Demo Project: Terraform and AWS EKS

Automate provisioning EKS cluster with Terraform

This guide demonstrates how to automate the creation of an AWS EKS cluster using Terraform. It includes setting up a VPC, creating the EKS cluster with managed worker nodes, deploying an Nginx app, and handling cleanup. The EKS cluster and infrastructure are defined using pre-configured Terraform modules.

Step 1: Set Up a New Terraform Project

1. **Create a New Git Branch**

   From the `main` branch, create a new branch: `git checkout -b feature/eks`

2. **Delete `main.tf` (if exists)**

   Remove the existing Terraform file: `rm main.tf`

Step 2: VPC Configuration with Terraform Module

Create a file called `vpc.tf` to define the VPC using a pre-configured Terraform module using as reference: https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/latest

This simplifies the VPC creation process by leveraging the reusable Terraform module, which includes configurations for subnets, NAT gateways, route tables, and more.

2.1 VPC Configuration (`vpc.tf`)

```
provider "aws" {
  region = "us-east-1"
}

variable vpc_cidr_block {}
variable private_subnet_cidr_blocks {}
variable public_subnet_cidr_blocks {}

data "aws_availability_zones" "azs" {}

module "myapp-vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.1.2"

  name = "myapp-vpc"
  cidr = var.vpc_cidr_block
  private_subnets = var.private_subnet_cidr_blocks
  public_subnets = var.public_subnet_cidr_blocks
  azs = data.aws_availability_zones.azs.names

  enable_nat_gateway = true
  single_nat_gateway = true
  enable_dns_hostnames = true

  tags = {
    "kubernetes.io/cluster/myapp-eks-cluster" = "shared"
```

```
  }

  public_subnet_tags = {
    "kubernetes.io/cluster/myapp-eks-cluster" = "shared"
    "kubernetes.io/role/elb" = 1
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/myapp-eks-cluster" = "shared"
    "kubernetes.io/role/internal-elb" = 1
  }
}
```

2.2 Configure `terraform.tfvars`

Create a file named `terraform.tfvars` to specify values for the variables:

```
vpc_cidr_block = "10.0.0.0/16"
private_subnet_cidr_blocks = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
public_subnet_cidr_blocks = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
```

Step 3: EKS Cluster and Worker Nodes

Create a new file called `eks-cluster.tf` for the EKS configuration using a Terraform module using as reference: https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest

This allows you to leverage pre-built configurations for EKS clusters and simplifies the setup of IAM roles, security groups, and other necessary components.

3.1 EKS Cluster Configuration (`eks-cluster.tf`)

```
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "19.17.2"
```

```
  cluster_name = "myapp-eks-cluster"
  cluster_version = "1.27"
  cluster_endpoint_public_access  = true

  subnet_ids = module.myapp-vpc.private_subnets
  vpc_id = module.myapp-vpc.vpc_id

  tags = {
    environment = "development"
    application = "myapp"
  }

  eks_managed_node_groups = {
    dev = {
      min_size     = 1
      max_size     = 3
      desired_size = 3

      instance_types = ["t2.small"]
    }
  }
}
```

Step 4: Initialize, Plan, and Apply Terraform Configuration

1. **Initialize Terraform:** `terraform init`
2. **Preview Changes:** `terraform plan`
3. **Apply the Configuration:** `terraform apply -auto-approve`

Step 5: Deploy an Nginx Application

1. **Configure Environment:**
   - Update the Kubeconfig file to allow `kubectl` to connect to the EKS cluster: `aws eks update-kubeconfig --name myapp-eks-cluster --region us-east-1`
   - Ensure that AWS CLI, `kubectl`, and AWS IAM Authenticator are installed.
2. **Nginx Deployment Configuration (nginx-config.yaml):**

   Create the deployment and service file for Nginx:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
```

```
  labels:
    app: nginx
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

3. Deploy Nginx to EKS: `kubectl apply -f /path/to/nginx-config.yaml`
4. Check the EKS Nodes: `kubectl get nodes`
5. Verify Deployment Status: `kubectl get pod -w`

   Example output:

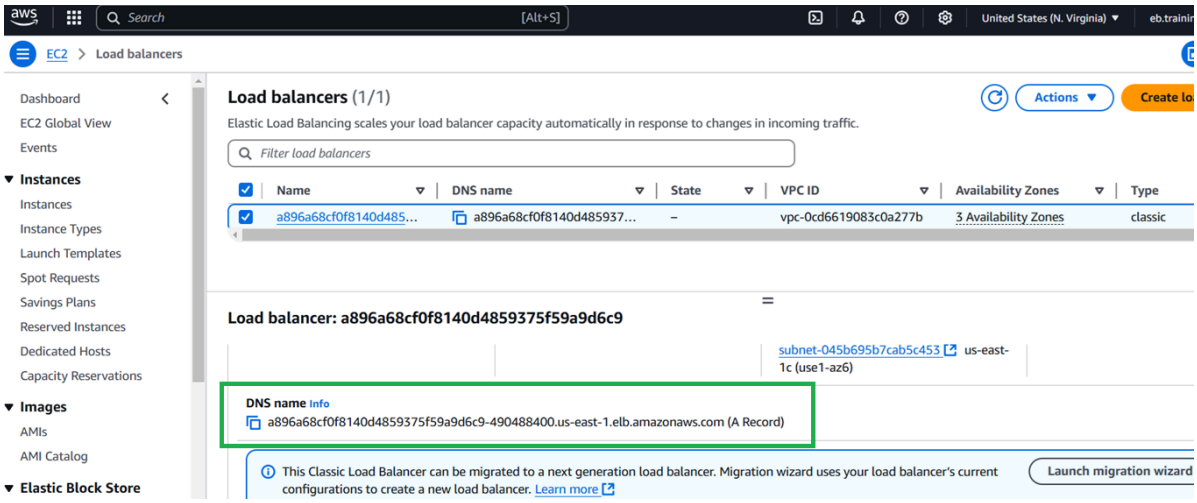
```
NAME                READY   STATUS    RESTARTS   AGE
nginx-55f598f8d-tnqqs   1/1     Running   0          15s
```

6. Check Services for LoadBalancer IP: `kubectl get svc`

   You should see an external LoadBalancer IP for the Nginx service:

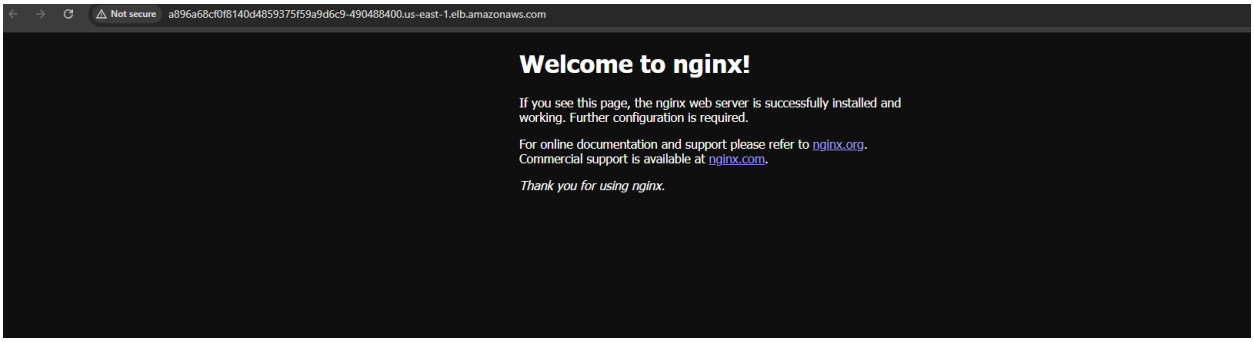
```
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP
kubernetes   ClusterIP      172.20.0.1     <none>
nginx        LoadBalancer   172.20.43.75   a896a68cf0f8140d4859375f59a9d6
```

   You can also see within the AWS console:



7. **Input the LoadBalancer DNS into a browser to confirm Nginx is accessible.**

http://a896a68cf0f8140d4859375f59a9d6c9-490488400.us-east-1.elb.amazonaws.com/



Step 6: Clean Up Resources

1. Delete the Nginx Deployment:

   `kubectl delete -f /mnt/c/Users/eduar/devops_projects2/05-terraform/nginx-config.yaml`

2. Destroy All Terraform Resources: `terraform destroy --auto-approve`

3. Verify All Resources Are Deleted: `terraform state list`

   Ensure no resources are listed in the output.

Troubleshoot:

Issue: Subnet Deletion Errors

If Terraform fails with a `DependencyViolation` error when deleting subnets or an Internet Gateway:

```
Error: deleting EC2 Subnet (subnet-045b695b7cab5c453): DependencyViolati
    status code: 400, request id: 9f8b7dae-152b-4c14-af21-98a4425d6819
```

Solution:

1. **Check for Load Balancers:**

   Ensure all Load Balancers (created by Kubernetes) are deleted. You can manually delete any remaining Load Balancers from the AWS console.

2. **Suggested Workflow:**

   - Delete Kubernetes resources first: `kubectl delete -f /path/to/nginx-config.yaml`
   - Run Terraform destroy: `terraform destroy --auto-approve`

By following this workflow, all dependent resources will be cleaned up before Terraform attempts to destroy the infrastructure.
