

Implementing AWS EKS Karpenter Using Terraform

Table of Contents

S.no	Content	Page no
1	What is Karpenter, AWS EKS, Terraform	2 – 3
2	Cluster Autoscaller & Karpenter & AWS Fargate	3 – 4
3	Create AWS VPC Using Terraform	4 – 9
4	Create EKS Cluster Using Terraform	9 – 11
5	Create Karpenter Controller IAM Role	11 – 13
6	Deploy Karpenter to EKS	13 – 15
7	Create Karpenter Provisioner	15 – 18
8	Demo: Automatic Node Provisioning	18 – 23

1. What is Karpenter, AWS EKS, Terraform

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

Karpenter:

Karpenter is an open-source project developed by AWS that provides an advanced cluster autoscaler for Kubernetes. It extends the functionality of the native Kubernetes Cluster Autoscaler by adding support for AWS-specific features such as 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, Karpenter 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 integrates 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 a managed Kubernetes service offered by AWS (Amazon Web Service). It simplifies the process of deploying, managing, and scaling a 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 its 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 CloudWatch. 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 a 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 Autoscaler & Karpenter & AWS Fargate

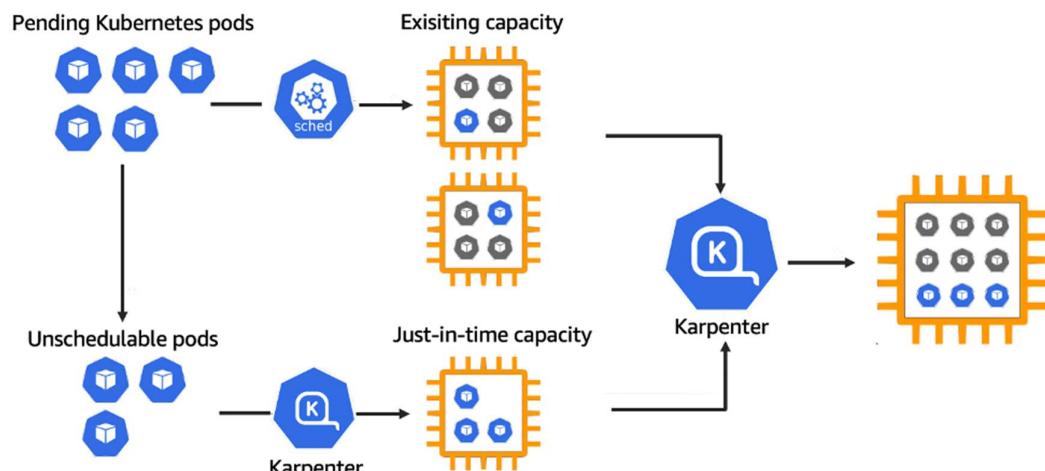
- ⇒ Cluster AutoScaler, Karpenter and AWS Fargate are all tools and services related to container orchestration and management, commonly used in the context of Kubernetes.

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.

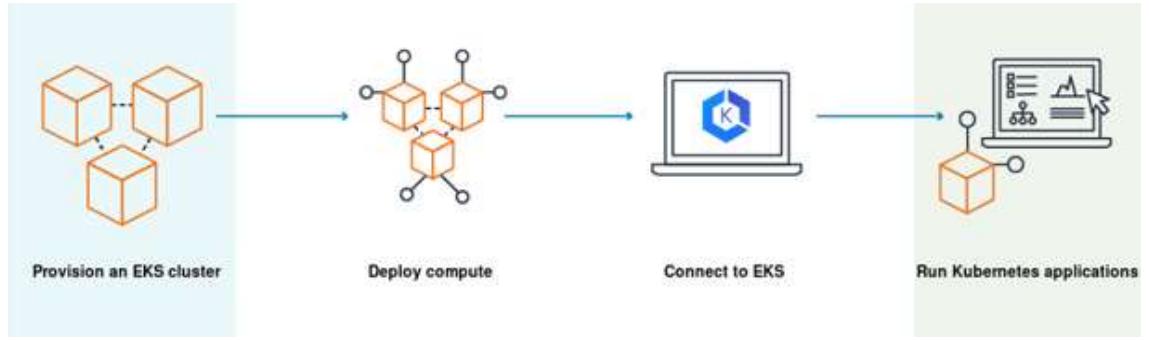
Karpenter Karpenter is an open-source project developed by AWS that provides automated node provisioning for Kubernetes clusters. It enables users to provision and manage the underlying infrastructure such as AWS EC2 instances in AWS required to run Kubernetes workloads. Karpenter works with the Cluster 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.



```
File Edit Selection View Go Run Terminal Help < > Terraform_Karpenter_New
EXPLORER ... 8-iam-oldctf 9-karpenter-controller-role.tf controller-trust-policy.json 10-karpenter-helm.tl provisioner.yaml deployment.yaml terraform.lock.hcl
> TERRAFORM PROVIDER ...
> .terraform provider...
> aws
> helm
> tls
> terraform.lock.hcl
1 # This file is maintained automatically by "terraform init".
2 # Manual edits may be lost in future updates.
3
4 provider "registry.terraform.io/hashicorp/aws" {
5   version  = "4.6.7"
6   constraints = "> 4.0"
7   hashes = [
8     "hh:08a3017ecc24352bf245f2fcfe9d9c25b58e50d516877b3ffee3bef34f860",
9     "zh:1987609846a0de019834ec569a6448db0c2518b8a71b5c8a7b07b244febddac6",
10    "dh:24959568b2ad8dcfa242e36ee7e91fc070e6144f418048c4ce2d0ba5183",
11    "ah:4a002990b94fd6d22d8c2bf1bb88780ffef7919995d9c1ff79aaef8f1",
12    "mn:559a2b5ca0687c0dec1e3512562f7a51e930674b1c2f6f06e29",
13    "gh:6ab07d1a3bb6057595d4d219b5fda874c7f3469b9cfa17d0fedfb4deefef4b7",
14    "sh:768b3bf126c3b7d975c7e50d2b207e9f9997c141aa38c3262642ab043",
15    "bh:7e65177e69984b547083c138b97742d7fe68487cef49ec0d94dbf9d1362",
16    "rb:8b562a818915fbd8b959257095215a0c5f6f467caa1ba95c58ba5f6e043f9b",
17    "hb:9b1a845846a07f811a4b2e3d88ad1a3fbcc0b580d04f4a25",
18    "ch:9c385d01a58b054e2adfd5279c8bc7cbd2d6c5c7d6a333e61092331f38af7cf",
19    "sh:b3c4a5f2821a894f17787d975c7e50d2b207e9f9997c141ab273d9555ad32b5a198bb4816b3b",
20    "dh:dac317f1db2a699628d5cab3598a704ac81d3825ea1d305bb3bbd032b1c6addfae0c",
21    "zh:dc7d4386228d85cab3598a704ac81d3825ea1d305bb3bbd032b1c6addfae0c",
22    "fb:fac0d2deadf9ec53d8a792f6666e1e73a03a611c57cbc4b86ac2821619b1d",
23  ]
24 }
25
26 provider "registry.terraform.io/hashicorp/helm" {
27   version  = "2.12.1"
28   constraints = "> 2.6"
29   hashes = [
30     "hh:xw1Va6ab/XWfTr2h3i50Lj6g9zTe4VavSnyKw3f9AI=",
31     "zh:6d23fb1662703f2fe786de3c795d848c776406eccc57a76784d088807b15004",
32     "dh:75a53hcrkjh3a2A14875130a3f7fcaad5e6f5a6f170303a3747a19ab085138"
```

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

The screenshot shows the VS Code interface with the Terraform extension active. The left sidebar is the Explorer view, showing a tree structure of Terraform files. The main editor area displays the contents of the file `0-provider.tf`. The code defines an AWS provider with the region set to "us-east-1". It also specifies required providers for AWS and Helm.

```
variable "cluster_name" {
  default = "demo"
}

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

terraform {
  required_version = "~> 1.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
    helm = {
      source  = "hashicorp/helm"
      version = "~> 2.6"
    }
  }
}
```

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

The screenshot shows the VS Code interface with the Terraform extension active. The left sidebar is the Explorer view, showing a tree structure of Terraform files. The main editor area displays the contents of the file `1-vpc.tf`. The code defines an AWS VPC resource with specific parameters for enabling EFS support.

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  # Must be enabled for EFS
  enable_dns_support  = true
  enable_dns_hostnames = true

  tags = {
    Name = "main"
  }
}
```

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



```
1 1-vpc.tf
2 2-igw.tf
3 3-subnets.tf
4 4-nat.tf
5 5-routes.tf
6 6-eks.tf
9
10 provider "aws" {
11   region = "us-east-1"
12 }
13
14
```

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.



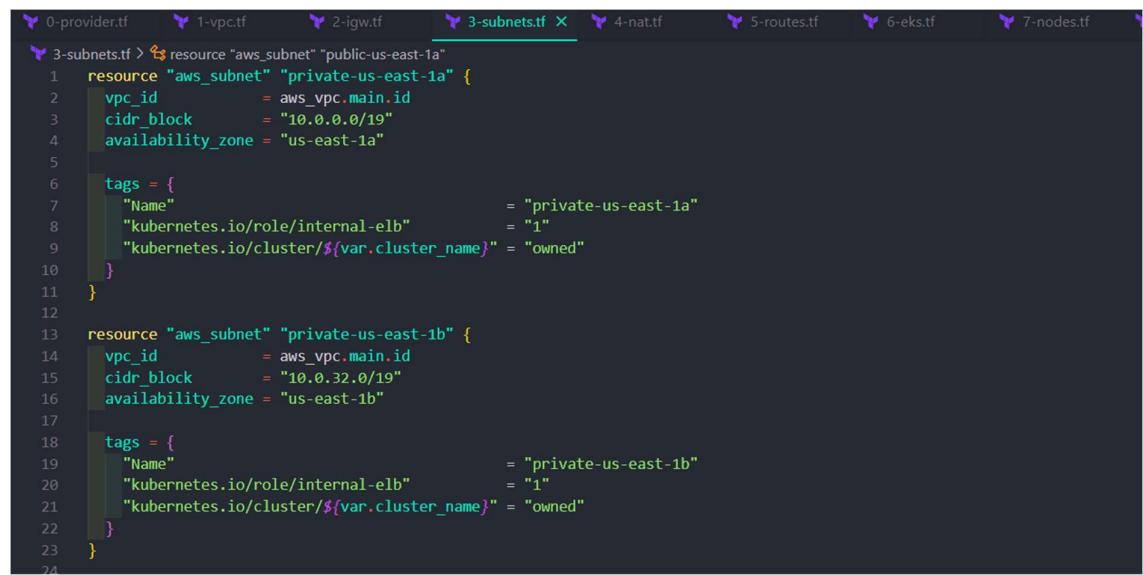
```
EXPLORER ... 0-provider.tf 1-vpc.tf 2-igw.tf 3-subnets.tf 4
> OPEN EDITORS 1 unsaved
< TERRAFORM_KARPENTER...
  .terraform\provider
    aws
    helm
    tls
    .terraform.lock.hcl
  0-provider.tf
```

2-igw.tf > resource "aws_internet_gateway" "igw" {
 vpc_id = aws_vpc.main.id
 tags = {
 Name = "igw"
 }
}

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



```
0-provider.tf 1-vpc.tf 2-igw.tf 3-subnets.tf 4-nat.tf 5-routes.tf 6-eks.tf 7-nodes.tf
3-subnets.tf > resource "aws_subnet" "private-us-east-1a" {
  vpc_id      = aws_vpc.main.id
  cidr_block  = "10.0.0.0/19"
  availability_zone = "us-east-1a"

  tags = {
    "Name"          = "private-us-east-1a"
    "kubernetes.io/role/internal-elb" = "1"
    "kubernetes.io/cluster/${var.cluster_name}" = "owned"
  }
}

resource "aws_subnet" "private-us-east-1b" {
  vpc_id      = aws_vpc.main.id
  cidr_block  = "10.0.32.0/19"
  availability_zone = "us-east-1b"

  tags = {
    "Name"          = "private-us-east-1b"
    "kubernetes.io/role/internal-elb" = "1"
    "kubernetes.io/cluster/${var.cluster_name}" = "owned"
  }
}
```

```

24
25 resource "aws_subnet" "public-us-east-1a" {
26   vpc_id           = aws_vpc.main.id
27   cidr_block       = "10.0.64.0/19"
28   availability_zone = "us-east-1a"
29   map_public_ip_on_launch = true
30
31   tags = {
32     "Name"          = "public-us-east-1a"
33     "kubernetes.io/role/elb" = "1"
34     "kubernetes.io/cluster/${var.cluster_name}" = "owned"
35   }
36
37
38 resource "aws_subnet" "public-us-east-1b" {
39   vpc_id           = aws_vpc.main.id
40   cidr_block       = "10.0.96.0/19"
41   availability_zone = "us-east-1b"
42   map_public_ip_on_launch = true
43
44   tags = {
45     "Name"          = "public-us-east-1b"
46     "kubernetes.io/role/elb" = "1"
47     "kubernetes.io/cluster/${var.cluster_name}" = "owned"
48   }
49
50

```

Create a NAT Gateway

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

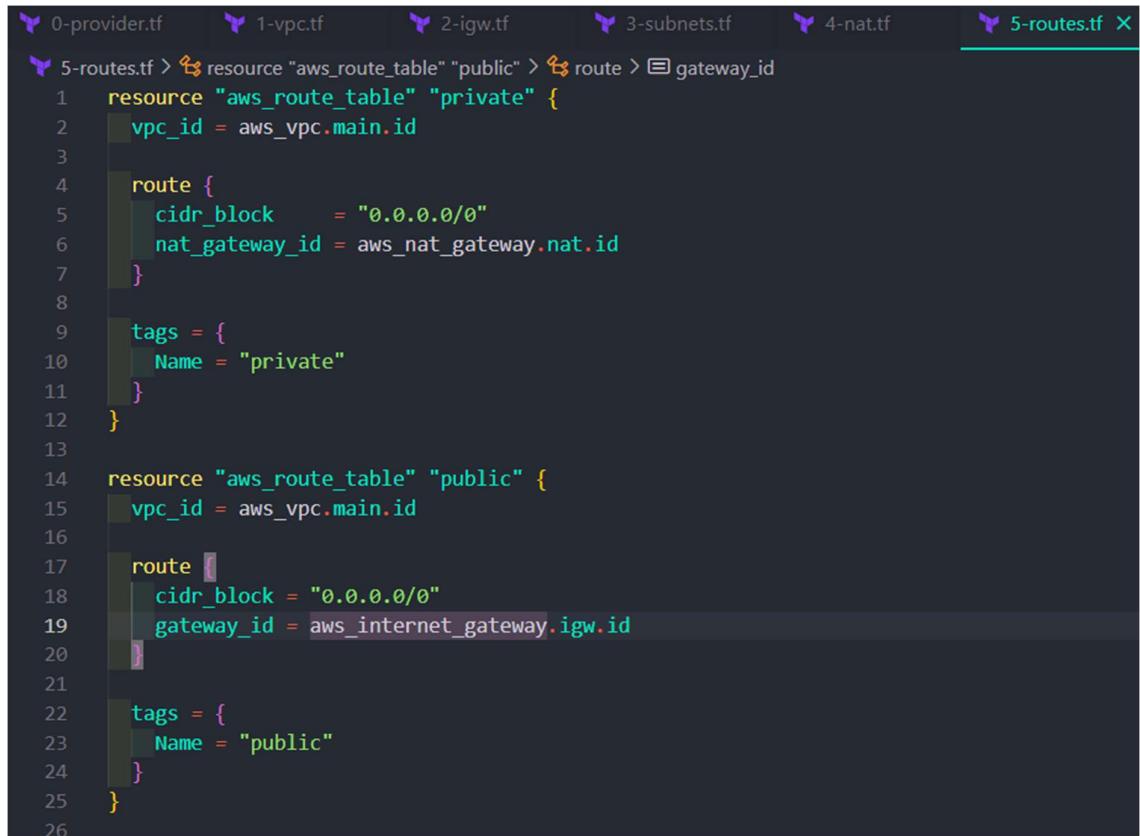
```

1  resource "aws_eip" "nat" {
2    vpc = true
3
4    tags = {
5      Name = "nat"
6    }
7
8
9    resource "aws_nat_gateway" "nat" {
10      allocation_id = aws_eip.nat.id
11      subnet_id     = aws_subnet.public-us-east-1a.id
12
13      tags = {
14        Name = "nat"
15      }
16
17      depends_on = [aws_internet_gateway.igw]
18    }
19

```

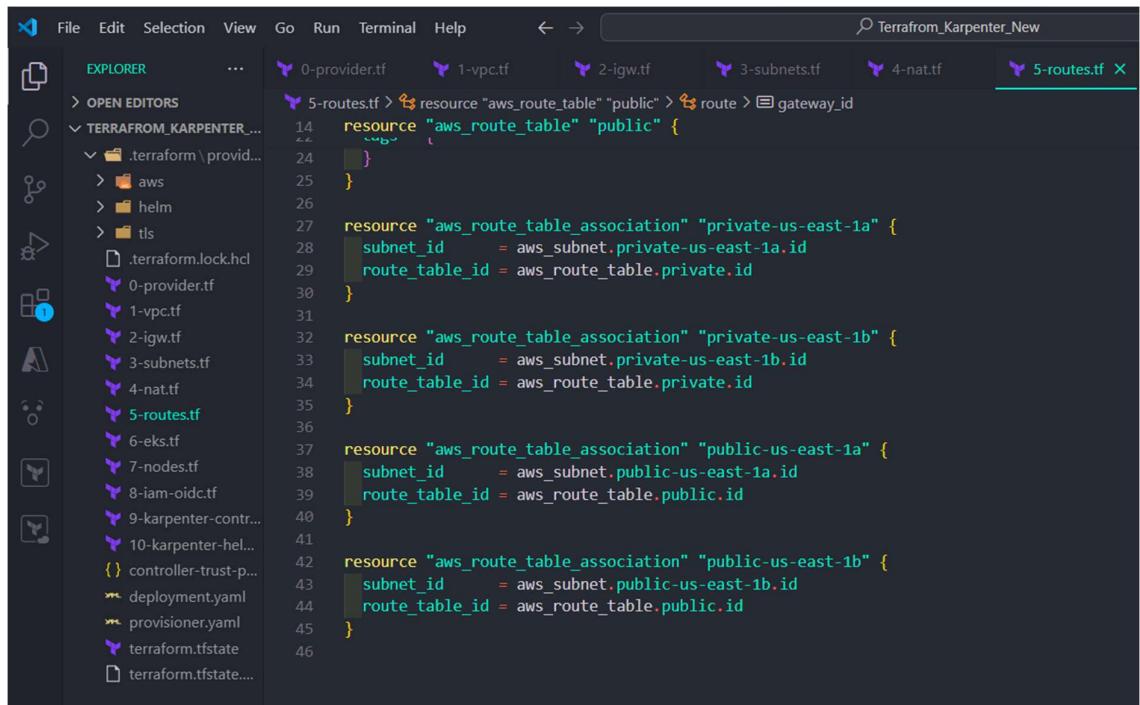
Create a Route

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



```
0-provider.tf    1-vpc.tf      2-igw.tf      3-subnets.tf   4-nat.tf      5-routes.tf X
5-routes.tf > resource "aws_route_table" "public" > route > gateway_id
1   resource "aws_route_table" "private" {
2     vpc_id = aws_vpc.main.id
3
4     route {
5       cidr_block      = "0.0.0.0/0"
6       nat_gateway_id = aws_nat_gateway.nat.id
7     }
8
9     tags = {
10    Name = "private"
11  }
12}
13
14 resource "aws_route_table" "public" {
15   vpc_id = aws_vpc.main.id
16
17   route {
18     cidr_block = "0.0.0.0/0"
19     gateway_id = aws_internet_gateway.igw.id
20   }
21
22   tags = {
23     Name = "public"
24   }
25}
26
```

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



```
File Edit Selection View Go Run Terminal Help ← → ⌘ Terraform_Karpenter_New
EXPLORER ... 0-provider.tf    1-vpc.tf      2-igw.tf      3-subnets.tf   4-nat.tf      5-routes.tf X
TERRAFORM_KARPENTER... .terraform\provider
  aws
  helm
  tls
  terraform.lock.hcl
  0-provider.tf
  1-vpc.tf
  2-igw.tf
  3-subnets.tf
  4-nat.tf
  5-routes.tf
  6-eks.tf
  7-nodes.tf
  8-iam-oidc.tf
  9-karpenter-contr...
  10-karpenter-hel...
  controller-trust-p...
  deployment.yaml
  provisioner.yaml
  terraform.tfstate
  terraform.tfstate....
```

```
5-routes.tf > resource "aws_route_table" "public" > route > gateway_id
14   resource "aws_route_table" "public" {
15     ...
16   }
17
18   resource "aws_route_table_association" "private-us-east-1a" {
19     subnet_id      = aws_subnet.private-us-east-1a.id
20     route_table_id = aws_route_table.private.id
21   }
22
23   resource "aws_route_table_association" "private-us-east-1b" {
24     subnet_id      = aws_subnet.private-us-east-1b.id
25     route_table_id = aws_route_table.private.id
26   }
27
28   resource "aws_route_table_association" "public-us-east-1a" {
29     subnet_id      = aws_subnet.public-us-east-1a.id
30     route_table_id = aws_route_table.public.id
31   }
32
33   resource "aws_route_table_association" "public-us-east-1b" {
34     subnet_id      = aws_subnet.public-us-east-1b.id
35     route_table_id = aws_route_table.public.id
36   }
```

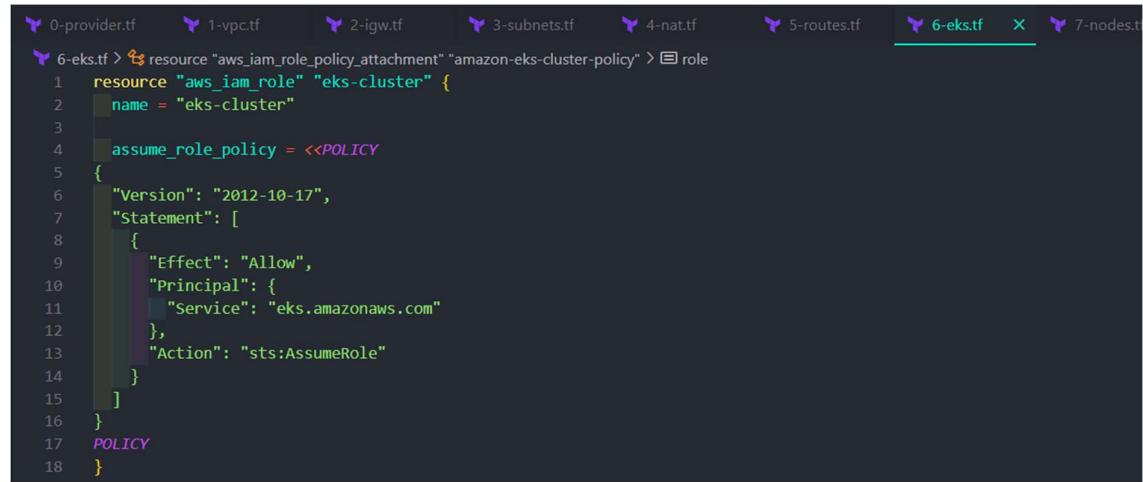
Let's initialize terraform and create all those components with terraform apply

```
terraform init
```

```
terraform apply
```

4. Create EKS Cluster Using Terraform

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

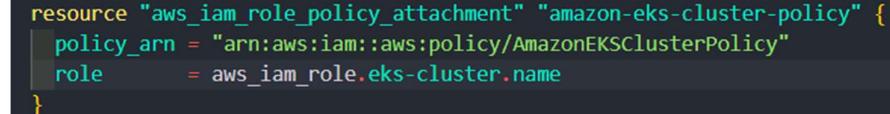


```
resource "aws_iam_role" "eks-cluster" {
  name = "eks-cluster"

  assume_role_policy = <>POLICY<>

  policy = [
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "eks.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  ]
}
```

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



```
resource "aws_eks_cluster" "cluster" {
  name     = var.cluster_name
  version  = "1.22"
  role_arn = aws_iam_role.eks-cluster.arn

  vpc_config {
    endpoint_private_access = false
    endpoint_public_access   = true
    public_access_cidrs     = ["0.0.0.0/0"]

    subnet_ids = [
      aws_subnet.private-us-east-1a.id,
      aws_subnet.private-us-east-1b.id,
      aws_subnet.public-us-east-1a.id,
      aws_subnet.public-us-east-1b.id
    ]
  }

  depends_on = [aws_iam_role_policy_attachment.amazon-eks-cluster-policy]
}
```

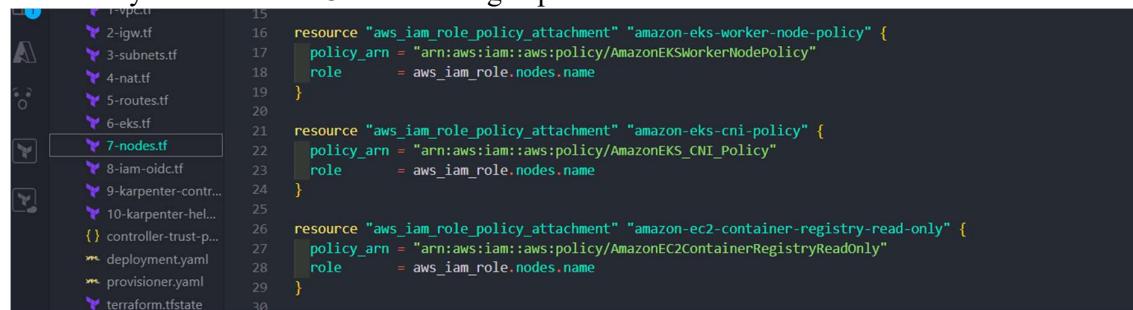
Now we need to create a Kubernetes nodes,

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.



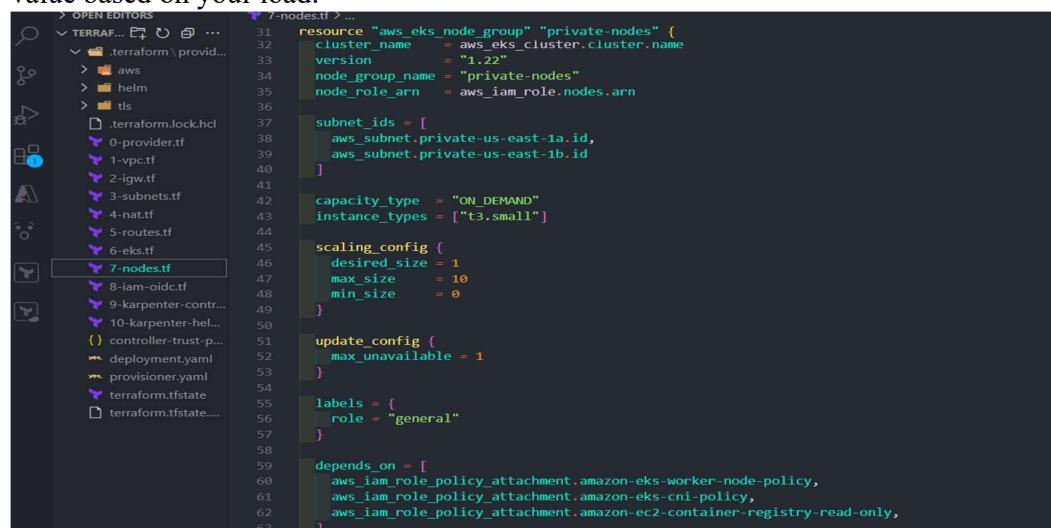
```
> OPEN EDITORS
  ✓ TERRAFORM PROVIDER ... .terraform\provider...
    ✓ aws
    ✓ helm
    ✓ tls
      .terraform.lock.hcl
    ✓ 0-provider.tf
    ✓ 1-vpc.tf
    ✓ 2-igw.tf
    ✓ 3-subnets.tf
    ✓ 4-nat.tf
    ✓ 5-routes.tf
    ✓ 6-eks.tf
    ✓ 7-nodes.tf > ...
      resource "aws_iam_role" "nodes" {
        name = "eks-node-group"
        assume_role_policy = jsonencode({
          Statement = [
            {
              Action = "sts:AssumeRole"
              Effect = "Allow"
              Principal = {
                Service = "ec2.amazonaws.com"
              }
            }
          ]
        })
        Version = "2012-10-17"
      }
```

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



```
15 resource "aws_iam_role_policy_attachment" "amazon-eks-worker-node-policy" {
16   policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
17   role       = aws_iam_role.nodes.name
18 }
19
20 resource "aws_iam_role_policy_attachment" "amazon-eks-cni-policy" {
21   policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
22   role       = aws_iam_role.nodes.name
23 }
24
25 resource "aws_iam_role_policy_attachment" "amazon-ec2-container-registry-read-only" {
26   policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
27   role       = aws_iam_role.nodes.name
28 }
29
30 }
```

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.



```
> OPEN EDITORS
  ✓ TERRAFORM PROVIDER ... .terraform\provider...
    ✓ aws
    ✓ helm
    ✓ tls
      .terraform.lock.hcl
    ✓ 0-provider.tf
    ✓ 1-vpc.tf
    ✓ 2-igw.tf
    ✓ 3-subnets.tf
    ✓ 4-nat.tf
    ✓ 5-routes.tf
    ✓ 6-eks.tf
    ✓ 7-nodes.tf > ...
      resource "aws_eks_node_group" "private-nodes" {
        cluster_name  = aws_eks_cluster.cluster.name
        version       = "1.22"
        node_group_name = "private-nodes"
        node_role_arn = aws_iam_role.nodes.arn
        subnet_ids   = [
          aws_subnet.private-us-east-1a.id,
          aws_subnet.private-us-east-1b.id
        ]
        capacity_type = "ON_DEMAND"
        instance_types = ["t3.small"]
        scaling_config {
          desired_size = 1
          max_size     = 10
          min_size     = 0
        }
        update_config {
          max_unavailable = 1
        }
        labels = {
          role = "general"
        }
        depends_on = [
          aws_iam_role_policy_attachment.amazon-eks-worker-node-policy,
          aws_iam_role_policy_attachment.amazon-eks-cni-policy,
          aws_iam_role_policy_attachment.amazon-ec2-container-registry-read-only,
        ]
      }
```

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`

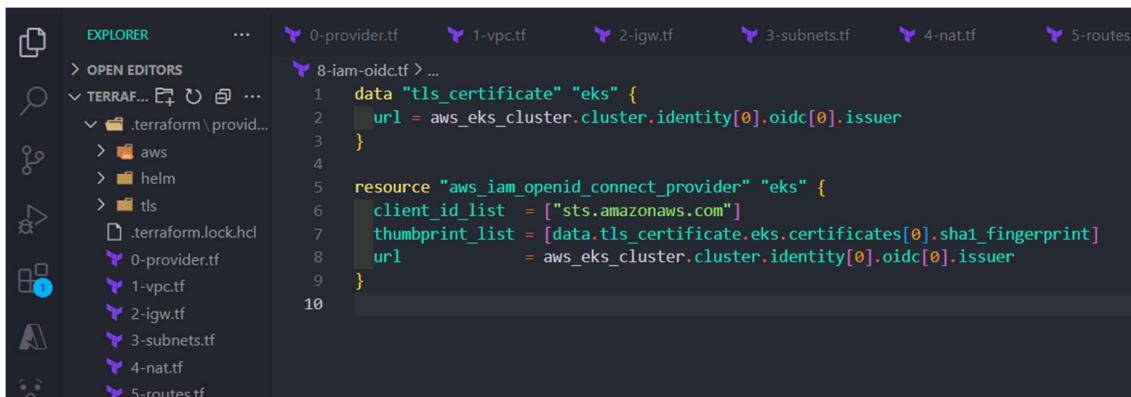
As I mentioned before, if you decide to create a separate IAM role and instance profile you would need to edit the auth configmap to add the ARN of the new role.

⇒ `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 nodes. 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.



```
data "tls_certificate" "eks" {
  url = aws_eks_cluster.cluster.identity[0].oidc[0].issuer
}

resource "aws_iam_openid_connect_provider" "eks" {
  client_id_list = ["sts.amazonaws.com"]
  thumbprint_list = [data.tls_certificate.eks.certificates[0].sha1_fingerprint]
  url           = aws_eks_cluster.cluster.identity[0].oidc[0].issuer
}
```

Get the certificate and use it in terraform resource.

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

```

1-vpc.tf    2-igw.tf    3-subnets.tf    4-nat.tf    5-routes.tf    6-eks.tf    7-nodes.tf
9-karpenter-controller-role.tf > resource "aws_iam_policy" "karpenter_controller"
1   data "aws_iam_policy_document" "karpenter_controller_assume_role_policy" {
2     statement {
3       actions = ["sts:AssumeRoleWithWebIdentity"]
4       effect  = "Allow"
5     }
6     condition {
7       test    = "StringEquals"
8       variable = "${replace(aws_iam_openid_connect_provider.eks.url, "https://", "")}:sub"
9       values   = ["system:serviceaccount:karpenter:karpenter"]
10    }
11   principals {
12     identifiers = [aws_iam_openid_connect_provider.eks.arn]
13     type        = "Federated"
14   }
15 }
16 }
17 }
18
19
20
21
22
23

```

Then create a karpenter controller role and attach that policy

```

18
19   resource "aws_iam_role" "karpenter_controller" {
20     assume_role_policy = data.aws_iam_policy_document.karpenter_controller_assume_role_policy.json
21     name               = "KarpenterController"
22   }
23

```

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

```

22   }
23
24   resource "aws_iam_policy" "karpenter_controller" {
25     policy = file("./controller-trust-policy.json")
26     name   = "KarpenterController"
27   }
28

```

Then Attached it to the role as well.

```

29   resource "aws_iam_role_policy_attachment" "aws_load_balancer_controller_attach" {
30     role      = aws_iam_role.karpenter_controller.name
31     policy_arn = aws_iam_policy.karpenter_controller.arn
32   }
33

```

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.

```

33
34   resource "aws_iam_instance_profile" "karpenter" {
35     name = "KarpenterNodeInstanceProfile"
36     role = aws_iam_role.nodes.name
37   }
38

```

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

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying the project structure under 'TERRAFORM_KARPENTER...' which includes providers (aws, helm, tls), Terraform files (0-provider.tf, 1-vpc.tf, 2-igw.tf, 3-subnets.tf, 4-nat.tf, 5-routes.tf, 6-eks.tf, 7-nodes.tf, 8-iam-oidc.tf, 9-karpenter-controller-role.tf, 10-karpenter-helm.tf), deployment.yaml, provisioner.yaml, and terraform.tfstate. The main editor pane shows the content of controller-trust-policy.json, which defines two statements for the 'Karpenter' role. The first statement allows various EC2 actions, and the second statement, under a condition where the resource has a tag 'karpenter', allows EC2 termination.

```

{
  "Statement": [
    {
      "Action": [
        "ssm:GetParameter",
        "iam:PassRole",
        "ec2:RunInstances",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceTypes",
        "ec2:DescribeInstanceTypeOfferings",
        "ec2:DescribeAvailabilityZones",
        "ec2:DeleteLaunchTemplate",
        "ec2:CreateTags",
        "ec2:CreateLaunchTemplate",
        "ec2:CreateFleet"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Sid": "Karpenter"
    },
    {
      "Action": "ec2:TerminateInstances",
      "Condition": {
        "StringLike": {
          "ec2:ResourceTag/Name": "*karpenter*"
        }
      },
      "Effect": "Allow",
      "Resource": "*",
      "Sid": "ConditionalEC2Termination"
    }
  ]
}

```

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.

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying the project structure under 'TERRAFORM_KARPENTER...' which includes providers (aws, helm, tls), Terraform files (0-provider.tf, 1-vpc.tf, 2-igw.tf, 3-subnets.tf, 4-nat.tf, 5-routes.tf, 6-eks.tf, 7-nodes.tf, 8-iam-oidc.tf, 9-karpenter-controller-role.tf, 10-karpenter-helm.tf), deployment.yaml, provisioner.yaml, and terraform.tfstate. The main editor pane shows the content of 10-karpenter-helm.tf, which defines a provider block for 'helm' using the AWS EKS provider. It specifies the host as the AWS EKS cluster endpoint and uses base64decode to get the certificate authority from the cluster's certificate authority data.

```

provider "helm" {
  kubernetes {
    host           = aws_eks_cluster.cluster.endpoint
    cluster_ca_certificate = base64decode(aws_eks_cluster.cluster.certificate_authority[0].data)
  }

  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name", aws_eks_cluster.cluster.id]
    command     = "aws"
  }
}

```

```
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 carpenter pod in it's dedicated namespace.

```
kubectl get pods -n carpenter
```

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: carpenter.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: carpenter.k8s.aws/instance-family  
      operator: In  
      values: [c5, m5, r5]  
    # Exclude small instance sizes  
    - key: carpenter.k8s.aws/instance-size  
      operator: NotIn  
      values: [nano, micro, small, large]  
  providerRef:  
    name: my-provider  
---  
apiVersion: carpenter.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.

The screenshot shows the AWS EC2 Instances page with the following details:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
i-0733d17e0732ea9aa	i-0733d17e0732ea9aa	Running	t1.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-52-205-37-1
i-0f839035709f2fd58	i-0f839035709f2fd58	Running	t3.small	2/2 checks passed	View alarms +	us-east-1b	-
i-01139d8196ee4fd9f	i-01139d8196ee4fd9f	Running	c5.4xlarge	2/2 checks passed	View alarms +	us-east-1b	ec2-18-208-232-0
i-0ee9556ac4b1460b0	i-0ee9556ac4b1460b0	Running	c5.2xlarge	2/2 checks passed	View alarms +	us-east-1b	ec2-5-88-24-26-0
demo-eks-kar...	i-00c667096c5943f8d	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c	-

Autoscaler Group created

The screenshot shows the AWS Auto Scaling Groups page with the following details:

Auto Scaling group name	Desired capacity	Desired capacity type	Amazon Resource Name (ARN)
demo-eks-karpenter-020240311194431053200000008	1	Units (number of instances)	arn:aws:autoscaling:us-east-1:472670613712:autoScalingGroup:37337e5de390-4cf1-b332-60d8075fc0da:autoScalingGroupName/demo-eks-karpenter-020240311194431053200000008
Date created	Minimum capacity	Status	
Tue Mar 12 2024 01:14:30 GMT+0530 (India Standard Time)	1	-	
	Maximum capacity		

Also VPC created on AWS

The screenshot shows the AWS VPCs page with the following details:

VPC ID	Name	State	IPv4 CIDR	IPv6 CIDR	DHCP options
vpc-0e77cab9117a02cb6	demo-eks-karpenter	Available	10.0.0.0/16	-	dopt-0c
vpc-0618bd55044d82c33	main	Available	10.0.0.0/16	-	dopt-0c
vpc-0500f61a225e301ba	-	Available	172.31.0.0/16	-	dopt-0c

vpc-0e77cab9117a02cb6 / demo-eks-karpenter

Details

VPC ID vpc-0e77cab9117a02cb6	State Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP option set dopt-0ca47861c6f0c51eb	Main route table rtb-0cfba01e93e89c116	Main network ACL acl-0c24221e641a9a11c
Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool -	IPv6 CIDR (Network border group) -
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups -	Owner ID 472670613712	

Subnets, Route Table, Internet Gateway are created on AWS

Subnets (16) Info

Name	Subnet ID	State	VPC	IPv4 CIDR
-	subnet-06aa540e277f62fb	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-0d6385765bachbd15	Available	vpc-0618bd55044d82c33 main	10.0.64.0/19
-	subnet-005bdc2c2a863c779	Available	vpc-0500f61a225e301ba	172.31.64.0/20
-	subnet-0b9a22bdec375b17e4	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-0fff945626e51bc4	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

Name	Route table ID	Explicit subnet associations	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-04d27ff9d54af939d6	3 subnets	-	No	vpc-0e77cab9117a02cb6

Internet gateways (3) Info

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

Name	NAT gateway ID	Connectivity...	State	State message	Primary public IP	Primary priva...
nat	nat-05106c745506b678f	Public	Available	-	50.16.208.139	10.0.65.123
demo-eks-karpenter...	nat-072d06969a5fc0e3	Public	Available	-	52.54.121.147	10.0.101.57

Finally, AWS EKS Created

The screenshot shows the AWS EKS Cluster Details page for a cluster named 'eks-demos'. The 'Cluster info' section indicates the cluster is Active, running Kubernetes version 1.22, with End of support and EKS as the provider. The 'Overview' tab is selected, showing tabs for Resources, Compute, Networking, Add-ons, Access, Observability, Upgrade insights (1), and Update history. The 'Details' section contains fields for API server endpoint, OpenID Connect provider URL, Created date (March 12, 2024), Cluster ARN, and Platform version (eks.24). It also lists Certificate authority and Cluster IAM role ARN.

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
1  ---
2  apiVersion: apps/v1
3  kind: Deployment
4  metadata:
5    name: nginx-deployment
6    labels:
7      app: nginx
8  spec:
9    replicas: 5
10   selector:
11     matchLabels:
12       app: nginx
13   template:
14     metadata:
15       labels:
16         app: nginx
17     spec:
18       containers:
19         - name: nginx
20           image: nginx:1.14.2
21           resources:
22             requests:
23               cpu: "4"
24               memory: 4Gi
25             ports:
26               - containerPort: 80
27

```

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 deployment with 4 replicas.

```
MINGW64 /d/ SEM/Terraform_Karpenter_New - □ X
since it previously failed to schedule, adding: toleration for PreferNoSchedule taints ["commit": "0e43-04-1413-42:01.6572", "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "toleration": {"key": "node-role.kubernetes.io/master", "operator": "Equal", "value": "true", "duration": 0, "retries": 0}, "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "version": "0.1.0"}, "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "version": "0.1.0"}]
2024-04-1413-42:01.6572 ERROR controller.provisioning Could not schedule pod, incompatible with provisioner "default", no instance type satisfied resources ["cpu": "4", "memory": "4Gi"] "pod[\"name\": \"nginx-deployment-56bb996c87-tgpmw\", \"spec\": {\"containers\": [\"nginx\", {\"image\": \"nginx:latest\", \"ports\": [{\"containerPort\": 80}], \"resources\": {\"limits\": {\"cpu\": \"4\", \"memory\": \"4Gi\"}, \"requests\": {\"cpu\": \"4\", \"memory\": \"4Gi\"}}}], \"schedulerName\": \"aws-node-scheduler\", \"taints\": [{\"key\": \"node-role.kubernetes.io/master\", \"operator\": \"Exists\", \"value\": \"\"}], \"tolerations\": [{\"key\": \"node-role.kubernetes.io/master\", \"operator\": \"Equal\", \"value\": \"true\", \"duration\": 0, \"retries\": 0}], \"taints\": [{\"key\": \"node-role.kubernetes.io/master\", \"operator\": \"Exists\", \"value\": \"\"}], \"version\": \"0.1.0\"}], \"status\": {\"conditions\": [{\"lastTransitionTime\": \"2024-04-14T13:42:01.657Z\", \"status\": \"True\", \"type\": \"SchedulingFailed\"}], \"observedGeneration\": 1, \"phase\": \"Pending\", \"reason\": \"SchedulingFailed\", \"status\": \"False\"}}]
2024-04-1413-42:01.6572 DEBUG controller.provisioning Relaxing soft constraints for pod "nginx-deployment-56bb996c87-tgpmw" because it previously failed to schedule, adding: toleration for PreferNoScheduled taints ["commit": "0e43-04-1413-42:01.6572", "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "toleration": {"key": "node-role.kubernetes.io/master", "operator": "Equal", "value": "true", "duration": 0, "retries": 0}, "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "version": "0.1.0"}, "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "version": "0.1.0"}]
2024-04-1413-42:01.6572 ERROR controller.provisioning Could not schedule pod, incompatible with provisioner "default", no instance type satisfied resources ["cpu": "4", "memory": "4Gi"] "pod[\"name\": \"nginx-deployment-56bb996c87-tgpmw\", \"spec\": {\"containers\": [\"nginx\", {\"image\": \"nginx:latest\", \"ports\": [{\"containerPort\": 80}], \"resources\": {\"limits\": {\"cpu\": \"4\", \"memory\": \"4Gi\"}, \"requests\": {\"cpu\": \"4\", \"memory\": \"4Gi\"}}}], \"schedulerName\": \"aws-node-scheduler\", \"taints\": [{\"key\": \"node-role.kubernetes.io/master\", \"operator\": \"Exists\", \"value\": \"\"}], \"tolerations\": [{\"key\": \"node-role.kubernetes.io/master\", \"operator\": \"Equal\", \"value\": \"true\", \"duration\": 0, \"retries\": 0}], \"taints\": [{\"key\": \"node-role.kubernetes.io/master\", \"operator\": \"Exists\", \"value\": \"\"}], \"version\": \"0.1.0\"}], \"status\": {\"conditions\": [{\"lastTransitionTime\": \"2024-04-14T13:42:01.657Z\", \"status\": \"True\", \"type\": \"SchedulingFailed\"}], \"observedGeneration\": 1, \"phase\": \"Pending\", \"reason\": \"SchedulingFailed\", \"status\": \"False\"}}]
2024-04-1413-42:01.6572 DEBUG controller.provisioning Relaxing soft constraints for pod "nginx-deployment-56bb996c87-tgpmw" because it previously failed to schedule, adding: toleration for PreferNoScheduled taints ["commit": "0e43-04-1413-42:01.6572", "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "toleration": {"key": "node-role.kubernetes.io/master", "operator": "Equal", "value": "true", "duration": 0, "retries": 0}, "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "version": "0.1.0"}, "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "version": "0.1.0"}]
2024-04-1413-42:01.6572 INFO controller.provisioning Found 1 provisionalable pod(s) [{}]
2024-04-1413-42:01.6572 INFO controller.provisioning Computed new node(s) will fit 1 pod(s) [{}]
2024-04-1413-42:01.6572 INFO controller.provisioning [\"nginx-deployment-56bb996c87-tgpmw\"] [{}]
2024-04-1413-42:16.8312 DEBUG controller.provisioning.cloudprovider Discovered subnets [{"subnet": "0961180f0d47bd1b5 (us-east-1b)", "subnet": "0de385765bacbd15 (us-east-1a)", "subnet": "0737831a83918503 (us-east-1a)", "subnet": "08e4cc1218e510 (us-east-1b)", "subnet": "08e4cc1218e510 (us-east-1b)", "taints": [{"key": "node-role.kubernetes.io/master", "operator": "Exists", "value": ""}], "version": "0.1.0"}]
2024-04-1413-42:16.9602 DEBUG controller.provisioning.cloudprovider Discovered security group [{"key": "sg-010e0cb240cff396", "version": "0.1.0"}]
2024-04-1413-42:16.9632 DEBUG controller.provisioning.cloudprovider Discovered kubernetes cluster [{}]
2024-04-1413-42:17.0072 DEBUG controller.provisioning.cloudprovider Discovered ami-0f03fc73f782a92e5 for query {"aws/service": "eks", "optimized ami": "1.22/amazon-linx/2/recommended,image_id": "ami-0f03fc73f782a92e5", "provisioner": "default"}
2024-04-1413-42:17.1527 DEBUG controller.provisioning.cloudprovider Created launch template "Karpenter-Template-1050842747143871" [{}]
2024-04-1413-42:17.1527 DEBUG controller.provisioning.cloudprovider [{}]
MINGW64 /d/ SEM/Terraform_Karpenter_New - □ X
nginx-deployment-56bb996c87-2x8f9 1/1 Running 0 57m
nginx-deployment-56bb996c87-mmskn 1/1 Running 0 57m
nginx-deployment-56bb996c87-tgpmw 1/1 Running 0 57m

USER@LAPTOP-PHNQK9FM MINGW64 /d/ SEM/Terraform_Karpenter_New [master]
$ kubectl get pods
NAME STATUS RESTARTS AGE
nginx-deployment-56bb996c87-2x8f9 1/1 Running 0 19h
nginx-deployment-56bb996c87-mrcfc 0/1 Pending 0 19h
nginx-deployment-56bb996c87-tgpmw 1/1 Running 0 19h
nginx-deployment-56bb996c87-mmskn 1/1 Running 0 19h
nginx-deployment-56bb996c87-tgpmw 1/1 Running 0 19h

USER@LAPTOP-PHNQK9FM MINGW64 /d/ SEM/Terraform_Karpenter_New [master]
$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
ip-10-0-113-177.ec2.internal Ready <none> 19h V1.22.17-eks-0a21954
ip-10-0-114-49.ec2.internal Ready <none> 19h V1.22.17-eks-0a21954
ip-10-0-44-110.ec2.internal Ready <none> 20h V1.22.17-eks-0a21954

USER@LAPTOP-PHNQK9FM MINGW64 /d/ SEM/Terraform_Karpenter_New [master]
$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
ip-10-0-113-177.ec2.internal Ready <none> 42h V1.22.17-eks-0a21954
ip-10-0-114-49.ec2.internal Ready <none> 42h V1.22.17-eks-0a21954
ip-10-0-44-110.ec2.internal Ready <none> 44h V1.22.17-eks-0a21954

USER@LAPTOP-PHNQK9FM MINGW64 /d/ SEM/Terraform_Karpenter_New [master]
$ kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx-deployment-56bb996c87-2x8f9 1/1 Running 0 42h
nginx-deployment-56bb996c87-mrcfc 0/1 Pending 0 42h
nginx-deployment-56bb996c87-tgpmw 1/1 Running 0 42h
nginx-deployment-56bb996c87-mmskn 1/1 Running 0 42h
nginx-deployment-56bb996c87-tgpmw 1/1 Running 0 42h

USER@LAPTOP-PHNQK9FM MINGW64 /d/ SEM/Terraform_Karpenter_New [master]
$ kubectl apply -f deployment.yaml
deployment.apps/nginx-deployment unchanged

USER@LAPTOP-PHNQK9FM MINGW64 /d/ SEM/Terraform_Karpenter_New [master]
$ kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx-deployment-56bb996c87-2x8f9 0/1 Pending 0 43h
nginx-deployment-56bb996c87-mrcfc 0/1 Pending 0 43h
nginx-deployment-56bb996c87-tgpmw 1/1 Running 0 43h
nginx-deployment-56bb996c87-mmskn 1/1 Running 0 43h
nginx-deployment-56bb996c87-tgpmw 1/1 Running 0 43h

USER@LAPTOP-PHNQK9FM MINGW64 /d/ SEM/Terraform_Karpenter_New [master]
$ |
```

① 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

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

aws | Services Search [Alt+S]

EC2 Dashboard X

EC2 Global View

Events

Console-to-Code [Preview](#)

▼ Instances

Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Resources

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		

The screenshot shows two separate AWS EC2 management pages:

- Instances (1/1) Info:** Displays a single instance named "demo-eks-kar...". It is running, has an instance type of t2.micro, and is located in the us-east-1c availability zone. It has 2/2 checks passed.
- Auto Scaling groups (1/1) Info:** Displays one Auto Scaling group named "demo-eks-kar...". It has a desired capacity of 1 and a minimum of 1. The ARN for the Auto Scaling group is listed as well.

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

```
Enter a value: yes
module.eks.kubernetes_config_map.aws_auth[0]: Creating...
module.eks.aws_launch_configuration.workers[0]: Creating...
module.eks.aws_launch_configuration.workers[0]: Creation complete after 4s [id=demo-eks-kar...-020240311194431053200000001]
module.eks.aws_launch_configuration.workers[0]: Destroying... [id=...]
module.eks.aws_auto_scaling_group[0]: Modifying... [id=...]
module.eks.aws_auto_scaling_group[0]: Complete after 1s [id=...]
module.eks.aws_launch_configuration.workers[0] (deposed object c1d6227): Destroying... [id=...]
module.eks.aws_launch_configuration.workers[0]: Destruction complete after 1s

| Error: Post "http://localhost/api/v1/namespaces/kube-system/configmaps": dial tcp [::]:80: connectex: No connection could be made because the target machine actively refused it.

  with module.eks.kubernetes_config_map.aws_auth[0],
  on .terraform/modules/eks/aws_auth.tf line 63, in resource "kubernetes_config_map" "aws_auth",
  63: resource "kubernetes_config_map" "aws_auth" {
```

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

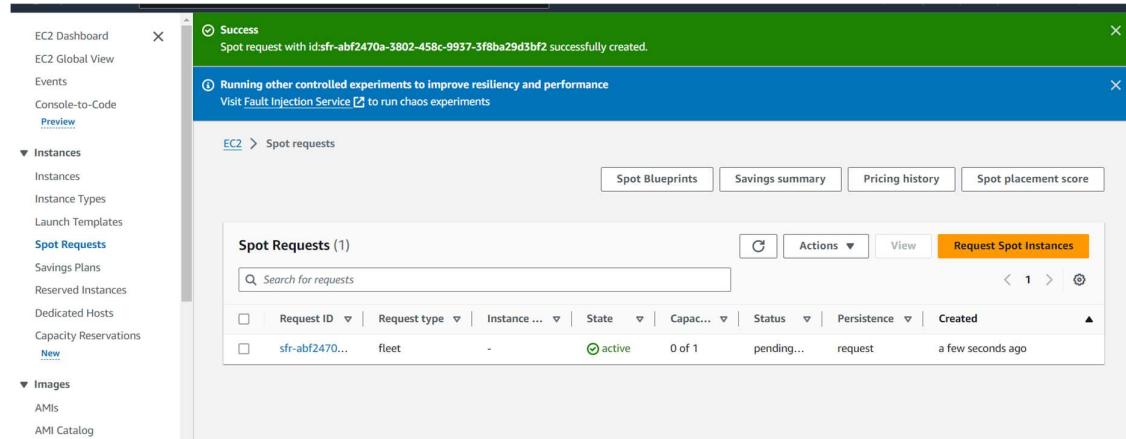
kubernetes_namespace.New_namespace: Creating...
module.eks.kubernetes_config_map.aws_auth[0]: Creating...
kubernetes_namespace.New_namespace: Creation complete after 4s [id=my-first-namespace]
module.eks.kubernetes_config_map.aws_auth[0]: Creation complete after 2s [id=kube-system/aws-auth]

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

USER@LAPTOP-PHNK09FM 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)
$ |
```