EKS Cluster

Introduction:

EKS cluster is provided by AWS as a PAAS service.

Objective:

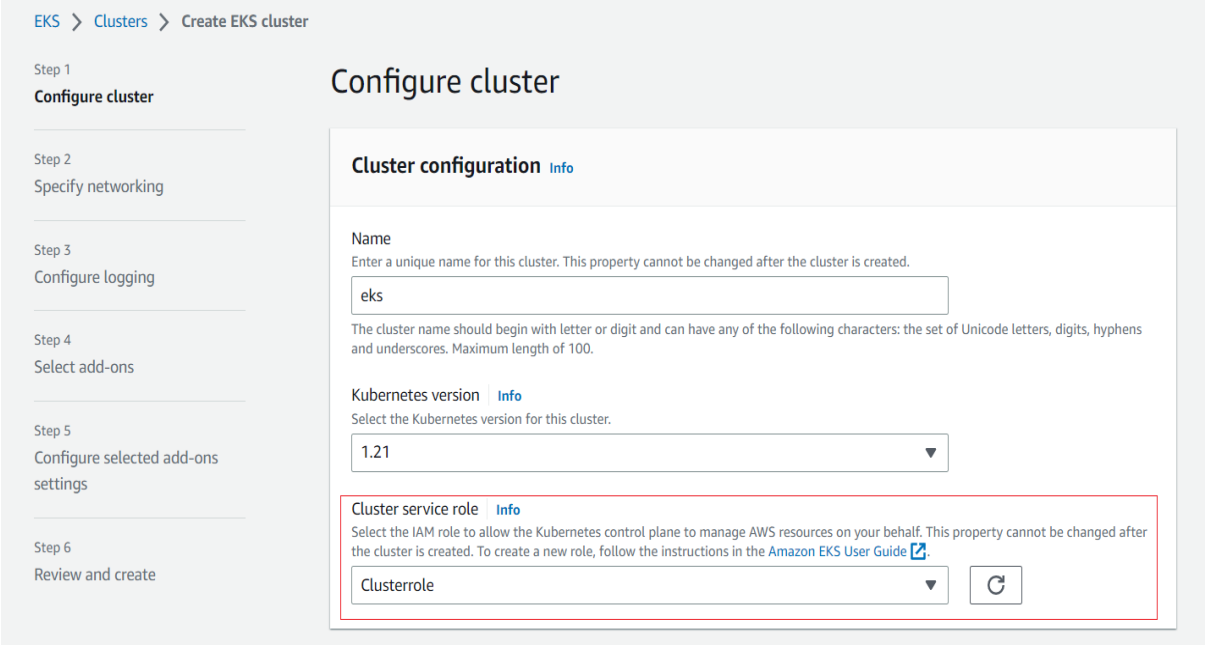
1. Create an EKS cluster

1. Create an EKS Cluster

Step 1:

In AWS, search for **Elastic Kubernetes Service** and click it.

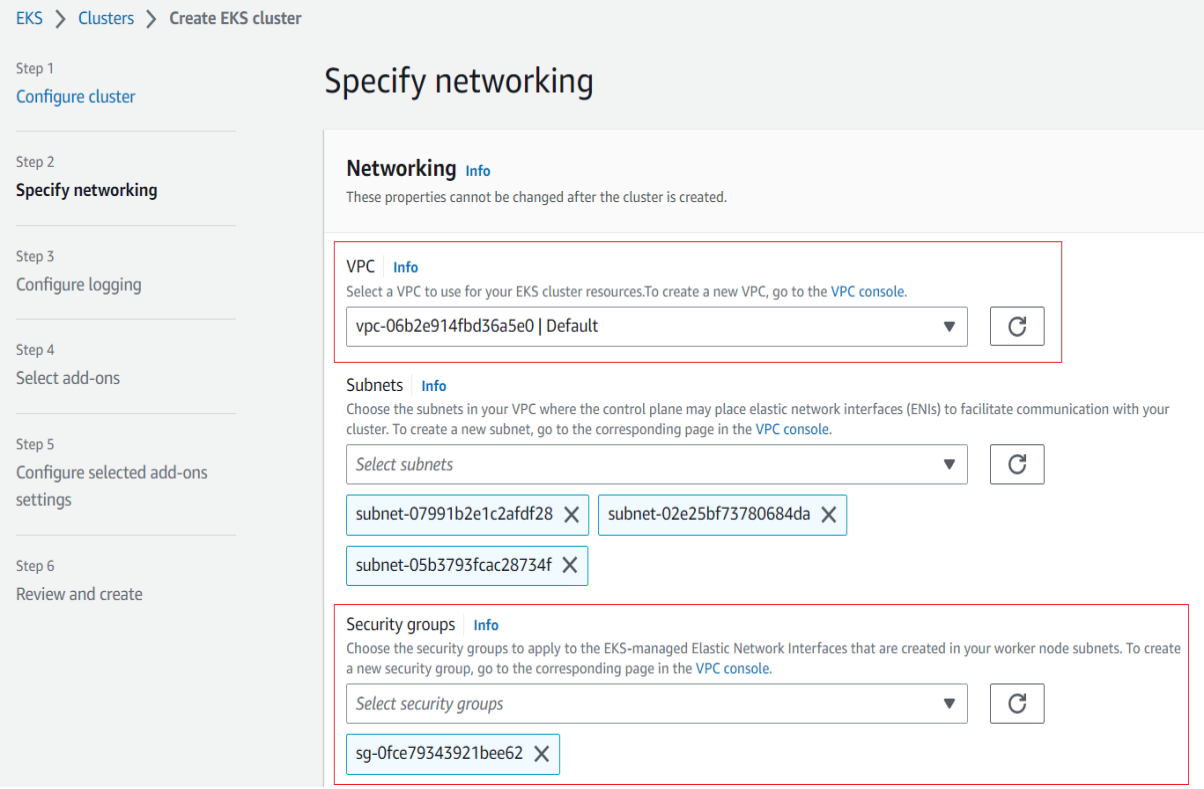
Go for create a cluster and fill the below information.



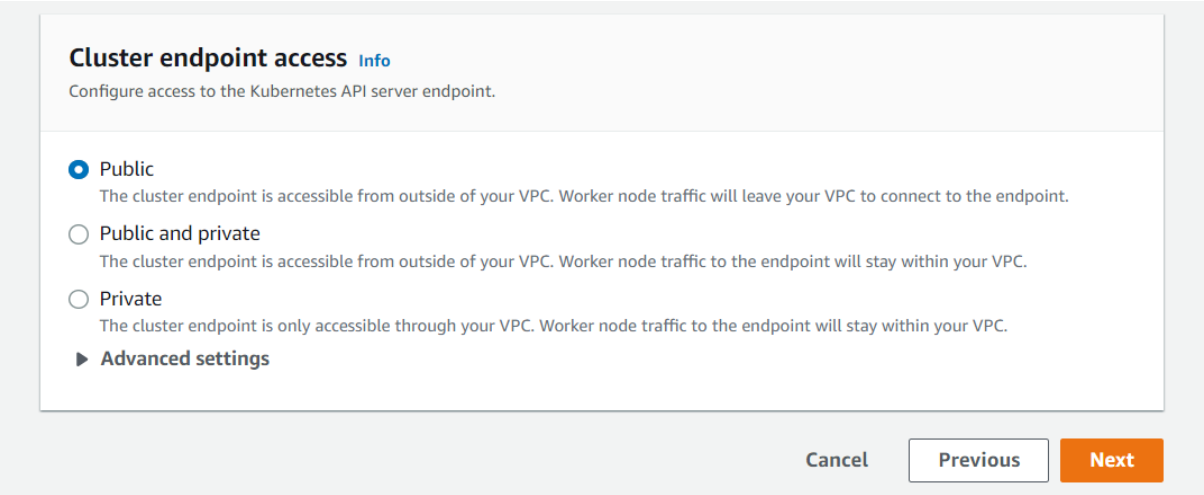
Above we are providing it a role Clusterrole which is having the below permission.

- AmazonEKSClusterPolicy : IAM role to allow the Kubernetes control plane to manage AWS resources on your behalf.

Click next and then follow the below steps.



Above we have selected the default VPC and the subnets related to that VPC. We have also selected the default security group and make sure that necessary ports are open. In our example have opened all the ports.



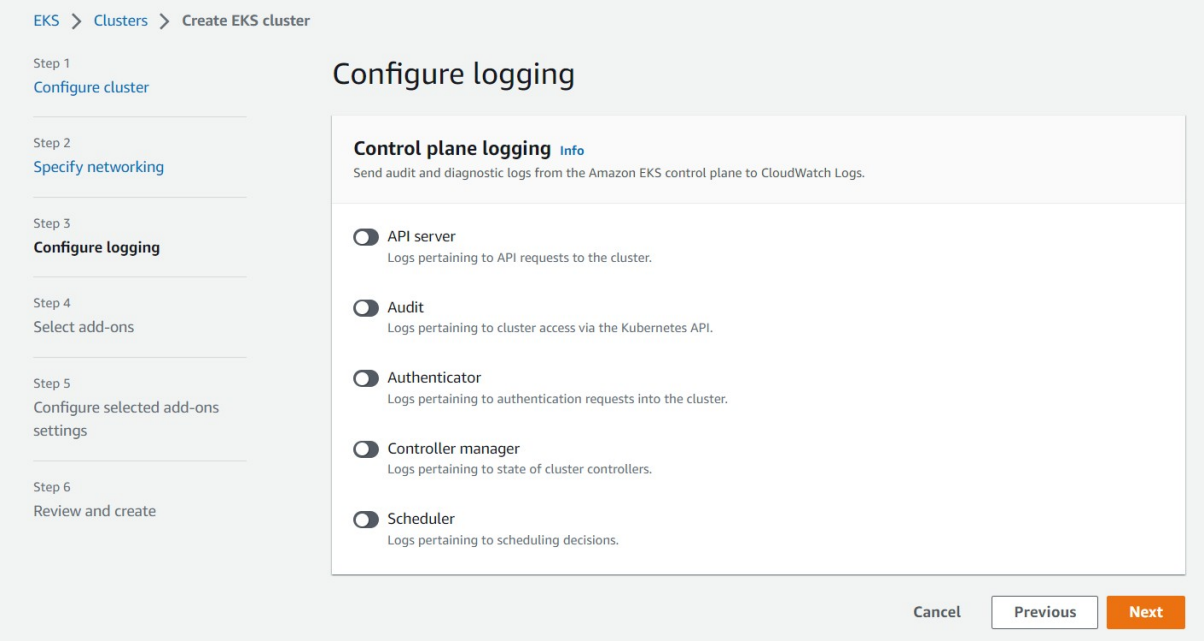
Now select the Cluster endpoint access, we have a three options which are Public, Public and private , Private.

Public
The cluster endpoint is accessible from outside of your VPC. Worker node traffic will leave your VPC to connect to the endpoint.

Public and private
The cluster endpoint is accessible from outside of your VPC. Worker node traffic to the endpoint will stay within your VPC.

Private
The cluster endpoint is only accessible through your VPC. Worker node traffic to the endpoint will stay within your VPC.

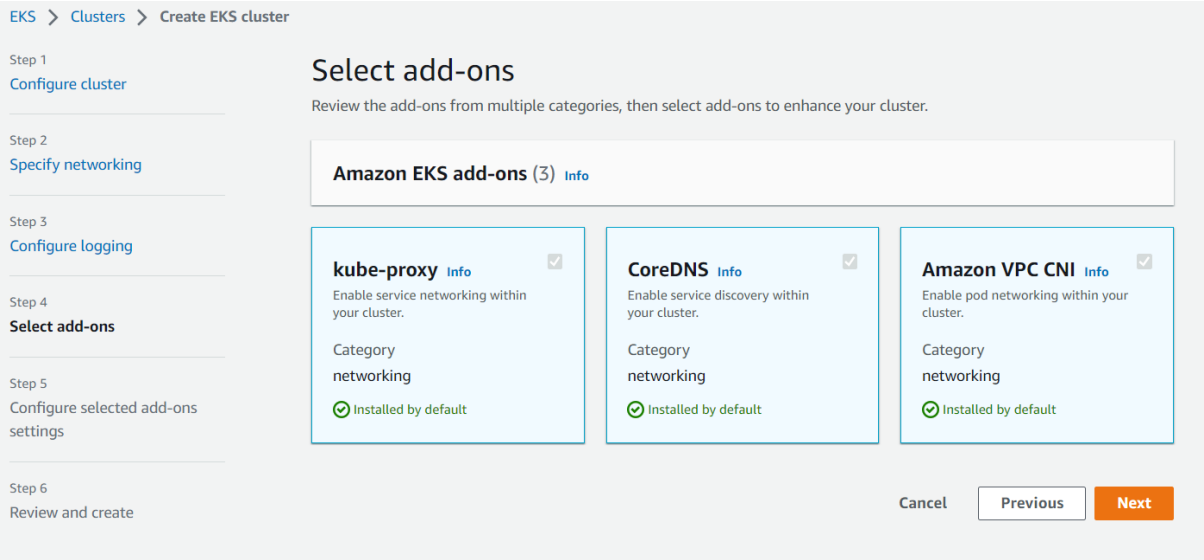
Choose as per your business requirement and click Next.



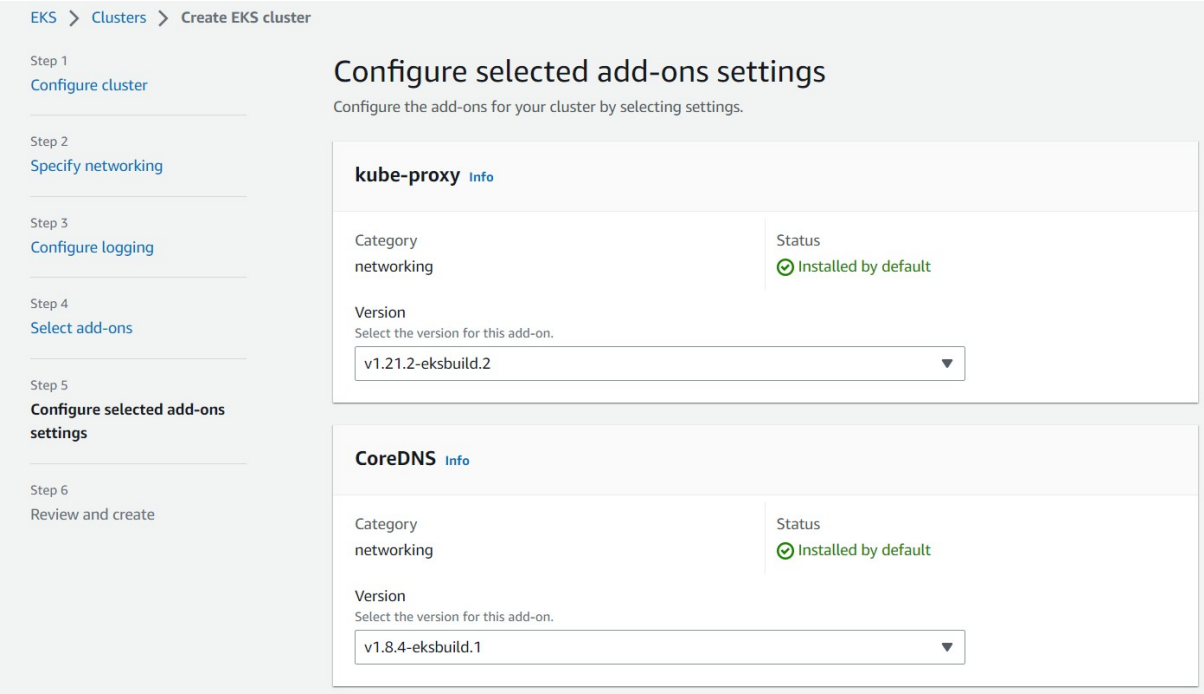
Above we have the logging option, we are keeping everything default and click Next.

Now AWS will install EKS add-ons which will be used for the networking part. These add-ons are.

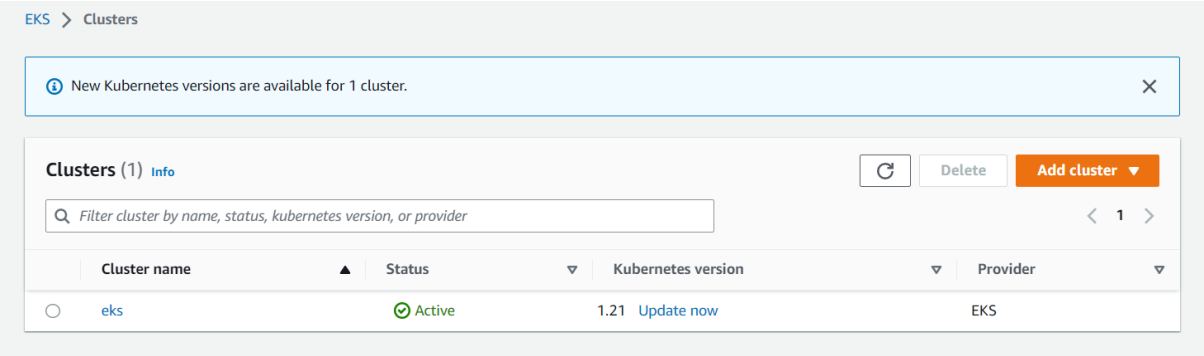
1. Kube-proxy
2. CoreDNS
3. Amazon VPC CNI



Click Next and it will ask for the versions for add-ons. We are keeping it default and click Next.



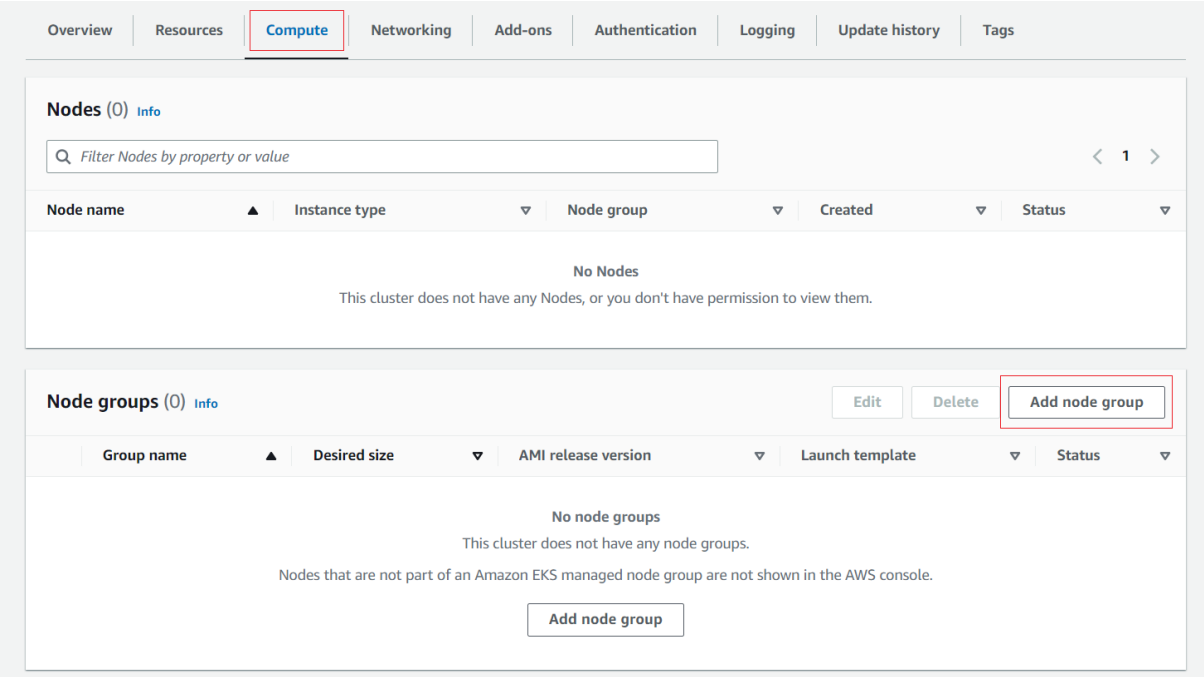
Next page will ask us to **review and create**. Go ahead and wait for our cluster.



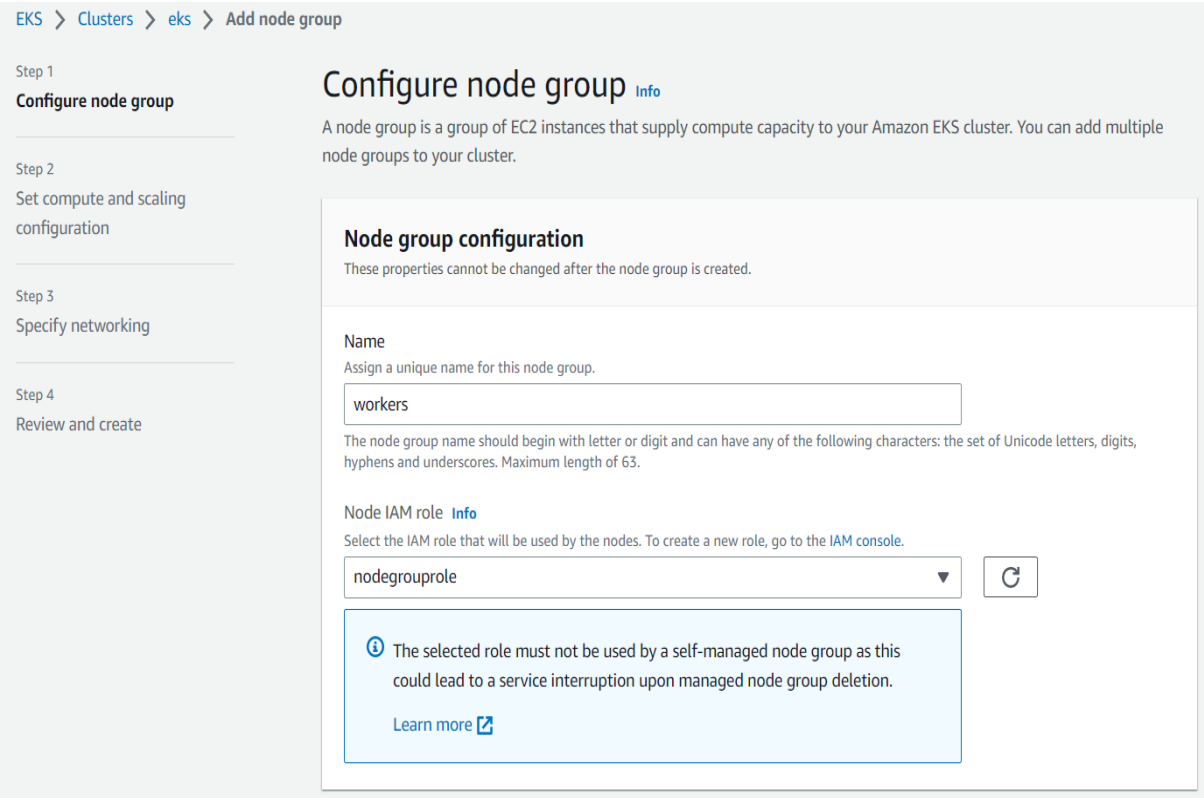
Now our cluster is in active mode, we need to add nodes to it.

Step 2: Add nodes to the cluster

Now click our cluster and go to compute section. Further we need to add a node group which will be having the worker nodes.

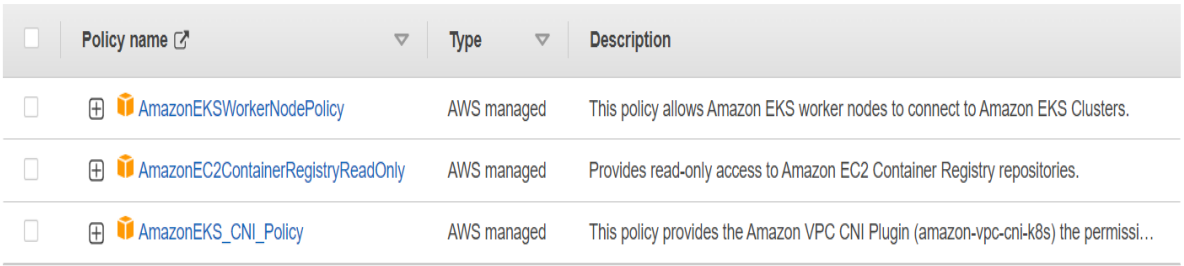


Below we have given a name **workers** to our node group and also a role has been assigned to it.



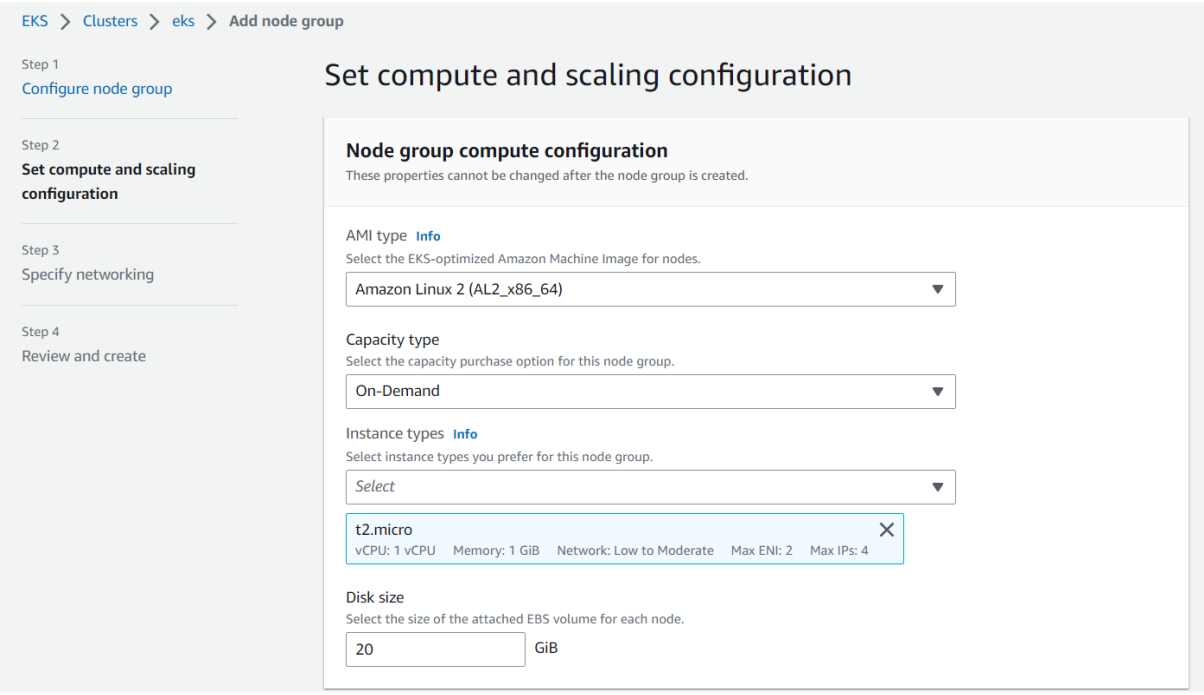
The role we have assigned is having three permission required cluster formation.

AmazonEKSWorkerNodePolicy	This policy allows Amazon EKS worker nodes to connect to Amazon EKS Clusters.
AmazonEC2ContainerRegistryReadOnly	Provides read-only access to Amazon EC2 Container Registry repositories.
AmazonEKS_CNI_Policy	This policy provides the Amazon VPC CNI Plugin (amazon-vpc-cni-k8s) the permissions it requires to modify the IP address configuration on your EKS worker nodes. This permission set allows the CNI to list, describe, and modify Elastic Network Interfaces on your behalf.



Then click next and set compute and scaling configuration.

We have used t2.micro instance type and 20GB in disk size.



Node group scaling configuration

Desired size

Set the desired number of nodes that the group should launch with initially.

2 nodes

Minimum size

Set the minimum number of nodes that the group can scale in to.

2 nodes

Maximum size

Set the maximum number of nodes that the group can scale out to.

2 nodes

Node group update configuration Info

Maximum unavailable

Set the maximum number or percentage of unavailable nodes to be tolerated during the node group version update.

- (●) Number — Enter a number
- ( ) Percentage — Specify a percentage

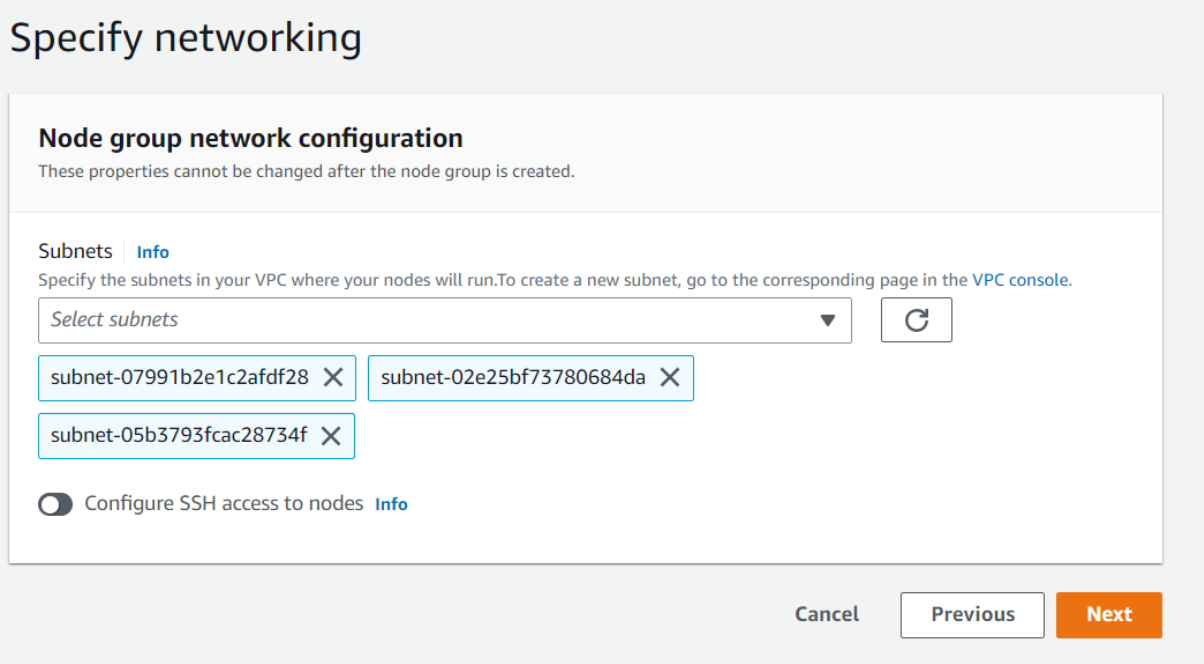
Value

1 node

We have set the desired, minimum and maximum nodes to 2.

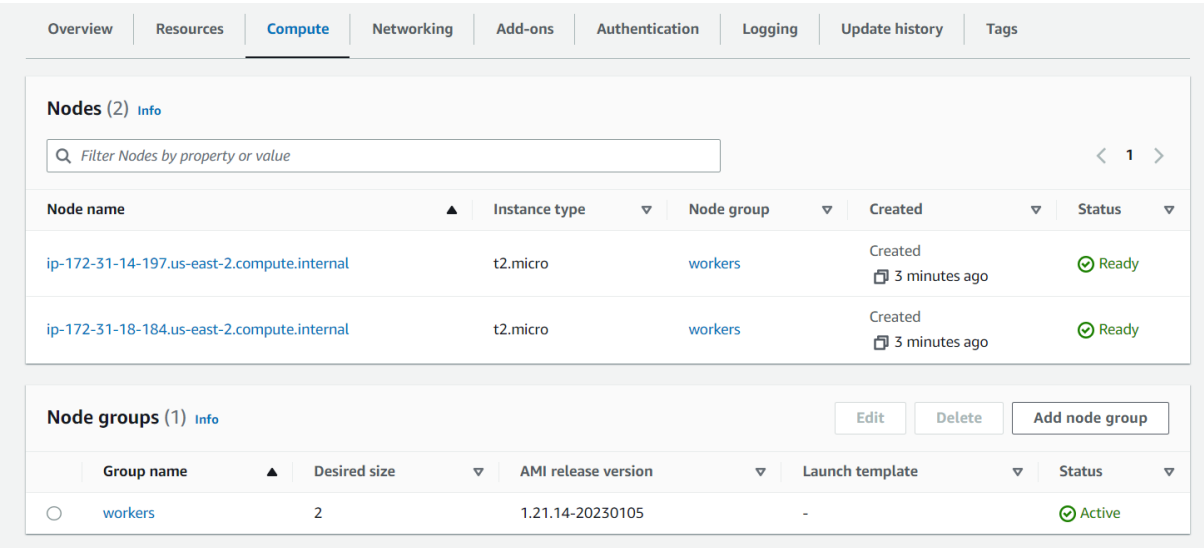
Maximum unavailable number has been set to 1 only.

Click next.



We are using the same subnets which are a part of our default VPC.

Click **Next**, further click **Review and Create**.



Now our cluster is completely ready and worker nodes are active too.

Step3: Access the cluster

Now we will be accessing the cluster from our machine. We need below packages as prerequisites.

1. AWS Installation
2. Kubectl Installation
3. Update kubeconfig file

1. **AWS Installation**:

   Use the below command to install AWS in our machine.

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

```
unzip awscliv2.zip
```

```
sudo ./aws/install
```

   Once the AWS installation is down, we can check the status using below command.

```
aws –version
```

   Now further configure aws using aws configure command. Also use **AWS access Key ID** and **AWS Secret Access Key** to configure it.

```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]: us-east-2
Default output format [None]:
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~#
```

2. **Kubectl Installation:**

   use the below command to install Kubectl package.

   Download the kubectl binary for your cluster's Kubernetes version from Amazon S3

```
curl -O
https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2022-10-31/bin/
linux/amd64/kubectl
```

Apply execute permissions to the binary

```
chmod +x ./kubectl
```

Copy the binary to a folder in your PATH. If you have already installed a version of kubectl, then we recommend creating a $HOME/bin/kubectl and ensuring that $HOME/bin comes first in your $PATH.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export
PATH=$PATH:$HOME/bin
```

Now our Kubectl package has been installed.

3. **Update kubeconfig file:**

Now our last step is to update the kubeconfig file in order to access the cluster. Use the below command.

```
aws eks --region us-east-2 update-kubeconfig --name eks
```

```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# aws eks --region us-east-2 update-kubeconfig --name eks
Added new context arn:aws:eks:us-east-2:685421549691:cluster/eks to /root/.kube/config
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl get nodes
NAME                                         STATUS   ROLES    AGE   VERSION
ip-172-31-14-197.us-east-2.compute.internal   Ready    <none>   43m   v1.21.14-eks-fb459a0
ip-172-31-18-184.us-east-2.compute.internal   Ready    <none>   43m   v1.21.14-eks-fb459a0
root@ip-172-31-33-180:~#
```

Now our cluster is completely ready and we can access it.

Let’s run a pod using below command.

```
kubectl run pod –image nginx
```

```
root@ip-172-31-33-180:~# kubectl run pod --image nginx
pod/pod created
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod    1/1     Running   0          9s
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE   IP             NODE                                          NOMINATED NODE   READINESS GATES
pod    1/1     Running   0          76s   172.31.2.150   ip-172-31-14-197.us-east-2.compute.internal   <none>           <none>
root@ip-172-31-33-180:~#
```