policy: Creating...
aws_iam_role_policy_attachment.demo-node-AmazonEKSWorkerNodePolicy: Creating...
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSClusterPolicy: Creation complete after 1s [id=terraform-eks-demo-cluster-20250124140048
196700000001]
aws_iam_role_policy_attachment.demo-node-AmazonEC2ContainerRegistryReadOnly: Creation complete after 1s [id=terraform-eks-demo-node-20250124
140048507900000003]
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSVPCResourceController: Creation complete after 1s [id=terraform-eks-demo-cluster-202501
241400483330000000002]
aws_iam_role_policy_attachment.demo-node-AmazonEKS_CNI_Policy: Creation complete after 1s [id=terraform-eks-demo-node-20250124140048672200000
0004]
aws_iam_role_policy_attachment.demo-node-AmazonEKSWorkerNodePolicy: Creation complete after 1s [id=terraform-eks-demo-node-20250124140048819
100000005]
aws_vpc.demo: Creation complete after 5s [id=vpc-015116bd437a4b354]
aws_internet_gateway.demo: Creating...
aws_subnet.demo[0]: Creating...
aws_subnet.demo[1]: Creating...
```

```
aws_subnet.demo[1]: Creating...
aws_security_group.demo-cluster: Creating...
aws_internet_gateway.demo: Creation complete after 2s [id=igw-0231b873f9979a7a0]
aws_route_table.demo: Creating...
aws_route_table.demo: Creation complete after 3s [id=rtb-0001f16ac9418e37c]
aws_security_group.demo-cluster: Creation complete after 6s [id=sg-077344b207c384866]
aws_security_group_rule.demo-cluster-ingress-workstation-https: Creating...
aws_security_group_rule.demo-cluster-ingress-workstation-https: Creation complete after 1s [id=sgrule-827502948]
aws_subnet.demo[0]: Still creating... [10s elapsed]
aws_subnet.demo[1]: Still creating... [10s elapsed]
aws_subnet.demo[0]: Creation complete after 13s [id=subnet-00cef28410ec7ae21]
aws_subnet.demo[1]: Creation complete after 13s [id=subnet-02c1b14a21b7b6648]
aws_route_table_association.demo[0]: Creating...
aws_route_table_association.demo[1]: Creating...
aws_eks_cluster.demo: Creating...
aws_route_table_association.demo[1]: Creation complete after 1s [id=rtbassoc-0fbc31687274e8f0a]
aws_route_table_association.demo[0]: Creation complete after 1s [id=rtbassoc-0cd23c50cb05fde9d]
aws_eks_cluster.demo: Still creating... [10s elapsed]
aws_eks_cluster.demo: Still creating... [20s elapsed]
aws_eks_cluster.demo: Still creating... [30s elapsed]
aws_eks_cluster.demo: Still creating... [40s elapsed]
aws_eks_cluster.demo: Still creating... [50s elapsed]
aws_eks_cluster.demo: Still creating... [1m0s elapsed]
```

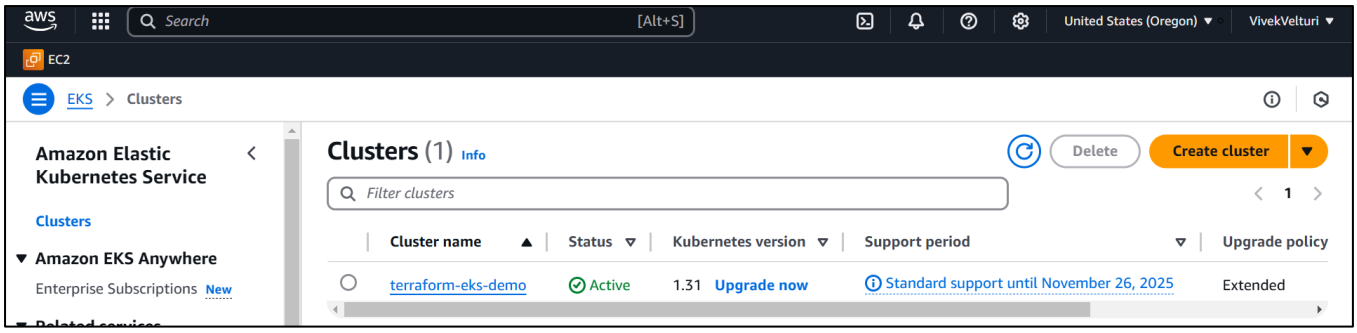
```
aws_eks_cluster.demo: Still creating... [7m50s elapsed]
aws_eks_cluster.demo: Still creating... [8m0s elapsed]
aws_eks_cluster.demo: Still creating... [8m10s elapsed]
aws_eks_cluster.demo: Still creating... [8m20s elapsed]
aws_eks_cluster.demo: Creation complete after 8m30s [id=terraform-eks-demo]
aws_eks_node_group.demo: Creating...
aws_eks_node_group.demo: Still creating... [10s elapsed]
aws_eks_node_group.demo: Still creating... [20s elapsed]
aws_eks_node_group.demo: Still creating... [30s elapsed]
aws_eks_node_group.demo: Still creating... [40s elapsed]
aws_eks_node_group.demo: Still creating... [50s elapsed]
aws_eks_node_group.demo: Still creating... [1m0s elapsed]
aws_eks_node_group.demo: Still creating... [1m10s elapsed]
aws_eks_node_group.demo: Still creating... [1m20s elapsed]
aws_eks_node_group.demo: Still creating... [1m30s elapsed]
aws_eks_node_group.demo: Still creating... [1m40s elapsed]
aws_eks_node_group.demo: Still creating... [1m50s elapsed]
aws_eks_node_group.demo: Creation complete after 1m52s [id=terraform-eks-demo:demo]
```

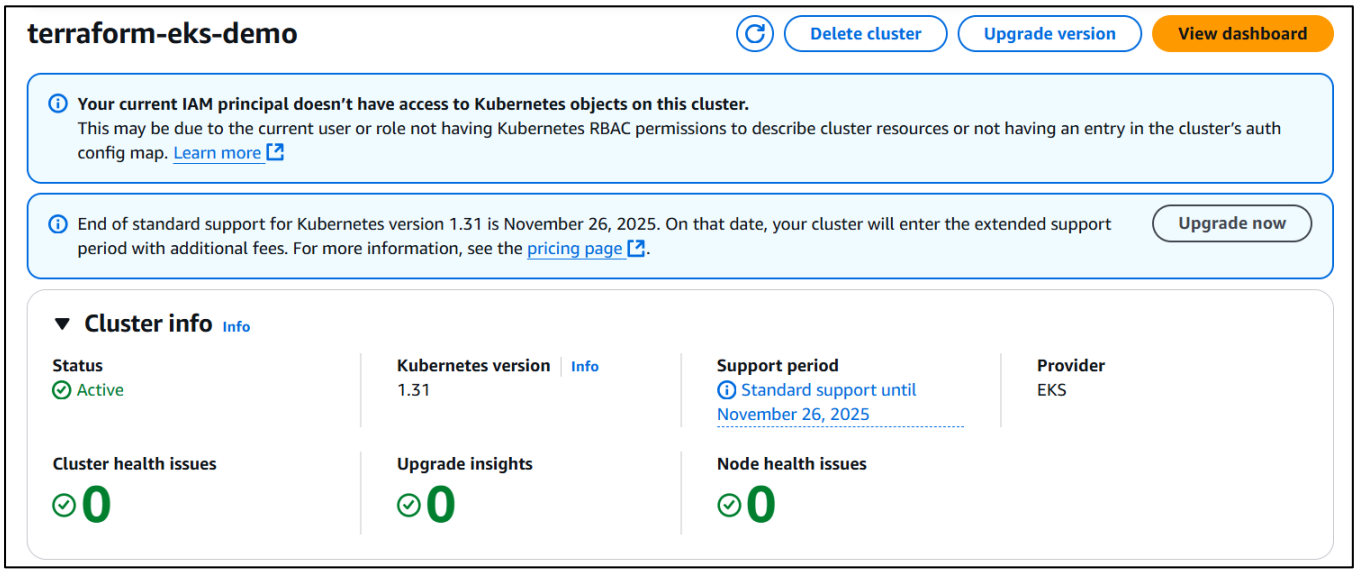
```
Apply complete! Resources: 18 added, 0 changed, 0 destroyed.

Outputs:

config_map_aws_auth = <<EOT


apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: arn:aws:iam::245712304097:role/terraform-eks-demo-node
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes

EOT
kubeconfig = <<EOT
```

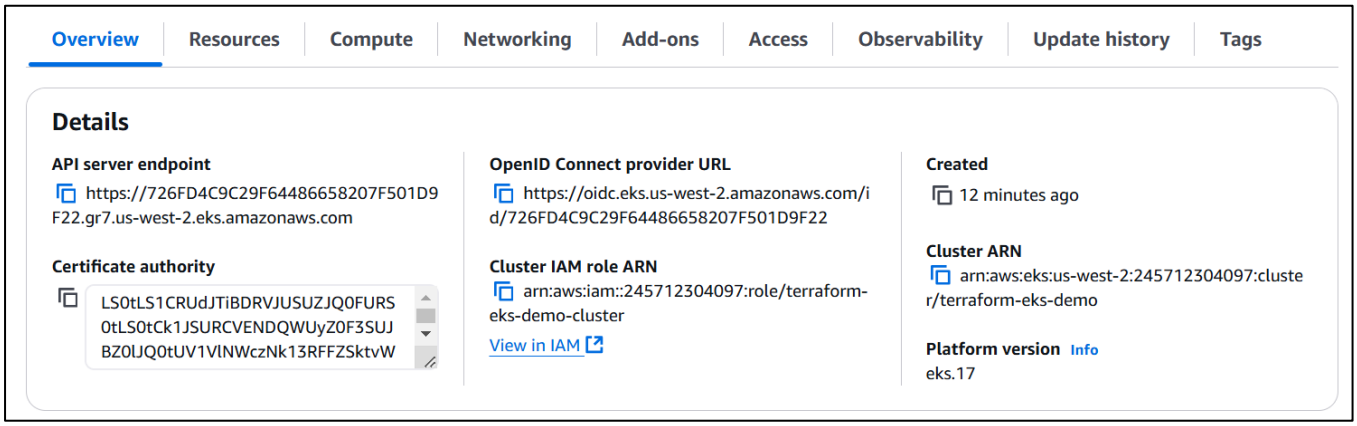
```
apiVersion: v1
clusters:
- cluster:
    server: https://726FD4C9C29F64486658207F501D9F22.gr7.us-west-2.eks.amazonaws.com
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lJQ0tUV1VlNWczNk13RFFZSktvWklodmNOQVFFTEJRQXdG
VEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TlRBeE1qUXhOREF4TURsYUZ3MHpOVEF4TWpJeE5EQTJNRGxhTUJVeApFekFSQmdOVkJBTVRDbXQxWW1WeWJtVjBaWE13
Z2dFaU1BMEdDU3FHU0liM0RRRUJBUVVBQTRJQkR3QXdnZ0VLCkFvSUJBUUROTDh2KzUxcVRPd25uQ0tnaGNQTks0WXI5cFZBbStRdUF4QnJLdS9uclBvWit0QnM4SVBFNkZ3ejUKcHU2
NlpTUDQwcjlCZC9RdlBVSGtVMGFSYnUzN2FIMFc3YStYTis0T0tEbUhHdW9hdHNBTy9qd1M2Z1dmVlhkQgo3WUJhRHpuelVLRkZVK2JrNUJnSmxHV2Q4QUhVTjRvSHZ3NTRIWG9FbFZy
dUliV21yM2dGRXZ1bkJmMW9FbVhHCklOTjFVYmJHSjlJZU0zb2NMd2QrbjYxa0dvSGNjNEFZREx0c2hVemhBZlNzcFhUSTlENHdyMXFOZzNWTEJYcUMKcUtiemhjNEZkYklSVlNsRHdR
WW1nU2xBOFFTY2ZrNTlxVE53SHpyZ2t5dndwWHhDSmNEbzNWdzBmb1lvT0xlcwpXVisrNW83cldMZGx5VmxPV2E0SDM3LzdnVks1QWdNQkFBR2pXVEJYTUE0R0ExVWREd0VCL3dRRUF3
SUNwREFQCkJnTlZIUk1CQWY4RUJUQURBUUgvTUIwR0ExVWREZ1FXQkJTT2trT2Nsb0NqVlh4RHRTTU42Qjgxbm05NEZEQVYKQmdOVkhSRUVEakFNZ2dwcmRXSmxjbTVsZEdWek1BMEdD
U3FHU0liM0RRRUJDd1VBQTRJQkFRQjg5N3J5d0dLUgplYngzZWM4c0FZeU1oWGp4eUhkYU5CeVNqQXFlMXB5VC9ZM1FmU3RjY08reEV2eGt4VEU5NEVCbDlpdERlTi9JCndmeVRVQ21O
TFZWcEJTSmRzWEdHUVFpdFRWakVrVUpVQjBRRGFzRFBjc2xlcDVHa0Y4eDBtTzJjbHFuUXlNcUsKOTJmWWlCOGgvck1DbUdhOXJSQlBNc0YycXBPMFNvbEQwakFlQ1FncEFseWdKZWhp
dTh4cURjOGhOQ2FVTjVLLwp3YXBCYnFKTndKUEkxR1ptT0NDTmE4NTdXcFBZVHdPdjNvZnhscU5CRkpUaHBhdFdFYUtDaEpWdFAvZGxtcXl2ClFINmlOODN5WGgwTWh1Sm9lMEtubDNx
RXhBeE5HYlRMQzM0am1sa1lEUGs4c3NLZy9WWU9ScW1uUXptSnA0WlAKY2FBK2J5cnlUcGRNCi0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
```

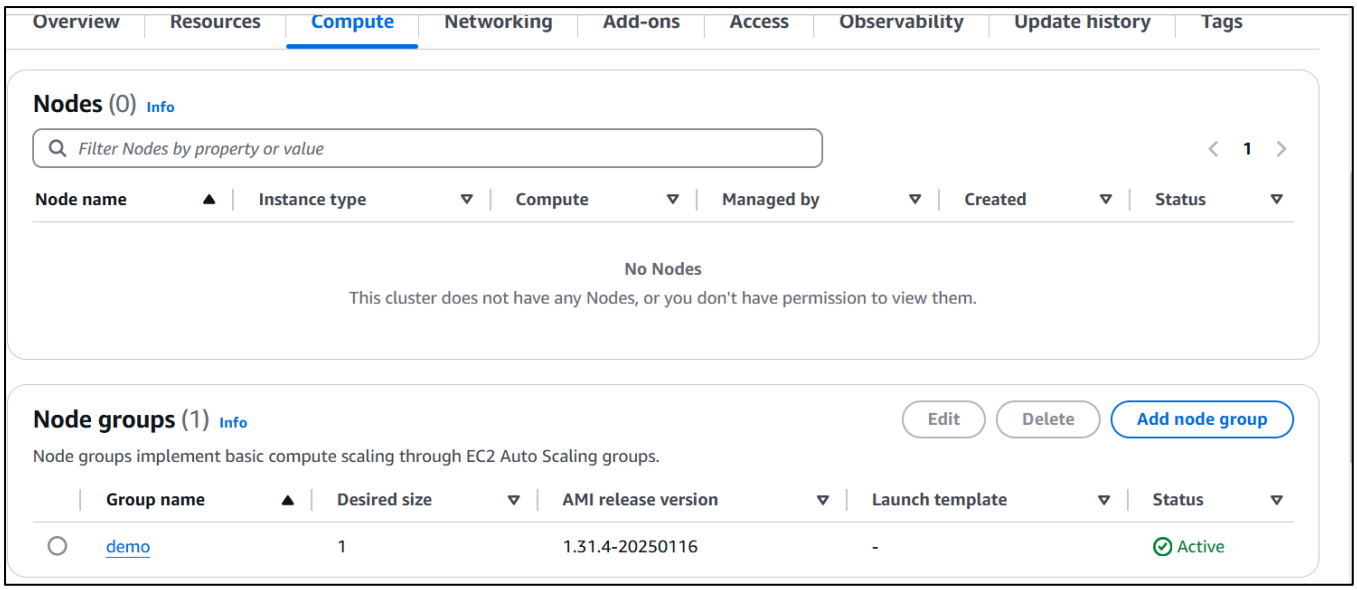
```
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      command: aws-iam-authenticator
      args:
        - "token"
        - "-i"
        - "terraform-eks-demo"

EOT
PS E:\AWS\DevOps\K8s\eks-cluster-terraform>
```

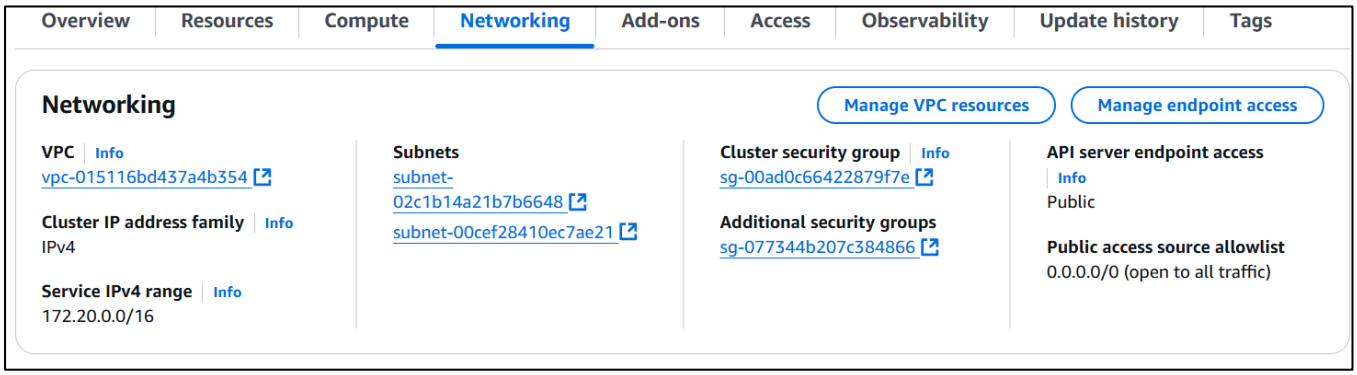
- After running the `terraform apply` command, log in to your AWS Management Console and navigate to the region where the resources were deployed using the Terraform script. Verify that an EKS cluster has been created and check the EC2 dashboard to confirm that an instance has been launched for the node.

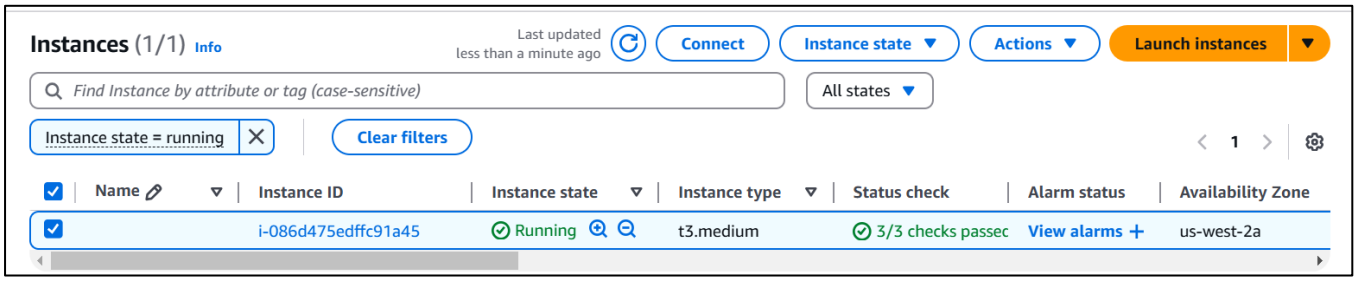


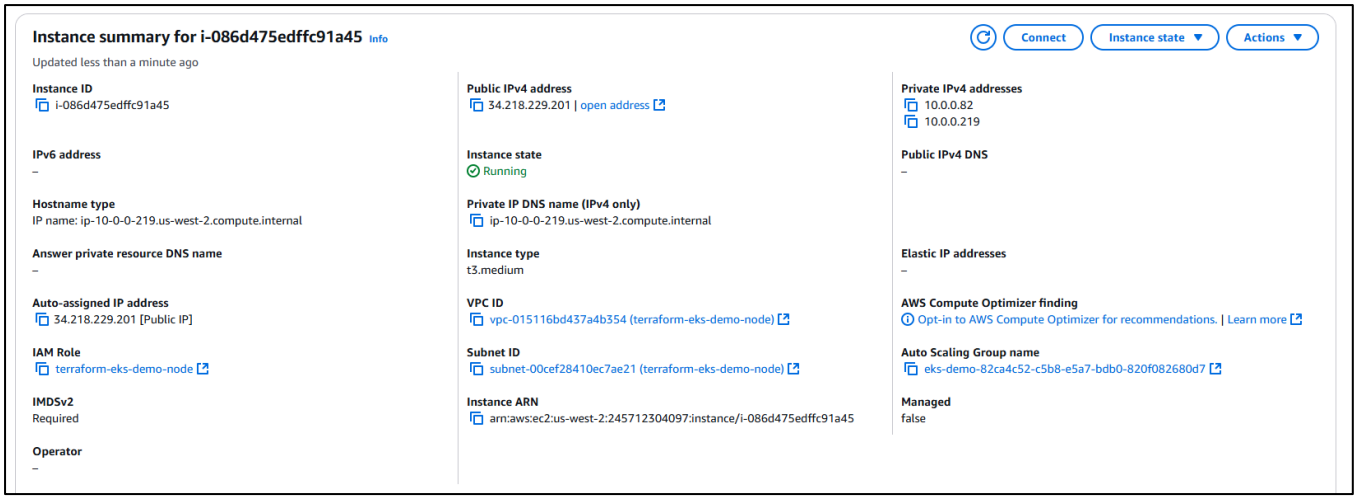


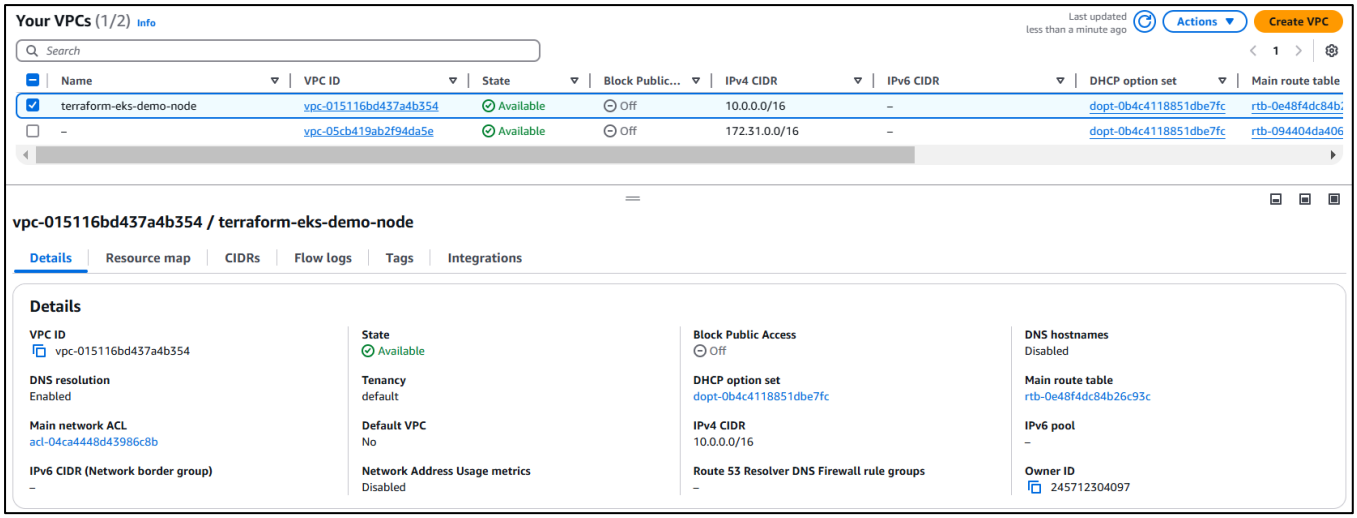


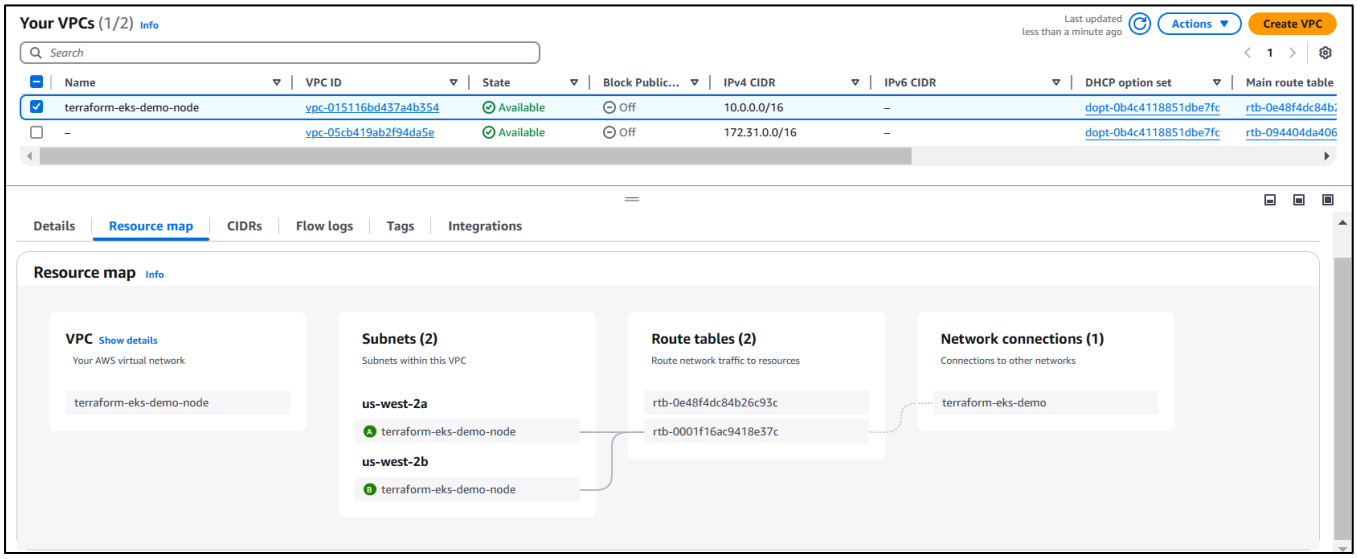

Overview
Resources
Compute
Networking
Add-ons
Access
Observability
Update history
Tags
Nodes (0) Info
Filter Nodes by property or value
1
Node name
Instance type
Compute
Managed by
Created
Status
No Nodes
This cluster does not have any Nodes, or you don't have permission to view them.
Node groups (1) Info
Edit
Delete
Add node group
Node groups implement basic compute scaling through EC2 Auto Scaling groups.
Group name
Desired size
AMI release version
Launch template
Status
demo
1
1.31.4-20250116
-
Active

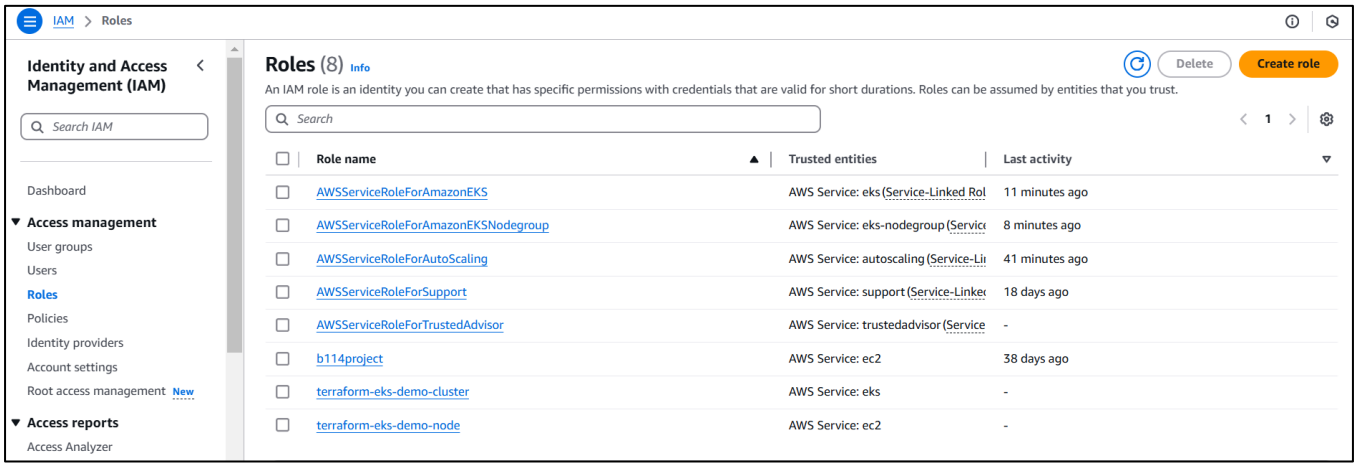

Overview
Resources
Compute
Networking
Add-ons
Access
Observability
Update history
Tags
Networking
Manage VPC resources
Manage endpoint access
VPC Info
vpc-015116bd437a4b354
Cluster IP address family Info
IPv4
Service IPv4 range Info
172.20.0.0/16
Subnets
subnet-02c1b14a21b7b6648
subnet-00cef28410ec7ae21
Cluster security group Info
sg-00ad0c66422879f7e
Additional security groups
sg-077344b207c384866
API server endpoint access Info
Public
Public access source allowlist
0.0.0.0/0 (open to all traffic)


Instances (1/1) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
i-086d475edffc91a45
Running
t3.medium
3/3 checks passec
View alarms
us-west-2a


Instance summary for i-086d475edffc91a45 Info
Updated less than a minute ago
Connect
Instance state
Actions
Instance ID
i-086d475edffc91a45
IPv6 address
-
Hostname type
IP name: ip-10-0-0-219.us-west-2.compute.internal
Answer private resource DNS name
-
Auto-assigned IP address
34.218.229.201 [Public IP]
IAM Role
terraform-eks-demo-node
IMDSv2
Required
Operator
-
Public IPv4 address
34.218.229.201 | open address
Instance state
Running
Private IP DNS name (IPv4 only)
ip-10-0-0-219.us-west-2.compute.internal
Instance type
t3.medium
VPC ID
vpc-015116bd437a4b354 (terraform-eks-demo-node)
Subnet ID
subnet-00cef28410ec7ae21 (terraform-eks-demo-node)
Instance ARN
arn:aws:ec2:us-west-2:245712304097:instance/i-086d475edffc91a45
Private IPv4 addresses
10.0.0.82
10.0.0.219
Public IPv4 DNS
-
Elastic IP addresses
-
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
Auto Scaling Group name
eks-demo-82ca4c52-c5b8-e5a7-bdb0-820f082680d7
Managed
false







- The `kubectl` utility, which we installed earlier on our local computer, serves as the command-line interface to communicate with Kubernetes clusters. Once the EKS cluster has been successfully created in AWS, `kubectl` allows us to manage the cluster and its resources. This includes tasks such as deploying applications, scaling services,

monitoring workloads, and managing configurations. By connecting `kubectl` to the cluster, we can execute commands directly from our local machine to interact with the Kubernetes environment running in AWS.

- To verify the connection between `kubectl` and the EKS cluster, use the following command: `kubectl get pods`

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform> kubectl get pods
E0124 19:48:39.644463    9912 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:808
0/api?timeout=32s\": dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it."
E0124 19:48:39.654812    9912 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:808
0/api?timeout=32s\": dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it."
E0124 19:48:39.661757    9912 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:808
0/api?timeout=32s\": dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it."
E0124 19:48:39.665025    9912 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:808
0/api?timeout=32s\": dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it."
E0124 19:48:39.679772    9912 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:808
0/api?timeout=32s\": dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it."
Unable to connect to the server: dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it.
PS E:\AWS\DevOps\K8s\eks-cluster-terraform>
```

- If you encounter an error such as **`unable to connect to the server`**, it indicates that `kubectl` is not yet configured to interact with the cluster. To establish this connection, use the following command:

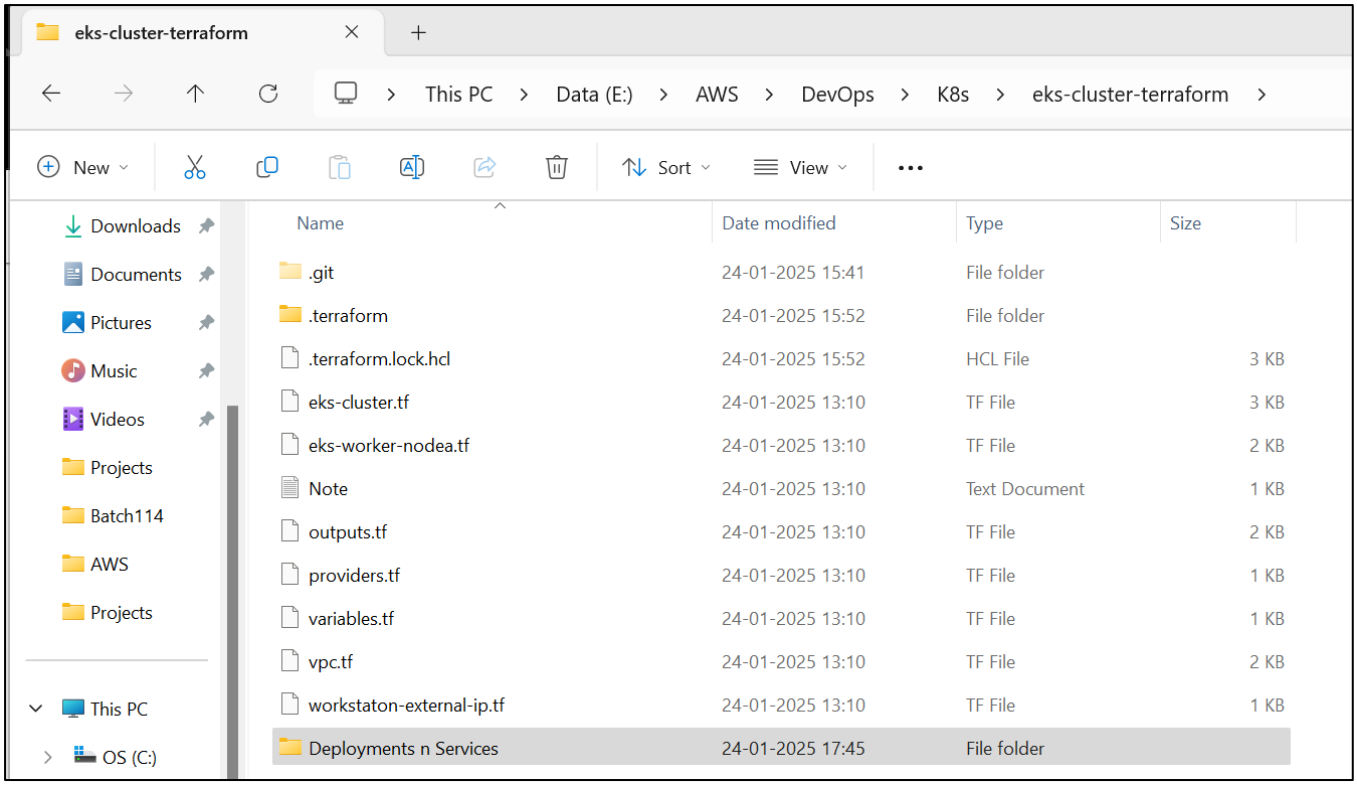
  `aws eks --region <region> update-kubeconfig --name <cluster-name>`

  For example, if your cluster is in the `us-west-2` region and its name is `terraform-eks-demo`, the command would be: `aws eks --region us-west-2 update-kubeconfig --name terraform-eks-demo`

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform> aws eks --region us-west-2 update-kubeconfig --name terraform-eks-demo
Added new context arn:aws:eks:us-west-2:245712304097:cluster/terraform-eks-demo to C:\Users\vivek\.kube\config
PS E:\AWS\DevOps\K8s\eks-cluster-terraform>
```

KUBERNETES DEPLOYMENT CREATION

- Next, let's set up a deployment. Begin by creating a folder named **Deployment n Services** inside the **eks-cluster-terraform** directory, where your Terraform scripts are stored. Inside this folder, create a YAML file called **deployment.yml**. Obtain the deployment script from the official Kubernetes website, paste it into the file, and save it using **Ctrl+S**. This YAML file will define a deployment configuration with a replica set of 3, which will create and manage pods within the cluster.

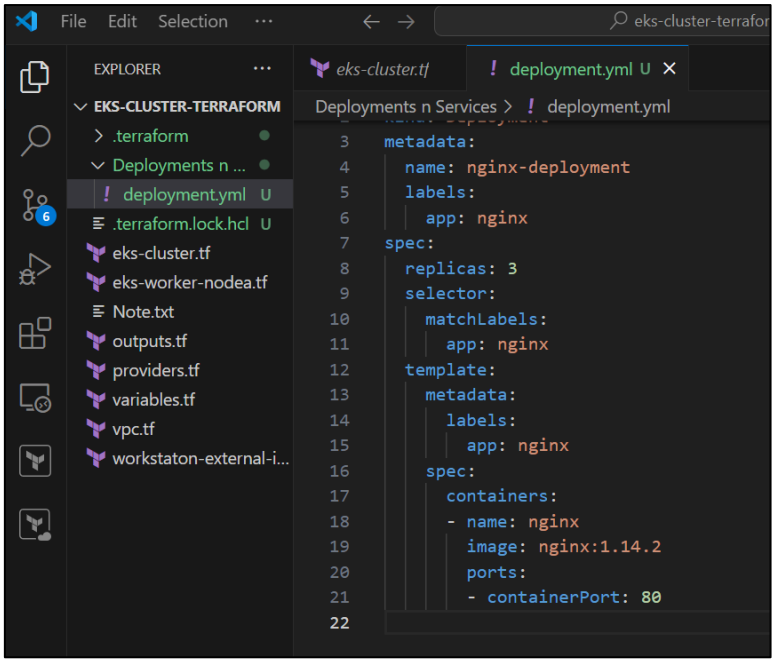


Creating a Deployment

The following is an example of a Deployment. It creates a ReplicaSet to bring up three `nginx` Pods:

controllers/nginx-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```



- After saving the deployment file, run the command `kubectl apply -f deployment.yml` to create the pods. Once the command is executed, you can verify the pods have been created by running `kubectl get pods`.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f deployment.yml
deployment.apps/nginx-deployment created
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-92q4w    1/1     Running   0          30s
nginx-deployment-d556bf558-cbrg8    1/1     Running   0          30s
nginx-deployment-d556bf558-qr2fw    1/1     Running   0          30s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-d556bf558-92q4w   1/1     Running   0          63s
pod/nginx-deployment-d556bf558-cbrg8   1/1     Running   0          63s
pod/nginx-deployment-d556bf558-qr2fw   1/1     Running   0          63s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   172.20.0.1   <none>        443/TCP   30m

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   3/3     3            3           65s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-d556bf558    3         3         3       65s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

- If we delete any pods created through the deployment, new pods will automatically be created to maintain the specified replica set defined in the deployment configuration.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-92q4w    1/1     Running   0          6m31s
nginx-deployment-d556bf558-cbrg8    1/1     Running   0          6m31s
nginx-deployment-d556bf558-qr2fw    1/1     Running   0          6m31s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl delete pod nginx-deployment-d556bf558-92q4w
pod "nginx-deployment-d556bf558-92q4w" deleted
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-cbrg8    1/1     Running   0          6m56s
nginx-deployment-d556bf558-qr2fw    1/1     Running   0          6m56s
nginx-deployment-d556bf558-vkj9v    1/1     Running   0          10s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

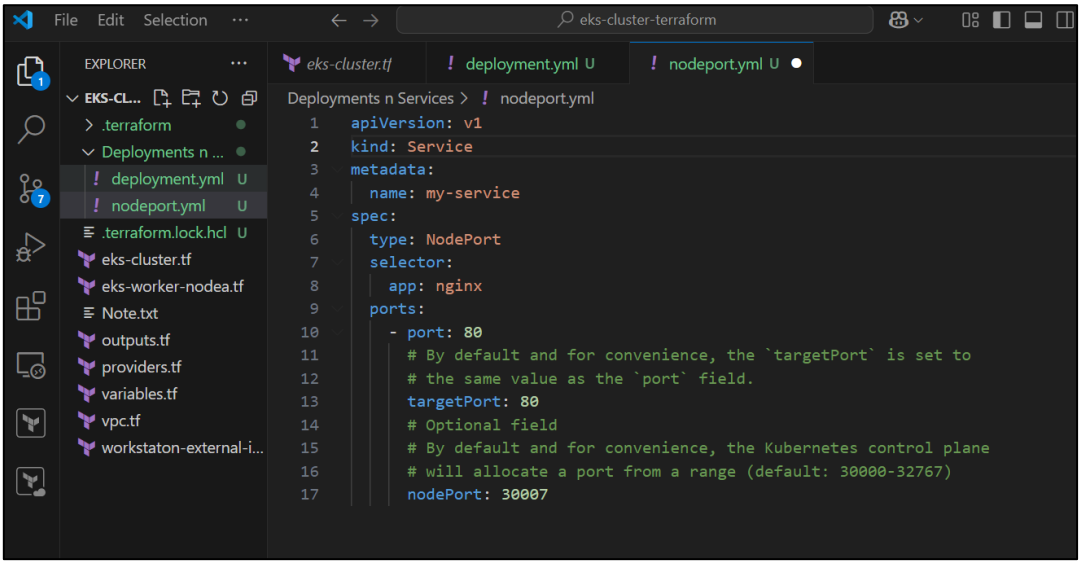
KUBERNETES NODEPORT: EXPOSING APPLICATIONS TO EXTERNAL TRAFFIC

- Next, visit the official Kubernetes website and download the YAML script for creating a NodePort service. Create a new file named **nodeport.yml** inside the **Deployments n Services** folder and paste the downloaded script into it.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - port: 80
      # By default and for convenience, the `targetPort` is set to
      # the same value as the `port` field.
      targetPort: 80
      # Optional field
      # By default and for convenience, the Kubernetes control plane
      # will allocate a port from a range (default: 30000-32767)
      nodePort: 30007
```

- In the NodePort script, you will notice the following selector:
  `selector:`
  `app.kubernetes.io/name: MyApp`
  However, in our deployment, the pods are created with the label `app: nginx`. To ensure the NodePort service targets the pods created by our deployment,

  replace:
  `app.kubernetes.io/name: MyApp`
  with:
  `app: nginx`
  This change will allow the NodePort service to apply correctly to the pods managed by the deployment.



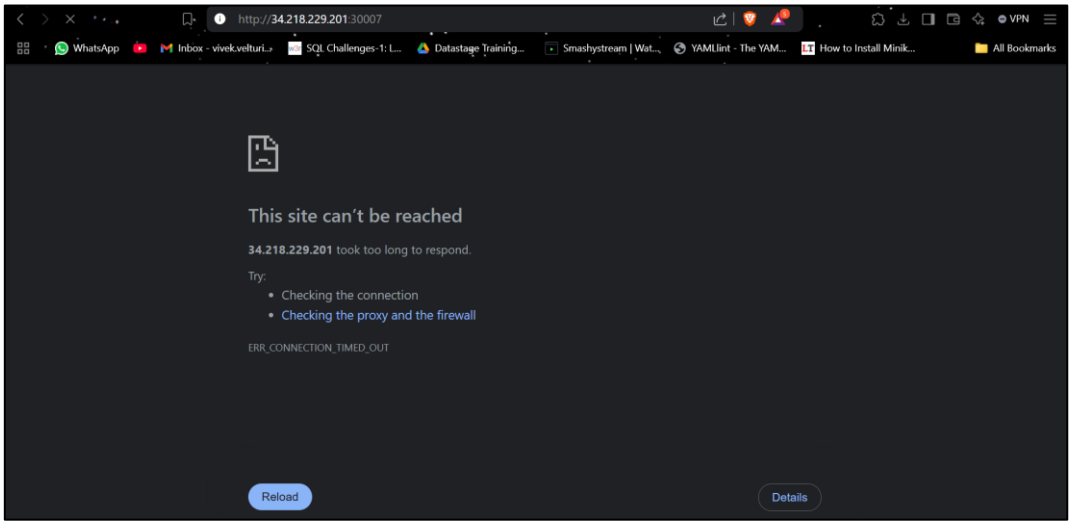
- After saving the nodeport.yml file, execute the command `kubectl apply -f nodeport.yml` to create the NodePort service. Once the service is successfully created, you can verify its status and details by using either `kubectl get svc` to list all services or `kubectl get all` to view all resources in the cluster, including the newly created service. These commands will confirm that the NodePort service has been applied and is ready to route traffic to the pods managed by your deployment.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f nodeport.yml
service/my-service created
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   172.20.0.1      <none>        443/TCP        31m
my-service   NodePort    172.20.23.213   <none>        80:30007/TCP   27s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-d556bf558-92q4w    1/1     Running   0          2m58s
pod/nginx-deployment-d556bf558-cbrg8    1/1     Running   0          2m58s
pod/nginx-deployment-d556bf558-qr2fw    1/1     Running   0          2m58s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP   172.20.0.1      <none>        443/TCP        31m
service/my-service   NodePort    172.20.23.213   <none>        80:30007/TCP   52s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment    3/3     3            3           3m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-d556bf558     3         3         3       3m
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

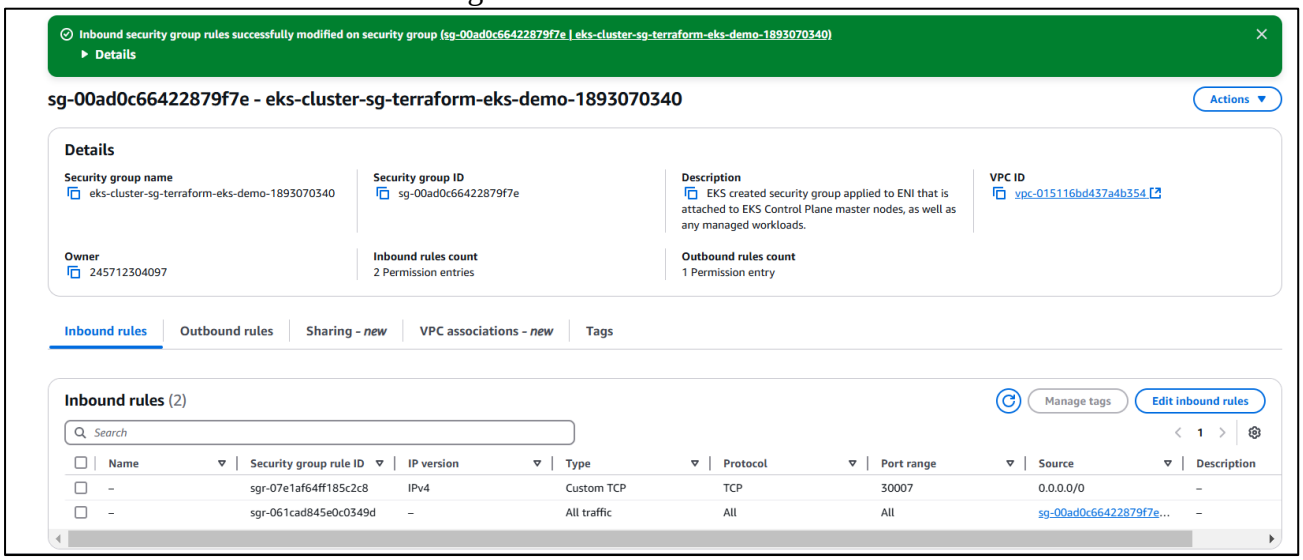
- In the NodePort script, we have specified port 80 for the host and port 30007 for the client. To verify the service, open your web browser and enter the public IP address of your node instance followed by `:30007`. For example, if your node instance's public IP is `34.218.229.201`, you would enter `http://34.218.229.201:30007` in the browser to check if the service is accessible.



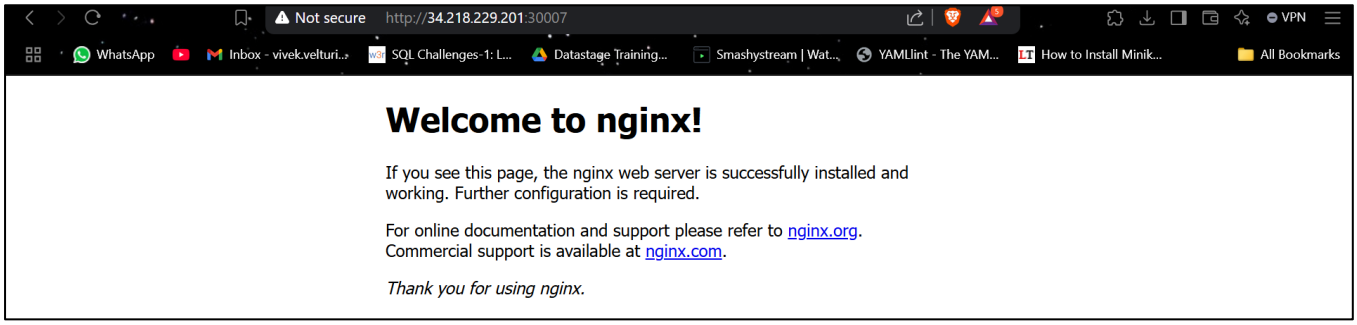
- If the service is not accessible, it's likely because port `30007` has not been added to the security group associated with the node instance.
- Go to the AWS Management Console.
    - Open the **EC2 Dashboard** and locate the node instance created as part of your cluster.
    - **Modify Security Group**:
        - Select the **Security Group** attached to the node instance.
        - Click on **Edit Inbound Rules** to modify the access rules.
        - **Add Inbound Rule**:
            - Add a new rule:
                - **Type**: Custom TCP

- **Port Range**: `30007`
- **Source**: `0.0.0.0/0` (to allow access from any IP address).

❖ Save the changes.

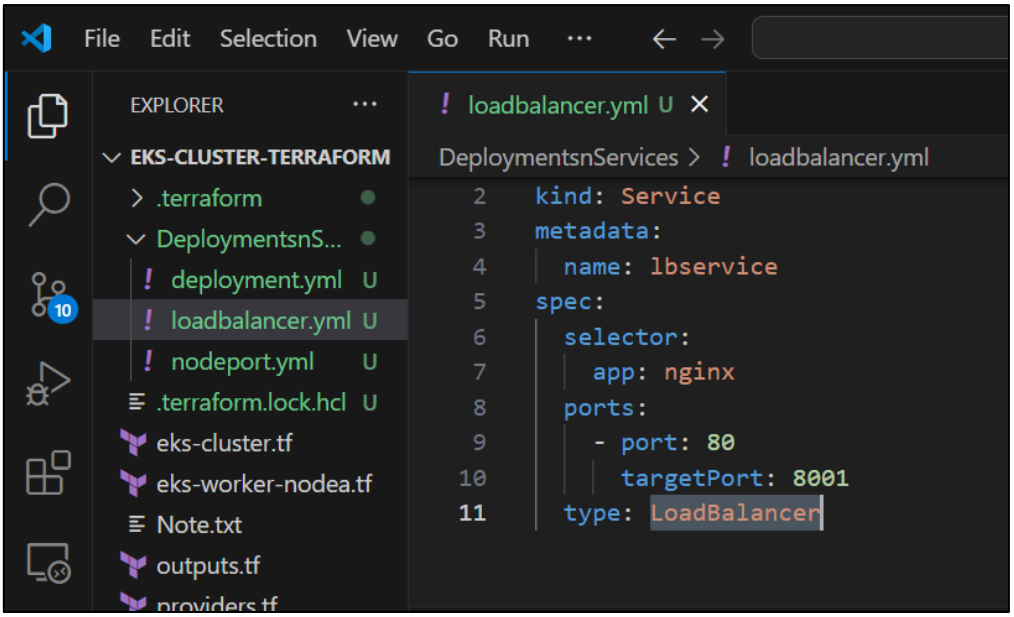


- **Test the Service Again,** Open your browser and enter the public IP address of the node followed by `:30007`.
  For example: `http:// 34.218.229.201:30007`.
- You should now be able to access the NGINX webpage.



KUBERNETES LOADBALANCER: EXPOSING APPS WITH EXTERNAL DNS:

- Next, visit the official Kubernetes website and download the YAML script for creating a Load Balancer service. Create a new file named **loadbalancer.yml** inside the **DeploymentsnServices** folder and paste the downloaded script into it.
- In the LoadBlancer script, make sure your selector has the same label as your pods, In our deployment, the pods are created with the label `app: nginx`. To ensure the loadbalancer service targets the pods created by our deployment,
  Replace the selector data to `app: nginx`
  This change will allow the loadbalancer service to apply correctly to the pods managed by the deployment.



- After saving the **loadbalancer.yml** file, execute the command `kubectl apply -f loadbalancer.yml` to create the LoadBalancer service. Once the service is successfully created, you can verify its status and details by using either `kubectl get svc` to list all services or `kubectl get all` to view all resources in the cluster, including the newly created service. These commands will confirm that the LoadBalancer service has been applied and is ready to route traffic to the pods managed by your deployment.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f loadbalancer.yml
service/lbservice created
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S
)        AGE
kubernetes   ClusterIP      172.20.0.1      <none>                                                                   443/TC
P        7m43s
lbservice    LoadBalancer   172.20.99.234   a7c01417196e54b849099cf7bcaaf437-466989463.us-west-2.elb.amazonaws.com   80:309
72/TCP   35s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

- Once your LoadBalancer service is up and running, it will provide an **external DNS** along with the associated ports (e.g., `a7c01417196e54b849099cf7bcaaf437-466989463.us-west-2.elb.amazonaws.com 80:30972/TCP`). Make sure to **allow access to these ports** in the **security group** of your nodes. After updating the security group rules, you can access the application using the external DNS.

KUBERNETES NAMESPACES: ORGANIZING CLUSTER RESOURCES

- To create and manage namespaces in Kubernetes, you can start by listing all existing namespaces using the command `kubectl get namespaces`. This will display the default namespaces: **default, kube-node-lease, kube-public,** and **kube-system**.
  To explore all resources within each namespace,
  you can use `kubectl get all -n <namespace-name>`.
  For example, to check resources in each default namespace,
  run

`kubectl get all -n default`,
`kubectl get all -n kube-node-lease`,
`kubectl get all -n kube-public`,
`kubectl get all -n kube-system`.
These commands will help you inspect all running resources within each namespace.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get namespaces
NAME              STATUS   AGE
default           Active   13m
kube-node-lease   Active   13m
kube-public       Active   13m
kube-system       Active   13m
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all -n default
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-d556bf558-6tx5z    1/1     Running   0          7m12s
pod/nginx-deployment-d556bf558-1j6p7    1/1     Running   0          7m12s
pod/nginx-deployment-d556bf558-zv9mf    1/1     Running   0          7m12s

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP
  PORT(S)        AGE
service/kubernetes   ClusterIP      172.20.0.1      <none>
  443/TCP        13m
service/lbservice    LoadBalancer   172.20.99.234   a7c01417196e54b849099cf7bcaaf437-466989463.us-west-2.elb.amazonaws.com
  80:30972/TCP   6m32s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   3/3     3            3           7m13s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-d556bf558    3         3         3       7m13s
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all -n kube-node-lease
No resources found in kube-node-lease namespace.
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all -n kube-public
No resources found in kube-public namespace.
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
pod/aws-node-j8717                  2/2     Running   0          11m
pod/coredns-7bb495d866-q2z9s        1/1     Running   0          13m
pod/coredns-7bb495d866-vbl2c        1/1     Running   0          13m
pod/kube-proxy-gqkc4                1/1     Running   0          11m

NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                  AGE
service/eks-extension-metrics-api   ClusterIP   172.20.120.215   <none>        443/TCP                  14m
service/kube-dns                    ClusterIP   172.20.0.10      <none>        53/UDP,53/TCP,9153/TCP   13m

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/aws-node     1         1         1       1            1           <none>          13m
daemonset.apps/kube-proxy   1         1         1       1            1           <none>          13m

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns   2/2     2            2           13m
```

- To create a new namespace in Kubernetes, you can use the `kubectl create namespace` command followed by the desired namespace name. For example, to **create a namespace named vivek**, run `kubectl create namespace vivek`. Once the

namespace is created, you can **verify its existence by** listing all namespaces using `kubectl get namespaces`. **To specifically check resources within the newly created vivek namespace**, use `kubectl get all -n vivek`. This ensures that the namespace has been successfully created and is ready for resource deployment.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl create namespace vivek
namespace/vivek created
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get namespaces
NAME              STATUS   AGE
default           Active   17m
kube-node-lease   Active   17m
kube-public       Active   17m
kube-system       Active   17m
vivek             Active   44s
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all -n vivek
No resources found in vivek namespace.
```

- To create a deployment in the **vivek** namespace, use the same `deployment.yml` file that was previously created. Simply apply the deployment using the command `kubectl apply -f deployment.yml -n vivek`. This will deploy the resources defined in the `deployment.yml` file within the **vivek** namespace. After the deployment is created, you **can verify its status by checking** the resources within the namespace using `kubectl get all -n vivek`. This command will list all the resources, including the newly created deployment, within the **vivek** namespace.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f deployment.yml -n vivek
deployment.apps/nginx-deployment created
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all -n vivek
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-d556bf558-4xmsg   1/1     Running   0          40s
pod/nginx-deployment-d556bf558-dltk2   1/1     Running   0          40s
pod/nginx-deployment-d556bf558-rjtql   1/1     Running   0          40s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   3/3     3            3           42s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-d556bf558    3         3         3       43s
```

- You may notice that the same deployment has been applied in both the **default** and **vivek** namespaces with the same label `app:nginx`. This demonstrates that you can create pods with the same label in different namespaces. However, you cannot create another deployment in the same namespace using the exact same label. If you attempt to deploy again with the same label in the same namespace, you will receive a message stating that no new pods will be created, as the deployment with the same label already exists in that namespace.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f deployment.yml -n vivek
deployment.apps/nginx-deployment unchanged
```

- You can **delete the vivek namespace using** the command `kubectl delete ns vivek`. **After deleting** the namespace, **if you run** the command `kubectl get all -n vivek`, **it will display "no resources found,"** indicating that the namespace and its resources have been successfully removed. **if you run** the command `kubectl get namespaces`, **you will notice that the vivek namespace no longer exists**, as it has been successfully deleted

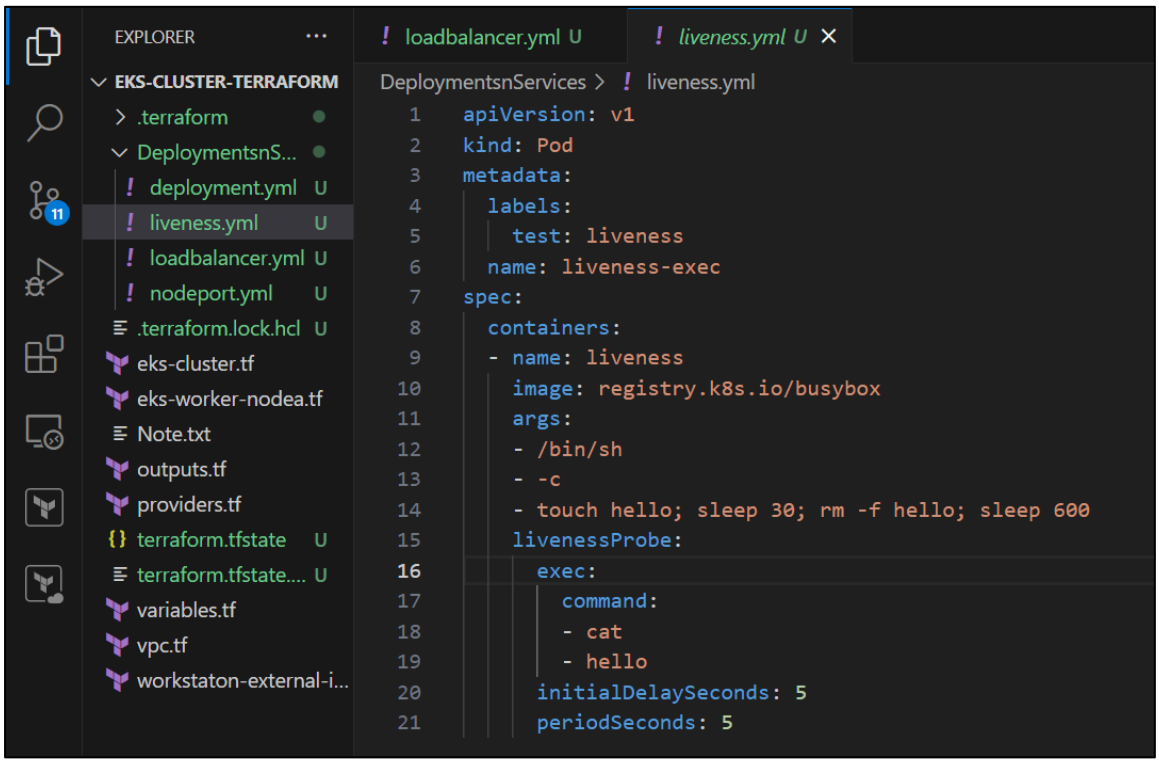
```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl delete ns vivek
namespace "vivek" deleted
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all -n vivek
No resources found in vivek namespace.
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get namespaces
NAME              STATUS   AGE
default           Active   22m
kube-node-lease   Active   22m
kube-public       Active   22m
kube-system       Active   22m
```

KUBERNETES LIVENESS PROBE: CHECKING POD HEALTH AND STABILITY:

- Let's install the liveness probe script. First, obtain the script and paste it into your **DeploymentsnServices** folder. This will make the script available for use within your deployment configuration.



```yaml
apiVersion: v1 # Specifies the API version, here 'v1' is used for Pod.
kind: Pod      # Defines the resource type, in this case, it's a Pod.
metadata:    # Metadata section that contains the name and labels for the Pod.
  labels:      # Labels to help categorize and identify the Pod.
    test: liveness          # The label 'test: liveness' is added to the Pod.
  name: liveness-exec       # The name of the Pod is set to 'liveness-exec'.
spec:                       # The specification of the Pod's desired state.
  containers:         # Defines the containers that should run inside the Pod.
  - name: liveness          # The container is named 'liveness'.
    image: registry.k8s.io/busybox  # Specifies the container image. 'busybox'
is a simple image with Unix utilities.
    args:                 # Arguments passed to the container when it starts.
    - /bin/sh             # The shell to run.
    - -c                  # Executes the following command as a string.
    - touch hello; sleep 30; rm -f hello; sleep 600     #   Creates   a   file
'hello', waits 30 seconds, deletes the file, and sleeps for 600 seconds.
    livenessProbe:  # Defines the liveness probe configuration for the container.
      exec:         # The liveness probe will run a command inside the container.
        command:   # Specifies the command to be executed as part of the liveness
check.
        - cat      # The command will attempt to 'cat' (read) the 'hello' file.
        - hello    # The file to be read by the 'cat' command.
      initialDelaySeconds: 5    # Delay before the first liveness probe runs,
set to 5 seconds.
      periodSeconds: 5          # Interval (in seconds) between each liveness
probe execution, set to 5 seconds.
```

In this configuration, we define a container named **liveness** running the **busybox** image from the Kubernetes registry. The container executes a shell command (`/bin/sh`) that first creates a file named `hello`, waits for 30 seconds, then deletes the file, and finally sleeps for 600 seconds. The purpose of this setup is to simulate a container that creates and deletes a file for health checking purposes. To monitor the container's health, we configure a **liveness probe**. The liveness probe runs a command inside the container that attempts to **cat** (read) the `hello` file. The probe checks the container's health 5 seconds after it starts, as defined by the `initialDelaySeconds` value. After the initial check, it will continue checking the health every 5 seconds, as specified by `periodSeconds`. If the `hello` file is missing when the probe runs, it will fail, signaling that the container is unhealthy.

- Run the command `kubectl apply -f liveness.yml` to apply the configuration. **You will observe the liveness probe in action, as the YAML file contains a shell script that simulates a Pod deadlock. This will allow you to see how the liveness probe functions by repeatedly checking the health of the container and handling any failure conditions. The probe will attempt to read the `hello` file, and since the file is deleted at certain intervals, it will detect the failure, demonstrating how the liveness probe works to ensure the container remains healthy or is restarted if necessary.**

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f liveness.yml
pod/liveness-exec created
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
liveness-exec                       1/1     Running   0          58s
nginx-deployment-d556bf558-6tx5z    1/1     Running   0          18m
nginx-deployment-d556bf558-lj6p7    1/1     Running   0          18m
nginx-deployment-d556bf558-zv9mf    1/1     Running   0          18m
```

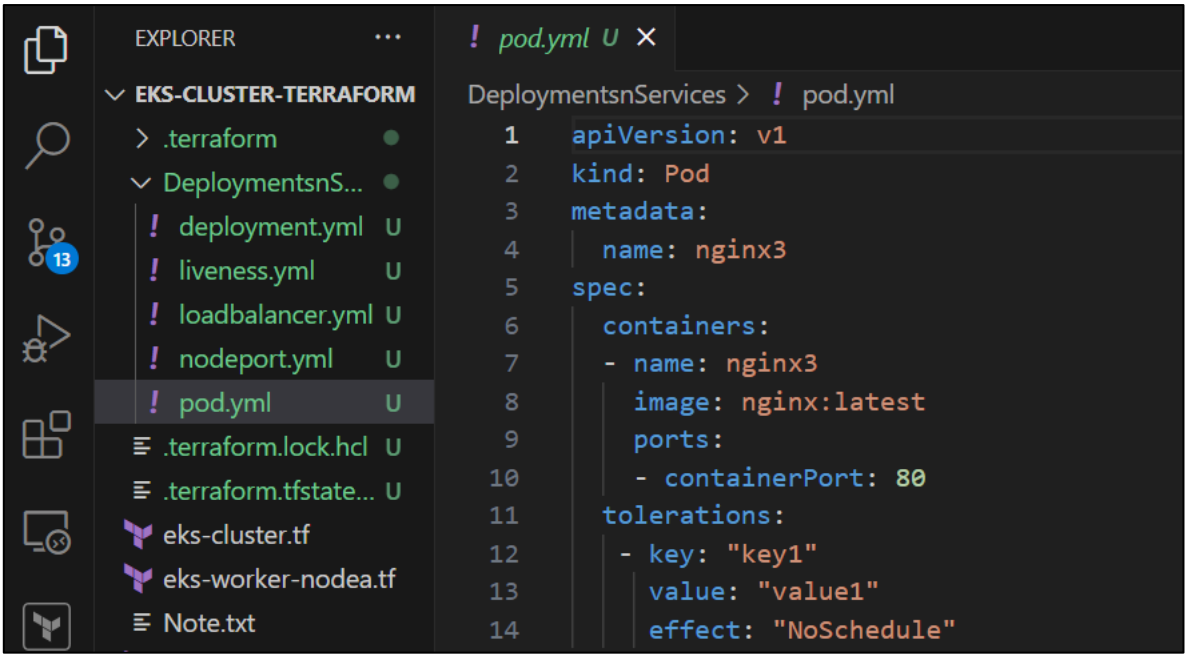
```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get pods -w
NAME                                READY   STATUS    RESTARTS     AGE
liveness-exec                       1/1     Running   0            58s
nginx-deployment-d556bf558-6tx5z    1/1     Running   0            18m
nginx-deployment-d556bf558-lj6p7    1/1     Running   0            18m
nginx-deployment-d556bf558-zv9mf    1/1     Running   0            18m
liveness-exec                       1/1     Running   1 (2s ago)   77s
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get pods -w
NAME                                READY   STATUS    RESTARTS     AGE
liveness-exec                       1/1     Running   0            58s
nginx-deployment-d556bf558-6tx5z    1/1     Running   0            18m
nginx-deployment-d556bf558-lj6p7    1/1     Running   0            18m
nginx-deployment-d556bf558-zv9mf    1/1     Running   0            18m
liveness-exec                       1/1     Running   1 (2s ago)   77s
liveness-exec                       1/1     Running   2 (1s ago)   2m31s
```

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get pods -w
NAME                                READY   STATUS    RESTARTS     AGE
liveness-exec                       1/1     Running   0            58s
nginx-deployment-d556bf558-6tx5z    1/1     Running   0            18m
nginx-deployment-d556bf558-lj6p7    1/1     Running   0            18m
nginx-deployment-d556bf558-zv9mf    1/1     Running   0            18m
liveness-exec                       1/1     Running   1 (2s ago)   77s
liveness-exec                       1/1     Running   2 (1s ago)   2m31s
liveness-exec                       1/1     Running   3 (1s ago)   3m46s
```

KUBERNETES TAINTS AND TOLERATIONS: MANAGING POD SCHEDULING:

- Let's configure Taints and Tolerations. First, obtain the **pod.yml** script and paste it into your **DeploymentsnServices** folder. This will make the script available for use within your deployment configuration. The script will define a taint on the nodes and tolerations in the pod configuration, allowing the pods to be scheduled on tainted nodes. Once the script is added, you can apply it to observe how Taints and Tolerations affect pod scheduling in your cluster.



- Execute the command `kubectl get nodes` to list all the nodes in your cluster. From the output, you will obtain the names of the nodes, which you can use to apply taints or configure tolerations for pod scheduling on specific nodes. This command will display the nodes along with their status, roles, and other relevant details.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform> kubectl get nodes
NAME                                        STATUS   ROLES    AGE    VERSION
ip-10-0-1-45.us-west-2.compute.internal     Ready    <none>   117s   v1.31.4-eks-aeac579
PS E:\AWS\DevOps\K8s\eks-cluster-terraform>
```

- To apply a taint to a node, execute the command `kubectl taint nodes <nodename> key1=value1:NoSchedule`, **replacing `<nodename>` with the actual name of the node you obtained earlier from the `kubectl get nodes` command**, in my case: `kubectl taint nodes ip-10-0-1-45.us-west-2.compute.internal key1=value1:NoSchedule`. **This will taint the node with the key-value pair `key1=value1` and the effect `NoSchedule`**, which means that no pods will be scheduled on this node unless they have a matching toleration. After applying the taint, the node will become unschedulable for any pods that do not explicitly tolerate the taint, ensuring that only the appropriate pods are scheduled on it.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform> kubectl taint nodes ip-10-0-1-45.us-west-2.compute.internal key1=value1:NoSchedule
node/ip-10-0-1-45.us-west-2.compute.internal tainted
PS E:\AWS\DevOps\K8s\eks-cluster-terraform>
```

- After applying the deployment YAML file using the command `kubectl apply -f deployment.yml`, execute `kubectl get all`. In the **Deployment** section, you will see **0/3** under the "READY" column, which means none of the three pods are running. **This occurs because the nodes have been tainted with the `NoSchedule` effect, preventing the pods from being scheduled on them**. As a result, the pods remain in a **Pending** state, as they are unable to find an appropriate node to run on. This illustrates how taints prevent pods from being scheduled unless they have matching tolerations.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f deployment.yml
deployment.apps/nginx-deployment created
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-d556bf558-k7mc4    0/1     Pending   0          14s
pod/nginx-deployment-d556bf558-kq5w6    0/1     Pending   0          13s
pod/nginx-deployment-d556bf558-msqtk    0/1     Pending   0          13s

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   172.20.0.1    <none>        443/TCP   11m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment    0/3     3            0           18s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-d556bf558     3         3         0       19s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

- Delete the deployment by running the command `kubectl delete deployment <deployment-name>`. You can find the **deployment-name** by executing `kubectl get all`, which will display all resources in the cluster, including the deployment name. Once you have identified the deployment name, replace `<deployment-name>` with the actual name and run the command.in my case : `kubectl delete deployment nginx-deployment`, This will delete the deployment and any associated resources, removing them from the cluster.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl delete deployment nginx-deployment
deployment.apps "nginx-deployment" deleted
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

- Now, apply the **pod.yml** file that we copied earlier, which contains the toleration criteria for the taint applied on the nodes. This YAML file includes the necessary tolerations that enable the pod to be scheduled on the tainted nodes. To do this, run the command `kubectl apply -f pod.yml`. By applying this configuration, the pod will be able to tolerate the taint on the nodes, allowing it to be scheduled and run on the tainted nodes, which was not possible before due to the absence of a matching toleration.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f pod.yml
pod/nginx3 created
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

- After running the command `kubectl get all`, you will observe that the pod created by the **pod.yml** file is now running. This is because the pod includes the necessary toleration criteria that match the taint applied to the node earlier. The toleration allows the pod to bypass the taint and get scheduled on the previously tainted node, ensuring it runs as expected.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all
NAME          READY   STATUS    RESTARTS   AGE
pod/nginx3    1/1     Running   0          56s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   172.20.0.1   <none>        443/TCP   14m
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

- To untaint the node, you can use the same taint command we applied earlier, but with a - at the end. The command would be: `kubectl taint nodes <nodename> key1=value1:NoSchedule-`, where `<nodename>` is the name of the node you previously tainted in my case: `kubectl taint nodes ip-10-0-1-45.us-west-2.compute.internal key1=value1:NoSchedule-`. Running this command will remove the taint from the node, allowing pods to be scheduled on it again without any restrictions. This will effectively restore the node's ability to accept pods, as the taint no longer prevents scheduling.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform> kubectl taint nodes ip-10-0-1-45.us-west-2.compute.internal key1=value1:NoSc
hedule-
node/ip-10-0-1-45.us-west-2.compute.internal untainted
PS E:\AWS\DevOps\K8s\eks-cluster-terraform>
```

- Now, if you run the deployment again as before, the pods will be created successfully this time. This is because we have removed the taint on the node, allowing the pods to be scheduled without any restrictions. Since the node is no longer tainted, Kubernetes will be able to schedule the pods on the node, and they will be created and run as expected.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl apply -f deployment.yml
deployment.apps/nginx-deployment created
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices> kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-d556bf558-hqqq8   1/1     Running   0          8s
pod/nginx-deployment-d556bf558-kpdxd   1/1     Running   0          8s
pod/nginx-deployment-d556bf558-m49dv   1/1     Running   0          8s
pod/nginx3                          1/1     Running   0          3m40s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   172.20.0.1   <none>        443/TCP   17m

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   3/3     3            3           10s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-d556bf558    3         3         3       10s
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices>
```

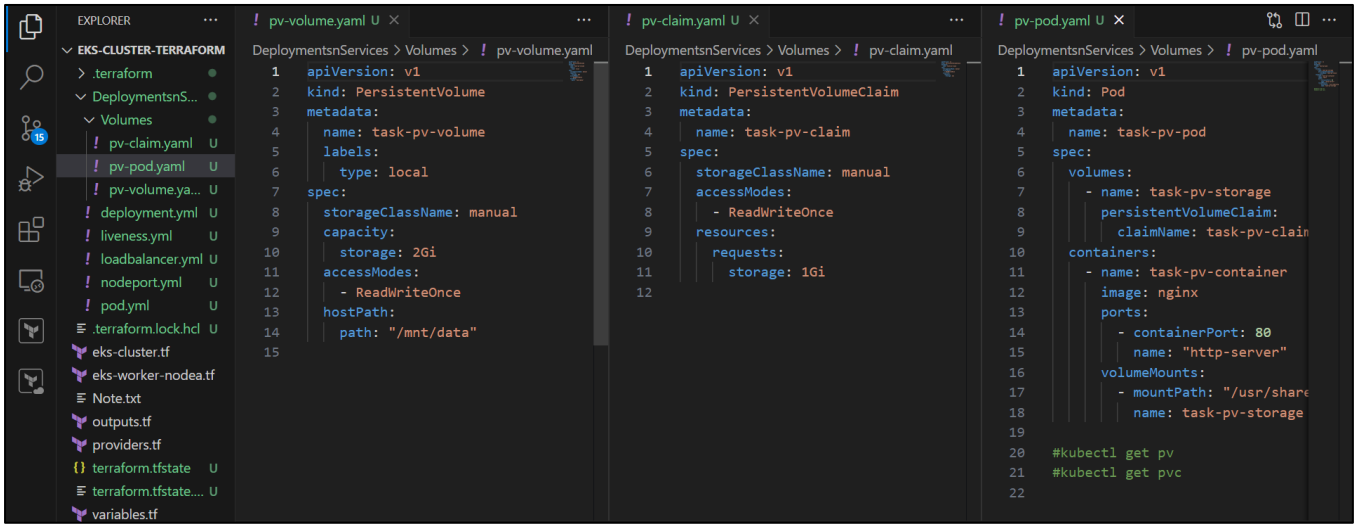
KUBERNETES PV AND PVC: MANAGING PERSISTENT STORAGE:

- Let's configure Persistent Volumes (PV) and Persistent Volume Claims (PVC). First, obtain the **pv-pvc.yml** script and paste it into the **volumes** subfolder inside your **DeploymentsnServices** folder. This will make the script available for use within your deployment configuration. The script will define a Persistent Volume (PV) and a corresponding Persistent Volume Claim (PVC), allowing the pods to access and store data persistently. Once the script is added, you can apply it to observe how PVs and PVCs enable persistent storage in your Kubernetes cluster.

- We have three YAML files: **pv-volume.yml**, **pv-claim.yml**, and **pv-pod.yml**.

    1. **pv-volume.yml**: This file is used to create a Persistent Volume (PV) in the cluster. It defines the storage resource that will be allocated and made available for use.
    2. **pv-claim.yml**: This file is used to create a Persistent Volume Claim (PVC). The PVC allows pods to request storage resources from the Persistent Volumes based on certain criteria like storage size and access modes.
    3. **pv-pod.yml**: This file creates a pod that uses the PVC defined in **pv-claim.yml** and the PV defined in **pv-volume.yml**. The pod will use the PVC to mount the storage, providing persistent data storage.

Place these files in the **volumes** subfolder inside the **DeploymentsnServices** folder to keep them organized and ready for deployment.



pv-volume.yml

```
apiVersion: v1  # Specifies the API version used for the resource. 'v1' is the stable version for core resources like PVs.
kind: PersistentVolume  # Defines the kind of resource being created. Here, it's a PersistentVolume (PV), representing a piece of storage in the cluster.
metadata:
  name: task-pv-volume  # The metadata section contains information about the resource. The PV is named 'task-pv-volume' for reference.
  labels:
```

```
    type: local  # Labels are key-value pairs used to categorize resources.
'type: local' indicates the volume is local storage.
spec:  # The spec section defines the desired state of the resource, detailing
the configuration for the Persistent Volume.
  storageClassName: manual  # Specifies the storage class of the Persistent
Volume. 'manual' means the volume is manually provisioned, not dynamically
created.
  capacity:
    storage: 2Gi  # Defines the storage capacity of the Persistent Volume. This
volume is allocated 2Gi (2 gigabytes) of storage.
  accessModes:
    - ReadWriteOnce  # Specifies that the volume can be mounted as read-write
by a single node at a time.
  hostPath:
    path: "/mnt/data"  # The volume is backed by a directory on the host
machine, located at '/mnt/data'.
```

This YAML file defines a **PersistentVolume** (PV) backed by local storage located at `/mnt/data` on the node. The volume has a storage size of **2Gi**, allows **ReadWriteOnce** access mode (meaning it can only be mounted read-write by one node at a time), and is manually managed under the **manual** storage class. The volume is labeled as **local** to indicate it uses local storage from the node.

pv-claim.yml

```
apiVersion: v1  # Specifies the API version used for the resource. 'v1' is the
stable version for core resources like PVCs.
kind: PersistentVolumeClaim  # Defines the kind of resource being created.
Here, it's a PersistentVolumeClaim (PVC), which allows a pod to request
storage.
metadata:
  name: task-pv-claim  # The metadata section contains information about the
resource. The PVC is named 'task-pv-claim' for reference.
spec:  # The spec section defines the desired state of the resource, detailing
the configuration for the Persistent Volume Claim.
  storageClassName: manual  # Specifies the storage class of the Persistent
Volume Claim. 'manual' means the claim will match a manually provisioned PV.
  accessModes:
    - ReadWriteOnce  # Specifies that the volume can be mounted as read-write
by a single node at a time.
  resources:
    requests:
      storage: 1Gi  # The PVC requests a Persistent Volume with a storage size
of 1Gi (1 gigabyte).
```

This YAML file defines a **PersistentVolumeClaim** (PVC) that requests a **1Gi** volume with **ReadWriteOnce** access mode. The PVC uses the **manual** storage class, meaning it will seek out a manually provisioned PV that matches the specified size and access mode. The claim is named **task-pv-claim** and, when applied, it will trigger Kubernetes to attempt to bind this claim to a matching Persistent Volume (PV).

pv-pod.yml

```yaml
apiVersion: v1  # Specifies the API version used for the resource. 'v1' is the stable version for core resources like Pods.
kind: Pod  # Defines the kind of resource being created. In this case, it's a Pod, which is the smallest deployable unit in Kubernetes.
metadata:
  name: task-pv-pod  # The metadata section contains information about the resource. The Pod is named 'task-pv-pod' for reference.
spec:  # The spec section defines the desired state of the resource, detailing the configuration for the Pod.
  volumes:
    - name: task-pv-storage  # Defines the volume inside the Pod. This volume is named 'task-pv-storage' for reference.
      persistentVolumeClaim:  # Specifies that the volume is backed by a Persistent Volume Claim (PVC).
        claimName: task-pv-claim  # The claimName references the PVC that was defined earlier ('task-pv-claim') to request the volume.
  containers:  # The containers section defines the container(s) that will run inside the Pod.
    - name: task-pv-container  # The container inside the Pod is named 'task-pv-container'.
      image: nginx  # Specifies the container image to use. In this case, it uses the official 'nginx' image.
      ports:
        - containerPort: 80  # Defines the container port that will be exposed. Port 80 is used for HTTP traffic.
          name: "http-server"  # Gives the container port a name ("http-server") for identification.
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"  # Specifies where inside the container the volume should be mounted.
          name: task-pv-storage  # Mounts the 'task-pv-storage' volume, which is linked to the Persistent Volume Claim.
```

This YAML file defines a **Pod** named **task-pv-pod** with a container running the **nginx** image. It mounts a **Persistent Volume** backed by the **task-pv-claim** PVC at the path `/usr/share/nginx/html` inside the container. The container exposes port 80 for HTTP traffic, and the volume provides persistent storage to the container. The volume is linked to the claim `task-pv-claim`, which ensures the Pod can access the storage defined by the Persistent Volume (PV).

- To apply the Persistent Volume (PV), Persistent Volume Claim (PVC), and Pod configurations, first navigate to the **Volumes** subfolder created in that you have created in **DeploymentsnServices**.Once inside the **Volumes** folder, execute the following:
    - Start by applying the **PersistentVolume** (PV) definition using the command: `kubectl apply -f pv-volume.yaml`. This will create the PV with the specified storage and access properties.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl apply -f pv-volume.yaml
persistentvolume/task-pv-volume created
```

- After applying the PV, you can verify its creation by running `kubectl get pv`. The output will display details about the PV, such as its name, capacity, access modes, and status (e.g., **Available**), indicating that it is ready to be claimed.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl get pv
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
task-pv-volume   2Gi        RWO            Retain           Available           manual         <unset>                          21s
```

- Next, apply the **PersistentVolumeClaim** (PVC) by running: `kubectl apply -f pv-claim.yaml`. This will create the PVC that requests the storage defined in the PV, ensuring the proper matching of storage and access modes.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl apply -f pv-claim.yaml
persistentvolumeclaim/task-pv-claim created
```

- Once applied, run `kubectl get pvc` to check the status of the PVC. The output will show that the PVC is **Bound**, meaning it has successfully claimed the appropriate PV.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl get pvc
NAME            STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
task-pv-claim   Bound    task-pv-volume   2Gi        RWO            manual         <unset>                 33s
```

- Finally, apply the **Pod** configuration by executing: `kubectl apply -f pv-pod.yaml`.This creates a Pod that mounts the Persistent Volume via the PVC and runs the nginx container with access to the persistent storage.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl apply -f pv-pod.yaml
pod/task-pv-pod created
```

- After applying the Pod, you can verify its creation by running `kubectl get pods`. The output will show the Pod's status (e.g., **Running**) along with its associated containers. This confirms that the Pod is successfully using the persistent storage. By following these steps and reviewing the outputs, you can ensure that each resource (PV, PVC, and Pod) is created and functioning correctly, with the PVC binding to the PV and the Pod utilizing the persistent storage.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-hqqq8    1/1     Running   0          10m
nginx-deployment-d556bf558-kpdxd    1/1     Running   0          10m
nginx-deployment-d556bf558-m49dv    1/1     Running   0          10m
nginx3                              1/1     Running   0          14m
task-pv-pod                         1/1     Running   0          32s
```

- **`task-pv-pod`** is the pod we created based on the previously applied PV and PVC YAML files.
- To inspect the details of this pod, execute the command `kubectl describe pod task-pv-pod`.

- This command provides comprehensive information about the pod, including its metadata, container configurations, current status, resource usage, and event history.
- Reviewing these details helps ensure that the pod is correctly configured and running as expected, and it allows you to troubleshoot any issues related to the persistent storage setup.

By following this order, the storage resources are first set up, followed by the Pod that utilizes them, ensuring that the Pod has the necessary storage to persist data.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl describe pod task-pv-pod
Name:             task-pv-pod
Namespace:        default
Priority:         0
Service Account:  default
Node:             ip-10-0-1-45.us-west-2.compute.internal/10.0.1.45
Start Time:       Sat, 01 Feb 2025 14:49:45 +0530
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.0.1.82
IPs:
  IP:  10.0.1.82
Containers:
  task-pv-container:
    Container ID:   containerd://274bd66f8704338ce5489f871b330dbbc6bf9df6a7f348808b90daccde2a5d15
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:0a399eb16751829e1af26fea27b20c3ec28d7ab1fb72182879dcae1cca21206a
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 01 Feb 2025 14:49:46 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /usr/share/nginx/html from task-pv-storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-qh869 (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  task-pv-storage:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  task-pv-claim
    ReadOnly:   false
  kube-api-access-qh869:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  116s  default-scheduler  Successfully assigned default/task-pv-pod to ip-10-0-1-45.us-west-2.compute.internal
  Normal  Pulling    116s  kubelet            Pulling image "nginx"
  Normal  Pulled     115s  kubelet            Successfully pulled image "nginx" in 622ms (622ms including waiting). Image size: 72080
558 bytes.
```

```
  Normal  Created    115s  kubelet            Created container task-pv-container
  Normal  Started    115s  kubelet            Started container task-pv-container
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes>
```

- To delete the Persistent Volume (PV), Persistent Volume Claim (PVC), and Pod, you should delete them in reverse order to ensure that dependencies are properly removed.
    - Start by deleting the **Pod** first, using the command `kubectl delete -f pv-pod.yaml`. This command will remove the Pod that was using the Persistent Volume via the PVC.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl delete -f pv-pod.yaml
pod "task-pv-pod" deleted
```

    - After deleting the Pod, you can verify its deletion by running `kubectl get pods`. The output should show no entries for the previously created Pod `task-pv-pod`.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-hqqq8    1/1     Running   0          20m
nginx-deployment-d556bf558-kpdxd    1/1     Running   0          20m
nginx-deployment-d556bf558-m49dv    1/1     Running   0          20m
nginx3                              1/1     Running   0          23m
```

    - Next, delete the **PersistentVolumeClaim** (PVC) using the command `kubectl delete -f pv-claim.yaml`. This will remove the PVC, which was previously bound to the PV.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl delete -f pv-claim.yaml
persistentvolumeclaim "task-pv-claim" deleted
```

    - After deleting the PVC, you can verify its deletion by running `kubectl get pvc`. The output should show no entries for the previously created PVC.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl get pvc
No resources found in default namespace.
```

    - Finally, delete the **PersistentVolume** (PV) using the command `kubectl delete -f pv-volume.yaml`. This will remove the PV.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl delete -f pv-volume.yaml
persistentvolume "task-pv-volume" deleted
```

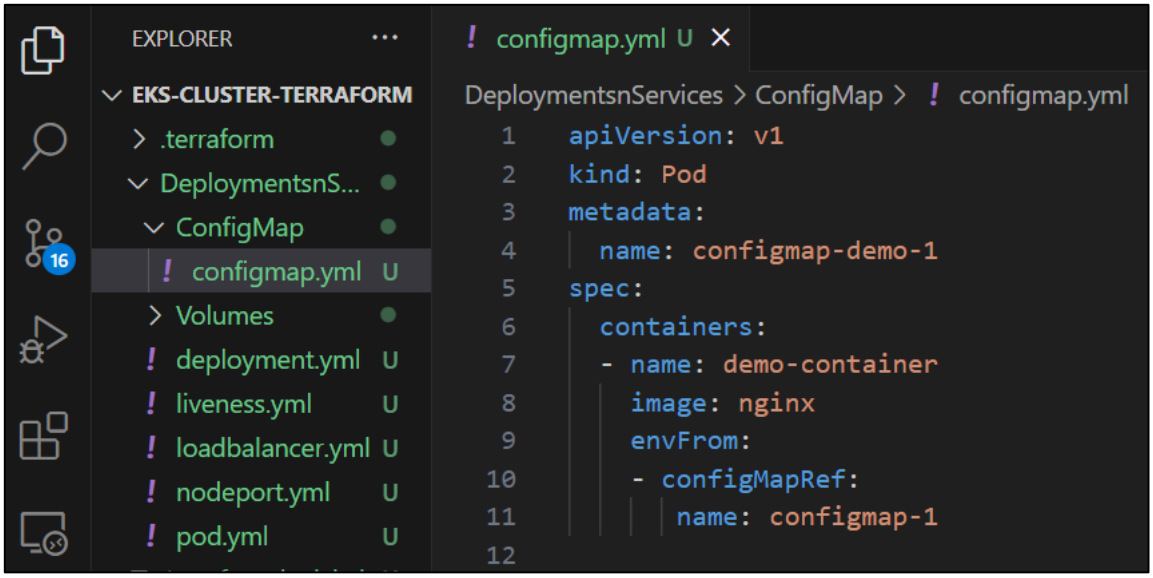
    - After deleting the PV, you can verify its deletion by running `kubectl get pv`. The output should show that the previously created PV is no longer present.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\Volumes> kubectl get pv
No resources found
```

By deleting the resources in this reverse order (Pod → PVC → PV), you ensure that the Pod, which relies on the PVC, is deleted first, followed by the PVC, and finally the PV, which is no longer needed once the PVC is deleted.

KUBERNETES CONFIGMAPS: MANAGING CONFIGURATION DATA:

- Let's configure ConfigMaps. First, obtain the configmap.yml script and paste it into the ConfigMap subfolder inside your **DeploymentsnServices** folder. This will make the script available for use within your deployment configuration. The script will define a ConfigMap, which stores non-sensitive configuration data in key-value pairs that can be used by your pods. Once the script is added, you can apply it to observe how ConfigMaps help manage and share configuration information across multiple pods in your Kubernetes cluster.



configmap.yml

```
apiVersion: v1   # Defines the API version, v1 is commonly used for Pod and other core Kubernetes objects.
kind: Pod    # Specifies the type of resource being defined, which is a Pod in this case.
metadata:    # Metadata about the Pod, such as its name and labels.
  name: configmap-demo-1       # The name of the Pod, in this case 'configmap-demo-1'.
spec:              # The specifications for the desired state of the Pod.
  containers:      # The list of containers that should be run within the Pod.
  - name: demo-container       # The name of the container, here it's named 'demo-container'.
    image: nginx   # Specifies the container image to use for the container, which is 'nginx' in this case.
    envFrom:             # Configures environment variables for the container.
    - configMapRef:      # Refers to a ConfigMap that will provide environment variables.
       name: configmap-1 # The name of the ConfigMap to reference, 'configmap-1', which holds key-value pairs.
```

This YAML file defines a Kubernetes Pod named `configmap-demo-1`, which runs a container with the `nginx` image.
The container, named `demo-container`, is configured to use environment variables sourced from a ConfigMap called `configmap-1`.
The `envFrom` directive with `configMapRef` specifies that the environment variables for this container should be set based on the key-value pairs stored in the `configmap-1` ConfigMap.
This setup allows the container to access configuration data from the ConfigMap as environment variables, making it easier to manage and update configuration without modifying the application code.

- Initially, when you run the command `kubectl get cm`, it will display a list of ConfigMaps that have been created by the Kubernetes system to manage cluster-related configurations. These ConfigMaps are automatically generated by the system during the cluster setup and typically include various settings needed for the proper operation of the cluster.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform> kubectl get cm
NAME                 DATA   AGE
kube-root-ca.crt     1      41m
```

- First, open the **ConfigMap** subfolder in your terminal and execute the command `kubectl get cm`.
  This will display the list of existing ConfigMaps created by the system, such as `kube-root-ca.crt`, `kube-proxy-config`, etc.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap> kubectl get cm
NAME                 DATA   AGE
kube-root-ca.crt     1      43m
```

Then, create a new ConfigMap named `configmap-1` with the command:

`kubectl create configmap configmap-1 --from-literal=name=first-configmap`.
This command creates a ConfigMap with a **key `name` and value `first-configmap`.**

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap> kubectl create configmap configmap-1 --from-literal=name=first-configmap
configmap/configmap-1 created
```

After that, run `kubectl get cm` again to verify the creation of the new ConfigMap. You will see that `configmap-1` has been added to the list of ConfigMaps, alongside the default ones.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap> kubectl get cm
NAME                 DATA   AGE
configmap-1          1      73s
kube-root-ca.crt     1      45m
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap>
```

- Apply the pod script that uses the ConfigMap by running the command `kubectl apply -f configmap.yml`.
  This command creates the pod as defined in the YAML file, which references the ConfigMap to load environment variables into the container.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap> kubectl apply -f configmap.yml
pod/configmap-demo-1 created
```

Next, execute `kubectl get pods` to list all the pods in your cluster.
From the output, note the name of the pod that was created using the ConfigMap.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
configmap-demo-1                    1/1     Running   0          32s
nginx-deployment-d556bf558-hqqq8    1/1     Running   0          29m
nginx-deployment-d556bf558-kpdxd    1/1     Running   0          29m
nginx-deployment-d556bf558-m49dv    1/1     Running   0          29m
nginx3                              1/1     Running   0          33m
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap>
```

We can observe that `configmap-demo-1` is the name of the pod created using the ConfigMap in our case.

This pod name confirms that the configuration specified in the ConfigMap was successfully applied during the pod's creation.

Once you have the pod name, run the command `kubectl exec -it <pod-name> -- print env`, replacing `<pod-name>` with the actual name, to open an interactive shell within the pod and print its environment variables. In my case `kubectl exec -it configmap-demo-1 -- printenv`
This command executes the `printenv` command inside the specified pod. This opens an interactive shell session within the container, where `printenv` then prints all the environment variables currently set in that container. This is useful for verifying that the expected environment variables—such as those injected by ConfigMaps are properly loaded into the pod.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap> kubectl exec -it configmap-demo-1 -- printenv
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=configmap-demo-1
NGINX_VERSION=1.27.3
NJS_VERSION=0.8.7
NJS_RELEASE=1~bookworm
PKG_RELEASE=1~bookworm
DYNPKG_RELEASE=1~bookworm
name=first-configmap
KUBERNETES_PORT_443_TCP=tcp://172.20.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=172.20.0.1
KUBERNETES_SERVICE_HOST=172.20.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://172.20.0.1:443
TERM=xterm
HOME=/root
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap>
```

- Next, run the command `kubectl describe configmap configmap-1` to view detailed information about the ConfigMap named `configmap-1`.

  This command displays metadata associated with the ConfigMap, such as its name, namespace, labels, annotations, and creation timestamp, providing context about when and how the ConfigMap was created.

  It also lists the key-value pairs stored within the ConfigMap, allowing you to verify that the expected configuration data is present.

  By reviewing this detailed output, you can ensure that the ConfigMap has been correctly configured and troubleshoot any discrepancies in the configuration data, which is essential for verifying that your deployment is set up as intended.

```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap> kubectl describe configmap configmap-1
Name:         configmap-1
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
name:
----
first-configmap


BinaryData
====

Events:  <none>
PS E:\AWS\DevOps\K8s\eks-cluster-terraform\DeploymentsnServices\ConfigMap>
```

DESTROYING THE EKS CLUSTER CREATED WITH TERRAFORM:

- To delete the EKS cluster and all associated resources created using Terraform, follow these steps:

1. Open the **terminal** in the same directory where you previously executed the Terraform commands (`terraform init`, `terraform plan`, and `terraform apply`).
2. Run the following command to start the deletion process: `terraform destroy`
3. Terraform will display a list of resources it plans to delete. Carefully review the plan, and when prompted, confirm the deletion by typing `yes`.

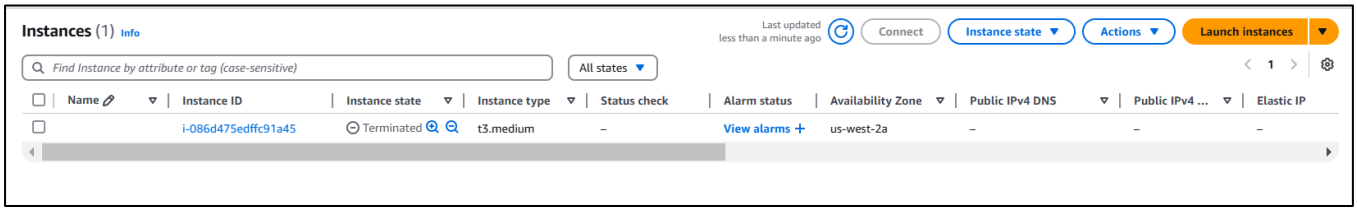
```
PS E:\AWS\DevOps\K8s\eks-cluster-terraform> terraform destroy
data.http.workstation-external-ip: Reading...
```

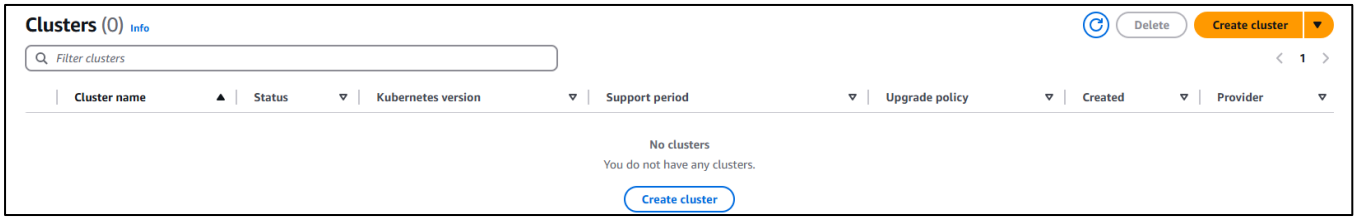
```
        } -> null
        # (4 unchanged attributes hidden)
    }

Plan: 0 to add, 0 to change, 18 to destroy.

Changes to Outputs:
  - config_map_aws_auth = <<-EOT
        apiVersion: v1
        kind: ConfigMap
```

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

```
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSClusterPolicy: Destroying... [id=terraform-eks-demo-cluster-20250124140048196700000001]
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSVPCResourceController: Destroying... [id=terraform-eks-demo-cluster-20250124140048333000
0000002]
aws_subnet.demo[1]: Destroying... [id=subnet-02c1b14a21b7b6648]
aws_subnet.demo[0]: Destroying... [id=subnet-00cef28410ec7ae21]
aws_security_group.demo-cluster: Destroying... [id=sg-077344b207c384866]
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSClusterPolicy: Destruction complete after 1s
aws_iam_role_policy_attachment.demo-cluster-AmazonEKSVPCResourceController: Destruction complete after 1s
aws_iam_role.demo-cluster: Destroying... [id=terraform-eks-demo-cluster]
aws_iam_role.demo-cluster: Destruction complete after 0s
aws_subnet.demo[0]: Destruction complete after 2s
aws_subnet.demo[1]: Destruction complete after 2s
aws_security_group.demo-cluster: Destruction complete after 2s
aws_vpc.demo: Destroying... [id=vpc-015116bd437a4b354]
aws_vpc.demo: Destruction complete after 1s

Destroy complete! Resources: 18 destroyed.
PS E:\AWS\DevOps\K8s\eks-cluster-terraform>
```

4. Wait for the process to complete. Terraform will remove all the resources it managed, including the EKS cluster, EC2 instances, and networking components.
5. After completion, you can verify the deletion by logging into the AWS Management Console and checking the **EKS** and **EC2** sections to ensure no resources remain.





This process ensures that all AWS resources created by your Terraform scripts are properly removed, avoiding unnecessary charges.