Implementing AWS EKS Karpenter Using Terraform

Table of Contents

1. **What is Karpenter, AWS EKS, Terraform**

⇨ Karpenter, AWS EKS and Terraform are all tools and technologies commonly used the realm of cloud-native application deployment and management, particularly within context of Kubernetes and Infrastructure as code.

Karpenter:

Karpenter is an open-source project developed by AWS that providers an advanced cluster autoscaler for Kubernetes. It extends the functionality of the native Kubernetes Cluster Autoscaler by adding support for AWS-specific features such EC2 Spot Instances. Karpenter aims to optimize cost and resource utilization for Kubernetes Cluster running on AWS by intelligently managing the Cluster's capacity and leveraging AWS services effectively. Also Spot instance support, carpenter offers sophisticated algorithms for capacity management, ensuring that Kubernetes Clusters dynamically adjust their size to meet application demand efficiently. This automated scaling capability not only optimizes resource utilization but also simplifies cluster management for operators, freeing them from manual intervention in response to fluctuating workloads. Moreover, Karpenter integrate seamlessly with other AWS Services, enabling Kubernetes cluster to leverage the full spectrum of AWS Offerings for enhanced functionality and performance.

AWS EKS (Elastic Kubernetes Service):

AWS EKS is managed Kubernetes service offered by AWS (Amazon Web Service). It simplifies the process of deploying, managing and scaling Kubernetes cluster on AWS infrastructure. With EKS, users can run Kubernetes applications without needing to install or manage the Kubernetes control plane. EKS integrates seamlessly with other AWS services, providing a highly available, secure, and scalable platform for containerized workloads. Kubernetes have one significant advantage that is it's deep integration with other AWS services. This integration extends beyond basic infrastructure components to include services such as AWS identity and Access management (IAM), AWS VPC, AWS CloudTrail and AWS Cloud Watch.Also it provides native support for Kubernetes tooling and APIs, ensuring compatibility with existing Kubernetes workflow and ecosystem tools. This compatibility enables users to leverage familiar Kubernetes features, such as Deployments, Services, and Persistent Volumes, without modification.

Terraform:

Terraform is also an open-source infrastructure as code (IaC) tool developed by HashiCorp. It allows users to define and provision infrastructure resources using declarative configuration files. Terraform supports multiple cloud providers, including AWS, Azure, Google Cloud Platform, and others. With Terraform, Infrastructure can be defined in code, enabling automation, version control, and reproducibility of infrastructure deployments. Terraform can be used to define and provision AWS resources required for running Kubernetes Cluster, including EKS Cluster, networking components, security policies and more. Terraform' s extendible architecture and vibrant ecosystem of plugins and modules provide users with a rich library of pre-built

Infrastructure components and best practices. These reusable modules cover a wide range of use cases, from provisioning Kubernetes cluster and networking infrastructure to configuring monitoring and security policies, allowing users to accelerate their infrastructure deployments and reduce manual effort.

2. Cluster Autoscaller & Karpenter & AWS Fargate

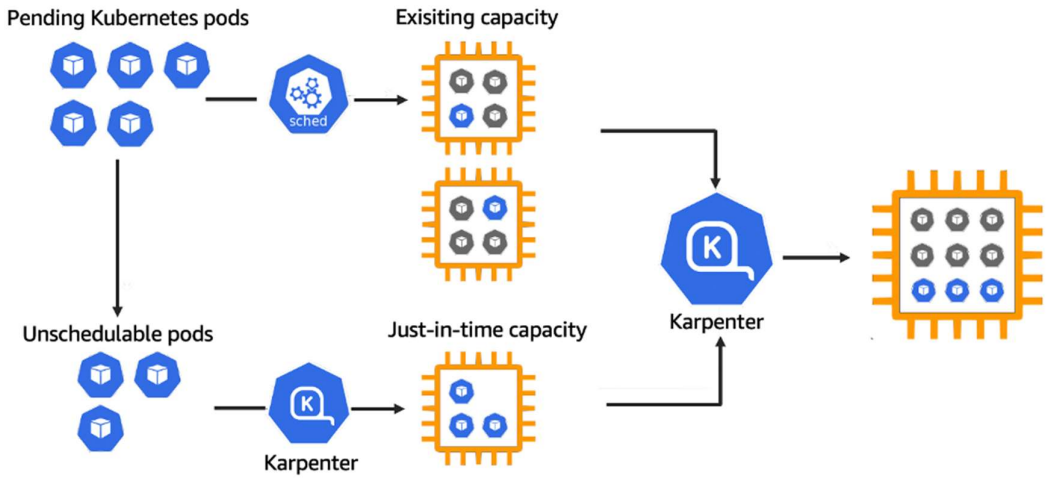
⇨ Cluster AutoScaler, Karpenter and AWS Fargate are all tools and services related to container orchestration and management, commonly used in the context of Kubernetes.

Cluster AutoScaler is a component of Kubernetes that automatically adjust the size of a Kubernetes cluster based on the current resource utilization. It scales nodes in and out of the cluster to ensure that there are enough resources available to schedule pods while also minimizing cost by removing underutilized nodes. This helps in optimizing the utilization of resources and ensures that applications have sufficient capacity to run efficiently.

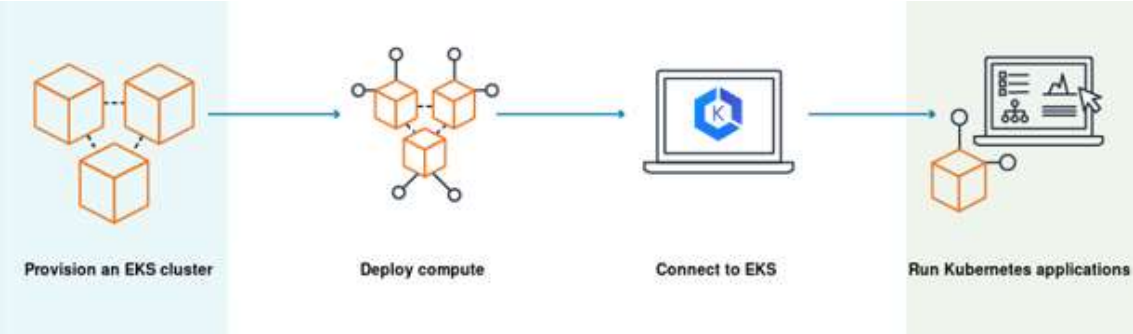
Karpenter Karpenter is an open-source project developed by AWS that provides automated node provisioning for Kubernetes clusters. It enables users to provision and manage the underlying infrastructure such as AWS EC2 instances in AWS required to run Kubernetes workloads. Karpenter works with the Cluster AutoScaler to dynamically provision and deprovision nodes based on workload demands. It supports multiple cloud providers, including AWS, Azure, and GCP.

AWS Fargate is a serverless compute engine for containers that allows users to run containers without managing the underlying infrastructure. With Fargate, users can focus on deploying and scaling their containerized applications without worrying about server provisioning, patching or scaling. Fargate abstracts away the EC2 instances, and users only pay for the vCPU and memory resources consumed by their containers, making it a cost-effective solution for running containerized workloads on AWS.

Karpenter

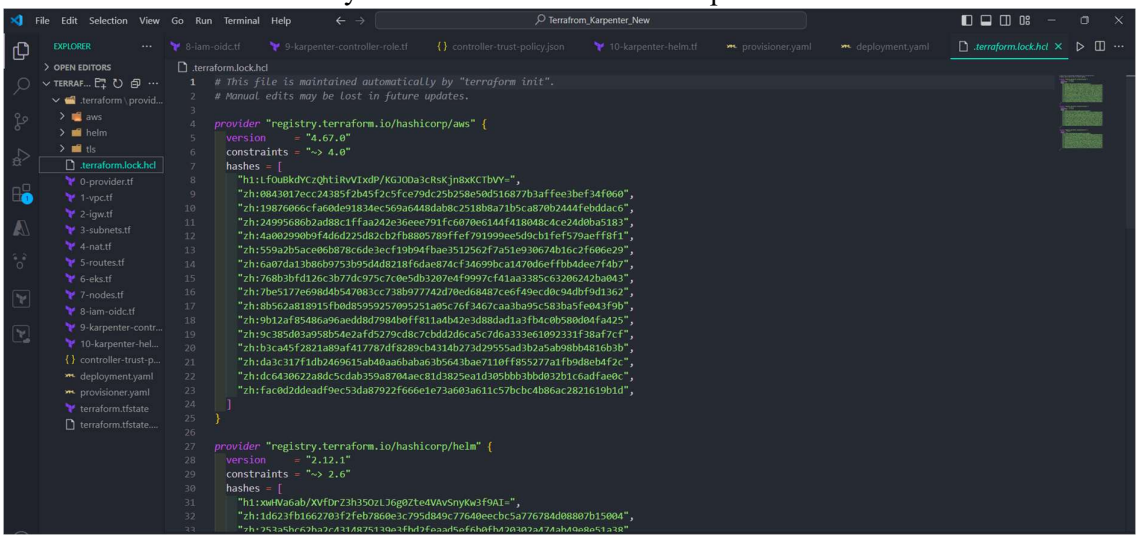


AWS EKS

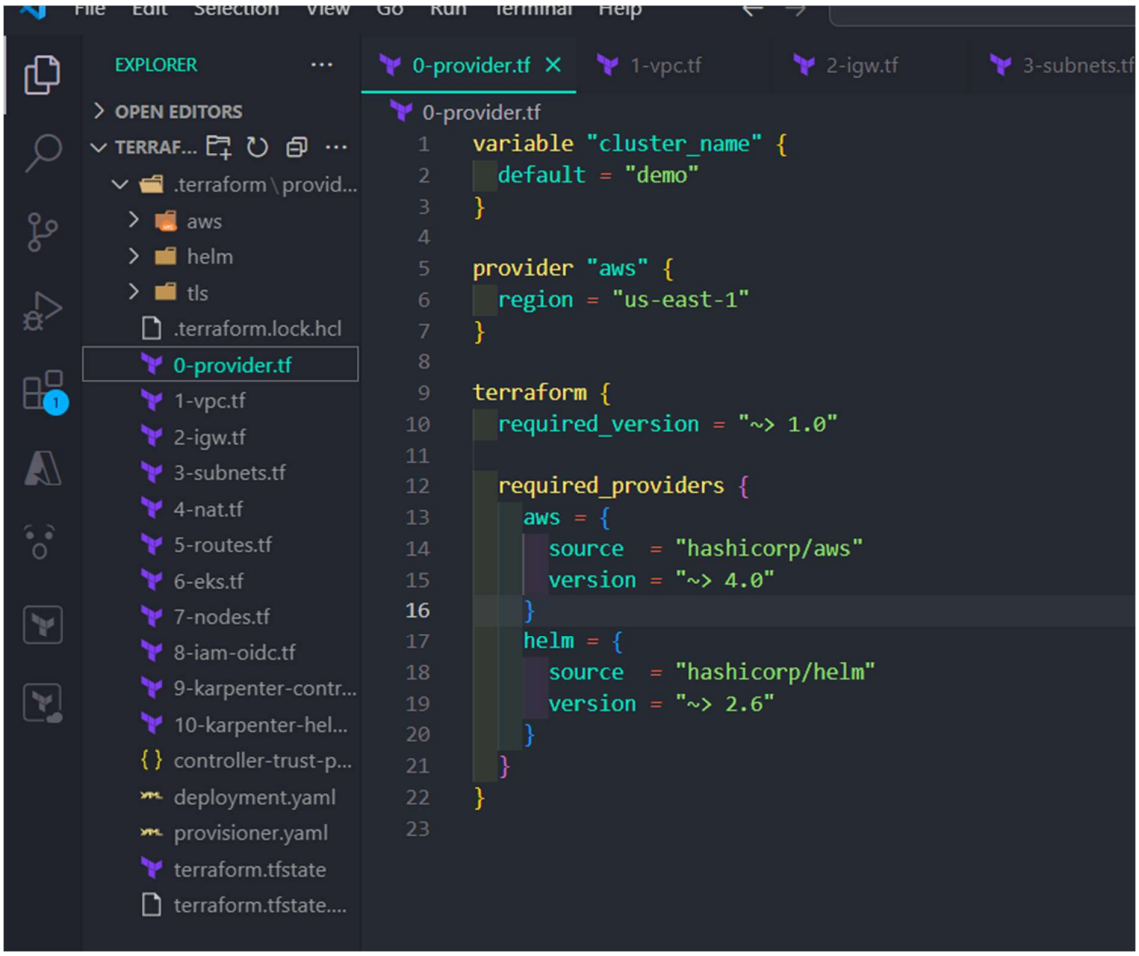


3. Create AWS VPC Using Terraform

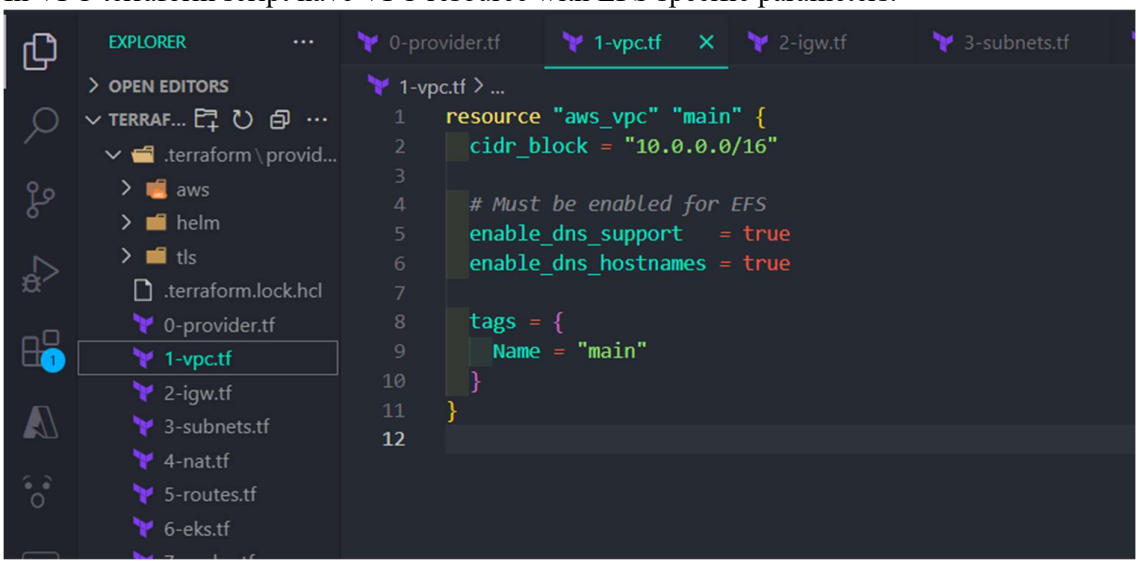
In new version of terraform you have the lock file with all provider version that I use.



Next is a provider with some variable such as EKS Cluster name and a region



In VPC terraform script have VPC resource with EFS specific parameters.



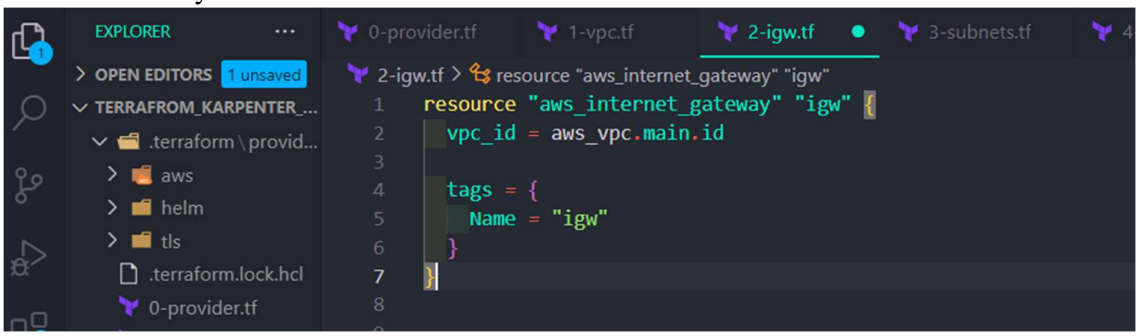
Create a Internet Gateway –

Step 1: Define Provider

Being by defining the AWS provider in your Terraform configuration file This file tells Terraform which cloud provider to use and which region

Step 2: Create a Internet Gateway resource

Define the internet gateway resource. Make sure to specify VPC ID to which the internet Gateway will be attached.



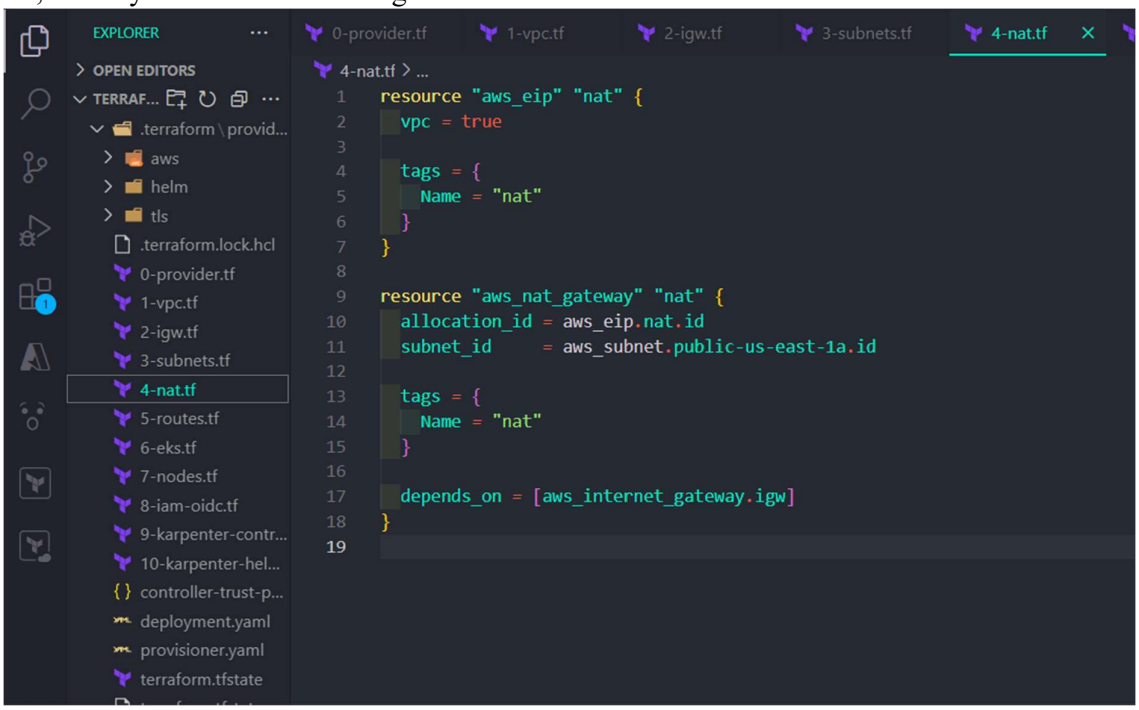
Create a Subnet

Create a Subnet resource, define the subnet resource, specifying its properties such as CIDR block, VPC ID, AZ etc.

I have created a 2 Private subnet and 2 Public subnet

Create a NAT Gateway

Define a NAT Gateway resource, specifying its properties such as subnet ID, allocation ID, and any other relevant settings.

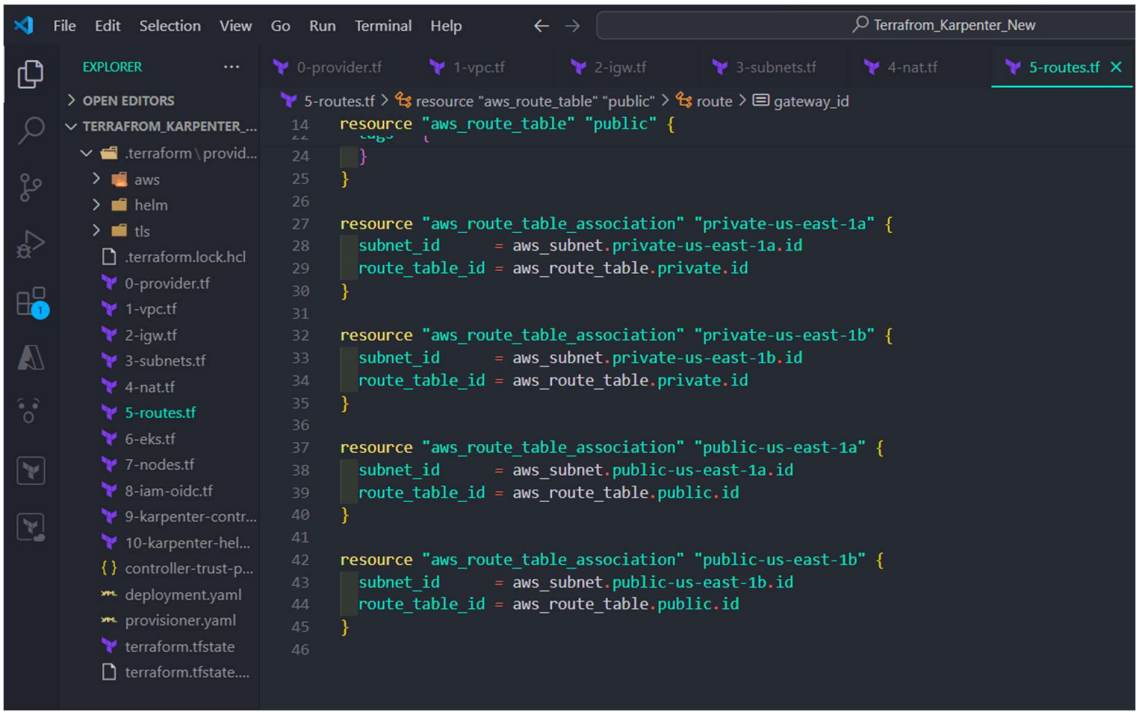


Create a Route

Define the route table resource and associate it with the desired subnet

```
resource "aws_route_table" "private" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.nat.id
  }

  tags = {
    Name = "private"
  }
}

resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "public"
  }
}
```

Define the Routes within the route table, specifying the destination CIDR blocks and target.



Let’s initialize terraform and create all those components with terraform apply
terraform init
terraform apply

4. Create EKS Cluster Using Terraform

Firstly, EKS requires an IAM role to access AWS API on your behave to create resources.

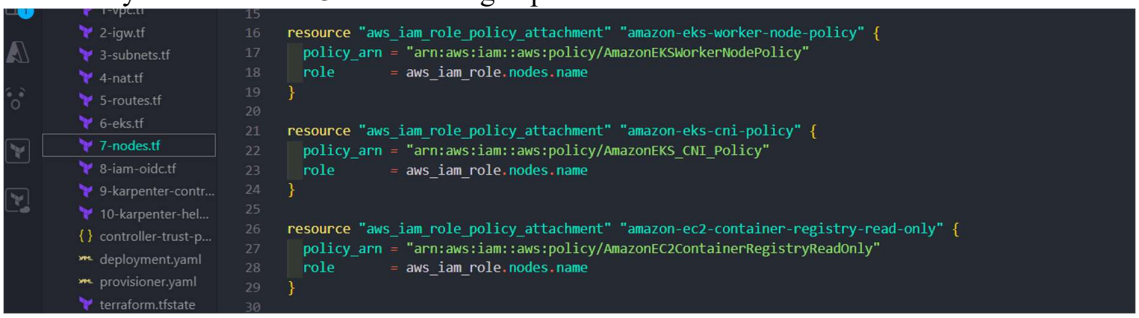
There is a single IAM policy that needs to be attached to that role : AmazonEKSClusterPolicy

```
resource "aws_iam_role_policy_attachment" "amazon-eks-cluster-policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.eks-cluster.name
}
```

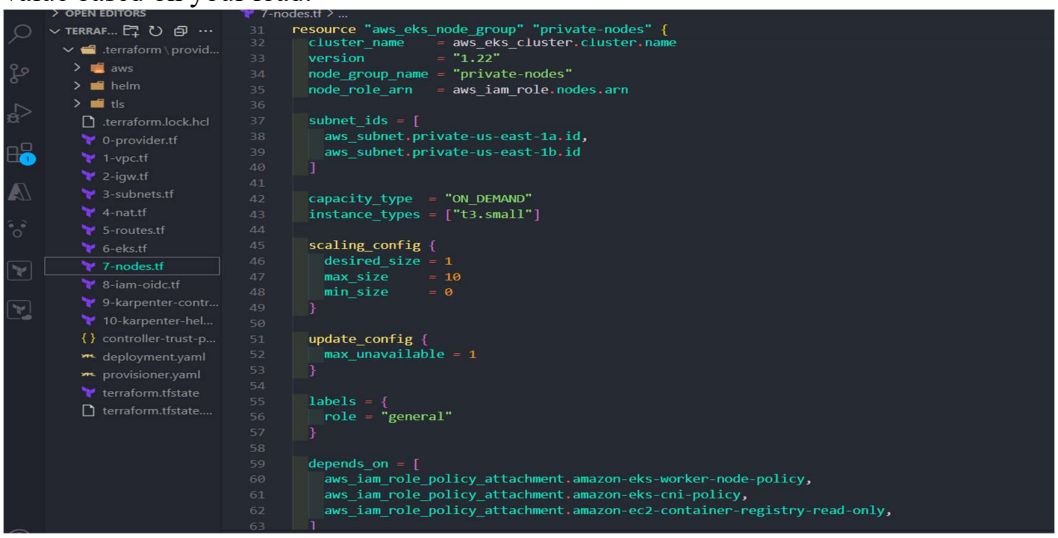
Next, we create a Cluster which is provide a cluster name and ARN of the role that we just created before

Now we need to create a Kubernetes nodes,
It's going to be used by the regular node pool and not karpenter, you have two option, either to use the same IAM role and create an instance profile for Karpenter or you can create a dedicated IAM role. But in this case, you would need manually update auth configmap to authorize nodes created by Karpenter with new IAM role to join the cluster. we will done later, now we are use the same IAM role.

We usually need to attach 3 AWS-managed policies as a base minimum.



Then it's going to be a EKS managed node group created as an AWS autoscaling group, with max, min, and desired size. So Your typical cluster autoscaller would adjust this value based on your load.



Now let's again apply the terraform to create an EKS Cluster.

terraform apply command run again

To connect to the cluster, you need to update the Kubernetes context with this command.

⇨ aws eks update-kubeconfig --name demo --region us-east-1

Then the quick check if we can reach Kubernetes. It should return the default k8s service.

⇨ kubectl get svc

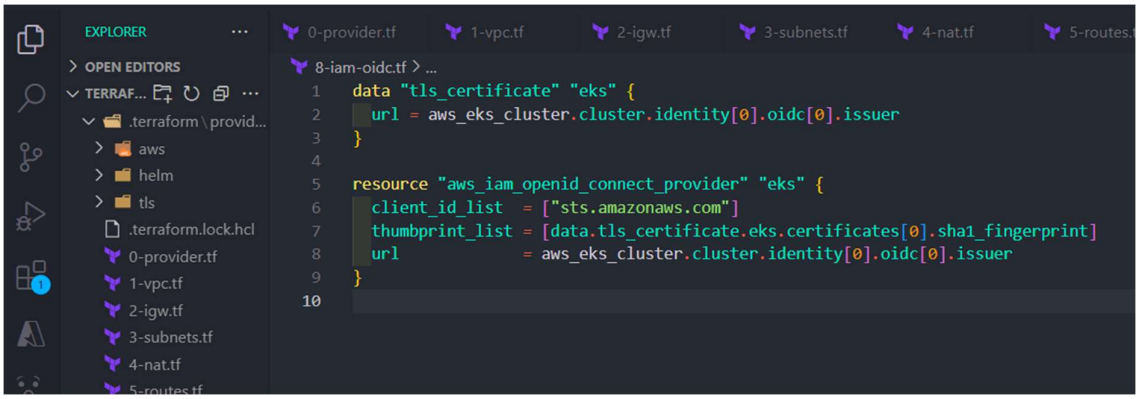
As I mentioned before, if you decide to create a separate IAM role and instance profile you would need to edit the auth configmap to add the ARN of the new role.

⇨ kubectl edit configmap aws-auth -n kube-system

5. Create Karpenter Controller IAM Role

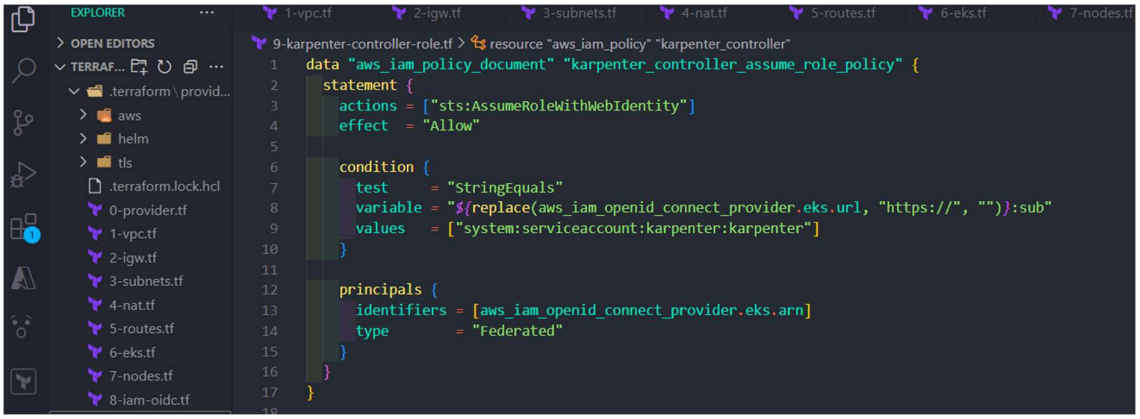
Karpenter needs permission to create EC2 instance in AWS. If you use a self-hosted Kubernetes cluster, for example by using kOps. You can add additional IAM policies to existing IAM role attached to Kubernetes nides. We use EKS, the best way to grant access to internal service would be with IAM roles for service accounts.

Firstly, we need to create OpenID Connect provider.



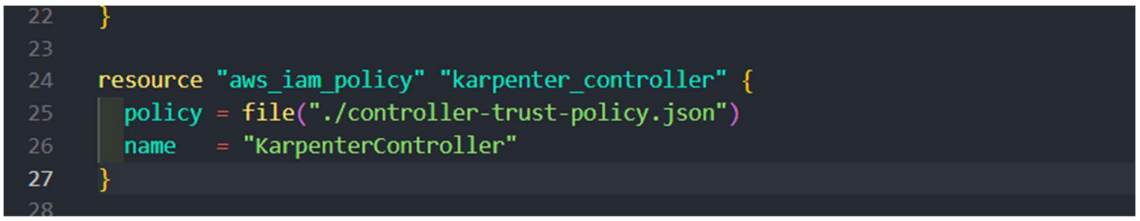
Get the certificate and use it in terraform resource.

Next is the trust policy to allow the Kubernetes a service to account to assume the IAM role. Make sure that you deploy Karpenter to the karpenter namespace within the same service account name.

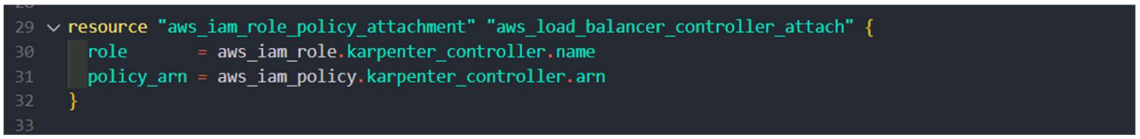


Then create a karpenter controller role and attach that policy

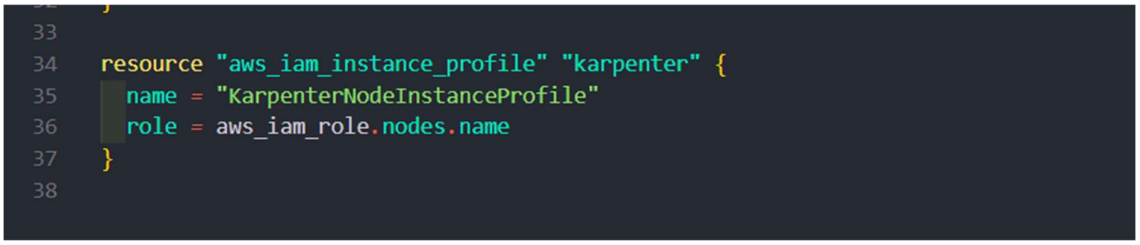
Next is a set of permissions that we need to grant to Karpenter to manage Kubernetes nodes.



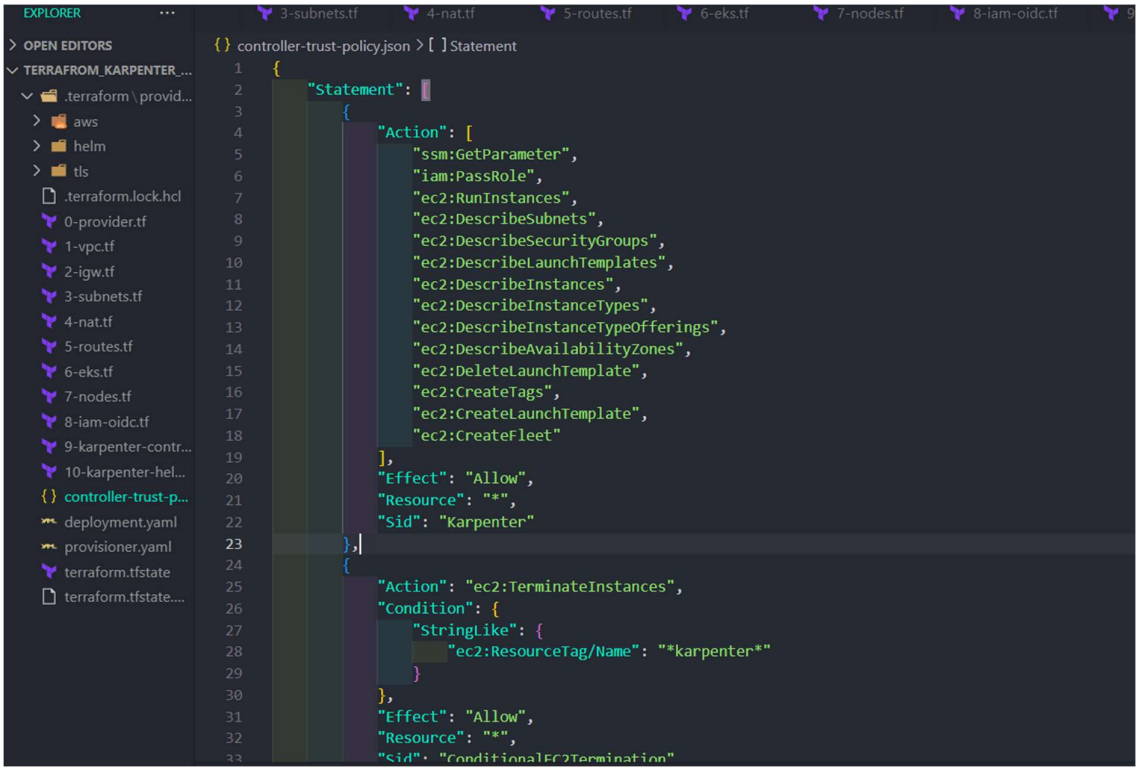
Then Attached it to the role as well.



Since we will be using the same IAM role, we need to create an IAM instance profile that karpenter will use to attach to EC2 Instances.



Let’s create the controller-trust-policy.json file



Since we have added an additional provider we need to initialize before we can apply

So same command run again

terraform init

terraform apply

6. Deploy Karpenter to EKS

To deploy Karpenter to our cluster, we're going to use **Helm.** First of all, you need to authenticate with EKS using the helm provider. Then the helm release.

```
resource "helm_release" "karpenter" {
  namespace        = "karpenter"
  create_namespace = true

  name       = "karpenter"
  repository = "https://charts.karpenter.sh"
  chart      = "karpenter"
  version    = "v0.13.1"
```

There are few important variables that we need to override.

First is the annotation for the Kubernetes service account

```
  set {
    name  = "serviceAccount.annotations.eks\\.amazonaws\\.com/role-arn"
    value = aws_iam_role.karpenter_controller.arn
  }
```

EKS Cluster name

```
  set {
    name  = "clusterName"
    value = aws_eks_cluster.cluster.id
  }
```

We also need a Cluster endpoint, karpenter will use it to join new nodes to EKS.

```
  set {
    name  = "clusterEndpoint"
    value = aws_eks_cluster.cluster.endpoint
  }
```

The final variable is the instance profile

```
    set {
      name  = "aws.defaultInstanceProfile"
      value = aws_iam_instance_profile.karpenter.name
    }

    depends_on = [aws_eks_node_group.private-nodes]
  }
```

Let's apply

terraform apply

Check if the helm was deployed successfully.

helm list - A

Then the karpenter pod in it's dedicated namespace.

kubectl get pods -n karpenter

7. Create Karpenter Provisioner

Before we can test Karpenter, we need to create a Provisioner. Karpenter defines a Custom Resource called a Provisioner to specify provisioning configuration.

```
---
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
```

I included some parameters such as TTL, also you can define the limit on how many nodes Karpenter can create. It's measured in CPU cores. You can define what types of EC2 family you want to use and exclude. Also, we need to create another Custom Resource: AWSNodeTemplate

```
spec:
  ttlSecondsAfterEmpty: 60 # scale down nodes after 60 seconds without workloads (excluding daemons)
  ttlSecondsUntilExpired: 604800 # expire nodes after 7 days (in seconds) = 7 * 60 * 60 * 24
  limits:
    resources:
      cpu: 100 # limit to 100 CPU cores
  requirements:
    # Include general purpose instance families
    - key: karpenter.k8s.aws/instance-family
      operator: In
      values: [c5, m5, r5]
    # Exclude small instance sizes
    - key: karpenter.k8s.aws/instance-size
      operator: NotIn
      values: [nano, micro, small, large]
  providerRef:
    name: my-provider
---
apiVersion: karpenter.k8s.aws/v1alpha1
kind: AWSNodeTemplate
metadata:
  name: my-provider
```

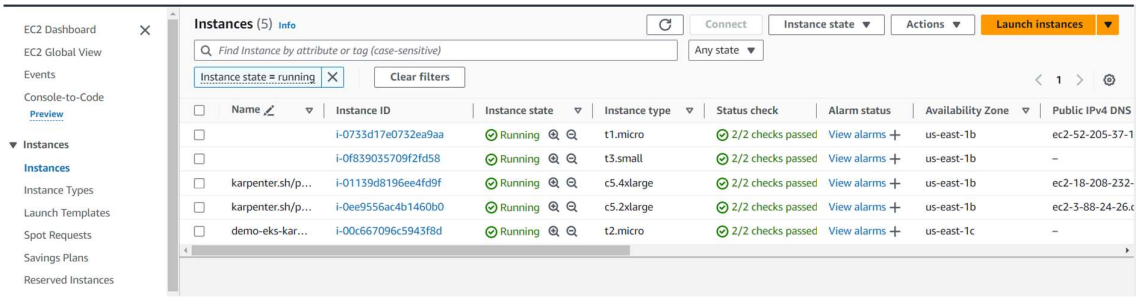
Here you need to use AWS tags to select subnets and Security groups.

```
spec:
  subnetSelector:
    kubernetes.io/cluster/demo: owned
  securityGroupSelector:
    kubernetes.io/cluster/demo: owned
```

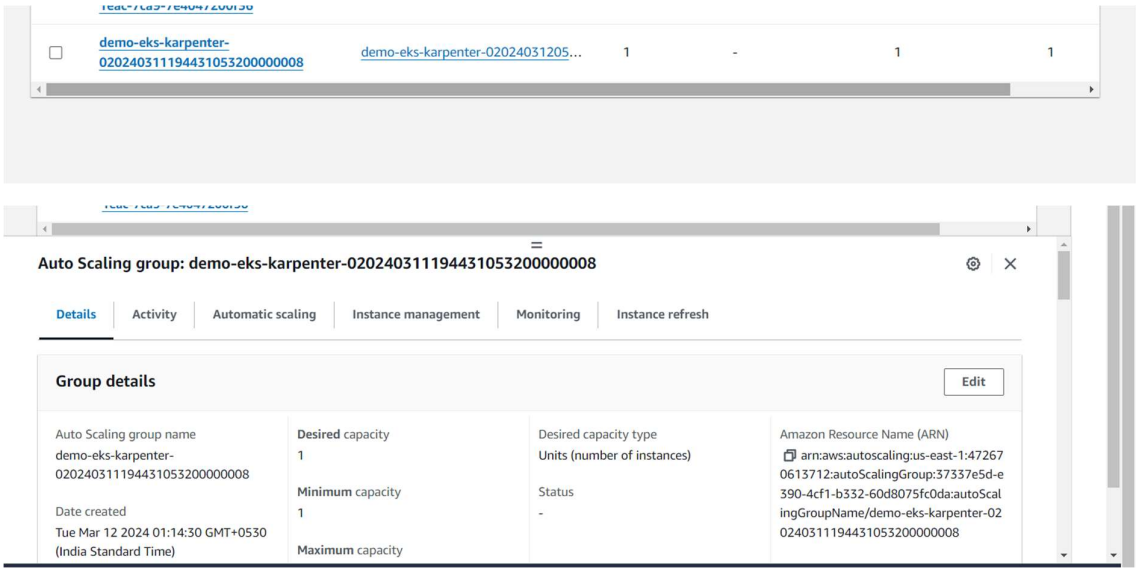
You need to replace the demo with your EKS cluster name.

Finally, use kubectl -f provisioner .yaml

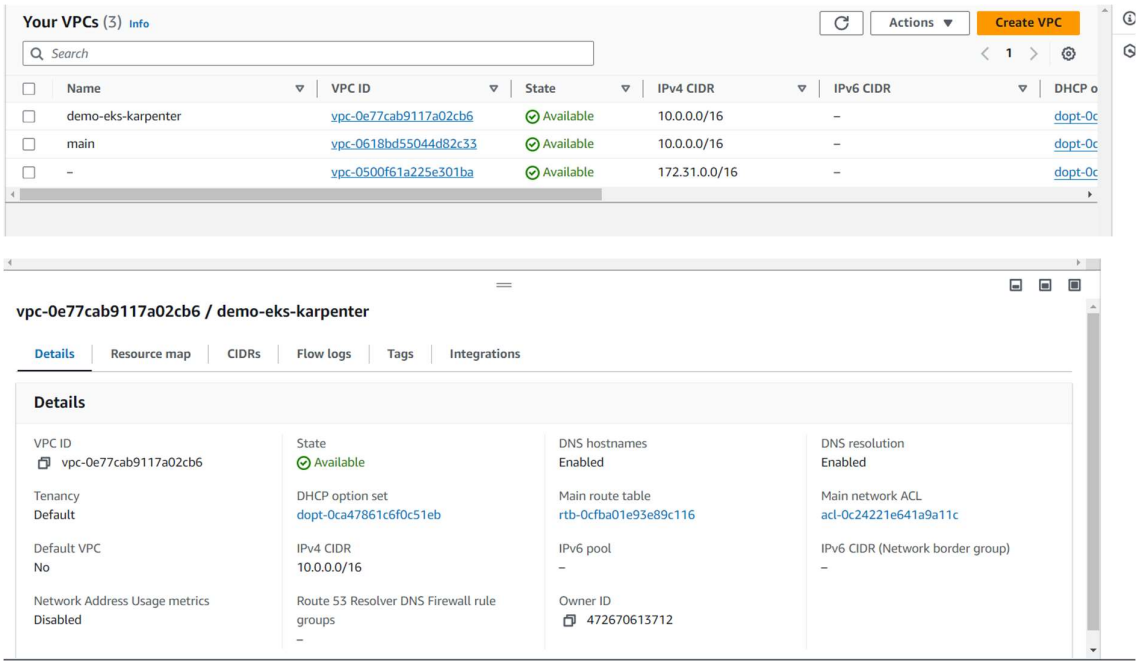
Go to AWS Account and see there create a instances.



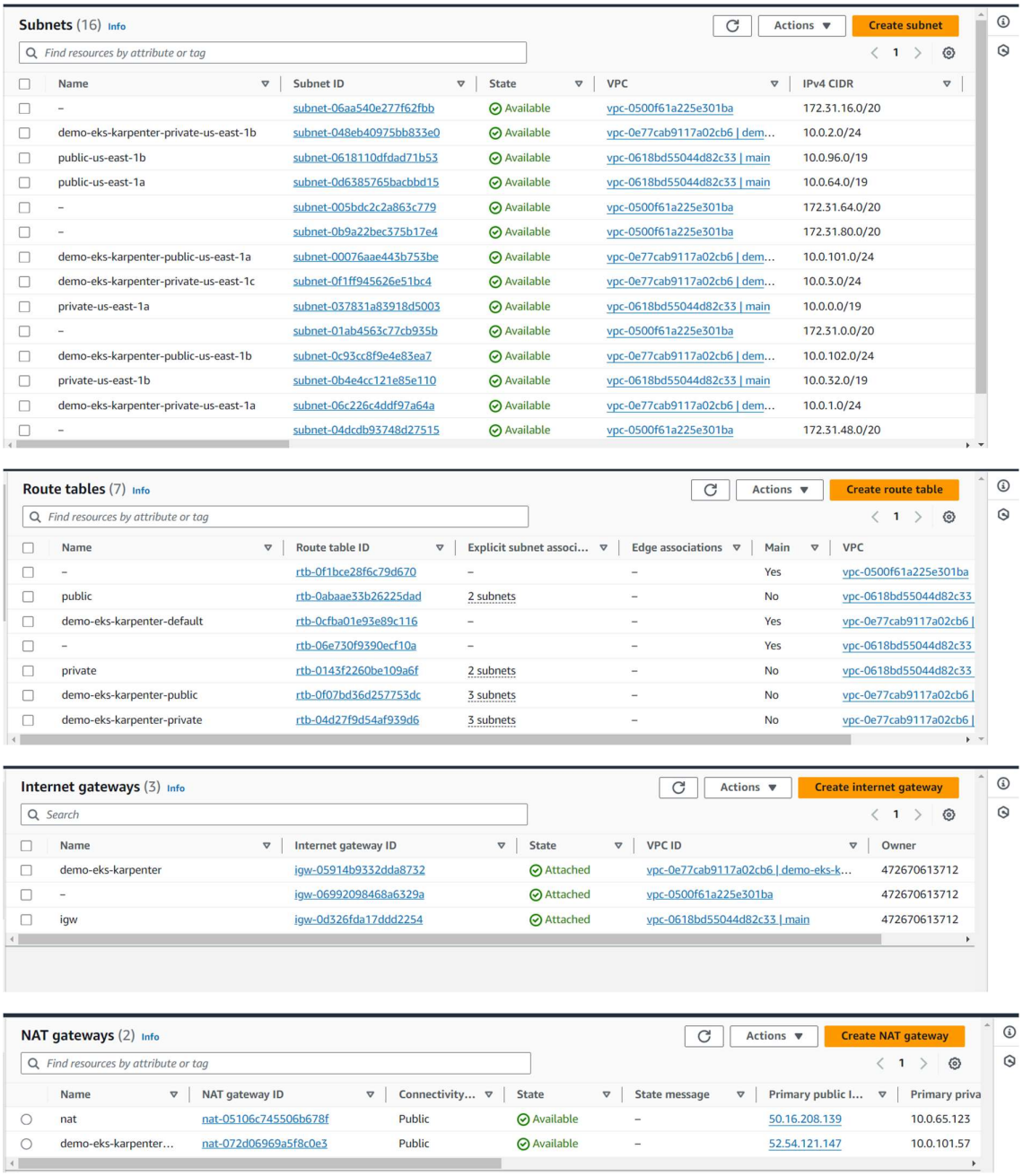
Autoscaler Group created



Also VPC created on AWS



Subnets, Route Table, Internet Gateway are created on AWS



Subnets (16) Info

Actions ▼ | Create subnet

Find resources by attribute or tag

1

Name	Subnet ID	State	VPC	IPv4 CIDR
–	subnet-06aa540e277f62fbb	Available	vpc-0500f61a225e301ba	172.31.16.0/20
demo-eks-karpenter-private-us-east-1b	subnet-048eb40975bb833e0	Available	vpc-0e77cab9117a02cb6 | dem...	10.0.2.0/24
public-us-east-1b	subnet-0618110dfdad71b53	Available	vpc-0618bd55044d82c33 | main	10.0.96.0/19
public-us-east-1a	subnet-0d6385765bacbbd15	Available	vpc-0618bd55044d82c33 | main	10.0.64.0/19
–	subnet-005bdc2c2a863c779	Available	vpc-0500f61a225e301ba	172.31.64.0/20
–	subnet-0b9a22bec375b17e4	Available	vpc-0500f61a225e301ba	172.31.80.0/20
demo-eks-karpenter-public-us-east-1a	subnet-00076aae443b753be	Available	vpc-0e77cab9117a02cb6 | dem...	10.0.101.0/24
demo-eks-karpenter-private-us-east-1c	subnet-0f1ff945626e51bc4	Available	vpc-0e77cab9117a02cb6 | dem...	10.0.3.0/24
private-us-east-1a	subnet-037831a83918d5003	Available	vpc-0618bd55044d82c33 | main	10.0.0.0/19
–	subnet-01ab4563c77cb935b	Available	vpc-0500f61a225e301ba	172.31.0.0/20
demo-eks-karpenter-public-us-east-1b	subnet-0c93cc8f9e4e83ea7	Available	vpc-0e77cab9117a02cb6 | dem...	10.0.102.0/24
private-us-east-1b	subnet-0b4e4cc121e85e110	Available	vpc-0618bd55044d82c33 | main	10.0.32.0/19
demo-eks-karpenter-private-us-east-1a	subnet-06c226c4ddf97a64a	Available	vpc-0e77cab9117a02cb6 | dem...	10.0.1.0/24
–	subnet-04dcdb93748d27515	Available	vpc-0500f61a225e301ba	172.31.48.0/20

Route tables (7) Info

Actions ▼ | Create route table

Find resources by attribute or tag

1

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC
–	rtb-0f1bce28f6c79d670	–	–	Yes	vpc-0500f61a225e301ba
public	rtb-0abaae33b26225dad	2 subnets	–	No	vpc-0618bd55044d82c33
demo-eks-karpenter-default	rtb-0cfba01e93e89c116	–	–	Yes	vpc-0e77cab9117a02cb6 |
–	rtb-06e730f9390ecf10a	–	–	Yes	vpc-0618bd55044d82c33
private	rtb-0143f2260be109a6f	2 subnets	–	No	vpc-0618bd55044d82c33
demo-eks-karpenter-public	rtb-0f07bd36d257753dc	3 subnets	–	No	vpc-0e77cab9117a02cb6 |
demo-eks-karpenter-private	rtb-04d27f9d54af939d6	3 subnets	–	No	vpc-0e77cab9117a02cb6 |

Internet gateways (3) Info

Actions ▼ | Create internet gateway

Search

1

Name	Internet gateway ID	State	VPC ID	Owner
demo-eks-karpenter	igw-05914b9332dda8732	Attached	vpc-0e77cab9117a02cb6 | demo-eks-k...	472670613712
–	igw-06992098468a6329a	Attached	vpc-0500f61a225e301ba	472670613712
igw	igw-0d326fda17ddd2254	Attached	vpc-0618bd55044d82c33 | main	472670613712

NAT gateways (2) Info

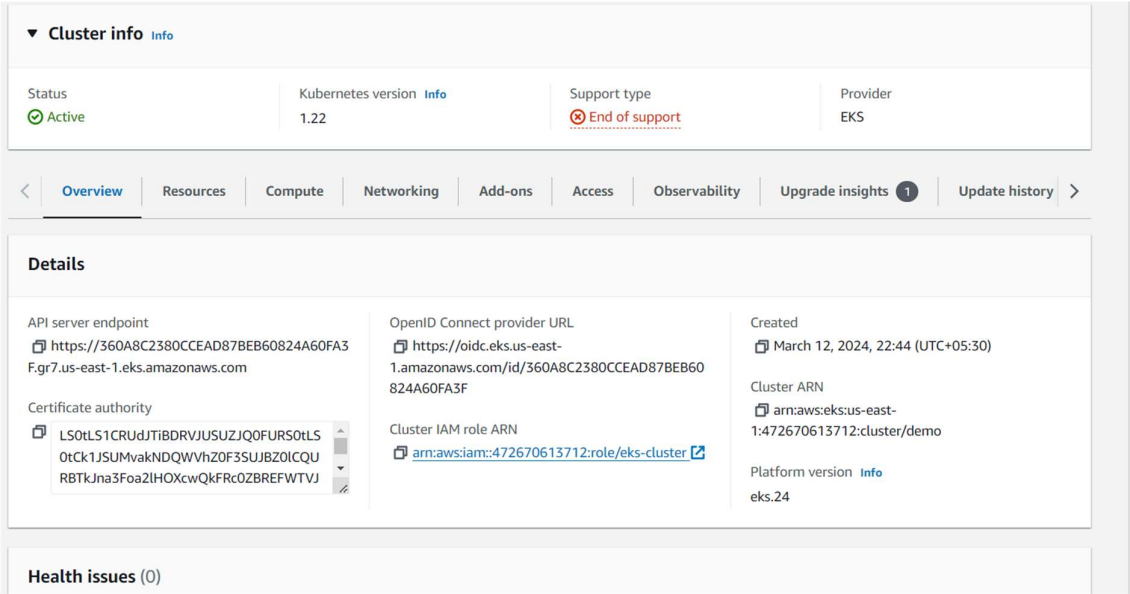
Actions ▼ | Create NAT gateway

Find resources by attribute or tag

1

Name	NAT gateway ID	Connectivity...	State	State message	Primary public I...	Primary priva
nat	nat-05106c745506b678f	Public	Available	–	50.16.208.139	10.0.65.123
demo-eks-karpenter...	nat-072d06969a5f8c0e3	Public	Available	–	52.54.121.147	10.0.101.57

Finally, AWS EKS Created



8. Demo: Automatic Node Provisioning

Lastly, let's created a Kubernetes deployment to test how quickly karpenter can create EC2 Instances and Schedule new pods

8-iam-oidc.tf | 9-karpenter-controller-role.tf | controller-trust-policy.json | 10-karpenter-helm.tf | terraform

deployment.yaml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        resources:
          requests:
            cpu: "4"
            memory: 4Gi
        ports:
        - containerPort: 80

```

When you just getting started with karpenter, it’s a good idea to check logs in case you get any errors.

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

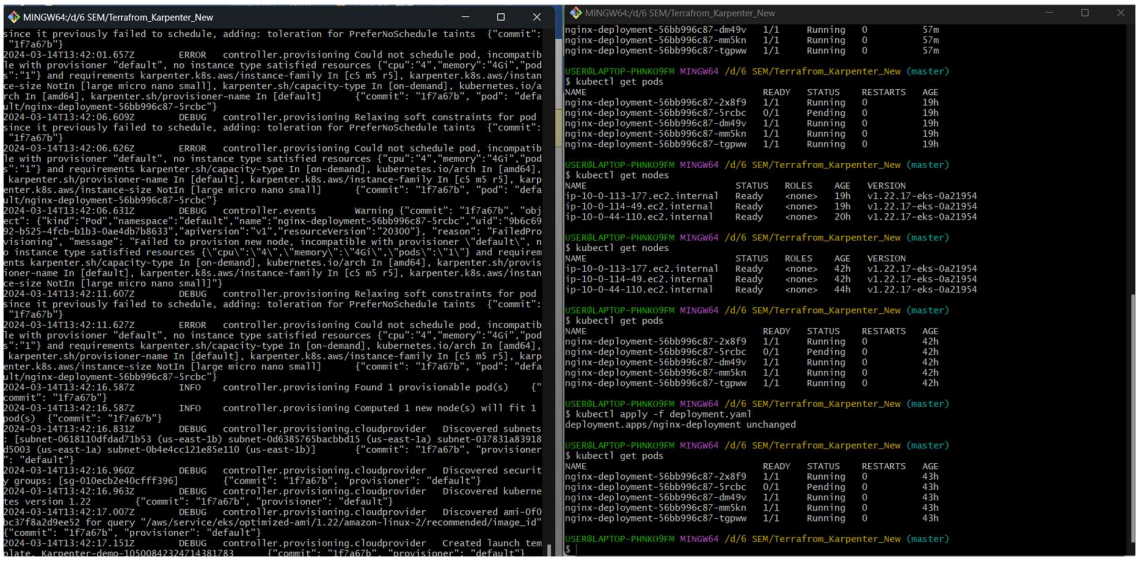
In another window, let’s run get pods

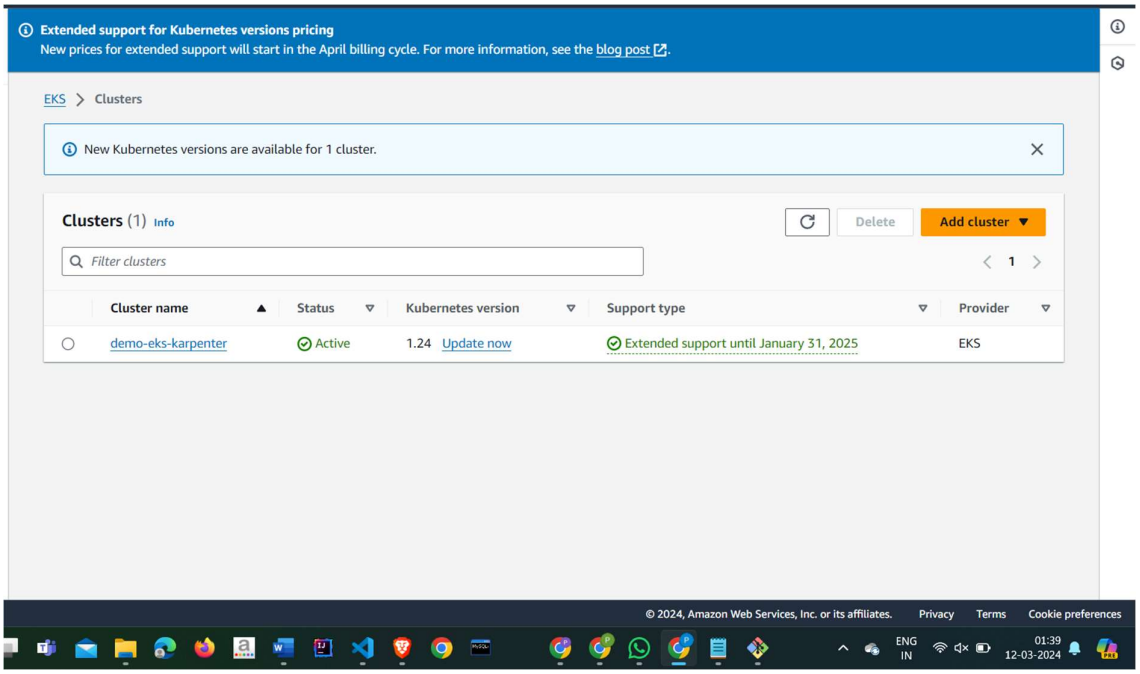
```
kubectl get pods
```

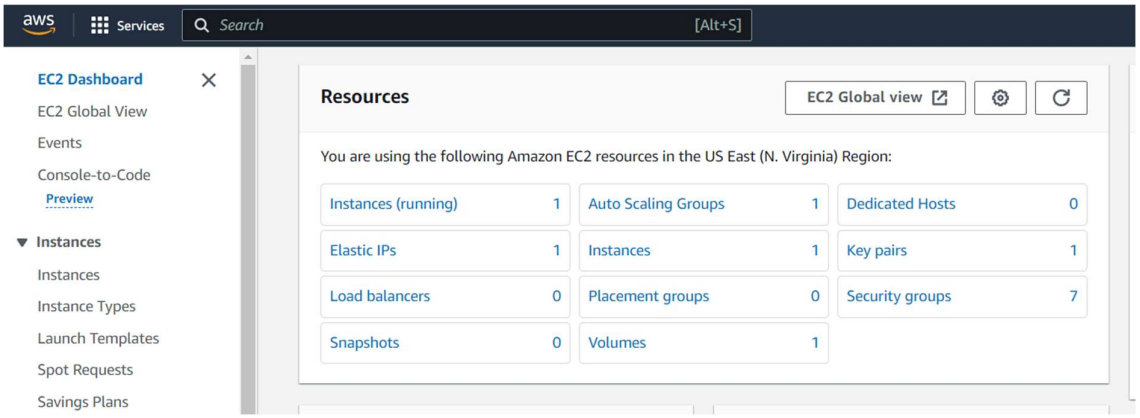
let’s get all the nodes available in the Kubernetes Cluster

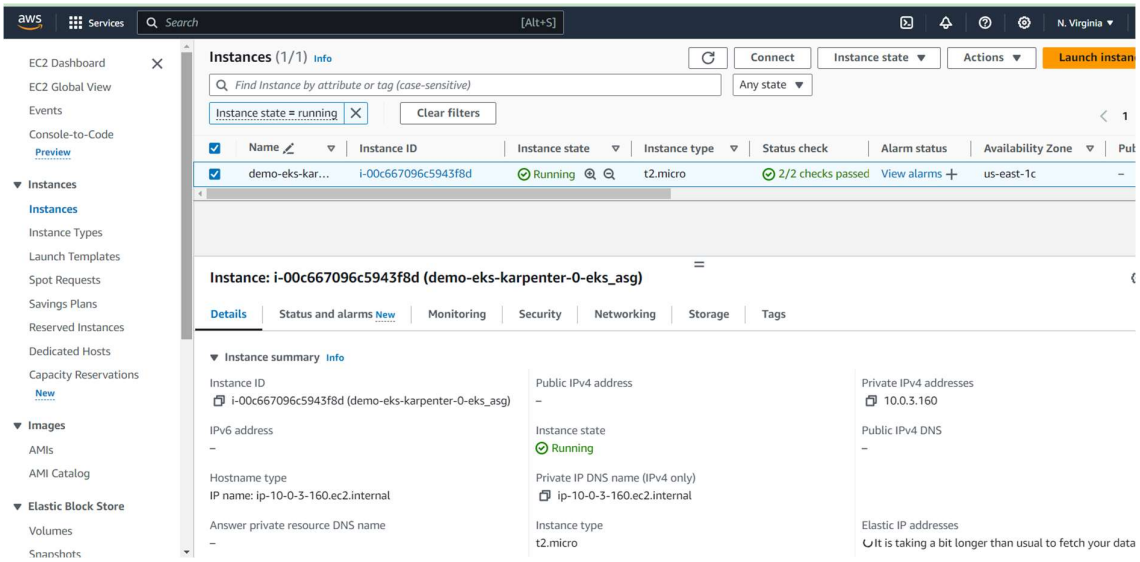
```
kubectl get nodes
```

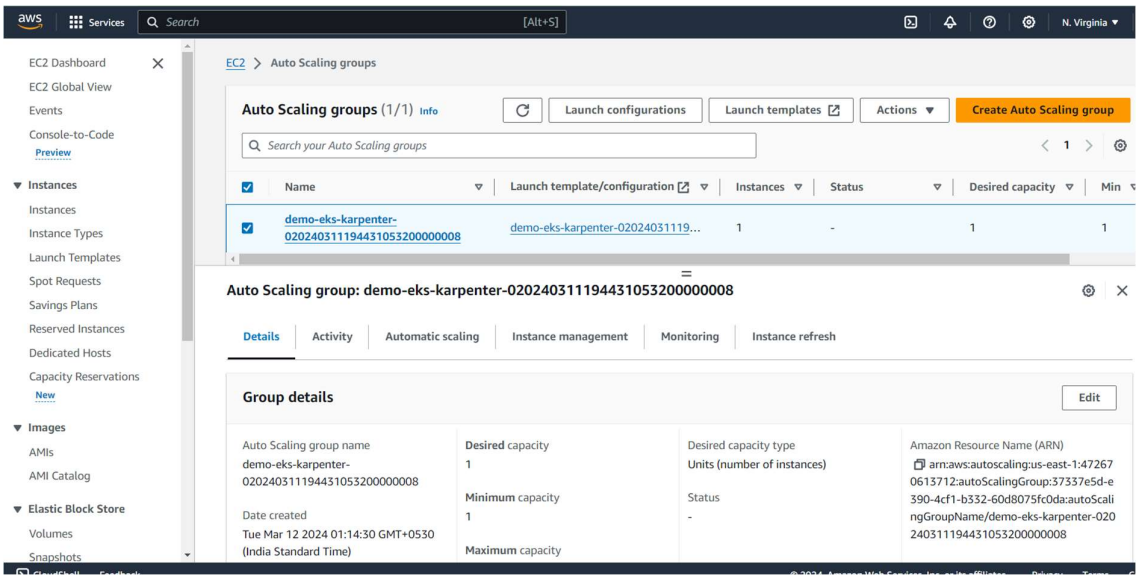
Finally, create the deployement with 4 replicas.



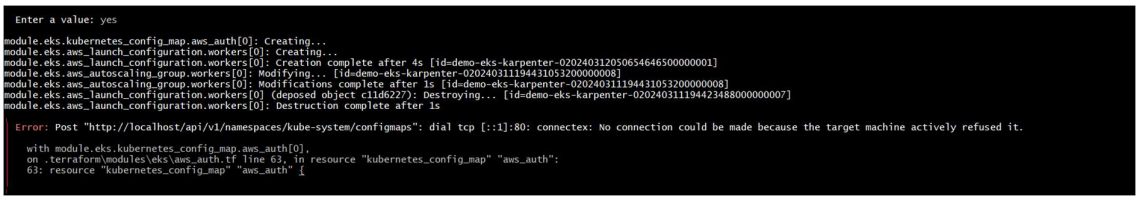

Extended support for Kubernetes versions pricing
New prices for extended support will start in the April billing cycle. For more information, see the blog post.
EKS > Clusters
New Kubernetes versions are available for 1 cluster.
Clusters (1) Info
Delete
Add cluster
Filter clusters
1
Cluster name
Status
Kubernetes version
Support type
Provider
demo-eks-karpenter
Active
1.24 Update now
Extended support until January 31, 2025
EKS
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
ENG
IN
01:39
12-03-2024


aws
Services
Search
[Alt+S]
EC2 Dashboard
EC2 Global View
Events
Console-to-Code
Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Resources
EC2 Global view
You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:
Instances (running) 1
Auto Scaling Groups 1
Dedicated Hosts 0
Elastic IPs 1
Instances 1
Key pairs 1
Load balancers 0
Placement groups 0
Security groups 7
Snapshots 0
Volumes 1





When you get a this type error then you can try to change you teraafrom main.tf file

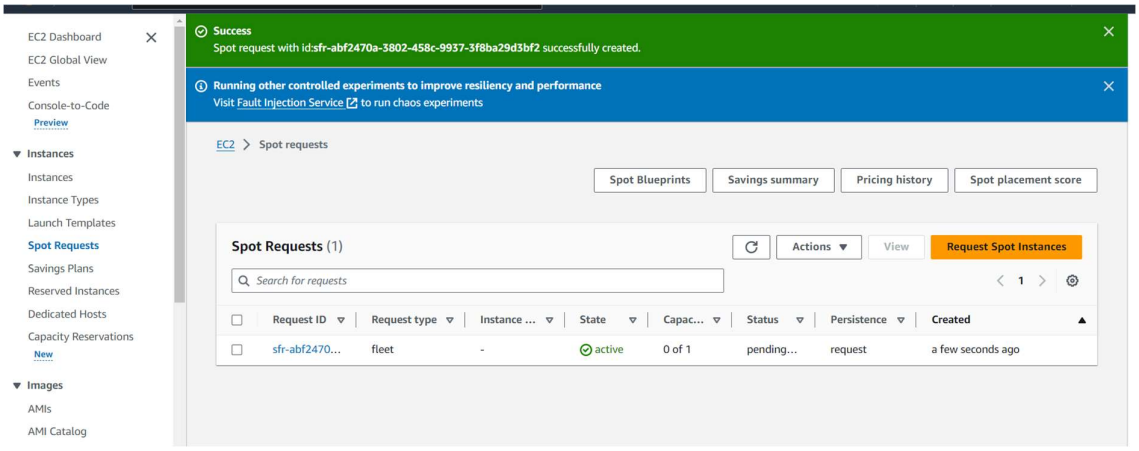


Change in main.tf

```
provider "kubernetes" {
  config_path    = "~/.kube/config"
  config_context = "arn:aws:eks:us-east-1:472670613712:cluster/demo-eks-karpenter"
}

resource "kubernetes_namespace" "New_namespace" {
  metadata {
    name = "my-first-namespace"
  }
}
```

```
  }

Plan: 2 to add, 0 to change, 0 to destroy.

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

kubernetes_namespace.New_namespace: Creating...
module.eks.kubernetes_config_map.aws_auth[0]: Creating...
kubernetes_namespace.New_namespace: Creation complete after 4s [id=my-first-namespace]
module.eks.kubernetes_config_map.aws_auth[0]: Creation complete after 2s [id=kube-system/aws-auth]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

USER@LAPTOP-PHNKO9FM MINGW64 /d/6 SEM/Terraform/aws-eks-karpenter (master)
$
```

Now we have the cluster ready. Now we need to install Karpenter in this EKS Cluster. Before we can install the Karpenter using the helm chart, we need to do Create an IAM role to give necessary access to Karpenter for managing the nodes like launching nodes, terminating nodes etc.



```
USER@LAPTOP-PHNKO9FM MINGW64 /d/6 SEM/Terrafrom_Karpenter_New (master)
$ kubectl apply -f deployment.yaml
deployment.apps/nginx-deployment unchanged

USER@LAPTOP-PHNKO9FM MINGW64 /d/6 SEM/Terrafrom_Karpenter_New (master)
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-56bb996c87-2x8f9   1/1     Running   0          56m
nginx-deployment-56bb996c87-5rcbc   0/1     Pending   0          56m
nginx-deployment-56bb996c87-dm49v   1/1     Running   0          56m
nginx-deployment-56bb996c87-mm5kn   1/1     Running   0          56m
nginx-deployment-56bb996c87-tgpww   1/1     Running   0          56m

USER@LAPTOP-PHNKO9FM MINGW64 /d/6 SEM/Terrafrom_Karpenter_New (master)
$ kubectl get nodes
NAME                           STATUS   ROLES    AGE    VERSION
ip-10-0-113-177.ec2.internal   Ready    <none>   56m    v1.22.17-eks-0a21954
ip-10-0-114-49.ec2.internal    Ready    <none>   56m    v1.22.17-eks-0a21954
ip-10-0-44-110.ec2.internal    Ready    <none>   123m   v1.22.17-eks-0a21954

USER@LAPTOP-PHNKO9FM MINGW64 /d/6 SEM/Terrafrom_Karpenter_New (master)
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-56bb996c87-2x8f9   1/1     Running   0          57m
nginx-deployment-56bb996c87-5rcbc   0/1     Pending   0          57m
nginx-deployment-56bb996c87-dm49v   1/1     Running   0          57m
nginx-deployment-56bb996c87-mm5kn   1/1     Running   0          57m
nginx-deployment-56bb996c87-tgpww   1/1     Running   0          57m

USER@LAPTOP-PHNKO9FM MINGW64 /d/6 SEM/Terrafrom_Karpenter_New (master)
$
```