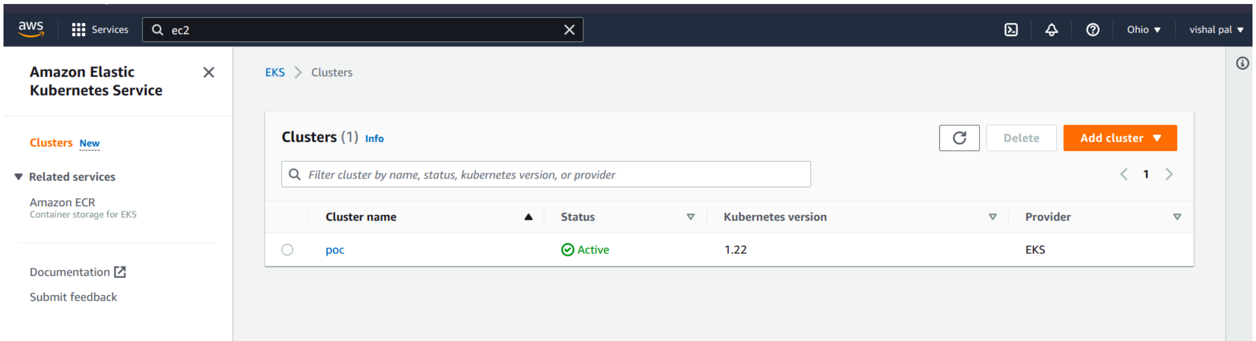
USING SPOT INSTANCE FOR DEPLOYMENT WITH NO DOWNTIME DURING RECLAMATION OF SPOT NODE USING NODE TERMINATION HANDLER.

CREATING EKS CLUSTER

We can refer this document to create a kubernetes cluster.
https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html.



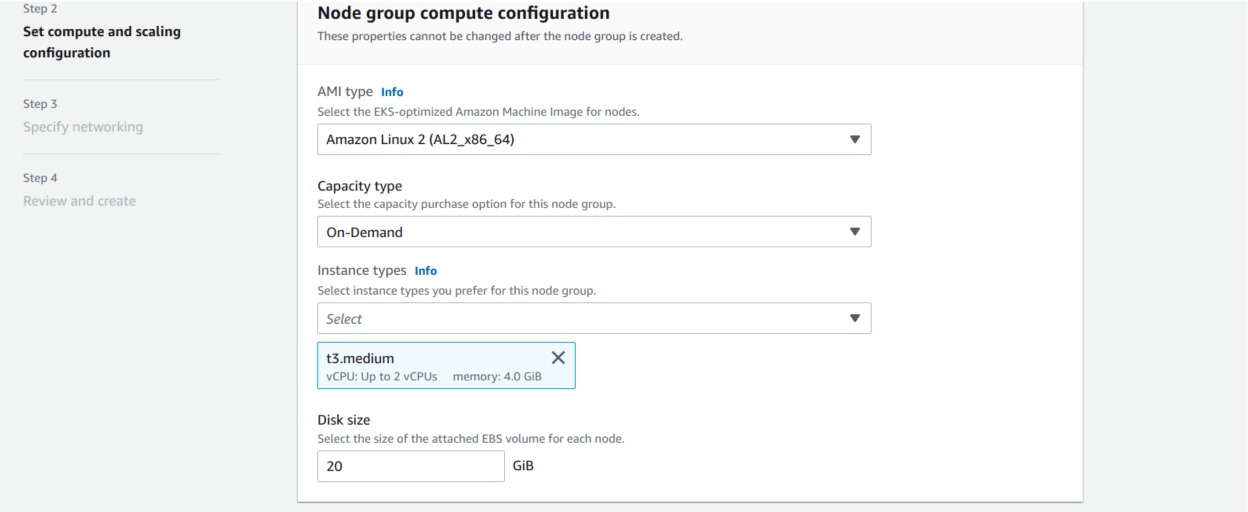
Clusters (1) Info

Cluster name	Status	Kubernetes version	Provider
poc	Active	1.22	EKS

CREATING A ON_DEMAND NODE GROUP

We can refer to this document to create a aws node group.
https://docs.aws.amazon.com/eks/latest/userguide/create-managed-node-group.html

In the Capacity type during creation of node group we will select ON_DEMAND



Node group compute configuration
These properties cannot be changed after the node group is created.

AMI type Info
Select the EKS-optimized Amazon Machine Image for nodes.
Amazon Linux 2 (AL2_x86_64)

Capacity type
Select the capacity purchase option for this node group.
On-Demand

Instance types Info
Select instance types you prefer for this node group.
Select
t3.medium
vCPU: Up to 2 vCPUs memory: 4.0 GiB

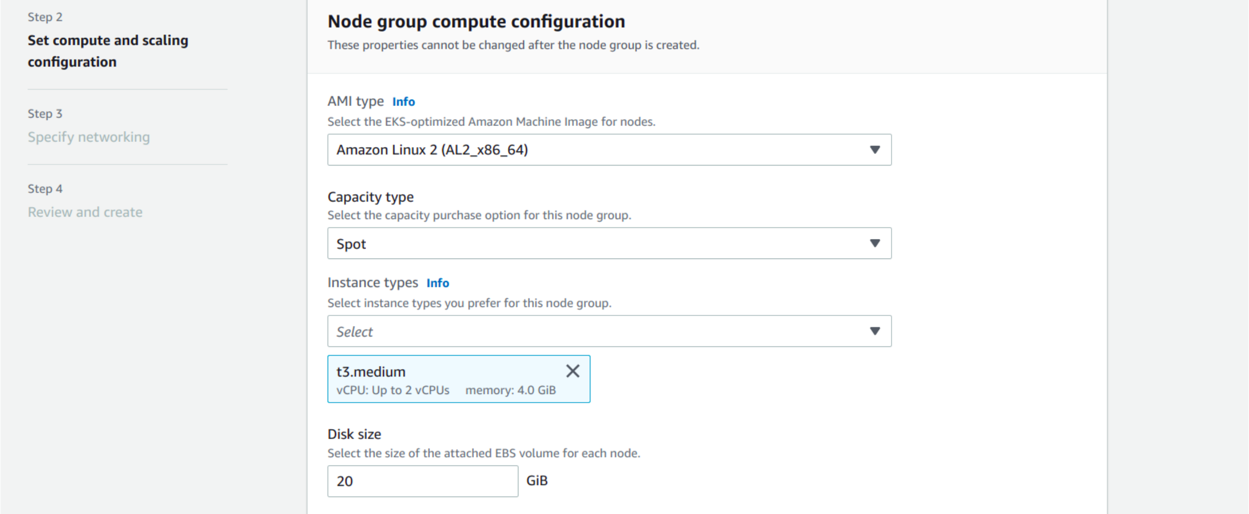
Disk size
Select the size of the attached EBS volume for each node.
20 GiB

After creation of an on_demand node group we will create a different node group for instance.

We will choose the minimum available node as 1
Maximum available node as 1
Desired nodes as 1
As we are only going to scale SPOT nodes.

CREATING A ON_SPOT NODE GROUP
We can refer to this document to create a spot aws node group.
https://docs.aws.amazon.com/eks/latest/userguide/create-managed-node-group.html

During creation of node group we will select capacity type as spot instance



Step 2
Set compute and scaling configuration

Step 3
Specify networking

Step 4
Review and create

Node group compute configuration
These properties cannot be changed after the node group is created.

AMI type Info
Select the EKS-optimized Amazon Machine Image for nodes.
Amazon Linux 2 (AL2_x86_64)

Capacity type
Select the capacity purchase option for this node group.
Spot

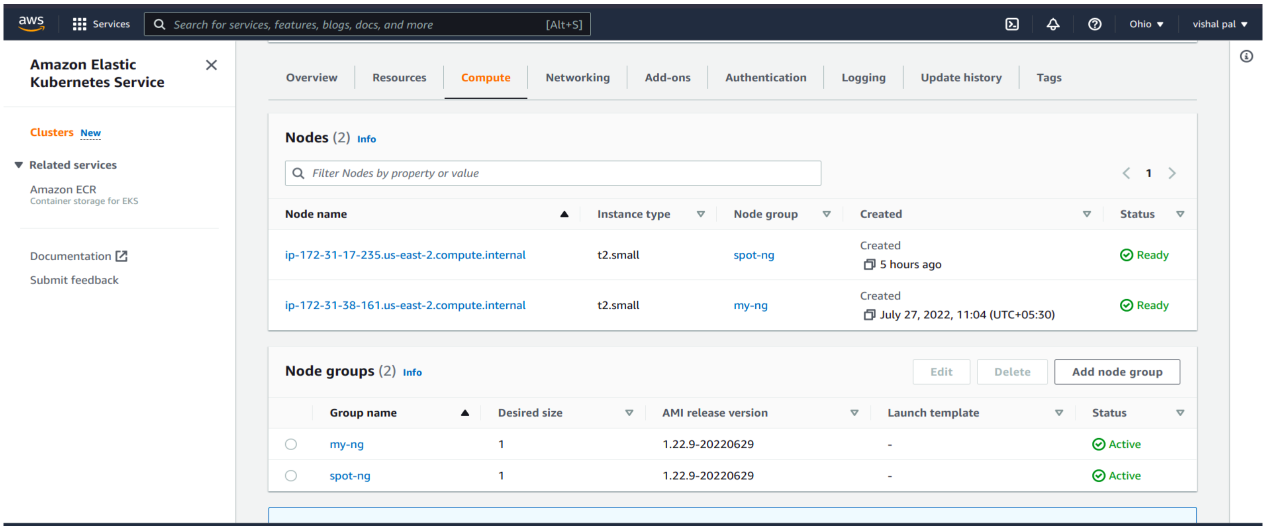
Instance types Info
Select instance types you prefer for this node group.
Select
t3.medium
vCPU: Up to 2 vCPUs memory: 4.0 GiB

Disk size
Select the size of the attached EBS volume for each node.
20 GiB

We make sure If we selected Spot for Capacity type, then we recommend specifying multiple instance types to enhance availability.
We will choose the minimum available node as 1
Maximum available node as 4
Desired nodes as 1

Make sure to add a label to this node group.
type = SPOT

After creation of node groups we will have one available node for each node group.



Nodes (2) Info

Node name	Instance type	Node group	Created	Status
ip-172-31-17-235.us-east-2.compute.internal	t2.small	spot-ng	Created 5 hours ago	Ready
ip-172-31-38-161.us-east-2.compute.internal	t2.small	my-ng	Created July 27, 2022, 11:04 (UTC+05:30)	Ready

Node groups (2) Info

	Group name	Desired size	AMI release version	Launch template	Status
○	my-ng	1	1.22.9-20220629	-	Active
○	spot-ng	1	1.22.9-20220629	-	Active

CREATING A DEPLOYMENT AND A SERVICE

We will create a deployment deployment.yaml file and apply it to launch the pods
$ Kubectl apply -f deployment.yaml

Next we will create a service file for the above deployment
$ Kubectl apply -f service.yaml

We will serve our deployment using ingress

INSTALLING INGRESS USING HELM

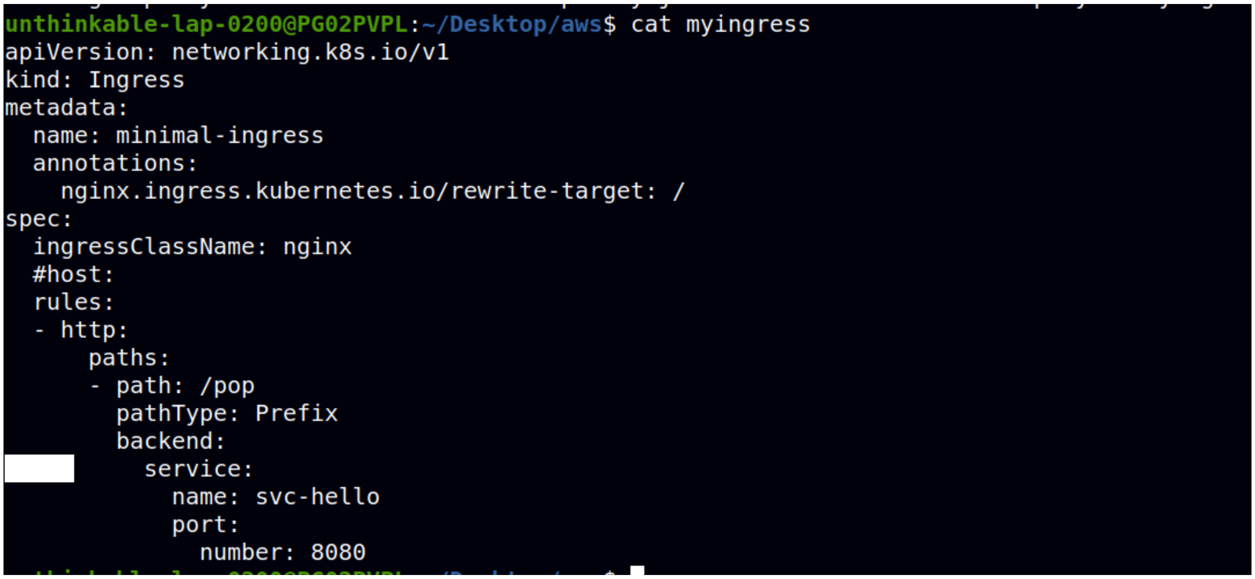
Enter the following command to add the Helm ingress-nginx repo:
$ helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

Command to create a namespace for ingress

```
$ kubectl create namespace nginx-ingress-sample
```

```
Command to deploy ingress resources using helm
$ helm install my-nginx ingress-nginx/ingress-nginx \
--namespace nginx-ingress-sample \
--set controller.metrics.enabled=true \
--set-string
controller.metrics.service.annotations."prometheus\.io/port"="10254" \
--set-string
controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

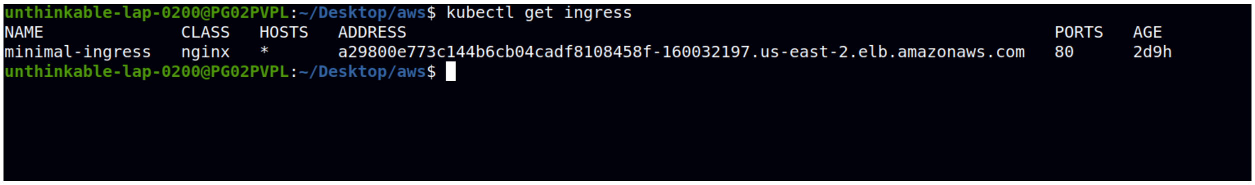
Now we will create a ingress file for our service



```
unthinkable-lap-0200@PG02PVPL:~/Desktop/aws$ cat myingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  #host:
  rules:
  - http:
      paths:
      - path: /pop
        pathType: Prefix
        backend:
          service:
            name: svc-hello
            port:
              number: 8080
```

Now we will apply the file
$ kubectl apply -f <ingress-file-name>

And now we will get our ingress load balancer url using command:
$ kubectl get ingress



```
unthinkable-lap-0200@PG02PVPL:~/Desktop/aws$ kubectl get ingress
NAME              CLASS   HOSTS   ADDRESS                                                                    PORTS   AGE
minimal-ingress   nginx   *       a29800e773c144b6cb04cadf8108458f-160032197.us-east-2.elb.amazonaws.com   80      2d9h
unthinkable-lap-0200@PG02PVPL:~/Desktop/aws$
```

INSTALLING KUBERNETES METRICS SERVER

https://docs.aws.amazon.com/eks/latest/userguide/metrics-server.html

Metrics Server is a scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines.

Use Case

- CPU/Memory based horizontal autoscaling (learn more about Horizontal Autoscaling)
- Automatically adjusting/suggesting resources needed by containers (learn more about Vertical Autoscaling)

We can use this commands:

$ kubectl apply -f
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

To get metrics server deployment

```
$ kubectl get deployment metrics-server -n kube-system
```

INSTALLING HORIZONTAL POD AUTOSCALER

https://docs.aws.amazon.com/eks/latest/userguide/horizontal-pod-autoscaler.html

We can directly use this commands :

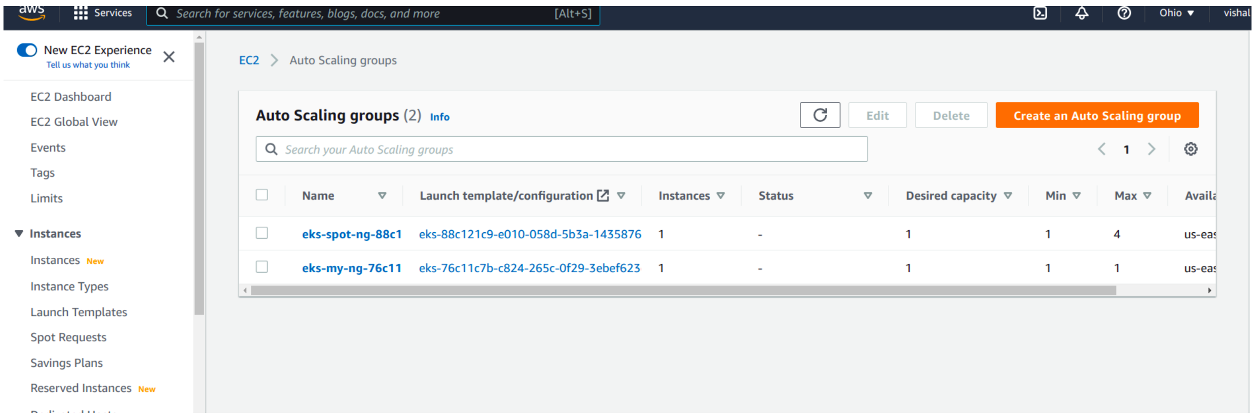
```
$ kubectl autoscale deployment <deployment-name> --cpu-percent=50 --min=1 --max=10
```

INSTALLING CLUSTER AUTOSCALER ON CLUSTER

The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail or are rescheduled onto other nodes. The Cluster Autoscaler is typically installed as a Deployment in your cluster.

We can refer to this document to install an autoscaler in our eks cluster:
https://docs.aws.amazon.com/eks/latest/userguide/autoscaling.html

After installing cluster autoscaler it wil create a autoscaling group for both out our node group in AWS autoscaling groups console



Auto Scaling groups (2) Info

Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max	Availa
eks-spot-ng-88c1	eks-88c121c9-e010-058d-5b3a-1435876	1	-	1	1	4	us-eas
eks-my-ng-76c11	eks-76c11c7b-c824-265c-0f29-3ebef623	1	-	1	1	1	us-eas

We will dynamically scale the spot node group
For creating a dynamic scaling policy
Go to auto scaling groups →Automatic scaling → Dynamic scaling policies →create a Dynamic Autoscaling policy.

We will choose cpu usage to our desired usage after that we want to scale out nodes.
And warmup time for nodes .

Make Sure scale in is enabled.

INSTALLING NODE TERMINATION HANDLER ON CLUSTER

To install node termination handler we will use these command

```
$ helm repo add eks https://aws.github.io/eks-charts
```

```
$ helm install aws-node-termination-handler \
            --namespace kube-system \
            --version 0.15.4 \
            --set nodeSelector.type= SPOT\
            eks/aws-node-termination-handler
```

To manage the Availability of our website we will create two deployments with a single service to handle no downtime during the reclamation of nodes by aws.

To make sure there is at least one pod available on each node we will use selectors to deploy on each node .

We can generate load on out load balancer endpoint using

```
$ kubectl run -i \
   --tty load-generator \
   --rm --image=busybox \
   --restart=Never \
   -- /bin/sh -c "while sleep 0.01; do wget -q -O- <out-end-point>; done"
```

To get HPA we can use:

```
$ kubectl get hpa
```

Thank you.