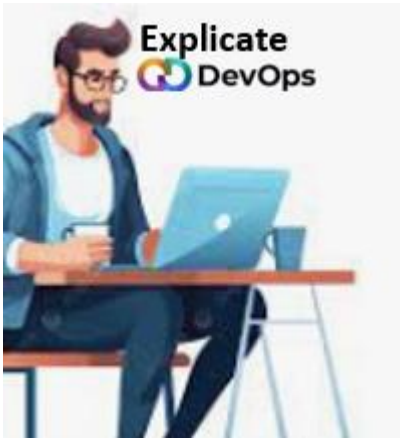
Kubernetes Autoscaler



By Ritesh Kumar Singh

Email Address: - riteshkumarsingh9559@gmail.com

LinkedIn: - https://www.linkedin.com/in/ritesh-kumar-singh-41113128b/

GitHub: - https://github.com/singhritesh85

या कुन्देन्दुतुषारहारधवला या शुभ्रवस्त्रावृता
या वीणावरदण्डमण्डितकरा या श्वेतपद्मासना।
या ब्रह्माच्युत शंकरप्रभृतिभिर्देवैः सदा वन्दिता
सा मां पातु सरस्वती भगवती निःशेषजाड्यापहा ॥

1. First Section

Kubernetes Autoscaler

1. EKS Cluster Autoscaler

As the name indicates EKS cluster autoscaler will scale the EKS Node-Group nodes between the minimum and maximum number of nodes of EKS Cluster. For the current scenario I will allow the nodes of EKS Node-Group to scale between 1 to 4. You can change it as per your project requirement.

```
scaling_config {
  desired_size = 1
  max_size     = 4
  min_size     = 1
}
```

Always remember the desired size can not be less than minimum size of the Autoscaling Group.

I used the terraform script present in the GitHub Repo https://github.com/singhritesh85/Autoscale.git at the path **Autoscale/Autoscale-EKS/Autoscale-EKS-using-cluster-autoscaler** to create the EKS Cluster and after creation of EKS Cluster run the manifests file **cluster-autoscaler.yaml** as shown in the screenshot attached below.

```
[root@ terraform-eks-withaddons-cluster-autoscaler]# kubectl apply -f cluster-autoscaler.yaml
serviceaccount/cluster-autoscaler created
clusterrole.rbac.authorization.k8s.io/cluster-autoscaler created
role.rbac.authorization.k8s.io/cluster-autoscaler created
clusterrolebinding.rbac.authorization.k8s.io/cluster-autoscaler created
rolebinding.rbac.authorization.k8s.io/cluster-autoscaler created
deployment.apps/cluster-autoscaler created
```

```
[root@ terraform-eks-withaddons-cluster-autoscaler]# kubectl get pods -n kube-system
NAME                                  READY   STATUS    RESTARTS   AGE
aws-node-                             2/2     Running   0          2m43s
cluster-autoscaler-                   1/1     Running   0          53s
coredns-                              1/1     Running   0          118s
coredns-                              1/1     Running   0          118s
ebs-csi-controller-                   6/6     Running   0          114s
ebs-csi-controller-                   6/6     Running   0          114s
ebs-csi-node-                         3/3     Running   0          114s
kube-proxy-                           1/1     Running   0          2m43s
snapshot-controller-                  1/1     Running   0          116s
snapshot-controller-                  1/1     Running   0          116s
[root@ terraform-eks-withaddons-cluster-autoscaler]# kubectl logs -f cluster-autoscaler- -n kube-system
```

As per the requirement new nodes will be added and before the Scale down it will wait for 10 minutes and then scale down.

When you will check the logs of cluster-autoscaler pod present in kube-system namespace then you can check how the scale up and scale down will be going on.

For the demonstration purpose I created a deployment named as dexter with number of pods required is 20 and I checked some of the pods are in running condition and other are in pending condition due to unavailability of compute resource.

```
[root@ ~]# kubectl apply -f deployment.yaml
deployment.apps/dexter created
```

```
[root@                ~]# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dexter
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 20
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: demo
        image: nginx
        ports:
        - containerPort: 80
```

```
[root@                ~]# kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
dexter-                   0/1     Pending   0          33s
dexter-                   1/1     Running   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   1/1     Running   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   1/1     Running   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   1/1     Running   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   0/1     Pending   0          33s
dexter-                   1/1     Running   0          33s
dexter-                   1/1     Running   0          33s
```

Then I checked the logs of cluster-autoscaler pod present in the kube-system namespace and found that two new nodes had been created and pods are in pending state launched on the nodes of the EKS Cluster.

```
[root@ ~]# kubectl get nodes
NAME                                        STATUS   ROLES    AGE     VERSION
ip-  .us-east-2.compute.internal   Ready    <none>   7m11s   v1.30.4-eks-a737599
ip-  us-east-2.compute.internal    Ready    <none>   17m     v1.30.4-eks-a737599
ip-  .us-east-2.compute.internal   Ready    <none>   7m14s   v1.30.4-eks-a737599
[root@ ~]# kubectl get pods -o wide
NAME                      READY   STATUS    RESTARTS   AGE    IP           NODE                                        NOMINATED NODE   READINESS GATES
dexter-                   1/1     Running   0          8m22s  10.  131     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  245     us-east-2.compute.internal    <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  246     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  51      .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  20      us-east-2.compute.internal    <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  238     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  190     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  215     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  136     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  91      us-east-2.compute.internal    <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  79      .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  97      .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  18      .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  45      .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  98      .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  52      us-east-2.compute.internal    <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  189     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  139     .us-east-2.compute.internal   <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  53      us-east-2.compute.internal    <none>           <none>
dexter-                   1/1     Running   0          8m22s  10.  180     us-east-2.compute.internal    <none>           <none>
```

Now I deleted the deployment dexter and found that two new scaled nodes had been deleted as shown in the screenshot attached below.

```
[root@                    ~]# kubectl delete -f deployment.yaml
deployment.apps "dexter" deleted
[root@                    ~]# kubectl get pods --watch
^C[root@                  ~]#
```

After unneed for 10 minutes the newly added nodes were deleted as can be seen from the logs of cluster-autoscaler pod present in the namespace kube-system.

```
[root@                    ~]# kubectl logs -f cluster-autoscaler-                    -n kube-system
```

```
I0316 11:27:40.519843       1 static_autoscaler.go:589] Scale down status: lastScaleUpTime=2025-03-16 11:01:34.694431923 +0000 UTC m=+487.611918864 lastScaleD
ownDeleteTime=2025-03-16 09:53:33.205512101 +0000 UTC m=-3593.877000953 lastScaleDownFailTime=2025-03-16 09:53:33.205512101 +0000 UTC m=-3593.877000953 scaleD
ownForbidden=false scaleDownInCooldown=false
I0316 11:27:40.519870       1 static_autoscaler.go:598] Starting scale down
I0316 11:27:40.519893       1 nodes.go:123] ip-10-    -175.us-east-2.compute.internal was unneeded for 10m2.295069559s
I0316 11:27:40.519905       1 nodes.go:123] ip-10-    -250.us-east-2.compute.internal was unneeded for 10m2.295069559s
I0316 11:27:40.581299       1 delete.go:103] Successfully added ToBeDeletedTaint on node ip-10-    -175.us-east-2.compute.internal
I0316 11:27:40.581654       1 event_sink_logging_wrapper.go:48] Event(v1.ObjectReference{Kind:"Node", Namespace:"", Name:"ip-10-    -175.us-east-2.compute.int
ernal", UID:"                                    ", APIVersion:"v1", ResourceVersion:"10001", FieldPath:""}): type: 'Normal' reason: 'ScaleDown' marked the no
de as toBeDeleted/unschedulable
I0316 11:27:40.613549       1 delete.go:103] Successfully added ToBeDeletedTaint on node ip-10-    -250.us-east-2.compute.internal
I0316 11:27:40.613581       1 actuator.go:161] Scale-down: removing empty node "ip-10-    -175.us-east-2.compute.internal"
I0316 11:27:40.613744       1 actuator.go:161] Scale-down: removing empty node "ip-10-    -250.us-east-2.compute.internal"
I0316 11:27:40.614445       1 event_sink_logging_wrapper.go:48] Event(v1.ObjectReference{Kind:"Node", Namespace:"", Name:"ip-10-    -250.us-east-2.compute.int
ernal", UID:"                                    ", APIVersion:"v1", ResourceVersion:"9994", FieldPath:""}): type: 'Normal' reason: 'ScaleDown' marked the nod
e as toBeDeleted/unschedulable
I0316 11:27:40.614650       1 actuator.go:244] Scale-down: waiting 5s before trying to delete nodes
I0316 11:27:40.628402       1 event_sink_logging_wrapper.go:48] Event(v1.ObjectReference{Kind:"ConfigMap", Namespace:"kube-system", Name:"cluster-autoscaler-s
tatus", UID:"                                    ", APIVersion:"v1", ResourceVersion:"11146", FieldPath:""}): type: 'Normal' reason: 'ScaleDownEmpty' Scale-do
wn: removing empty node "ip-10-    -175.us-east-2.compute.internal"
I0316 11:27:40.635365       1 event_sink_logging_wrapper.go:48] Event(v1.ObjectReference{Kind:"ConfigMap", Namespace:"kube-system", Name:"cluster-autoscaler-s
tatus", UID:"c467fc5a-5eb1-48c0-955d-ce381529b8a4", APIVersion:"v1", ResourceVersion:"11146", FieldPath:""}): type: 'Normal' reason: 'ScaleDownEmpty' Scale-do
wn: removing empty node "ip-10-    -250.us-east-2.compute.internal"
I0316 11:27:45.176218       1 reflector.go:559] k8s.io/autoscaler/cluster-autoscaler/utils/kubernetes/listers.go:347: Watch close - *v1.ReplicaSet total 10 it
ems received
I0316 11:27:45.866778       1 auto_scaling_groups.go:311] Terminating EC2 instance: i-0                    f
I0316 11:27:45.866810       1 aws_manager.go:161] DeleteInstances was called: scheduling an ASG list refresh for next main loop evaluation
I0316 11:27:45.867125       1 event_sink_logging_wrapper.go:48] Event(v1.ObjectReference{Kind:"ConfigMap", Namespace:"kube-system", Name:"cluster-autoscaler-s
tatus", UID:"                                    ", APIVersion:"v1", ResourceVersion:"11195", FieldPath:""}): type: 'Normal' reason: 'ScaleDownEmpty' Scale-do
wn: empty node ip-10-    -175.us-east-2.compute.internal removed
I0316 11:27:46.010965       1 auto_scaling_groups.go:311] Terminating EC2 instance: i-0                    6
I0316 11:27:46.010986       1 aws_manager.go:161] DeleteInstances was called: scheduling an ASG list refresh for next main loop evaluation
I0316 11:27:46.011124       1 event_sink_logging_wrapper.go:48] Event(v1.ObjectReference{Kind:"ConfigMap", Namespace:"kube-system", Name:"cluster-autoscaler-s
tatus", UID:"                                    ", APIVersion:"v1", ResourceVersion:"11195", FieldPath:""}): type: 'Normal' reason: 'ScaleDownEmpty' Scale-do
wn: empty node ip-10-    -250.us-east-2.compute.internal removed
I0316 11:27:46.571647       1 reflector.go:559] k8s.io/autoscaler/cluster-autoscaler/utils/kubernetes/listers.go:329: Watch close - *v1.ReplicationController
```

```
[root@                    ~]# kubectl get nodes
NAME                                        STATUS   ROLES    AGE   VERSION
ip-10-    -73.us-east-2.compute.internal    Ready    <none>   42m   v1.30.4-eks-a737599
```

2. Second Section

Karpenter

Karpenter is an open source kubernetes cluster autoscaler. Karpenter is 10 times faster than cluster autoscaler. I had created the EKS Cluster using the terraform script present in my GitHub Repo https://github.com/singhritesh85/Autoscale.git at the path **Autoscale/Autoscale-EKS/Autoscale-EKS-Cluster-using-Karpenter** and edited the configmap **aws-auth** present at the namespace **kube-system** then created the Karpenter controller using the **helm** run the manifests file **karpenter.yaml** as mentioned below.

```
[root@                ~]# kubectl get nodes --watch
NAME                                        STATUS   ROLES    AGE   VERSION
ip-10-    -12.us-east-2.compute.internal    Ready    <none>   12m   v1.30.4-eks-a     9
ip-10-    -158.us-east-2.compute.internal   Ready    <none>   12m   v1.30.4-eks-a     9
```

Then create the deployment with 40 pods and found that pods are in pending state due to unavailability of compute resources. Which you can check with the command **kubectl describe pod <pod-name> -n namespace**.

```
[root@                ~]# vim deployment.yaml
[root@                ~]# kubectl apply -f deployment.yaml
deployment.apps/dexter created
[root@                ~]# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dexter
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 40
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80


[root@                ~]# kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
dexter-                   0/1     Pending   0          19s
dexter-                   0/1     Pending   0          18s
dexter-                   0/1     Pending   0          18s
dexter-                   1/1     Running   0          19s
dexter-                   0/1     Pending   0          18s
dexter-                   1/1     Running   0          19s
dexter-                   1/1     Running   0          19s
dexter-                   0/1     Pending   0          18s
dexter-                   1/1     Running   0          19s
dexter-                   0/1     Pending   0          18s
dexter-                   1/1     Running   0          19s
dexter-                   1/1     Running   0          19s
```

```
[root@                ~]# kubectl edit cm aws-auth -n kube-system
```

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::0[redacted]6:role/eks-nodegroup-role-dev
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::0[redacted]6:role/karpenter-eks-noderole
      username: system:node:{{EC2PrivateDNSName}}
kind: ConfigMap
metadata:
  creationTimestamp: "2025-[redacted]"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "[redacted]"
  uid: b[redacted]7
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT (paste) --
```

Install the karpenter controller using the helm as shown in the screenshot attached below.

helm upgrade --install karpenter oci://public.ecr.aws/karpenter/karpenter --version "1.3.2" --namespace "karpenter" --create-namespace --set "settings.clusterName=eks-demo-cluster-dev" --set "serviceAccount.annotations.eks\.amazonaws\.com/role-arn=arn:aws:iam::02XXXXXXXXX6:role/karpenter-controller-role" --set controller.resources.requests.cpu=1 --set controller.resources.requests.memory=1Gi --set controller.resources.limits.cpu=1 --set controller.resources.limits.memory=1Gi

```
[root@[redacted] terraform-eks-withaddons-karpenter]# helm upgrade --install karpenter oci://public.ecr.aws/karpenter/karpenter --version "1.3.2" --namespace "karpenter" --create-namespace --set "settings.clusterName=eks-demo-cluster-dev" --set "serviceAccount.annotations.eks\.amazonaws\.com/role-arn=arn:aws:iam::0[redacted]6:role/karpenter-controller-role" --set controller.resources.requests.cpu=1 --set controller.resources.requests.memory=1Gi --set controller.resources.limits.cpu=1 --set controller.resources.limits.memory=1Gi
Release "karpenter" does not exist. Installing it now.
NAME: karpenter
LAST DEPLOYED: [redacted] 2025
NAMESPACE: karpenter
STATUS: deployed
REVISION: 1
TEST SUITE: None
[root@[redacted] terraform-eks-withaddons-karpenter]# kubectl get pods -n karpenter --watch
NAME                         READY   STATUS    RESTARTS   AGE
karpenter-8[redacted]b   1/1     Running   0          12s
karpenter-8[redacted]6   1/1     Running   0          12s
```

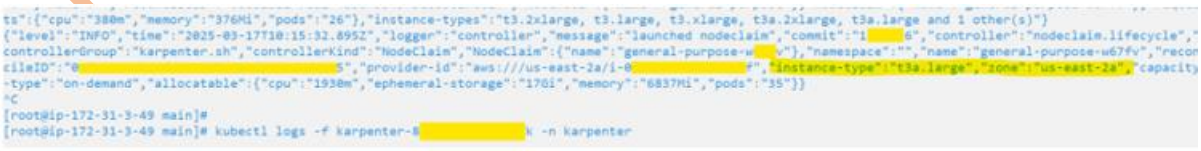
The two pods present in the namespace karpenter are for karpenter controller and then run the manifests file karpenter.yaml as shown in the screenshot attached below.

```
[root@                terraform-eks-withaddons-karpenter]# kubectl apply -f karpenter.yaml
nodepool.karpenter.sh/general-purpose created
ec2nodeclass.karpenter.k8s.aws/default created
```

For your reference I kept this yaml manifests file in GitHub Repo https://github.com/singhritesh85/Autoscale.git at the path **Autoscale/Autoscale-EKS/Autoscale-EKS-Cluster-using-Karpenter/terraform-eks-withaddons-karpenter**. You can edit this yaml manifests file depending on your project then apply.

```
[root@                terraform-eks-withaddons-karpenter]# cat karpenter.yaml
# This example NodePool will provision general purpose instances
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: general-purpose
  annotations:
    kubernetes.io/description: "General purpose NodePool for generic workloads"
spec:
  template:
    spec:
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: kubernetes.io/os
          operator: In
          values: ["linux"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]    #["spot"] ### You can select on-demand or spot instance depending on your requirement.
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["t"]   # Interested to launch t series instance          #["c", "m", "r"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["2"]   # Instances launched will be greated than 2
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
---
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: default
  annotations:
    kubernetes.io/description: "General purpose EC2NodeClass for running Amazon Linux 2 nodes"
spec:
  role: "karpenter-eks-noderole" # replace with your karpenter noderole
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "eks-demo-cluster-dev" # replace with your cluster name
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "eks-demo-cluster-dev" # replace with your cluster name
  amiSelectorTerms:
    - alias: al2023@latest # Amazon Linux 2023
[root@                terraform-eks-withaddons-karpenter]#
```

Then I checked the logs of the karpenter controller pod present in the namespace karpenter as shown in the screenshot attached below.



Finally, I found a new node had been created which joined the cluster as shown in the screenshot attached below.

```
[root@                    ]# kubectl get nodes
NAME                                        STATUS   ROLES    AGE   VERSION
ip-10-    -65.us-east-2.compute.internal    Ready    <none>   93m   v1.30.4-eks-a737599
ip-10-    -148.us-east-2.compute.internal   Ready    <none>   33m   v1.30.4-eks-a737599
ip-10-    -58.us-east-2.compute.internal    Ready    <none>   94s   v1.30.9-eks-5d632ec
```

You can provide your choice as per your project requirement in the file karpenter.yaml like which series instance type you want to get launched, whether you want on-demand or spot instances and other stuff.

I would like to mention here in industry at most of the cases the kubernetes cluster is available with the cluster-autoscaler and if you want to migrate to karpenter from cluster-autoscaler then do the below changes in your existing EKS Cluster.

1. Delete the yaml manifests for cluster-autoscaler using the command kubectl delete -f cluster-autoscaler.yaml.
2. Add the tags to subnets and security groups of the Nodes which were the part that EKS Cluster.
3. Add the IAM Role and Policy for Karpenter Controller and Karpenter Nodes Roles and Karpenter Nodes Policy (This changes you can perform in your existing terraform script).
4. Edit the configmap aws-auth present in the namespace kube-system.
5. Install the karpenter controller using the helm.
6. Do the changes in karpenter.yaml as per your project requirement like which series of instance you want to be get launched, on-demand/spot instances and other stuff then apply the manifests file using the command kubectl apply -f karpenter.yaml.

```
[root@              ~]# cd terraform-eks-withaddons-cluster-autoscaler/
[root@              terraform-eks-withaddons-cluster-autoscaler]# kubectl delete -f cluster-autoscaler.yaml
serviceaccount "cluster-autoscaler" deleted
clusterrole.rbac.authorization.k8s.io "cluster-autoscaler" deleted
role.rbac.authorization.k8s.io "cluster-autoscaler" deleted
clusterrolebinding.rbac.authorization.k8s.io "cluster-autoscaler" deleted
rolebinding.rbac.authorization.k8s.io "cluster-autoscaler" deleted
deployment.apps "cluster-autoscaler" deleted
```

Below changes had been done in vpc.tf file.

```
################################ Public Subnet ########################################

resource "aws_subnet" "public_subnet" {
  count = "${length(data.aws_availability_zones.azs.names)}"
  vpc_id     = "${aws_vpc.test_vpc.id}"
  availability_zone = "${element(data.aws_availability_zones.azs.names,count.index)}"
  cidr_block = "${element(var.public_subnet_cidr,count.index)}"
  map_public_ip_on_launch = true

  tags = {
    Name = "PublicSubnet-${var.env}-${count.index+1}"
    Environment = var.env            ##"${terraform.workspace}"
    "karpenter.sh/discovery" = "${var.eks_cluster}-${var.env}"
  }
}

################################ Private Subnet #######################################

resource "aws_subnet" "private_subnet" {
  count = "${length(data.aws_availability_zones.azs.names)}"                  ##"${length
  vpc_id     = "${aws_vpc.test_vpc.id}"
  availability_zone = "${element(data.aws_availability_zones.azs.names,count.index)}"
  cidr_block = "${element(var.private_subnet_cidr,count.index)}"

  tags = {
    Name = "PrivateSubnet-${var.env}-${count.index+1}"
    Environment = var.env              ##"${terraform.workspace}"
    "karpenter.sh/discovery" = "${var.eks_cluster}-${var.env}"
  }
}
```

Open the file cluster.tf and go to the end and edit with content as written below.

```
#############################################################################################
#####################
# Tag the EKS Cluster Security Group
#############################################################################################
#####################

resource "aws_ec2_tag" "cluster_security_group" {
  resource_id = aws_eks_cluster.eksdemo.vpc_config[0].cluster_security_group_id
  key         = "karpenter.sh/discovery"
  value       = "${var.eks_cluster}-${var.env}"

  depends_on = [aws_eks_node_group.eksnode]
}

#############################################################################################
#####################
# Karpenter Controller
#############################################################################################
#####################

data "aws_iam_policy_document" "karpenter_assume_role_policy" {
  statement {
    actions = ["sts:AssumeRoleWithWebIdentity"]
    effect  = "Allow"

    condition {
      test     = "StringEquals"
      variable = "${replace(aws_iam_openid_connect_provider.eksopidc.url, "https://", "")}:sub"
      values   = ["system:serviceaccount:karpenter:karpenter"]
    }

    principals {
```

```
    identifiers = [aws_iam_openid_connect_provider.eksopidc.arn]
    type        = "Federated"
   }
  }
}

resource "aws_iam_role" "karpenter_controller_role" {
  assume_role_policy = data.aws_iam_policy_document.karpenter_assume_role_policy.json
  name               = "karpenter-controller-role"
}

resource "aws_iam_role_policy_attachment" "karpenter_controller_policy_attach" {
  role       = aws_iam_role.karpenter_controller_role.name
  policy_arn = aws_iam_policy.karpenter_controller_policy.arn
}


resource "aws_iam_policy" "karpenter_controller_policy" {
  name = "karpenter-controller-policy"
  policy = jsonencode(
   {
    "Version" : "2012-10-17",
    "Statement" : [
     {
      "Sid" : "VisualEditor0",
      "Effect" : "Allow",
      "Action" : "ec2:TerminateInstances",
      "Resource" :
"arn:aws:ec2:${data.aws_region.reg.id}:${data.aws_caller_identity.G_Duty.account_id}:instance/*",
      "Condition" : {
       "ForAnyValue:StringLike" : {
```

```
          "ec2:DeleteLaunchTemplate",
          "ec2:CreateLaunchTemplate",
          "ec2:CreateTags"
        ],
        "Resource" :
"arn:aws:ec2:${data.aws_region.reg.id}:${data.aws_caller_identity.G_Duty.account_id}:launch-
template/*"
      },
      {
        "Sid" : "VisualEditor3",
        "Effect" : "Allow",
        "Action" : [
          "iam:PassRole",
          "ssm:GetParameter"
        ],
        "Resource" : [
          "arn:aws:iam::${data.aws_caller_identity.G_Duty.account_id}:role/karpenter-node*",
          "arn:aws:ssm:${data.aws_region.reg.id}:*:parameter/*"
        ]
      },
      {
        "Sid" : "karpenterInstanceProfile",
        "Effect" : "Allow",
        "Action" : [
          "iam:CreateInstanceProfile",
          "iam:TagInstanceProfile",
          "iam:AddRoleToInstanceProfile",
          "iam:RemoveRoleFromInstanceProfile",
          "iam:DeleteInstanceProfile"
        ],
        "Resource" : [
```

```
          "arn:aws:iam::${data.aws_caller_identity.G_Duty.account_id}:instance-profile/eks-demo-cluster-dev*"
        ]
      }
    ]
}
 )

}

################################################################################################################
# Nodes Spinned-up using Karpenter
################################################################################################################

# IAM Role which allows Karpenter to spin up nodes
data "aws_iam_policy_document" "karpenter_node_iam_role_policy" {
  statement {
    actions = ["sts:AssumeRole"]
    effect  = "Allow"

    principals {
      identifiers = ["ec2.amazonaws.com"]
      type        = "Service"
    }
  }
}

resource "aws_iam_role" "karpenter_node_iam_role" {
  assume_role_policy = data.aws_iam_policy_document.karpenter_node_iam_role_policy.json
  name               = "karpenter-eks-noderole"
```

```
}

resource "aws_iam_role_policy_attachment" "karpenter_workernode" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
}

resource "aws_iam_role_policy_attachment" "karpenter_cni" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
}

resource "aws_iam_role_policy_attachment" "karpenter_ecr" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
}

resource "aws_iam_role_policy_attachment" "karpenter_ssm" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
}

resource "aws_iam_instance_profile" "karpenter_instance_profile" {
  name = "karpenter-eks-instance-profile"
  role = aws_iam_role.karpenter_node_iam_role.name
}
```

```
      "ec2:ResourceTag/Name" : "*karpenter*"
    }
   }
  },
  {
   "Sid" : "VisualEditor1",
   "Effect" : "Allow",
   "Action" : [
    "eks:DescribeCluster",
    "ec2:DescribeSpotPriceHistory",
    "ec2:DescribeImages",
    "ec2:DescribeInstances",
    "ec2:DescribeInstanceTypeOfferings",
    "ec2:DescribeAvailabilityZones",
    "ec2:DescribeLaunchTemplates",
    "ec2:DescribeInstanceTypes",
    "ec2:RunInstances",
    "ec2:CreateFleet",
    "ec2:DescribeSubnets",
    "ec2:DescribeSecurityGroups",
    "ec2:CreateTags",
    "pricing:GetProducts",
    "iam:GetInstanceProfile",
    "iam:PassRole"
   ],
   "Resource" : "*"
  },
  {
   "Sid" : "VisualEditor2",
   "Effect" : "Allow",
   "Action" : [
```

```
            "ec2:DeleteLaunchTemplate",
            "ec2:CreateLaunchTemplate",
            "ec2:CreateTags"
          ],
          "Resource" :
"arn:aws:ec2:${data.aws_region.reg.id}:${data.aws_caller_identity.G_Duty.account_id}:launch-
template/*"
        },
        {
          "Sid" : "VisualEditor3",
          "Effect" : "Allow",
          "Action" : [
            "iam:PassRole",
            "ssm:GetParameter"
          ],
          "Resource" : [
            "arn:aws:iam::${data.aws_caller_identity.G_Duty.account_id}:role/karpenter-node*",
            "arn:aws:ssm:${data.aws_region.reg.id}:*:parameter/*"
          ]
        },
        {
          "Sid" : "karpenterInstanceProfile",
          "Effect" : "Allow",
          "Action" : [
            "iam:CreateInstanceProfile",
            "iam:TagInstanceProfile",
            "iam:AddRoleToInstanceProfile",
            "iam:RemoveRoleFromInstanceProfile",
            "iam:DeleteInstanceProfile"
          ],
          "Resource" : [
```

```
          "arn:aws:iam::${data.aws_caller_identity.G_Duty.account_id}:instance-profile/eks-demo-cluster-dev*"
        ]
      }
    ]
}
 )

}

################################################################################################################
# Nodes Spinned-up using Karpenter
################################################################################################################

# IAM Role which allows Karpenter to spin up nodes
data "aws_iam_policy_document" "karpenter_node_iam_role_policy" {
  statement {
    actions = ["sts:AssumeRole"]
    effect  = "Allow"

    principals {
      identifiers = ["ec2.amazonaws.com"]
      type        = "Service"
    }
  }
}

resource "aws_iam_role" "karpenter_node_iam_role" {
  assume_role_policy = data.aws_iam_policy_document.karpenter_node_iam_role_policy.json
  name               = "karpenter-eks-noderole"
```

```
}

resource "aws_iam_role_policy_attachment" "karpenter_workernode" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
}

resource "aws_iam_role_policy_attachment" "karpenter_cni" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
}

resource "aws_iam_role_policy_attachment" "karpenter_ecr" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
}

resource "aws_iam_role_policy_attachment" "karpenter_ssm" {
  role       = aws_iam_role.karpenter_node_iam_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
}

resource "aws_iam_instance_profile" "karpenter_instance_profile" {
  name = "karpenter-eks-instance-profile"
  role = aws_iam_role.karpenter_node_iam_role.name
}
```

I changed the min_size, max_size and desired_capacity and kept them same. Take the desired count such as it could bear all the pods required for the exiting projects running in this cluster. For the current scenario I kept it as 2.

```
  scaling_config {
    desired_size = 2
    max_size     = 2
    min_size     = 2
  }
```

```
[root@                main]# terraform apply -auto-approve
```

```
[root@                ~]# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dexter
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 30
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80


[root@                ~]# kubectl apply -f deployment.yaml
```

```
[root@ip-172-31-3-49 ~]# kubectl get pods --watch
NAME                      READY   STATUS              RESTARTS   AGE
dexter-6cfb64b7c5-2tdz2   1/1     Running             0          15s
dexter-6cfb64b7c5-4bwqh   1/1     Running             0          15s
dexter-6cfb64b7c5-4sbk7   0/1     Pending             0          15s
dexter-6cfb64b7c5-574pk   1/1     Running             0          15s
dexter-6cfb64b7c5-57sbv   0/1     Pending             0          15s
dexter-6cfb64b7c5-77d8q   1/1     Running             0          15s
dexter-6cfb64b7c5-7fqj5   1/1     Running             0          15s
```

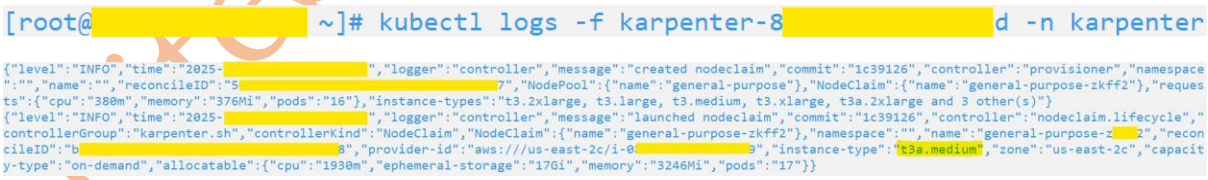
Pod Disruption Budget must had applied with the existing pods for your projects. Otherwise, if the application becomes unavailable then DevOps team which Email-ID was provided to the notification for Synthetic Monitoring (Unavailability for endpoint URL) will get notification. So please do this activity during non-production hours or take-off the email-id from the Alerting (However it is not suggestable to remove the DevOps team Email-ID from the Alerting) and after the activity add back to Alerting.

```
[root@ main]# kubectl get pods -n karpenter --watch
NAME                         READY   STATUS    RESTARTS   AGE
karpenter-8 h   0/1     Running   0          9s
karpenter-8 d   0/1     Running   0          9s
karpenter-8 d   1/1     Running   0          12s
karpenter-8 h   1/1     Running   0          12s
^C[root@ main]# cd ..
[root@ terraform-eks-withaddons-cluster-autoscaler]# vim karpenter.yaml
[root@ terraform-eks-withaddons-cluster-autoscaler]# kubectl apply -f karpenter.yaml
nodepool.karpenter.sh/general-purpose created
ec2nodeclass.karpenter.k8s.aws/default created
[root@ terraform-eks-withaddons-cluster-autoscaler]# cd
```

```
# This example NodePool will provision general purpose instances
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: general-purpose
  annotations:
    kubernetes.io/description: "General purpose NodePool for generic workloads"
spec:
  template:
    spec:
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: kubernetes.io/os
          operator: In
          values: ["linux"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]   #["spot"] ### You can select on-demand or spot instance depending on your requirement.
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["t"]   # Interested to launch t series instance          #["c", "m", "r"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["2"]   # Instances launched will be greated than 2
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
```

```
      name: default
---
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: default
  annotations:
    kubernetes.io/description: "General purpose EC2NodeClass for running Amazon Linux 2 nodes"
spec:
  role: "karpenter-eks-noderole" # replace with your karpenter noderole
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "eks-demo-cluster-dev" # replace with your cluster name
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "eks-demo-cluster-dev" # replace with your cluster name
  amiSelectorTerms:
    - alias: al2023@latest # Amazon Linux 2023
```

Checked the logs of karpenter controller pod and found a node had been created as shown in the screenshot attached below.



```
[root@                    ]# kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
dexter-6                2   1/1     Running   0          m
dexter-6                h   1/1     Running   0          m
dexter-6                7   1/1     Running   0          m
dexter-6                k   1/1     Running   0          m
dexter-6                v   1/1     Running   0          m
dexter-6                q   1/1     Running   0          m
dexter-6                5   1/1     Running   0          m
dexter-6                c   1/1     Running   0          m
dexter-6                5   1/1     Running   0          m
dexter-6                w   1/1     Running   0          m
dexter-6                h   1/1     Running   0          m
dexter-6                h   1/1     Running   0          m
dexter-6                8   1/1     Running   0          m
dexter-6                q   1/1     Running   0          m
dexter-6                t   1/1     Running   0          m
dexter-6                q   1/1     Running   0          m
dexter-6                w   1/1     Running   0          m
dexter-6                5   1/1     Running   0          m
dexter-6                5   1/1     Running   0          m
dexter-6                x   1/1     Running   0          m
dexter-6                s   1/1     Running   0          m
dexter-6                z   1/1     Running   0          m
dexter-6                n   1/1     Running   0          m
dexter-6                h   1/1     Running   0          m
dexter-6                v   1/1     Running   0          m
dexter-6                f   1/1     Running   0          m
dexter-6                6   1/1     Running   0          m
dexter-6                4   1/1     Running   0          m
dexter-6                c   1/1     Running   0          m
dexter-6                n   1/1     Running   0          m
```

```
[root@ip-172-31-3-49 ~]# kubectl get nodes
NAME                                        STATUS   ROLES    AGE    VERSION
ip-10-    -140.us-east-2.compute.internal   Ready    <none>          v1.30.4-eks-a    9
ip-10-    -139.us-east-2.compute.internal   Ready    <none>          v1.30.4-eks-a    9
ip-10-    -79.us-east-2.compute.internal    Ready    <none>          v1.30.9-eks-5    c
```

The above attached screenshot verifies the autoscaling had been done and migrated from cluster-autoscaler to karpenter

.

3. Third Section

Cluster-Autoscaler in AKS Cluster

I used the terraform script to created the AKS Cluster and enabled the cluster autoscaler in default node pool (system node pool) and user node pool by keeping **auto_scaling_enabled = true**. As shown in the screen shot attached below, I kept the min_count and max_count as 1 and 4 respectively, the node_count which indicates the desired node count kept as 1. The node_count cannot be less than min_count and cannot be less than max_count.

min_count ≤ node_count ≤ max_count

```
 auto_scaling_enabled  = true
 max_count             = 4
 node_count            = 1
 min_count             = 1
```

Initially just after creation of the AKS Cluster there were 2 nodes one for system pool and one for agent pool.

```
[root@                    ~]# kubectl get nodes
NAME                                STATUS   ROLES    AGE   VERSION
aks-agentpool-2       4-vmss000000  Ready    <none>         v1.29.0
aks-userpool-2        4-vmss000000  Ready    <none>         v1.29.0
```

You can check the logs of cluster autoscaling using the command **kubectl get events -n kube-system** or **kubectl describe cm cluster-autoscaler-status -n kube-system**.

For this demonstration I created a deployment with nginx image and kept the number of desired pods as 250.

```
[root@                    ~]# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dexter
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 250
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
[root@                    ~]# kubectl apply -f deployment.yaml
```

```
[root@                    ~]# kubectl get pods --watch
NAME                          READY   STATUS    RESTARTS   AGE
dexter-7                  p   1/1     Running   0          56s
dexter-7                  w   0/1     Pending   0          19s
dexter-7                  2   0/1     Pending   0          16s
dexter-7                  b   1/1     Running   0          55s
dexter-7                  j   0/1     Pending   0          20s
dexter-7                  g   0/1     Pending   0          16s
dexter-7                  f   1/1     Running   0          55s
dexter-7                  g   1/1     Running   0          55s
dexter-7                  b   1/1     Running   0          21s
dexter-7                  f   0/1     Pending   0          16s
dexter-7                  q   0/1     Pending   0          16s
dexter-7                  l   1/1     Running   0          22s
dexter-7                  n   1/1     Running   0          91s
dexter-7                  x   1/1     Running   0          21s
dexter-7                  6   0/1     Pending   0          19s
dexter-7                  l   1/1     Running   0          92s
dexter-7                  9   0/1     Pending   0          16s
dexter-7                  9   0/1     Pending   0          16s
```

As there were lack of compute resources due to which pods could not be scheduled on the nodes and are in pending state which can be seen in the screenshot attached above, you can also check the same using the command **kubectl describe pod <pod-name> -n namespace**.

Now the cluster-autoscaler in AKS will spin new nodes on which the pods can be scheduled and I found that cluster-autoscaler spun 2 new nodes so that it could schedule the pods to the nodes.

```
[root@                    ~]# kubectl get nodes
NAME                                  STATUS   ROLES    AGE      VERSION
aks-                              00  Ready    <none>            v1.29.0
aks-                              01  Ready    <none>            v1.29.0
aks-                              02  Ready    <none>            v1.29.0
aks-                              0   Ready    <none>            v1.29.0
```

```
[root@                    ~]# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
dexter-7                      1/1     Running   0
```

I checked the logs using the command **kubectl describe cm cluster-autoscaler-status -n kube-system** as shown below.

```
[root@                    ~]# kubectl describe cm cluster-autoscaler-status -n kube-system
```

```
Events:
  Type    Reason         Age   From                Message
  ----    ------         ----  ----                -------
  Normal  ScaledUpGroup  16m   cluster-autoscaler  Scale-up: setting group aks-                    4-vmss size to 3 instead of 1 (max: 4)
  Normal  ScaledUpGroup  16m   cluster-autoscaler  Scale-up: group aks-                  4-vmss size set to 3 instead of 1 (max: 4)
```

Finally, I deleted the deployment dexter using the command as shown in the screenshot attached below and found that no pod is running now.

```
[root@                          ~]# kubectl delete -f deployment.yaml
```

```
[root@                          ~]# kubectl get pods
No resources found in default namespace.
```

Then checked the logs using the command **kubectl describe cm cluster-autoscaler-status -n kube-system** as shown below and found that nodes are scaled down to the initial state.

```
Events:
  Type    Reason          Age   From                Message
  ----    ------          ----  ----                -------
  Normal  ScaledUpGroup   34m   cluster-autoscaler  Scale-up: setting group aks-                -vmss size to 3 instead of 1 (max: 4)
  Normal  ScaledUpGroup   34m   cluster-autoscaler  Scale-up: group aks-                -vmss size set to 3 instead of 1 (max: 4)
  Normal  ScaleDown       14m   cluster-autoscaler  Scale-down: removing node aks-                      01, utilization: {0.38526315789473686 0.232064075
54169126 0 cpu 0.38526315789473686}, pods to reschedule: metrics-server-5        b-x  z,metrics-server-5          t
  Normal  ScaleDownEmpty  14m   cluster-autoscaler  Scale-down: removing empty node "aks-                      02"
  Normal  ScaleDownEmpty  13m   cluster-autoscaler  Scale-down: empty node aks-                      02 removed
  Normal  ScaleDown       13m   cluster-autoscaler  Scale-down: node aks-                      01 removed with drain
```

```
[root@                          ~]# kubectl get nodes
NAME                                STATUS   ROLES    AGE    VERSION
aks-agentpool-2        4-vmss000000   Ready    <none>          v1.29.0
aks-userpool-2         4-vmss000000   Ready    <none>          v1.29.0
```


GitHub Repository: - https://github.com/singhritesh85/Autoscale.git

References: -

1. https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-autodiscover.yaml
2. https://karpenter.sh/docs/getting-started/getting-started-with-karpenter/
3. https://github.com/aws/karpenter-provider-aws/blob/main/examples/v1/general-purpose.yaml