EKS Cluster with Karpenter

This Terraform repository deploys an Amazon EKS cluster with Karpenter for autoscaling. It supports both x86 and ARM64 (Graviton) instances, allowing you to run workloads on the most cost-effective and performant hardware.

Features

- **EKS Cluster**: Deploys a managed EKS cluster with the latest Kubernetes version.
- **Karpenter**: Automatically provisions nodes based on workload requirements.
- **Multi-Architecture Support**: Supports both x86 and ARM64 (Graviton) instances.
- **Spot Instances**: Utilizes Spot instances for cost savings.
- **Terraform Modules**: Uses modular and reusable Terraform code.
- **Security Best Practices**: Implements least privilege IAM policies for Karpenter.

Prerequisites

- **Terraform**: Install Terraform v1.0+.
- **AWS CLI**: Configure the AWS CLI with credentials and the correct region.
- **kubectl**: Install kubectl to interact with the EKS cluster.
- **Helm**: Install Helm to deploy Karpenter.
- **Existing VPC**: Ensure you have an existing VPC and subnets in your AWS account.

Usage

1. Clone the Repository

git clone https://github.com/your-repo/eks-karpenter-terraform.git
cd eks-karpenter-terraform

2. Initialize Terraform

terraform init

3. Update Variables

Update the following variables in the `main.tf` file:

- `vpc_id`: Your existing VPC ID.

- `subnet_ids`: Your existing subnet IDs.
- `cluster_name`: Name of the EKS cluster.
- `karpenter_iam_role`: IAM role ARN for Karpenter.

4. Apply the Terraform Configuration

terraform apply

This will:

- Deploy an EKS cluster.
- Install Karpenter.
- Configure Karpenter to provision both x86 and ARM64 instances.

Deploying Workloads

1. Deploy a Workload on x86

Create a deployment with a node selector for amd64 (x86):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: x86-workload
spec:
  replicas: 2
  selector:
    matchLabels:
      app: x86-workload
  template:
    metadata:
      labels:
        app: x86-workload
    spec:
      containers:
        - name: nginx
          image: nginx
      nodeSelector:
        kubernetes.io/arch: amd64
```

Apply the deployment:

kubectl apply -f x86-workload.yaml

2. Deploy a Workload on ARM64 (Graviton)

Create a deployment with a node selector for arm64 (Graviton):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: arm64-workload
spec:
  replicas: 2
  selector:
    matchLabels:
      app: arm64-workload
  template:
    metadata:
      labels:
        app: arm64-workload
    spec:
      containers:
        - name: nginx
          image: nginx
      nodeSelector:
        kubernetes.io/arch: arm64
```

Apply the deployment:

kubectl apply -f arm64-workload.yaml

3. Verify Pod Scheduling

kubectl get pods -o wide
kubectl get nodes -o wide

You should see pods scheduled on x86 or ARM64 nodes based on the node selector.

Testing Karpenter Autoscaling

1. Deploy a Test Workload

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate
spec:
  replicas: 5
  selector:
    matchLabels:
      app: inflate
  template:
    metadata:
      labels:
        app: inflate
    spec:
      containers:
        - name: inflate
          image: public.ecr.aws/eks-distro/kubernetes/pause:3.7
          resources:
            requests:
              cpu: "1"
              memory: "512Mi"
```

Apply the deployment:

```
kubectl apply -f inflate.yaml
```

2. Monitor Node Provisioning

```
kubectl get nodes -w
```

Cleanup

To destroy the infrastructure and avoid unnecessary costs:

```
terraform destroy
```

Troubleshooting

1. Pods Stuck in Pending State

Check Karpenter logs:

kubectl logs -n karpenter -l app.kubernetes.io/name=karpenter

Verify the Provisioner configuration:

kubectl get provisioner default -o yaml

2. Nodes Not Provisioned

Ensure subnets and security groups are tagged correctly:

aws ec2 describe-subnets --subnet-ids <subnet-id> --query 'Subnets[*].Tags'
aws ec2 describe-security-groups --group-ids <security-group-id> --query 'SecurityGroups[*].Tags'

Tags should include:

Key: karpenter.sh/discovery
Value: <cluster-name>

3. Karpenter Not Running

Check the Karpenter pods:

kubectl get pods -n karpenter

4. IAM Role Issues

Ensure the IAM role for Karpenter has the necessary permissions:

aws iam get-role --role-name <karpenter-iam-role>

Confirm that it includes policies for:

- `ec2:RunInstances`
- `ec2:TerminateInstances`
- `ec2:DescribeInstances`
- `eks:DescribeCluster`