Setup AWS EKS Cluster With eksctl Tool

Introduction

Elastic Kubernetes Service (EKS) is a fully managed Kubernetes service from AWS. In this lab, you will work with the AWS command line interface **(aws-cli)** and console, using command line utilities like **eksctl** to launch an EKS cluster and **kubectl** to provision a Kubernetes deployment and pod running instances of nginx, and create a LoadBalancer service to expose your application over the internet.

We need to install first the aws-cli, eksctl & kubectl tools on a server

aws-cli: The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

eksctl: is a simple CLI tool for creating and managing clusters on EKS - Amazon's managed Kubernetes service for EC2. It is written in Go, uses CloudFormation.

Kubectl: allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs etc..

To setup the above tools we may use

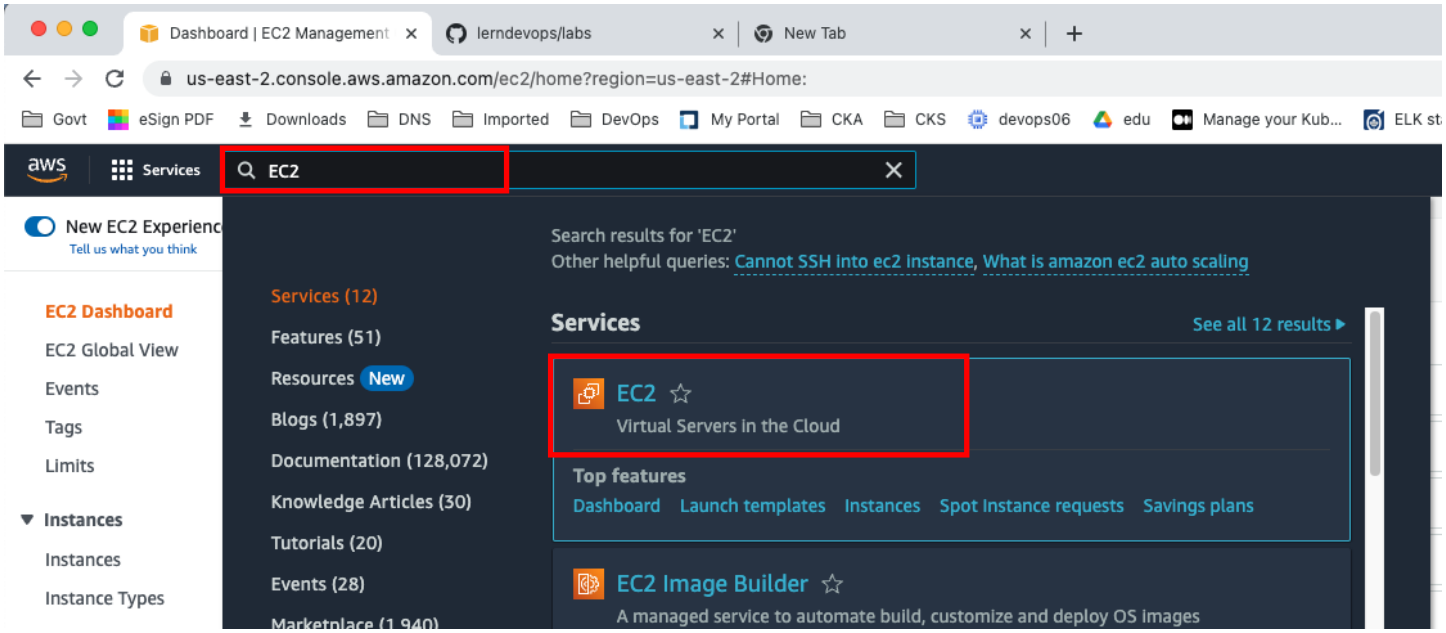
- **Use AWS Cloud Shell**
  - **here you will see both aws cli & kubectl already installed**
  - **you will need to install eksctl tool**
- **Create a EC2 VM & install**
- **install on your own laptops/desktops(windows or mac)**

go to: https://console.aws.amazon.com/

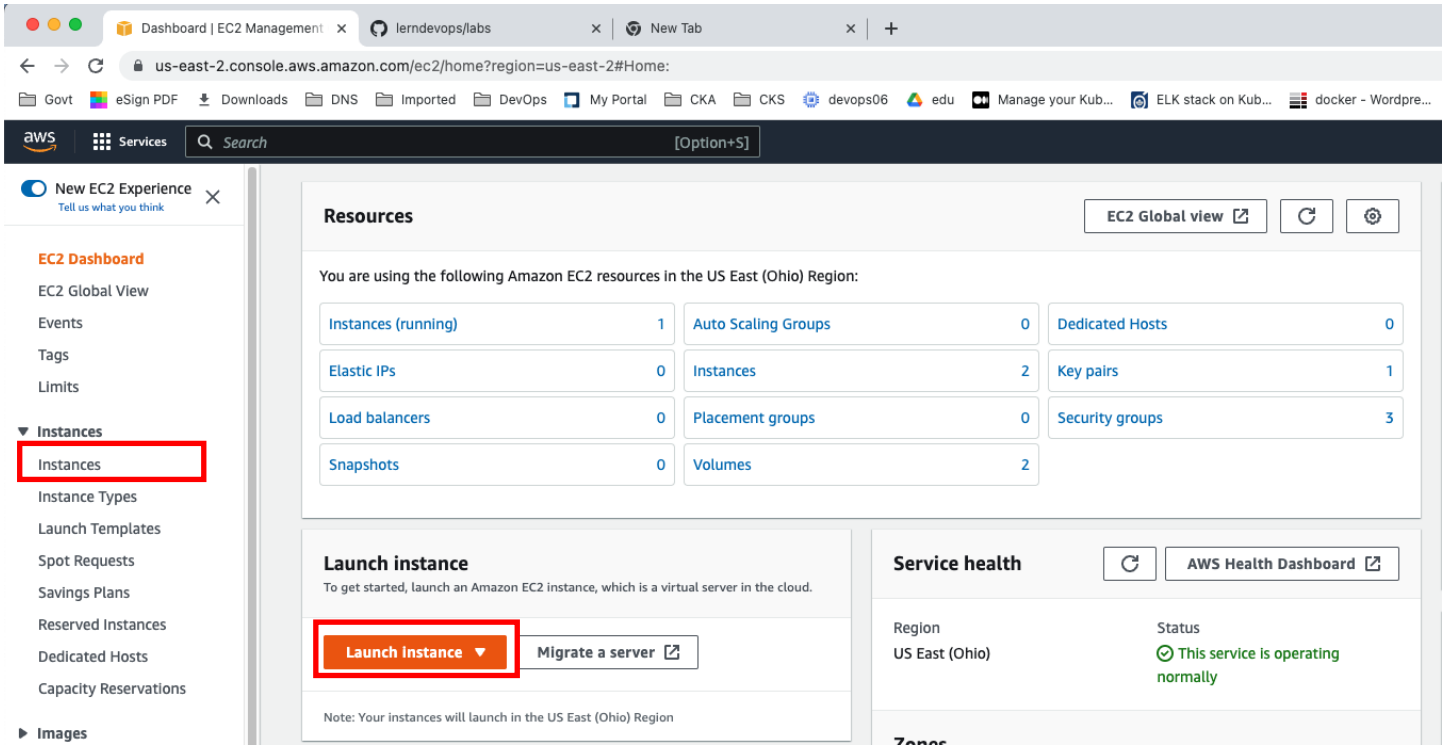
Login to your AWS account

Launch an EC2 Instance to Configure the Command Line Tools

Navigate to **EC2** > **Instances**.

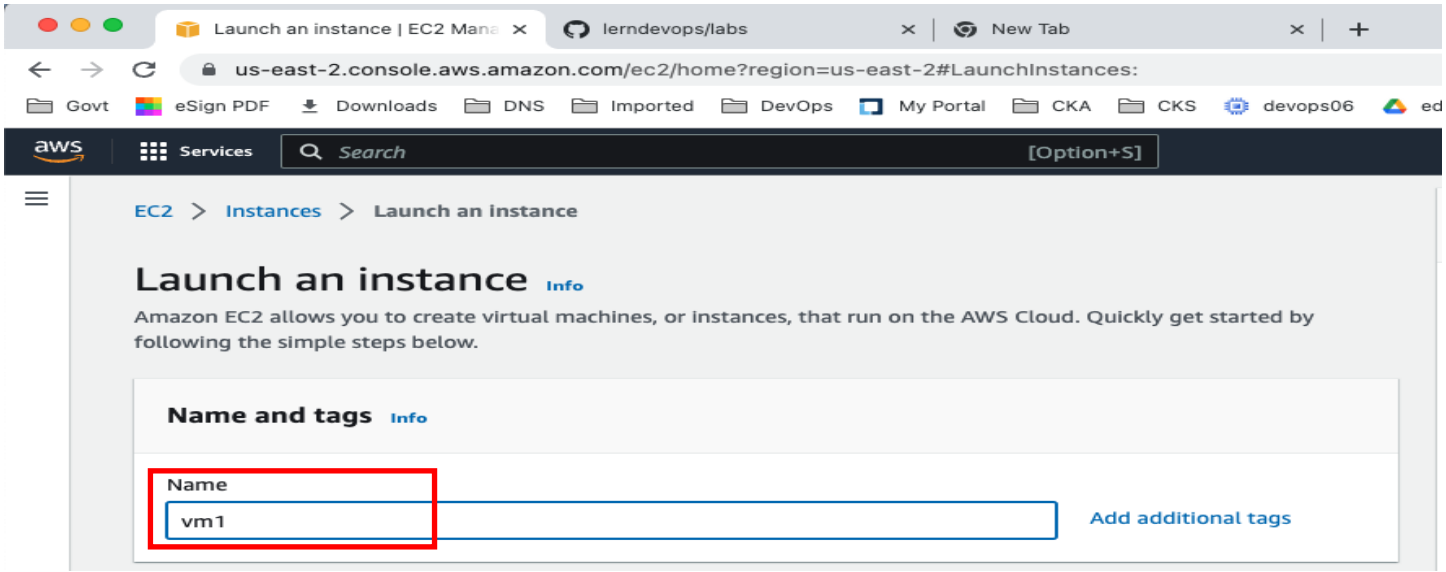


Click **Launch Instance**.

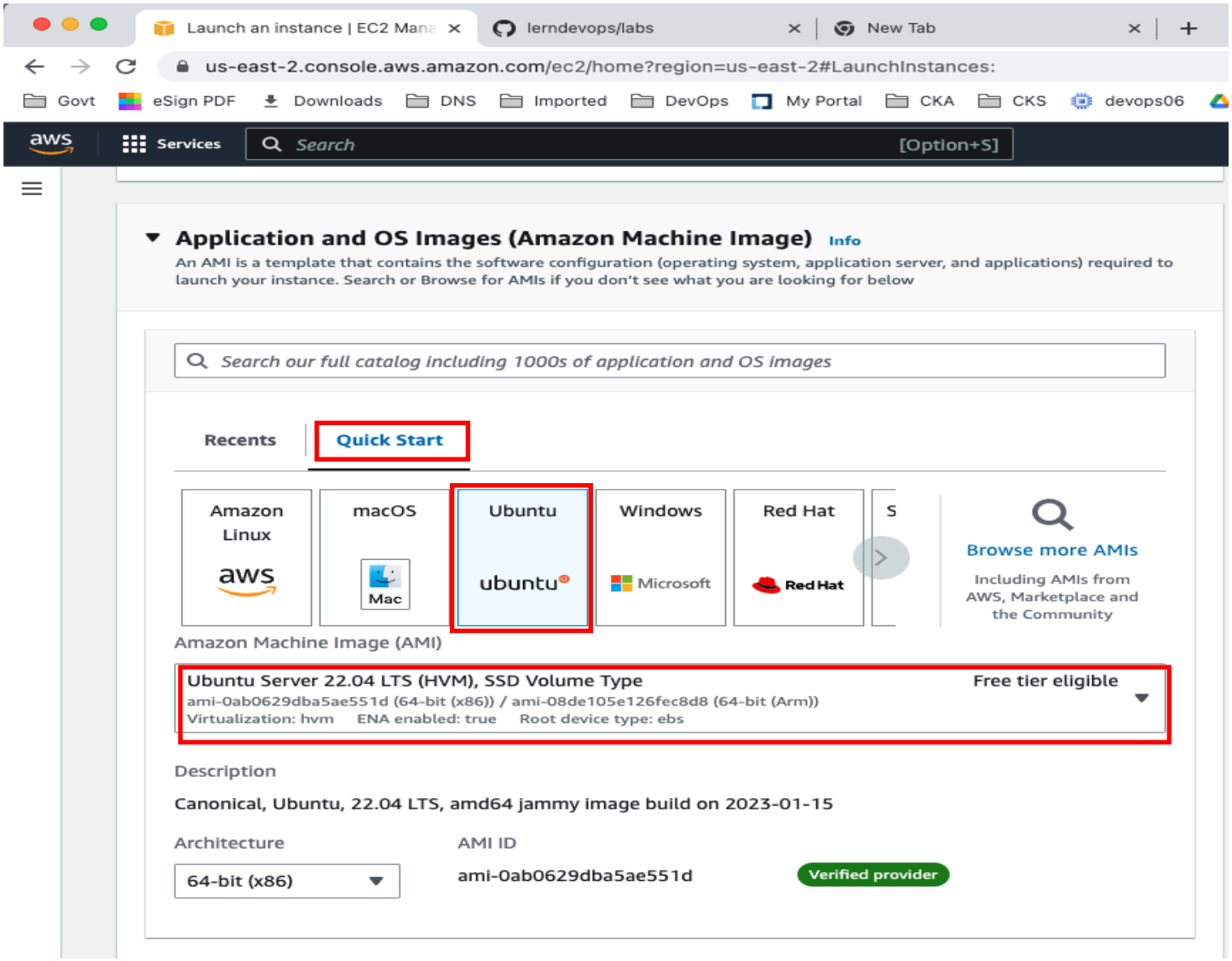


On the Launch an instance Page

Enter Name and tags: any value then scroll down

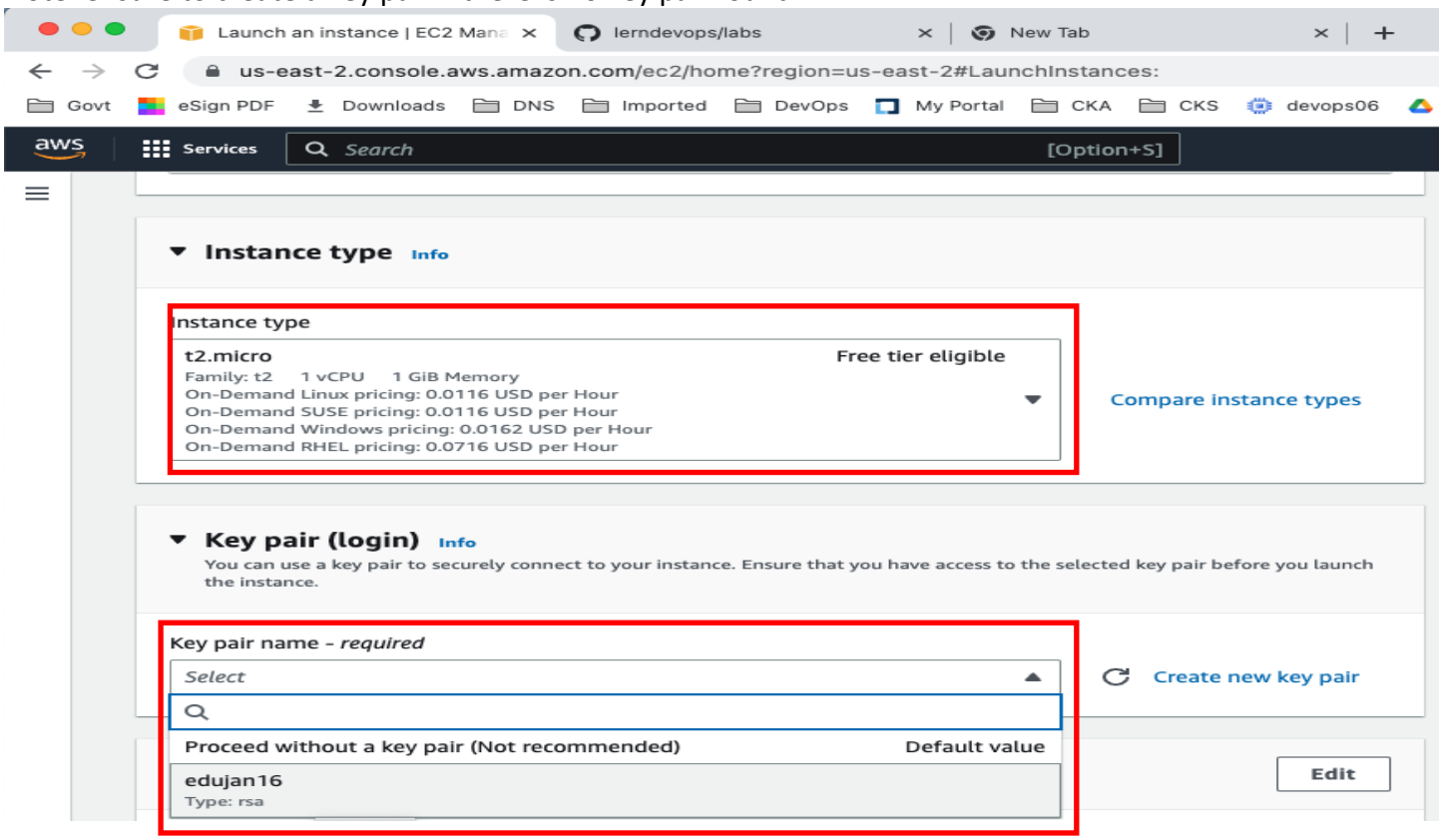


Choose an OS Image & Scroll down

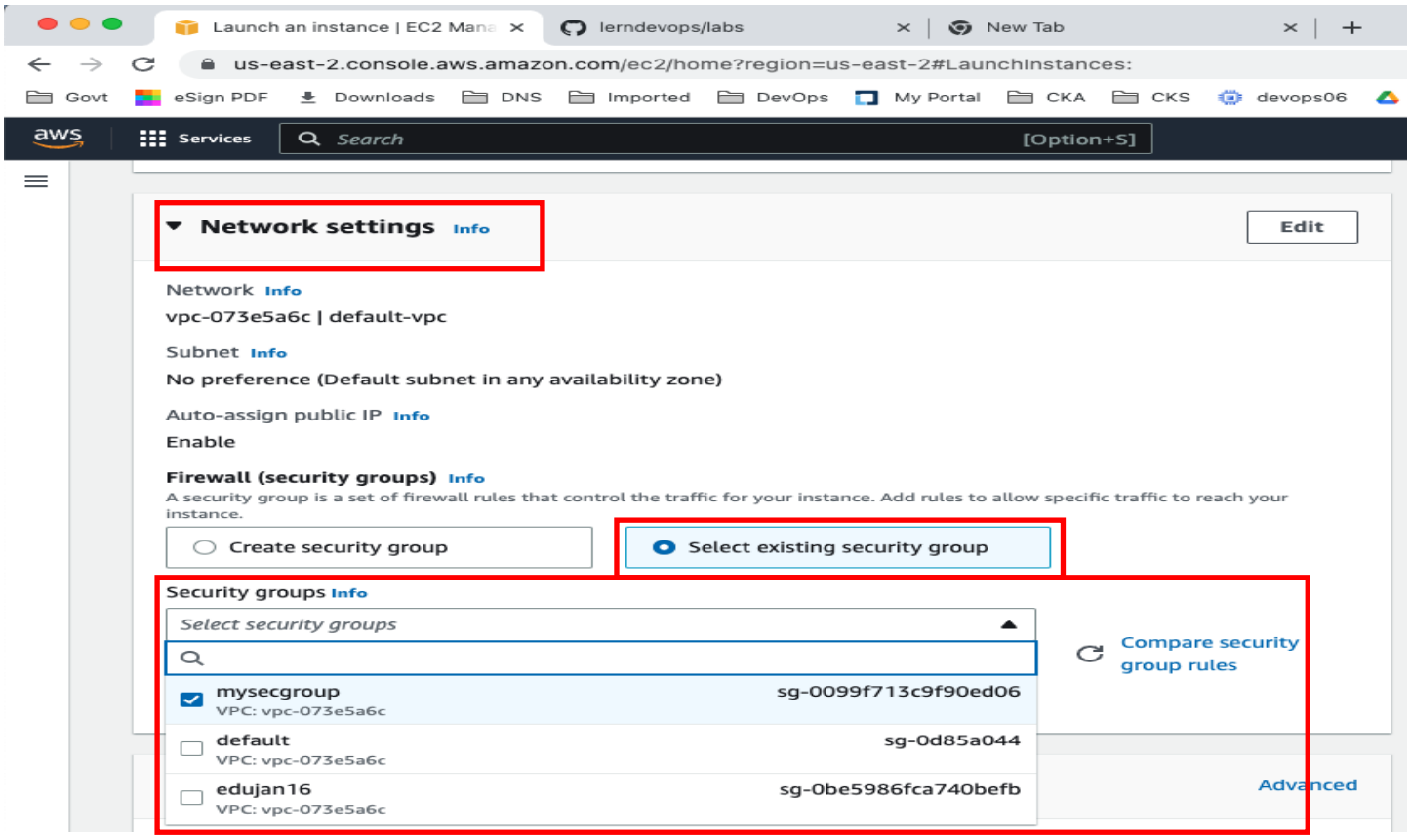


Choose Instance type & Key pair

Note: ensure to create a key pair if there is no key pair found

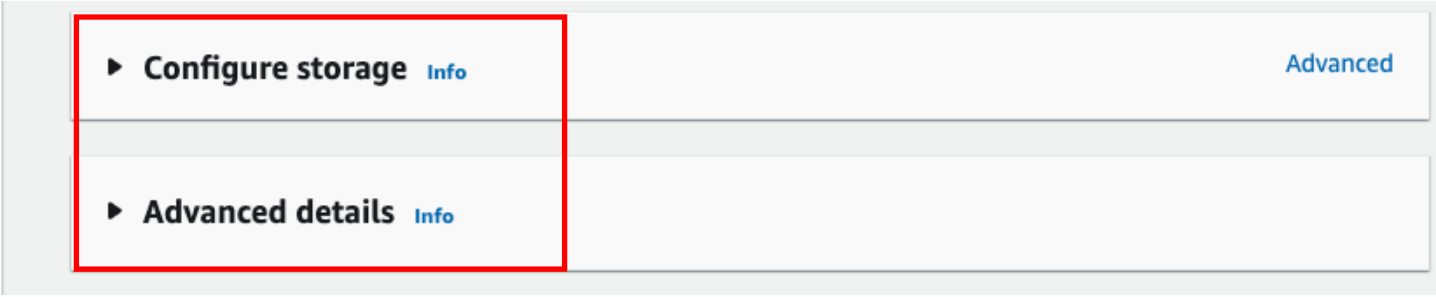


Network Settings: Choose an **existing Security Group** or **Create one** & Select as needed & Scroll Down

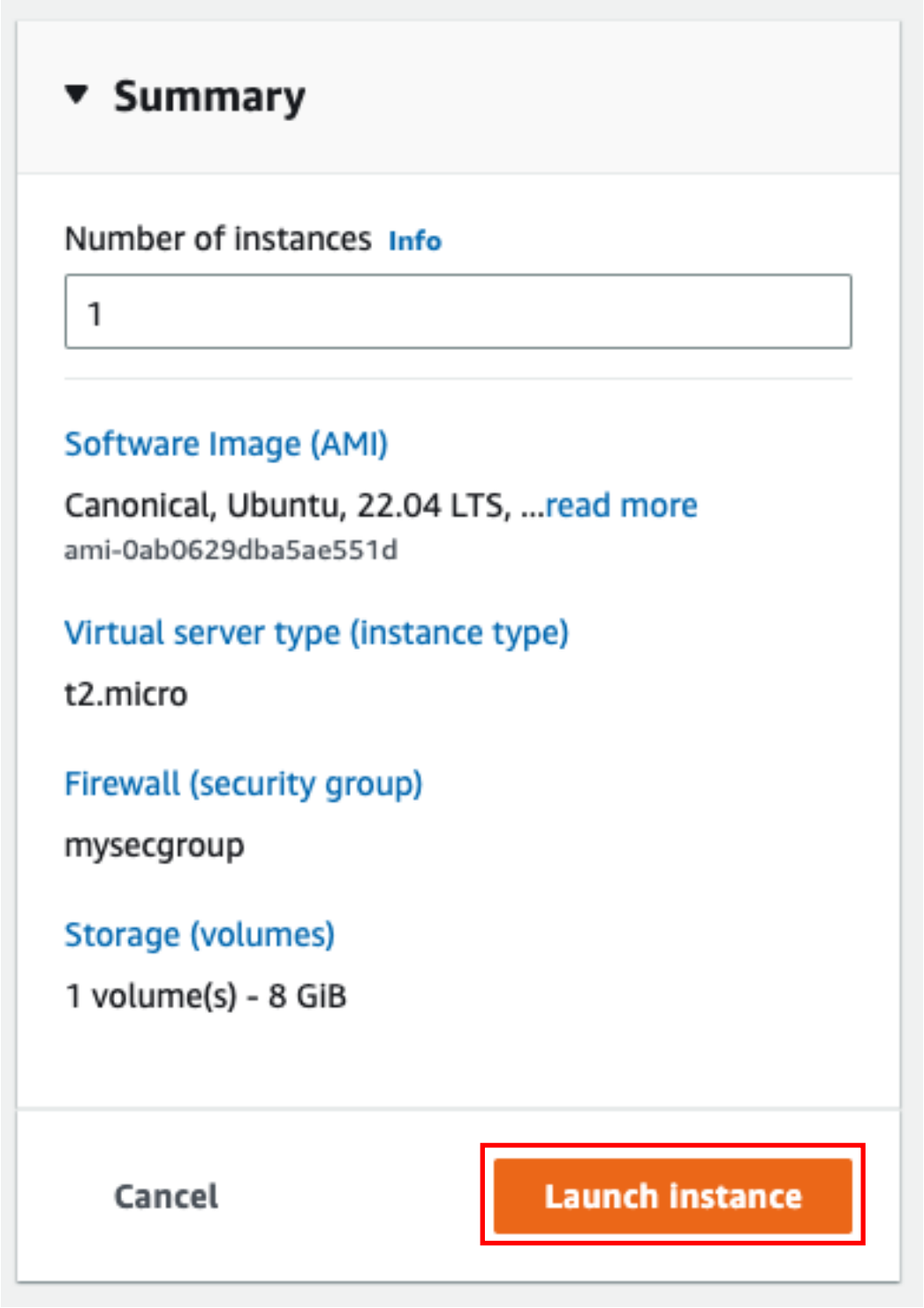


Configure Storage & Advanced details

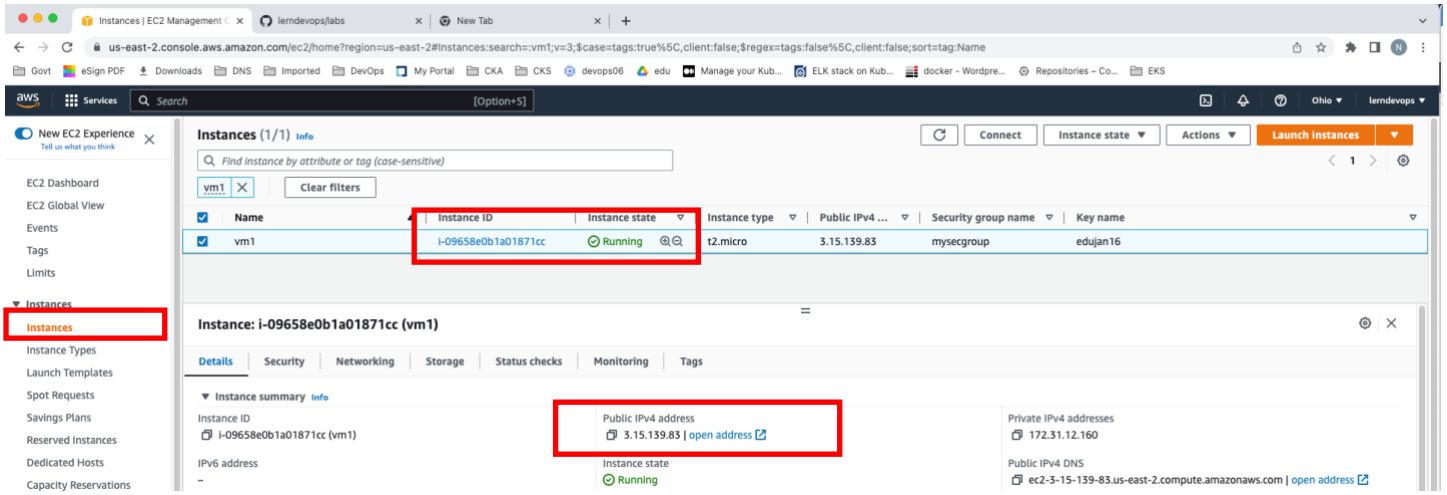
YOU DO NOT NEED TO CHANGE ANYTHING on Below Sections – leave the defaults values as they are.



Review the Summary & **Click on Launch Instance**



Then Go Back to the Instances Page -- You Should See the VM Running in couple of min



Login to VM/EC2 Created – Follow the below to Login

There are many ways to login to remote linux servers, some recommended options as below

Using AWS SSM: (common for windows or mac users)

https://github.com/lerndevops/labs/blob/master/cloud/aws/connect-to-aws-ec2-using-aws-ssm.pdf

MAC Users:

https://github.com/lerndevops/labs/blob/master/cloud/aws/connect-to-EC2-with-MAC-terminal.pdf

Windows Users:

https://github.com/lerndevops/labs/blob/master/cloud/aws/connect-to-EC2-with-mobaXterm.pdf

Once Logged In

Follow the installation Instruction from below to install the tools required

install AWS CLI:

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

install kubectl CLI:

https://kubernetes.io/docs/tasks/tools/

install eksctl CLI:

https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html

Validate versions after install:

- aws --version
- kubectl version --short --client
- eksctl version

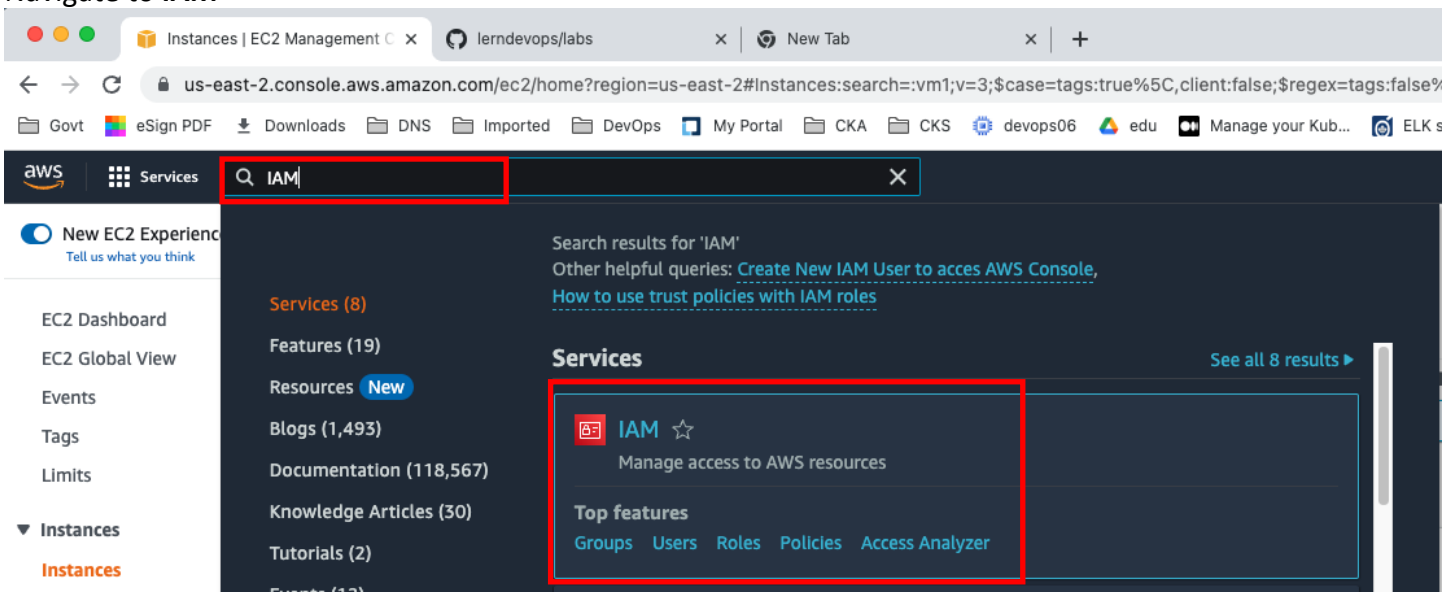
```
nareshwar@mbpro ~ %
nareshwar@mbpro ~ %
nareshwar@mbpro ~ % aws --version
aws-cli/2.9.19 Python/3.11.1 Darwin/22.3.0 source/arm64 prompt/off
nareshwar@mbpro ~ %
nareshwar@mbpro ~ % kubectl version --short --client
Flag --short has been deprecated, and will be removed in the future. The --short output will become the default.
Client Version: v1.26.1
Kustomize Version: v4.5.7
nareshwar@mbpro ~ %
nareshwar@mbpro ~ % eksctl version
0.127.0
nareshwar@mbpro ~ %
nareshwar@mbpro ~ %
```

Now To Authenticate to your AWS Account using aws command line tool we need to

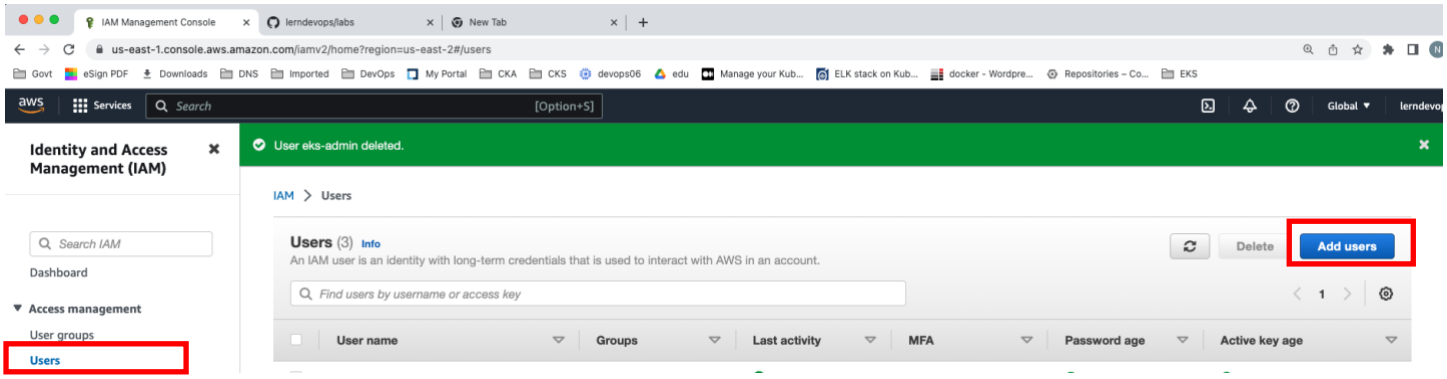
Create an IAM User with Admin Permissions

Note: you can fine grain your access level to IAM user accordingly

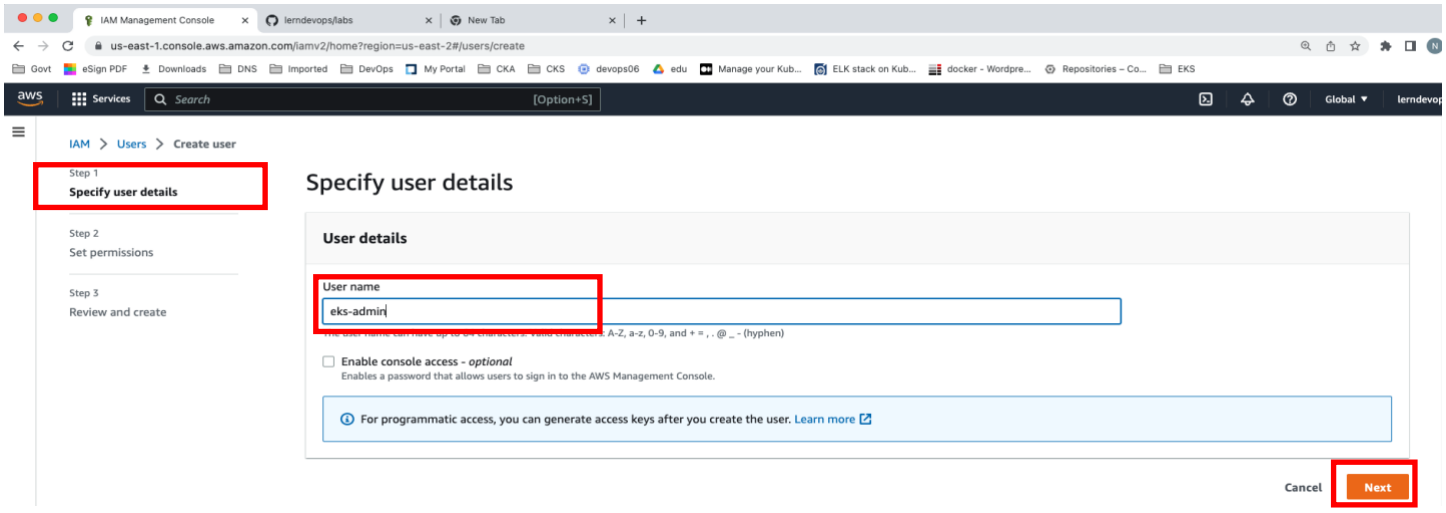
Navigate to **IAM**



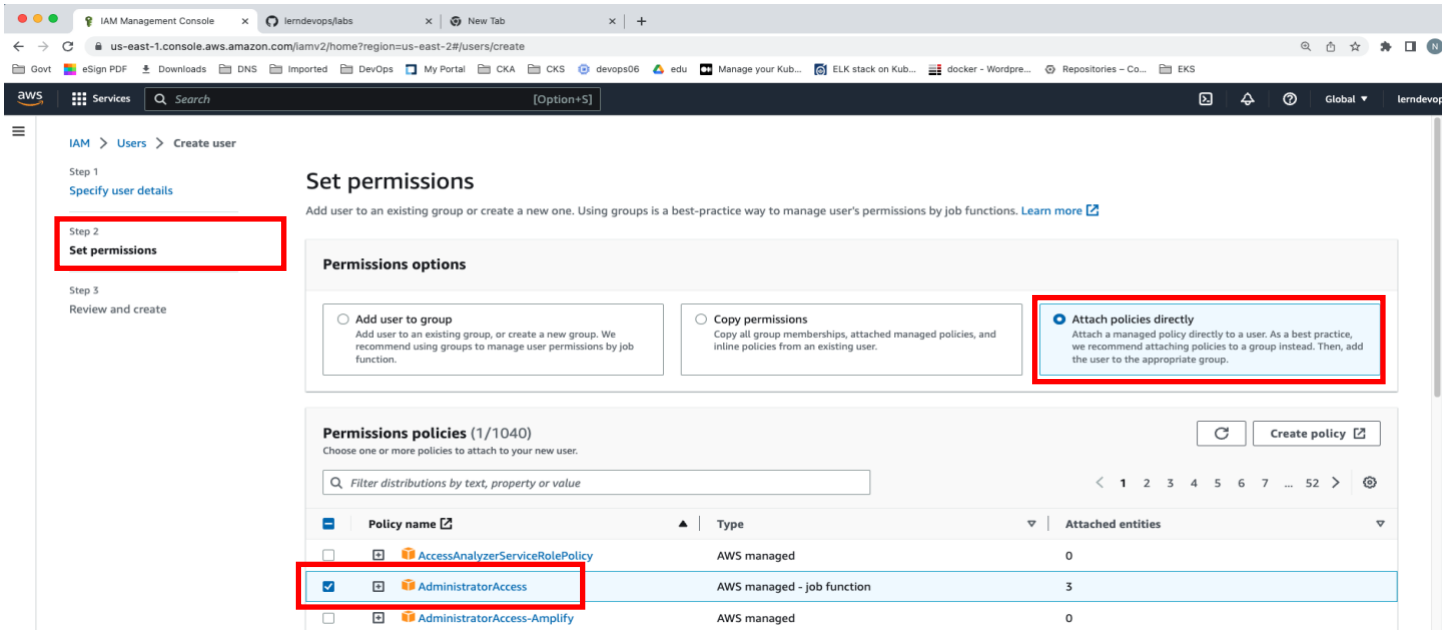
Users & Click **Add users**



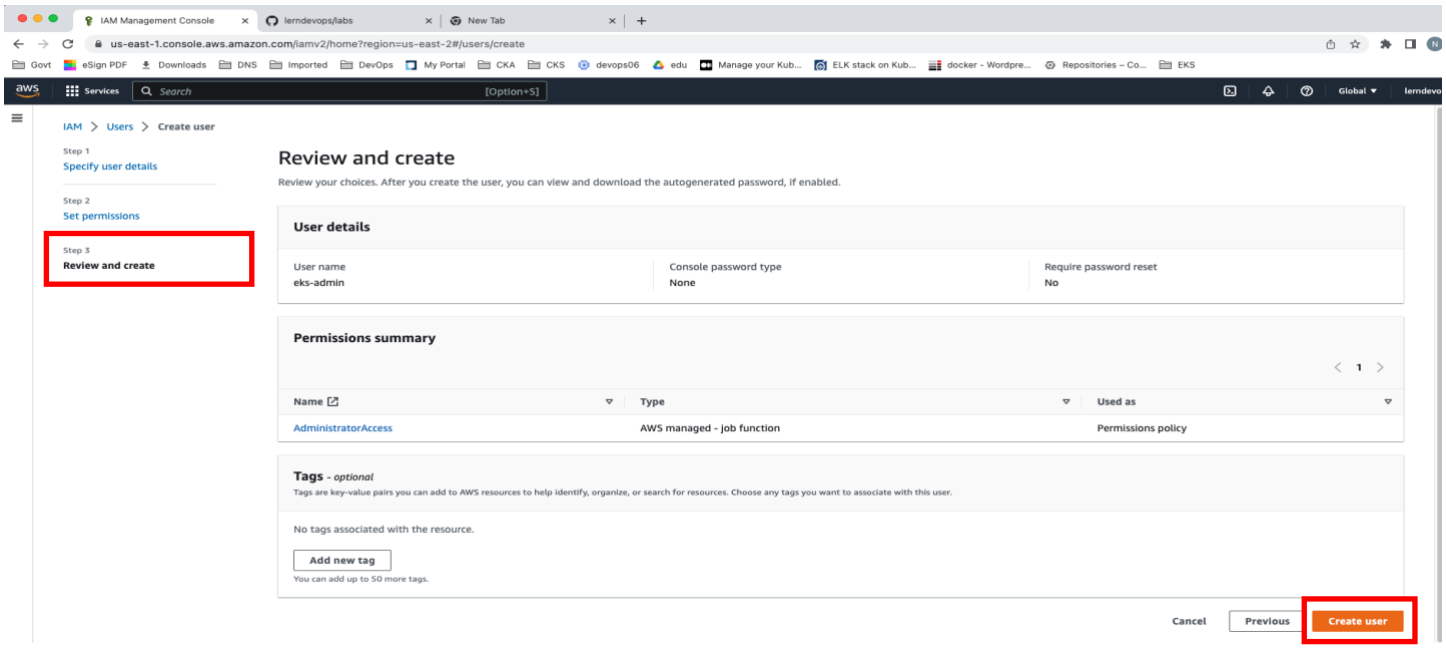
Specify user details & Click Next



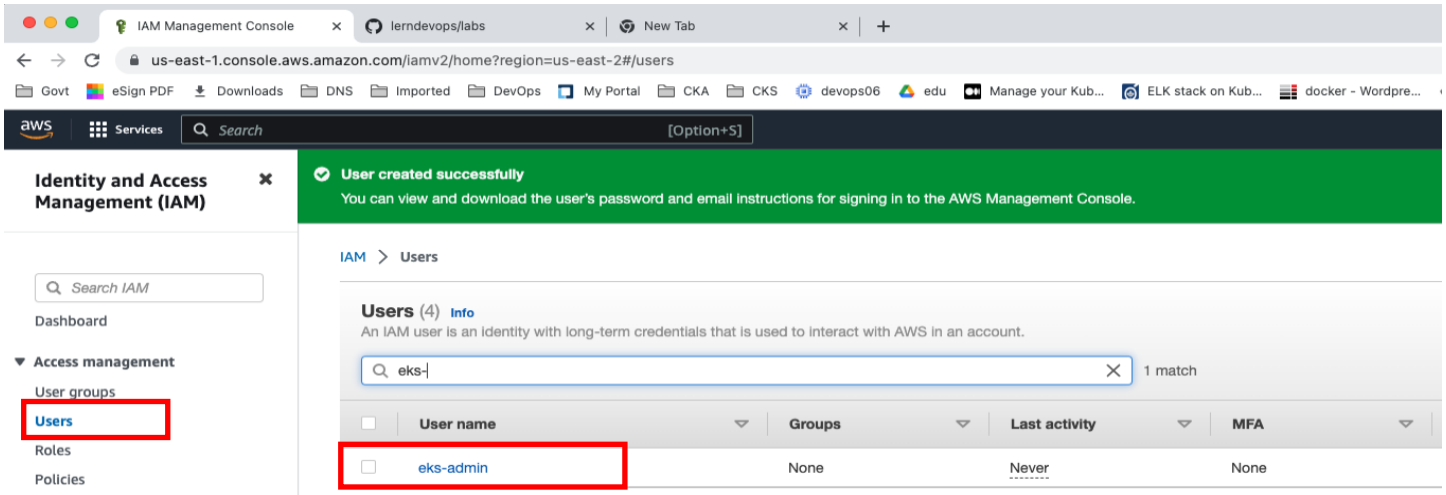
Set **Permissions** → Select **Attach policies directly** → Select **AdministratorAccess** → **Click Next**



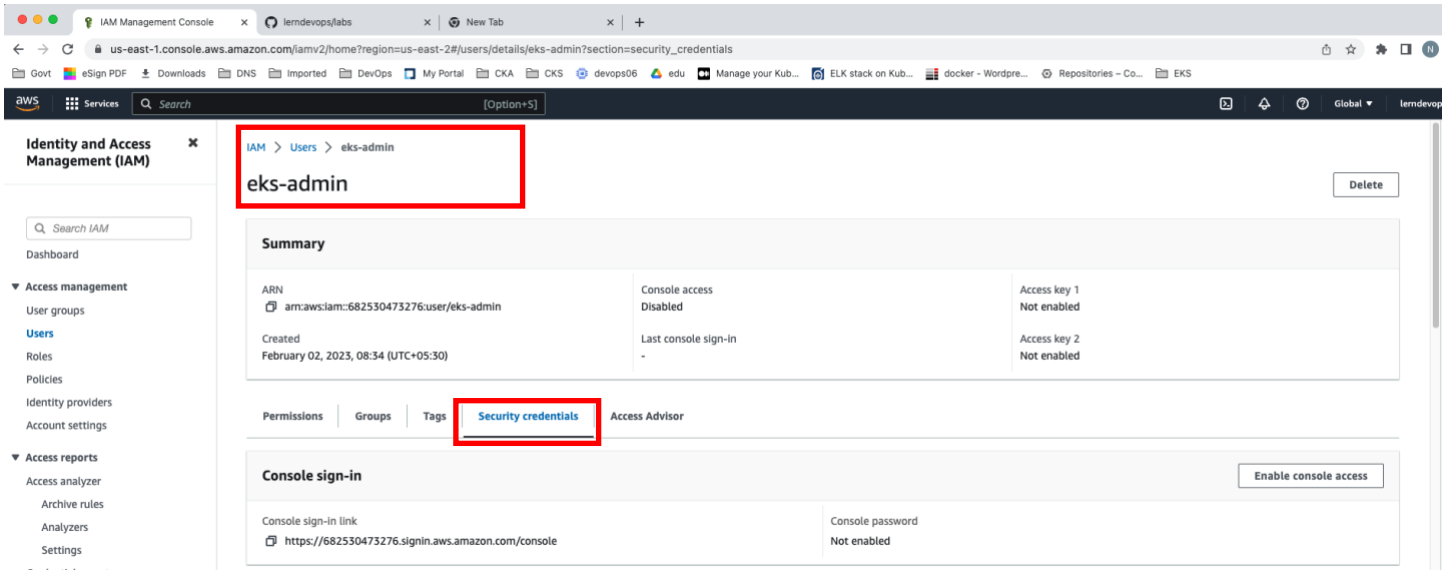
Review and Create → Click Create User



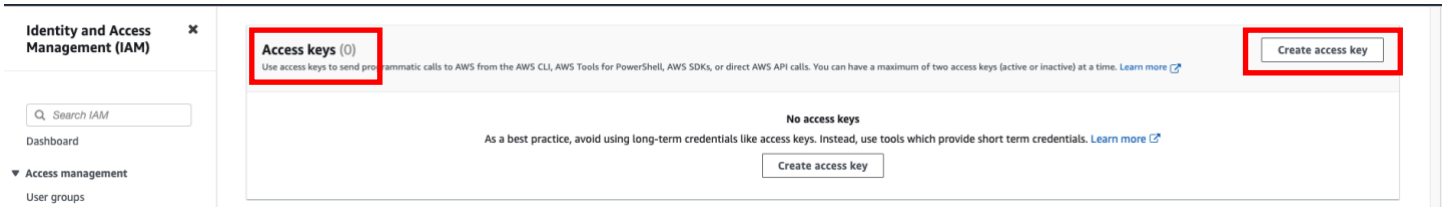
Once the User is created **Click on the User created**



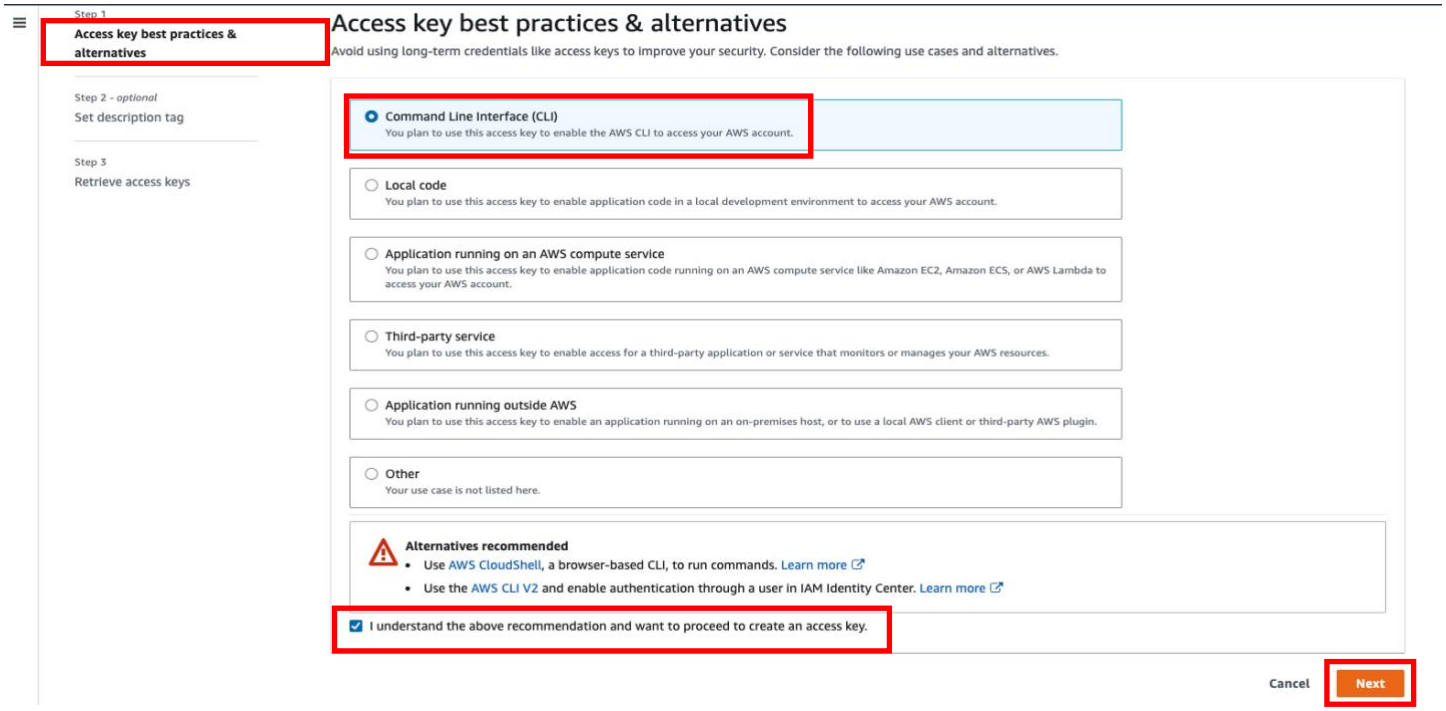
On the user page → Click on **Security Credential** Tab on user page



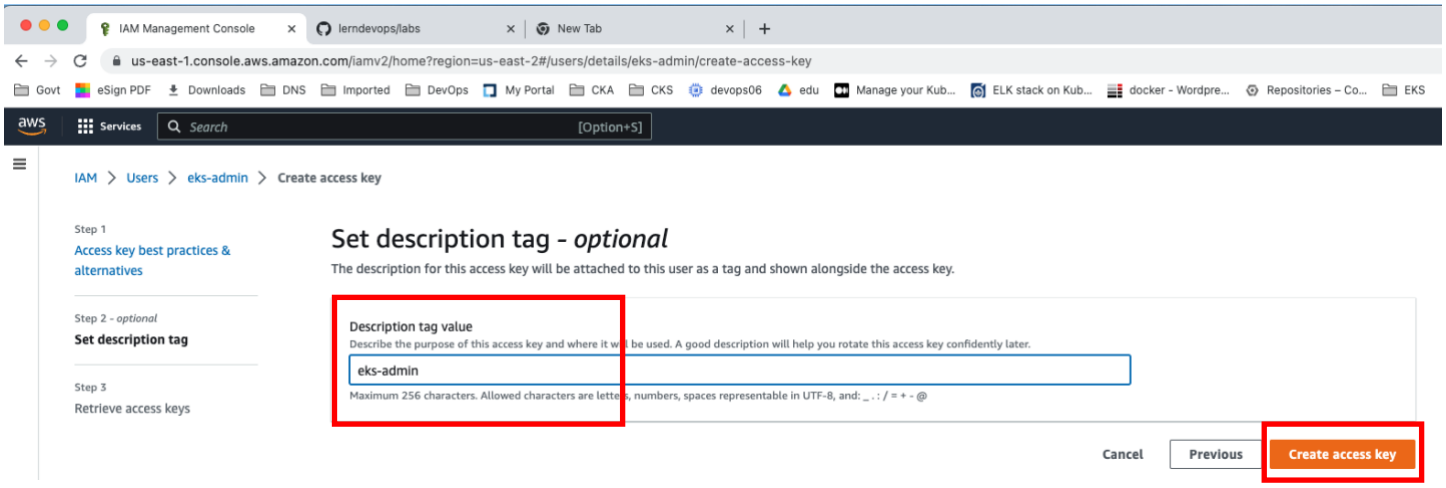
Scroll down to **"Access Keys"** Section & Click on **"Create access key"**



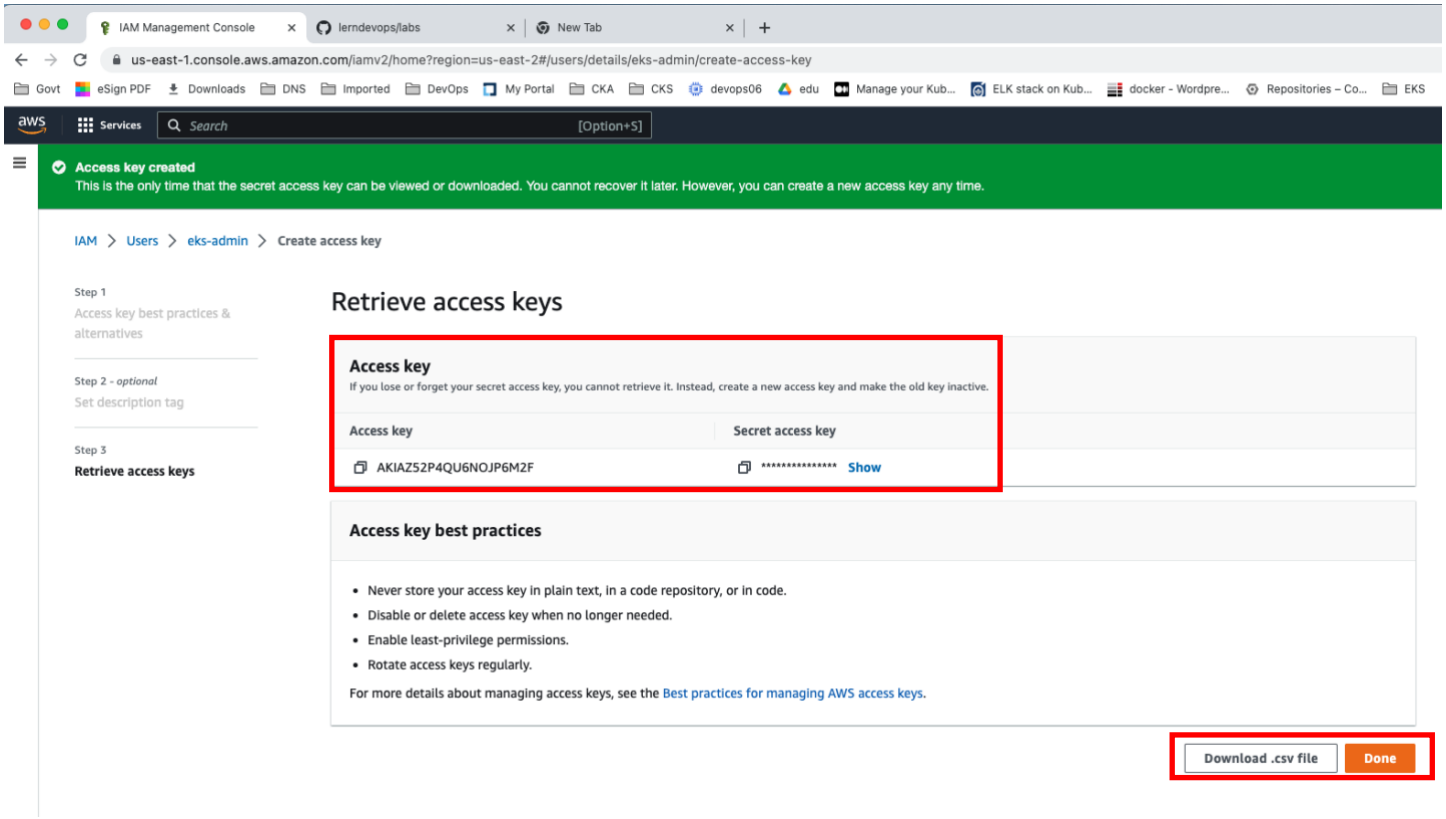
Access Key best Practices & alternatives → Choose **"Command Line Interface (CLI)"** option & Click **Next**



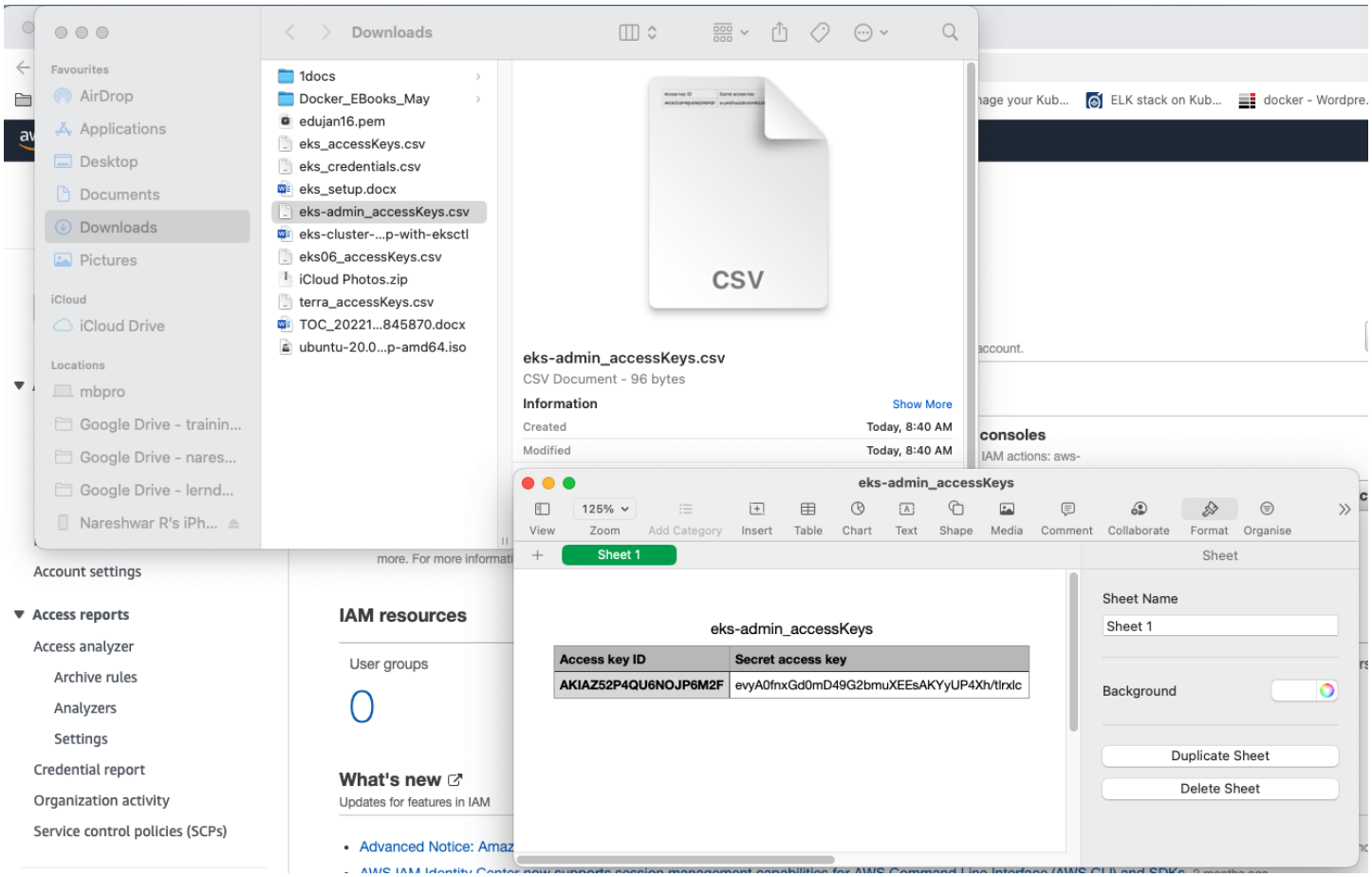
Enter a **"Description tag value"** & Click on **Create access key**



Save the **“Access Key” & “Secret Access Key”** generated on the page **OR**
Click on **“Download .csv file”**
then click on Done



Ensure the keys are downloaded or saved on your computer successfully



Now Let's Configure the keys created above for AWS CLI tool

Login to the server/vm where the aws cli tool is installed & run

aws configure

→ it will prompt for keys accordingly as below enter the values & validate

```
root@vm1:~#
root@vm1:~#
root@vm1:~# aws --version
aws-cli/2.9.20 Python/3.9.11 Linux/5.15.0-1028-aws exe/x86_64.ubuntu.22 prompt/off
root@vm1:~#
root@vm1:~# aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-1
Default output format [None]:
root@vm1:~#
root@vm1:~# aws sts get-caller-identity
{
    "UserId": "AIDAVVWXZIT7VDSXOHKCK",
    "Account": "[redacted]",
    "Arn": "arn:aws:iam::[redacted]:user/eks06"
}
root@vm1:~#
```

Let's Create a EKS Cluster with eksctl now

By default eksctl command line tool uses aws Cloud Formation template to create EKS cluster with default configurations as below

Ex: with command **"eksctl create cluster"**

A cluster will be created with default parameters:

- exciting auto-generated name, e.g., fabulous-mushroom-1527688624
- two m5.large worker nodes—this instance type suits most common use-cases, and is good value for money
- use the official AWS EKS AMI
- us-west-2 region
- a dedicated VPC (check your quotas)
- configures kubeconfig for kubectl tool to access the cluster

```
root@vm1:~#
root@vm1:~# eksctl create cluster
2023-02-02 04:36:29 [ℹ]  eksctl version 0.127.0
2023-02-02 04:36:29 [ℹ]  using region us-east-2
2023-02-02 04:36:29 [ℹ]  setting availability zones to [us-east-2b us-east-2c us-east-2a]
2023-02-02 04:36:29 [ℹ]  subnets for us-east-2b - public:192.168.0.0/19 private:192.168.96.0/19
2023-02-02 04:36:29 [ℹ]  subnets for us-east-2c - public:192.168.32.0/19 private:192.168.128.0/19
2023-02-02 04:36:29 [ℹ]  subnets for us-east-2a - public:192.168.64.0/19 private:192.168.160.0/19
2023-02-02 04:36:29 [ℹ]  nodegroup "ng-be340c9e" will use "" [AmazonLinux2/1.24]
2023-02-02 04:36:29 [ℹ]  using Kubernetes version 1.24
2023-02-02 04:36:29 [ℹ]  creating EKS cluster "attractive-party-1675312589" in "us-east-2" region with managed nodes
2023-02-02 04:36:29 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2 sequential tasks: { create cluster control plane "attractive-party-1675312589",
    2 sequential sub-tasks: {
        wait for control plane to become ready,
        create managed nodegroup "ng-be340c9e",
    }
}
2023-02-02 04:47:30 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-cluster"
2023-02-02 04:48:30 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-cluster"
2023-02-02 04:50:30 [ℹ]  building managed nodegroup stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 04:50:31 [ℹ]  deploying stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 04:50:31 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 04:51:01 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 04:51:40 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 04:52:51 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 04:54:09 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 04:54:09 [ℹ]  waiting for the control plane to become ready
2023-02-02 04:54:09 [✓]  saved kubeconfig as "/root/.kube/config"
2023-02-02 04:54:09 [ℹ]  no tasks
2023-02-02 04:54:09 [✓]  all EKS cluster resources for "attractive-party-1675312589" have been created
2023-02-02 04:54:09 [ℹ]  nodegroup "ng-be340c9e" has 2 node(s)
2023-02-02 04:54:09 [ℹ]  node "ip-192-168-34-170.us-east-2.compute.internal" is ready
2023-02-02 04:54:09 [ℹ]  node "ip-192-168-93-19.us-east-2.compute.internal" is ready
2023-02-02 04:54:09 [ℹ]  waiting for at least 2 node(s) to become ready in "ng-be340c9e"
2023-02-02 04:54:09 [ℹ]  nodegroup "ng-be340c9e" has 2 node(s)
2023-02-02 04:54:09 [ℹ]  node "ip-192-168-34-170.us-east-2.compute.internal" is ready
2023-02-02 04:54:09 [ℹ]  node "ip-192-168-93-19.us-east-2.compute.internal" is ready
2023-02-02 04:54:11 [ℹ]  kubectl command should work with "/root/.kube/config", try 'kubectl get nodes'
2023-02-02 04:54:11 [✓]  EKS cluster "attractive-party-1675312589" in "us-east-2" region is ready
root@vm1:~#

root@vm1:~# eksctl get clusters
NAME                    REGION          EKSCTL CREATED
attractive-party-1675312589     us-east-2       True
```

Validate the cluster

```
root@vm1:~#
root@vm1:~#
root@vm1:~# kubectl get nodes -o wide
NAME                                        STATUS   ROLES    AGE     VERSION               INTERNAL-IP      EXTERNAL-IP      OS-IMAGE         KERNEL-VERSION                  CONTAINER-RUNTIME
ip-192-168-34-170.us-east-2.compute.internal   Ready    <none>   2m58s   v1.24.9-eks-49d8fe8   192.168.34.170   18.118.148.199   Amazon Linux 2   5.4.228-131.415.amzn2.x86_64    containerd://1.6.6
ip-192-168-93-19.us-east-2.compute.internal    Ready    <none>   2m59s   v1.24.9-eks-49d8fe8   192.168.93.19    3.17.173.40      Amazon Linux 2   5.4.228-131.415.amzn2.x86_64    containerd://1.6.6
root@vm1:~#
root@vm1:~#
root@vm1:~# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE     IP               NODE                                           NOMINATED NODE   READINESS GATES
kube-system   aws-node-mshq4             1/1     Running   0          3m17s   192.168.93.19    ip-192-168-93-19.us-east-2.compute.internal    <none>           <none>
kube-system   aws-node-r9mmz             1/1     Running   0          3m16s   192.168.34.170   ip-192-168-34-170.us-east-2.compute.internal   <none>           <none>
kube-system   coredns-5c5677bc78-f8lgw   1/1     Running   0          11m     192.168.69.15    ip-192-168-93-19.us-east-2.compute.internal    <none>           <none>
kube-system   coredns-5c5677bc78-g4nhh   1/1     Running   0          11m     192.168.70.252   ip-192-168-93-19.us-east-2.compute.internal    <none>           <none>
kube-system   kube-proxy-2428j           1/1     Running   0          3m16s   192.168.34.170   ip-192-168-34-170.us-east-2.compute.internal   <none>           <none>
kube-system   kube-proxy-84mtn           1/1     Running   0          3m17s   192.168.93.19    ip-192-168-93-19.us-east-2.compute.internal    <none>           <none>
root@vm1:~#
root@vm1:~#
```

We can customize the all the configurations as required using cli options or using yaml as below

Ex: cli command

eksctl create cluster --name myekscluster --node-type t2.micro --nodegroup-name ng1

Like above we can put all parameters as required in cli but no so comfortable

instead we can write a simple yaml config file as below

vi cluster.yaml

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: basic-cluster
  region: eu-north-1
nodeGroups:
  - name: ng-1
    instanceType: m5.large
    desiredCapacity: 10
  - name: ng-2
    instanceType: m5.xlarge
    desiredCapacity: 2
```

then apply file as – **eksctl create cluster -f cluster.yaml**

we can also do a Dry Run to generate the cluster config yaml with default parameters used

```
root@vm1:~# eksctl create cluster --dry-run
apiVersion: eksctl.io/v1alpha5
availabilityZones:
- us-east-2b
- us-east-2c
- us-east-2a
cloudWatch:
  clusterLogging: {}
iam:
  vpcResourceControllerPolicy: true
  withOIDC: false
kind: ClusterConfig
kubernetesNetworkConfig:
  ipFamily: IPv4
managedNodeGroups:
- amiFamily: AmazonLinux2
  desiredCapacity: 2
  disableIMDSv1: false
  disablePodIMDS: false
  iam:
    withAddonPolicies:
      albIngress: false
      appMesh: false
      appMeshPreview: false
```

```
      autoScaler: false
      awsLoadBalancerController: false
      certManager: false
      cloudWatch: false
      ebs: false
      efs: false
      externalDNS: false
      fsx: false
      imageBuilder: false
      xRay: false
  instanceSelector: {}
  labels:
    alpha.eksctl.io/cluster-name: exciting-gopher-1675314821
    alpha.eksctl.io/nodegroup-name: ng-aa9510ed
  maxSize: 2
  minSize: 2
  name: ng-aa9510ed
  privateNetworking: false
  releaseVersion: ""
  securityGroups:
    withLocal: null
    withShared: null
  ssh:
    allow: false
    publicKeyPath: ""
  tags:
    alpha.eksctl.io/nodegroup-name: ng-aa9510ed
    alpha.eksctl.io/nodegroup-type: managed
  volumeIOPS: 3000
  volumeSize: 80
  volumeThroughput: 125
  volumeType: gp3
metadata:
  name: exciting-gopher-1675314821
  region: us-east-2
  version: "1.24"
privateCluster:
  enabled: false
  skipEndpointCreation: false
vpc:
  autoAllocateIPv6: false
  cidr: 192.168.0.0/16
  clusterEndpoints:
    privateAccess: false
    publicAccess: true
  manageSharedNodeSecurityGroupRules: true
  nat:
    gateway: Single
```

We can also delete the cluster as below

eksctl delete cluster --name <cluster-name>

```
root@vm1:~#
root@vm1:~# eksctl get clusters
NAME                    REGION          EKSCTL CREATED
attractive-party-1675312589     us-east-2       True
root@vm1:~#
root@vm1:~# eksctl delete cluster --name attractive-party-1675312589
2023-02-02 05:16:15 [ℹ]  deleting EKS cluster "attractive-party-1675312589"
2023-02-02 05:16:15 [ℹ]  will drain 0 unmanaged nodegroup(s) in cluster "attractive-party-1675312589"
2023-02-02 05:16:15 [ℹ]  starting parallel draining, max in-flight of 1
2023-02-02 05:16:15 [ℹ]  deleted 0 Fargate profile(s)
2023-02-02 05:16:16 [✓]  kubeconfig has been updated
2023-02-02 05:16:16 [ℹ]  cleaning up AWS load balancers created by Kubernetes objects of Kind Service or Ingress
2023-02-02 05:16:16 [ℹ]
2 sequential tasks: { delete nodegroup "ng-be340c9e", delete cluster control plane "attractive-party-1675312589" [async]
}
2023-02-02 05:16:16 [ℹ]  will delete stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:16:16 [ℹ]  waiting for stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e" to get deleted
2023-02-02 05:16:16 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:16:46 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:17:17 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:19:00 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:19:46 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:21:23 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:22:59 [ℹ]  waiting for CloudFormation stack "eksctl-attractive-party-1675312589-nodegroup-ng-be340c9e"
2023-02-02 05:22:59 [ℹ]  will delete stack "eksctl-attractive-party-1675312589-cluster"
2023-02-02 05:22:59 [✓]  all cluster resources were deleted
root@vm1:~#
```