DVS Technologies

Compiled and Scrutinized by
Mr. Shaan Shaik
(Senior DevOps Lead)

Words To The Students

Though we have taken utmost efforts to present you this book error free, but still it may contain some errors or mistakes. Students are encouraged to bring, if there are any mistakes or errors in this document to our notice. So that it may be rectified in the next edition of this document.

"Suppressing your doubts is Hindering your growth".

We urge you to work hard and make use of the facilities we are providing to you, because there is no substitute for hard work. We wish you all the best for your future.

"The grass isn't greener on the other side; the grass is greener where you water it."

You and your suggestions are valuable to us; Help us to serve you better. In case of any suggestions, grievance, or complaints, please feel free to write us your suggestions, grievance and feedback on the following

Dvs.training@gmail.com

1. Introduction to Kubernetes

What is Kubernetes:

Commonly referred to by its internal designation during the development as Google(k8s),Kubernetes is an open source container cluster manage.

When it was released in july 2015, Google donated it as "seed technology" to the newly formed Cloud native computing foundation established in a partenership with the linux foundation

Kubernetes primary goal is to provide a platform for automating deployment,scaling and operations of application containers across a cluster of hosts.

Design Overview:

kubernetes is built through the definition of a set of components (building blocks or "primitives") which,when used collectively,provide a method for the deployment, maintenance and scalability of container based application clusters.

These "primitives" are designed to be loosely coupled(i.e. where little to no knowledge of the other component definitions is needed to use) as well as easily extensible through an API. Both the internal components of kubernetes as well as the extensions and containers make use of this API.

Although kubernetes was designed and used internally at google to deploy and utilize "billion of containers", since it was open sourced under the Apache common license, it has since been adopted formally as a service available from each of the major cloud providers.

Components:

Building Block of Kubernetes:

Node(matser & workers)
Pods
Labels
Selectors
Controllers/Deployments
Services
Control Pane
API

Understanding Kubernetes Architecture :

Explain in detail about the architecture of K8s how it works.

Please check "HighLevel Architecture" figure for more information ...

From the above architecture we can conclude that we will have a "master controller" which will control all the other nodes. Here in k8s we called these nodes as "worker nodes", which means that each and every server (worker nodes) should have docker package installed. Along with this we are going to create our containers in seperate environment which we called it as "Pods". Hence you are not allowed to create as seperate entity. You should create the containers as part of the pods. If you want to increase the containers you can increase the containers as part of that pods. And finally all the worker nodes will be managed by our master controller.

Components of Kubernetes in detail over view :

worker nodes (Nodes):

You can think of these as "container clients". These are the individual hosts(physical or virtual) that docker is installed on and hosts the various containers within your managed cluster. So Finally "worker nodes" are nothing but our physical or virtula server where we have our container package installed and it acts like a slave to the master controller.

Pods:

A pod consists of one or more containers. Those containers are guaranteed (by the cluster controller) to be located on the same host machine in order to facilitate sharing of resources. Pods are assigned unique ips with in each cluster. These allow an application to use ports
without having to worry about conflicting port utilization.

Pods can contain definitions of disk volumes or share, and then provide access from those to
all the members(containers) with in the pod.

Finally, pod management is done through the API or delegated to a controller.

Labels:
Clients can attach "key-value pairs" to any object in the system (like pods or worker nodes).
These become the labels that identify them in the configuration and management of them.

Selectors:

Label selectors represent queries that are made against those labels. They resolve to the corresponding matching objects.
These two item are the primary way that grouping is done in kubernetes and determine which
components that a given operation applies to when indicated.

Controllers:

These are used in the management of your cluster. Controllers are the mechanism by which
your desired configuration state is enforced.Controllers manage a set of pods and depending on the desire configuration state, may enagage other controllers to handle replication and scaling (Replication ontroller)
of XX number of containers and pods across the clsuter. It is also responsible for replacing any container in a pod that fails (based on the desired state of the cluster). Other controllers that cna be engaged include a DaemonSet Controller (enforces a 1 to 1 ratio of pods to worker nodes) and job Controller(that runs pods to "completion", such as in batch jobs).

Each set of pods any controller manages, is determined by the label selectors that are part of its definition.

Services:

A pod consists of one or more containers. Those containers are guranteed (by the cluster controller) to be located on the same host machine in order to facilitate sharing of resources. This is so that pods can work together, like in a multi-tier application configuration. Each set of pods that define and implement a service (like Mysql or Apache) are defined by the label selector.

Kubernetes can then provide service discovery and handle routing with the static ip for each pod as well as load balancing (round robin based) connections to that service among the pods that match the label selector indicated. By default although a service is only exposed inside a cluster, it can also be exposed
outside a cluster as needed

On the Master Node following components will be installed :

API Server – It provides kubernetes API using Json / Yaml over http, states of
API objects are stored in etcd

Scheduler – It is a program on master node which performs the scheduling tasks like launching containers in worker nodes based on resource availability

Controller Manager – Main Job of Controller manager is to monitor replication controllers and create pods to maintain desired state.

etcd – It is a Key value pair data base. It stores configuration data of cluster and cluster state.

Kubectl utility – It is a command line utility which connects to API Server on port 6443. It is used by administrators to create pods, services etc.

On Worker Nodes following components will be installed ::

Kubelet – It is an agent which runs on every worker node, it connects to docker and takes care of creating, starting, deleting containers.

Kube-Proxy – It routes the traffic to appropriate containers based on ip address and port number of the incoming request. In other words we can say it is used for port translation.

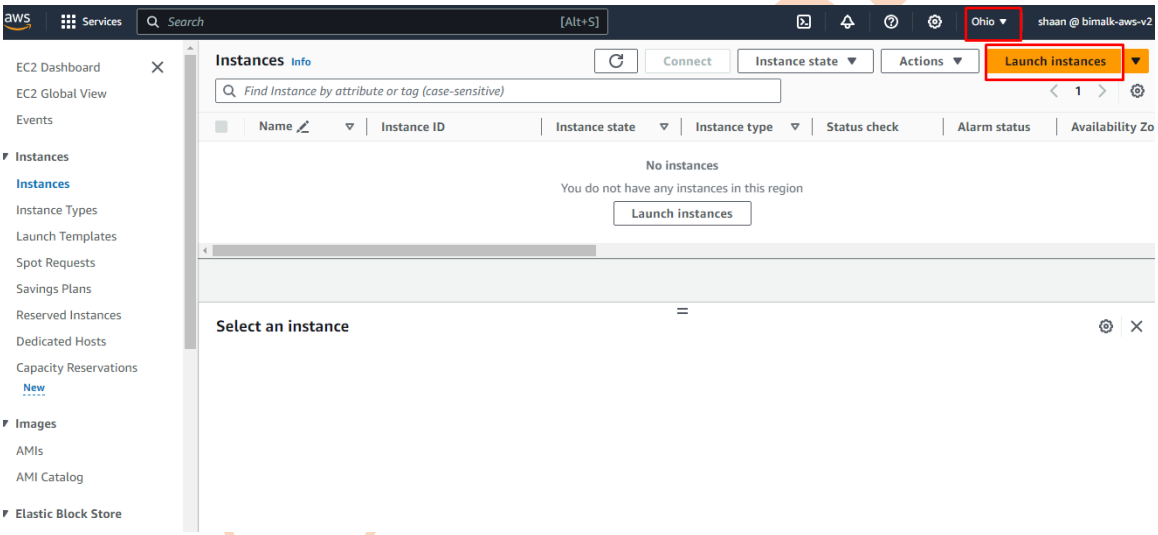
Pod – Pod can be defined as a multi-tier or group of containers that are deployed on a single worker node or docker host.

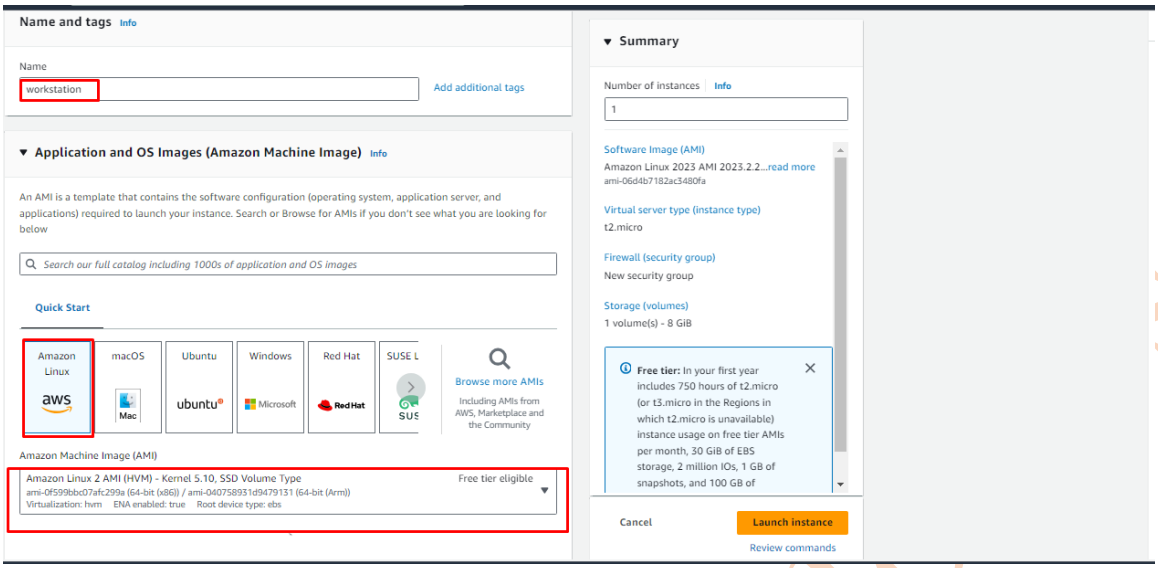
2 Preparing workstation

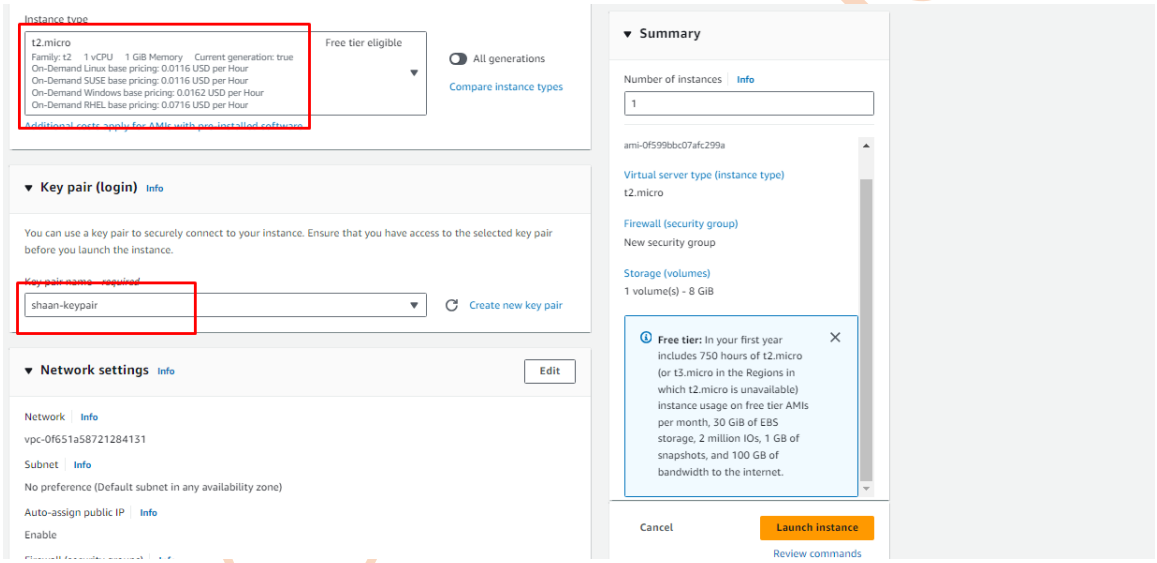
Read the below before you start (mandatory):

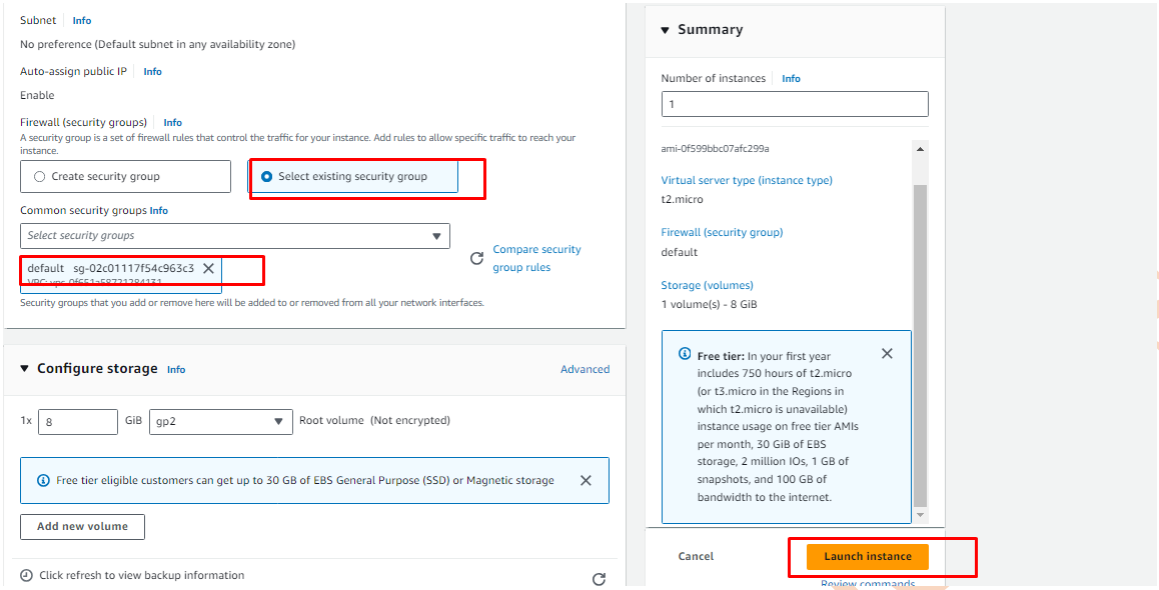
1 Since it's a workstation we can go ahead and create workstation with t2.micro
2 Make sure that you are configuring worker node groups with t2.medium configuration
3 Note that all the servers are getting created in the same network(vpc)
for example in future jenkins (t2.medium) etc ..
4 Make sure you are creating the infrastructure in OHIO (us-east-2)

1. Creating workstation:

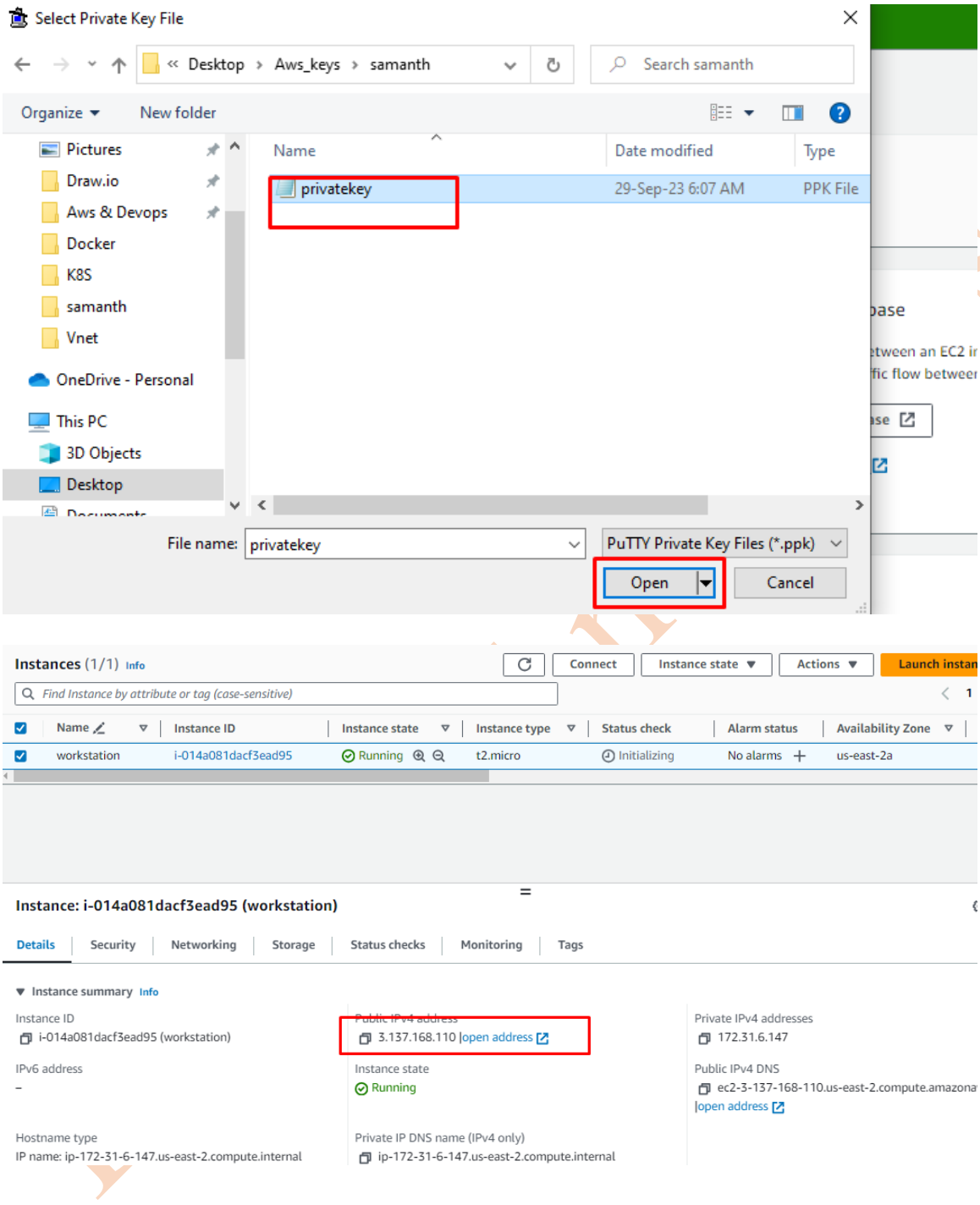


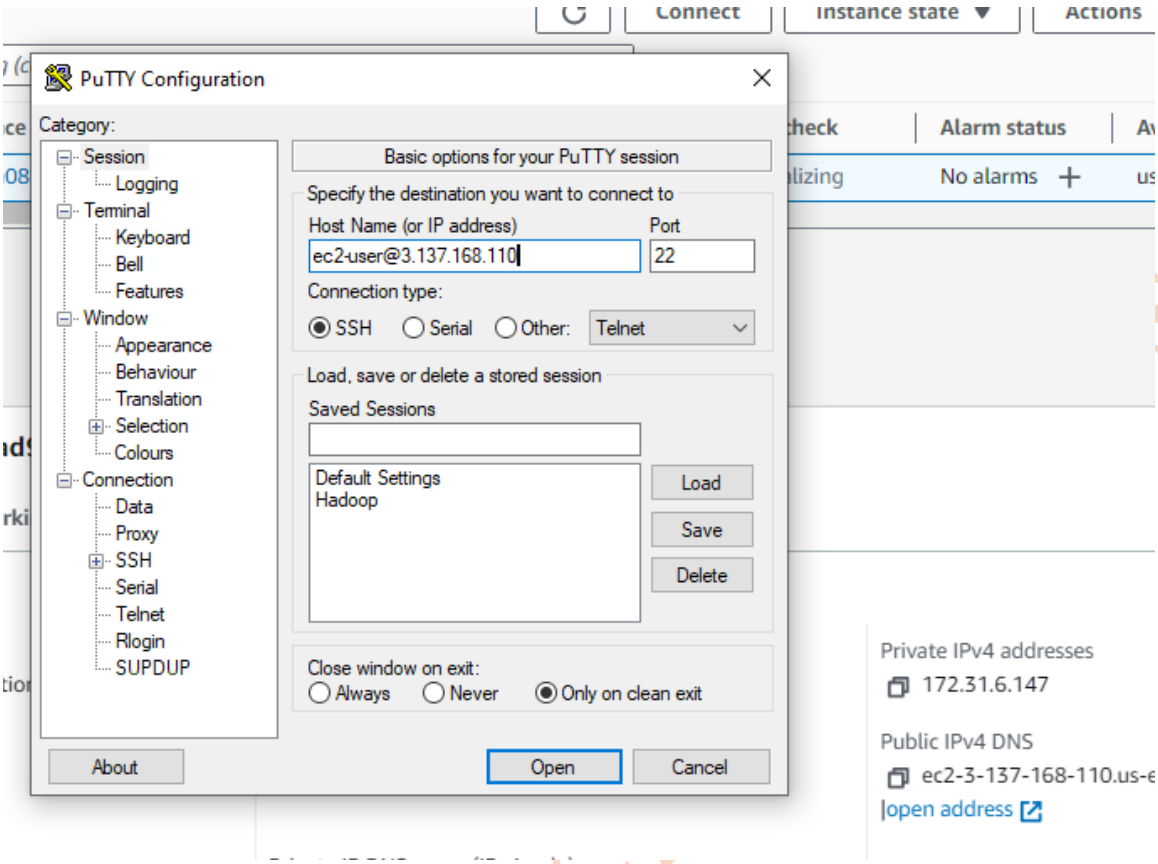

Name and tags Info
Name
workstation
Add additional tags
Application and OS Images (Amazon Machine Image) Info
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below
Search our full catalog including 1000s of application and OS images
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type
Free tier eligible
Summary
Number of instances Info
1
Software Image (AMI)
Amazon Linux 2023 AMI 2023.2.2...read more
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of
Cancel
Launch instance
Review commands

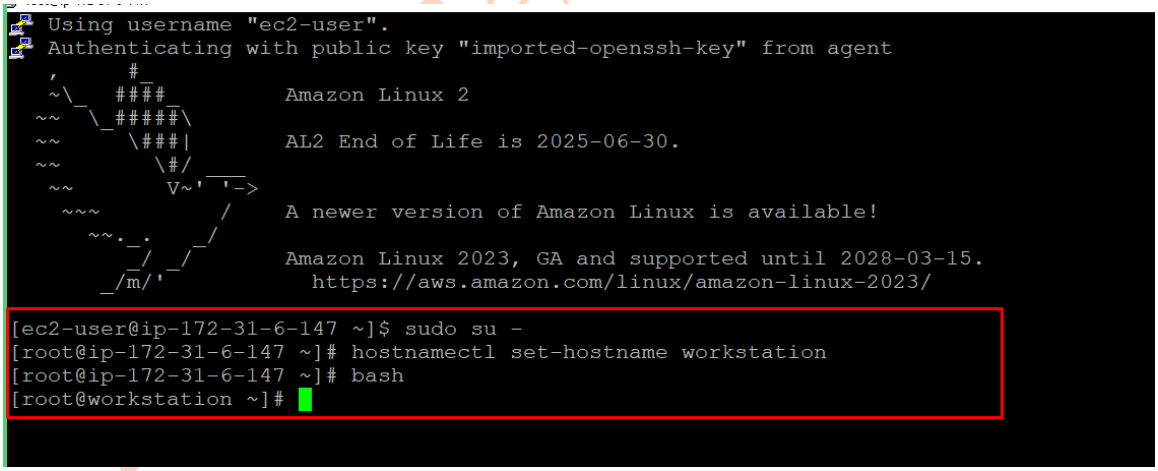

Instance type
t2.micro
Free tier eligible
All generations
Compare instance types
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
shaan-keypair
Create new key pair
Network settings Info
Edit
Network Info
vpc-0f651a58721284131
Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info
Enable
Summary
Number of instances Info
1
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Launch instance
Review commands



2. Login to the server and do the below


Select Private Key File
Desktop › Aws_keys › samanth
Search samanth
Organize
New folder
Pictures
Draw.io
Aws & Devops
Docker
K8S
samanth
Vnet
OneDrive - Personal
This PC
3D Objects
Desktop
Name
Date modified
Type
privatekey
29-Sep-23 6:07 AM
PPK File
File name: privatekey
PuTTY Private Key Files (*.ppk)
Open
Cancel
Instances (1/1) Info
Connect
Instance state
Actions
Launch instan
Find Instance by attribute or tag (case-sensitive)
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
workstation
i-014a081dacf3ead95
Running
t2.micro
Initializing
No alarms
us-east-2a
Instance: i-014a081dacf3ead95 (workstation)
Details
Security
Networking
Storage
Status checks
Monitoring
Tags
Instance summary Info
Instance ID
i-014a081dacf3ead95 (workstation)
3.137.168.110 |open address
Private IPv4 addresses
172.31.6.147
IPv6 address
Instance state
Running
Public IPv4 DNS
ec2-3-137-168-110.us-east-2.compute.amazona
|open address
Hostname type
IP name: ip-172-31-6-147.us-east-2.compute.internal
Private IP DNS name (IPv4 only)
ip-172-31-6-147.us-east-2.compute.internal


Connect
Instance state
Actions
PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
SSH
Serial
Telnet
Rlogin
SUPDUP
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
ec2-user@3.137.168.110
22
Connection type:
SSH
Serial
Other:
Telnet
Load, save or delete a stored session
Saved Sessions
Default Settings
Hadoop
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel
Alarm status
No alarms
Private IPv4 addresses
172.31.6.147
Public IPv4 DNS
ec2-3-137-168-110.us-e
open address


Using username "ec2-user".
Authenticating with public key "imported-openssh-key" from agent
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-6-147 ~]$ sudo su -
[root@ip-172-31-6-147 ~]# hostnamectl set-hostname workstation
[root@ip-172-31-6-147 ~]# bash
[root@workstation ~]#

3. **Installing all the required softwares to proceed with installation**

1. **Aws cli installation:**

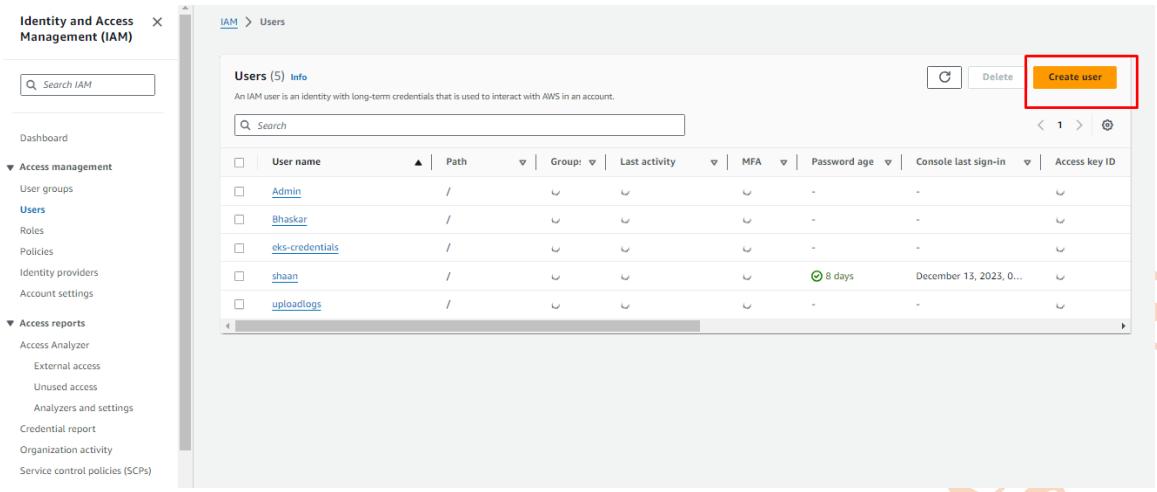
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

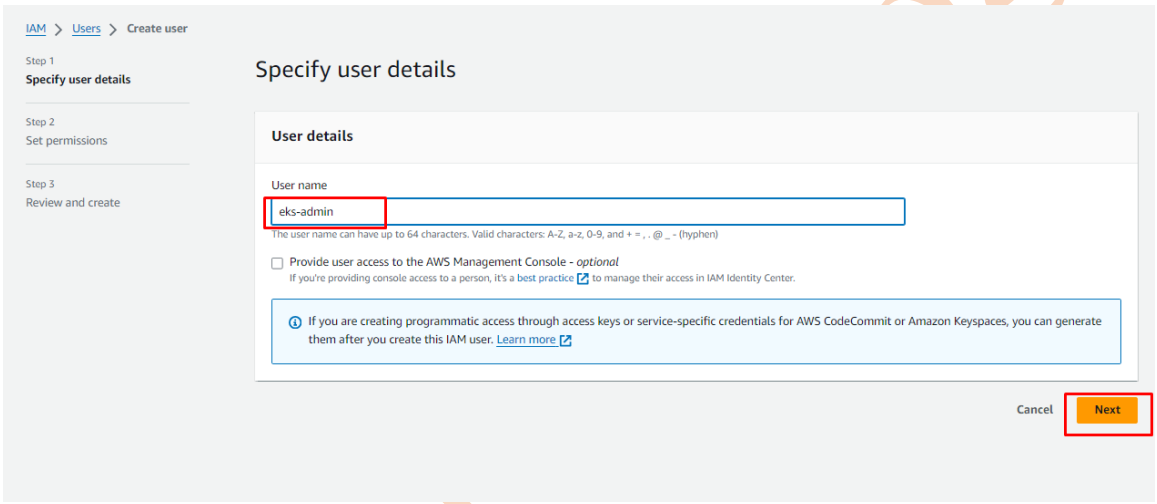
2. **Install eksctl latest binary**
https://eksctl.io/installation/
# for ARM systems, set ARCH to: `arm64`, `armv6` or `armv7`
ARCH=amd64
PLATFORM=$(uname -s)_$ARCH
curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$PLATFORM.tar.gz"
(Optional) Verify checksum
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt" | grep $PLATFORM | sha256sum --check
tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && /usr/bin/rm -f eksctl_$PLATFORM.tar.gz
sudo mv /tmp/eksctl /usr/local/bin
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc

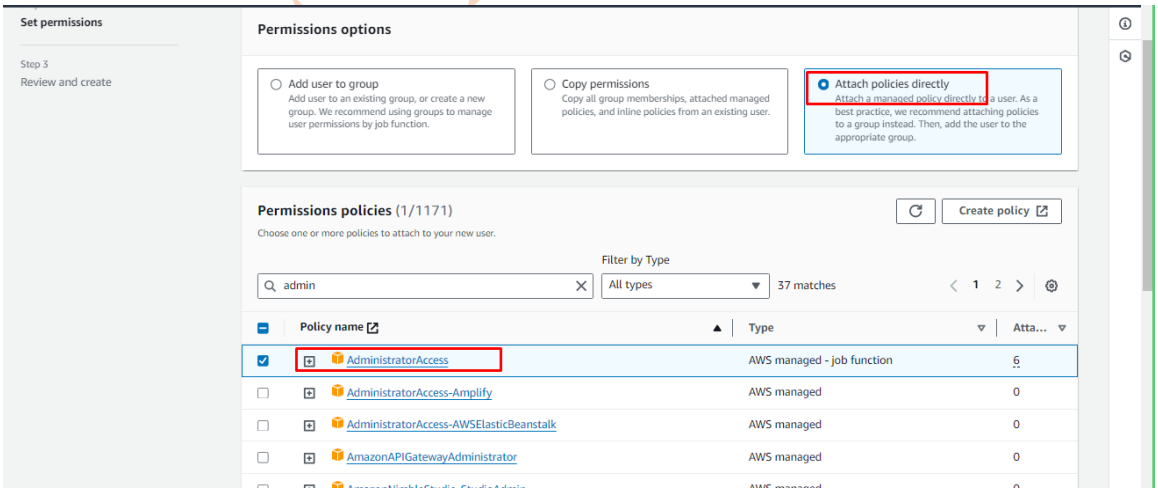
3. **Install latest of needed version of kubectl command**
https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl
chmod +x ./kubectl
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
kubectl version --client

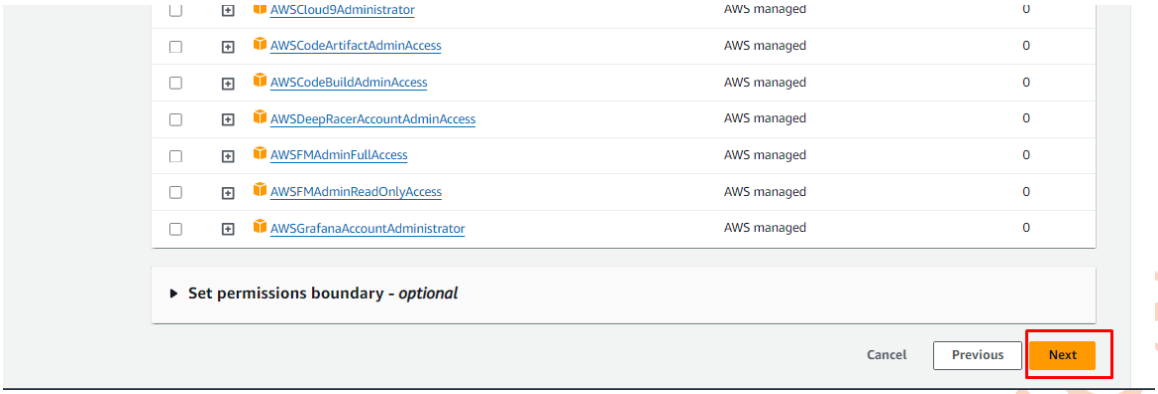
```
[root@workstation ~]# aws --version
aws-cli/2.15.0 Python/3.11.6 Linux/5.10.201-191.748.amzn2.x86_64 exe/x86_64.amzn.2 prompt/off
[root@workstation ~]# eksctl version
0.165.0
[root@workstation ~]# kubectl version --client
Client Version: v1.28.3-eks-e71965b
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
[root@workstation ~]#
```

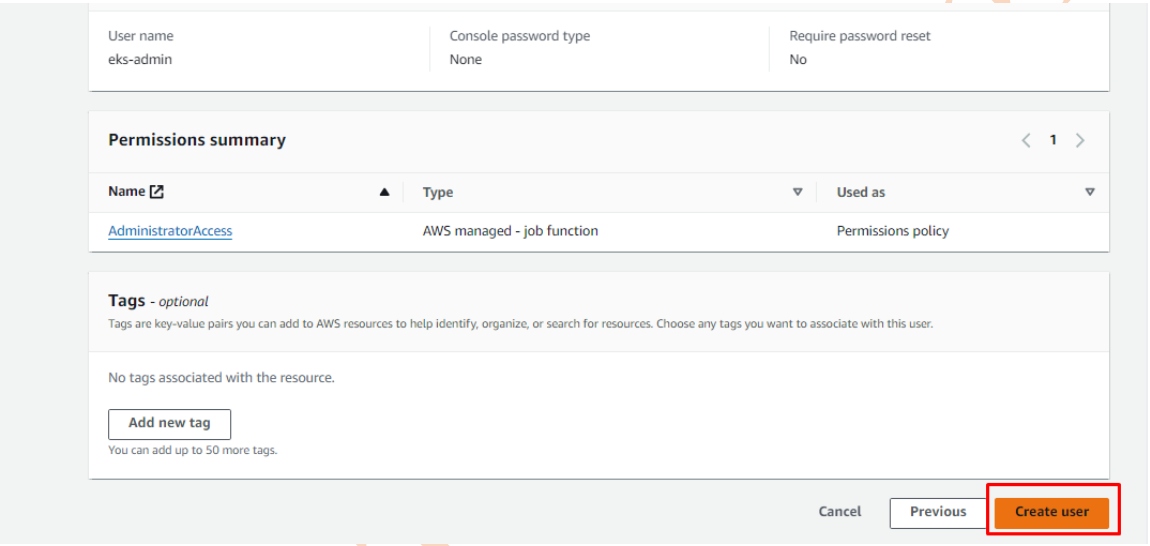
4. **Configuring aws cli**


Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access Analyzer
External access
Unused access
Analyzers and settings
Credential report
Organization activity
Service control policies (SCPs)
IAM > Users
Users (5) Info
An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.
Delete
Create user
Search
User name
Path
Group:
Last activity
MFA
Password age
Console last sign-in
Access key ID
Admin
Bhaskar
eks-credentials
shaan
8 days
December 13, 2023, 0...
uploadlogs

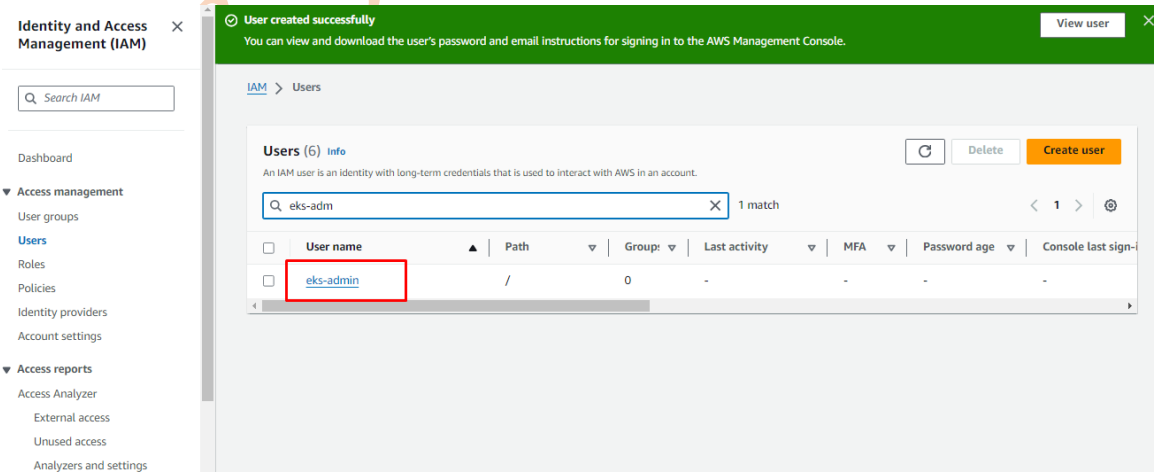

IAM > Users > Create user
Step 1
Specify user details
Step 2
Set permissions
Step 3
Review and create
Specify user details
User details
User name
eks-admin
The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)
Provide user access to the AWS Management Console - optional
If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.
If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more
Cancel
Next

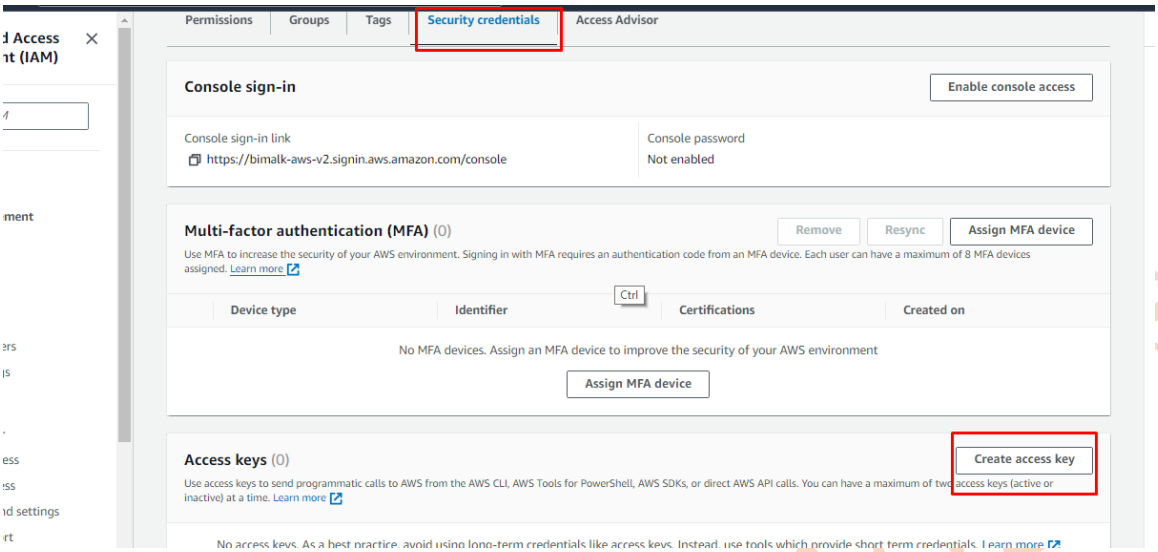

Set permissions
Step 3
Review and create
Permissions options
Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.
Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.
Permissions policies (1/1171)
Choose one or more policies to attach to your new user.
Create policy
Filter by Type
admin
All types
37 matches
Policy name
Type
Atta...
AdministratorAccess
AWS managed - job function
6
AdministratorAccess-Amplify
AWS managed
0
AdministratorAccess-AWSElasticBeanstalk
AWS managed
0
AmazonAPIGatewayAdministrator
AWS managed
0

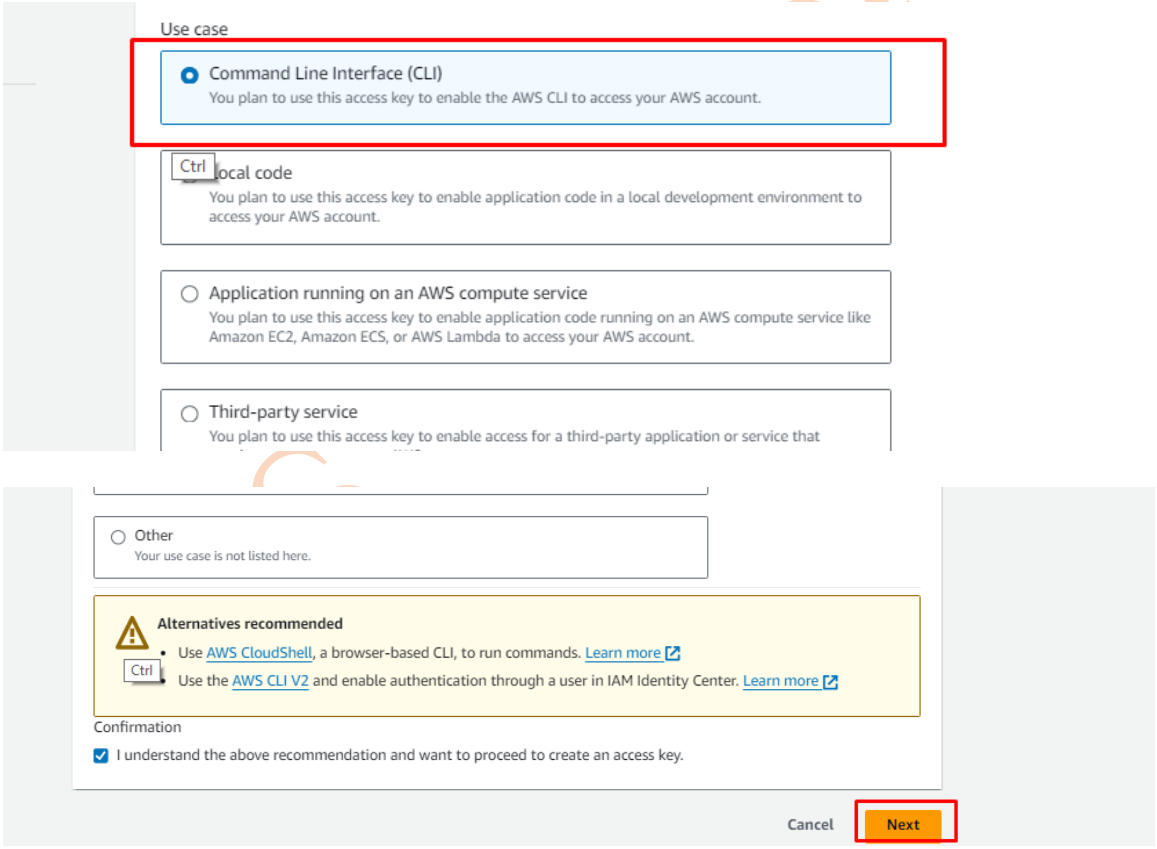


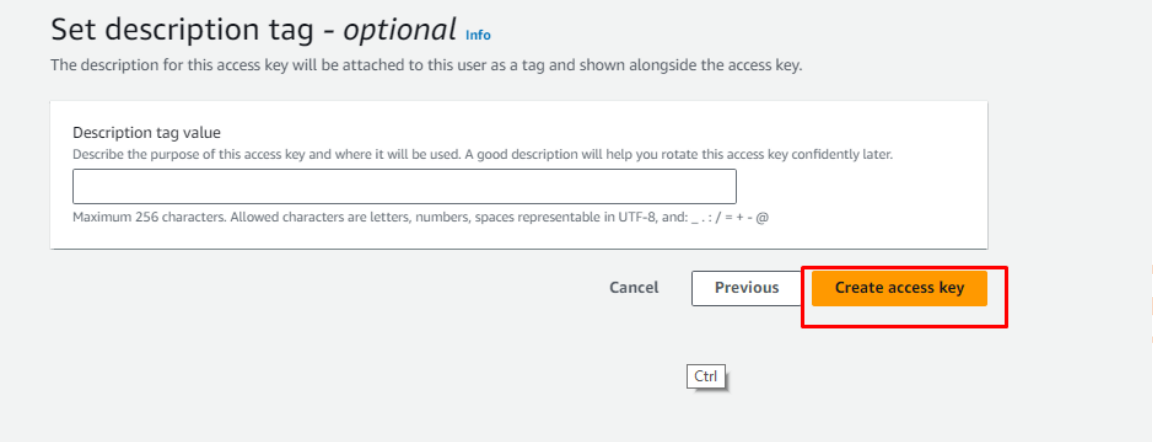


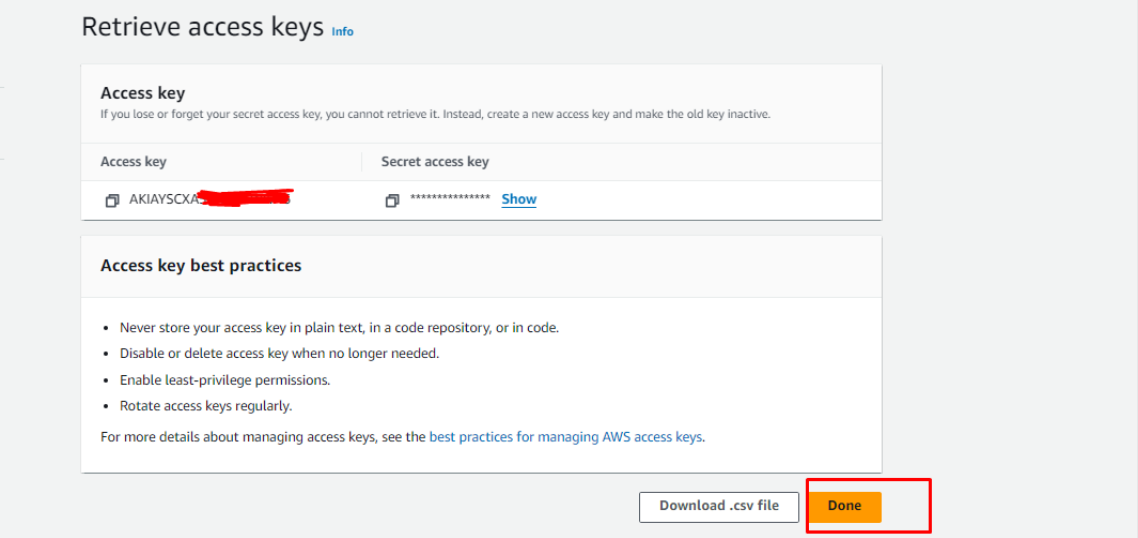
Now let's generate the credentials (access key,token)



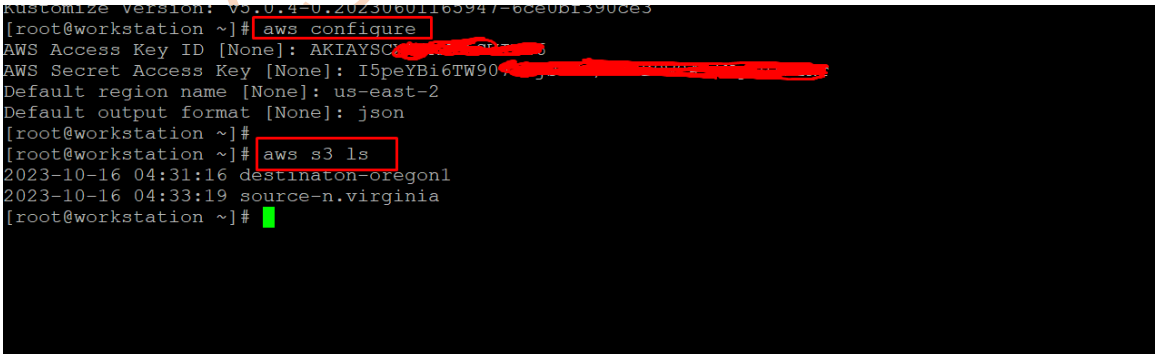

Permissions
Groups
Tags
Security credentials
Access Advisor
Console sign-in
Enable console access
Console sign-in link
https://bimalk-aws-v2.signin.aws.amazon.com/console
Console password
Not enabled
Multi-factor authentication (MFA) (0)
Remove
Resync
Assign MFA device
Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. Learn more
Device type
Identifier
Certifications
Created on
No MFA devices. Assign an MFA device to improve the security of your AWS environment
Assign MFA device
Access keys (0)
Create access key
Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more


Use case
Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.
Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.
Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
Third-party service
You plan to use this access key to enable access for a third-party application or service that
Other
Your use case is not listed here.
Alternatives recommended
Use AWS CloudShell, a browser-based CLI, to run commands. Learn more
Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. Learn more
Confirmation
I understand the above recommendation and want to proceed to create an access key.
Cancel
Next





Now you can go to workstation & do the aws configure as mentioned below



Finally we can see that we have successfully prepare our workstation with all necessary software's required.

3 Creating cluster with eksctl

Creating cluster:

eksctl create cluster --name dvseks cluster --region us-east-2 --instance-types t2.medium --managed --vpc-cidr 192.168.100.0/24 --node-private-networking --version 1.27 --without-nodegroup

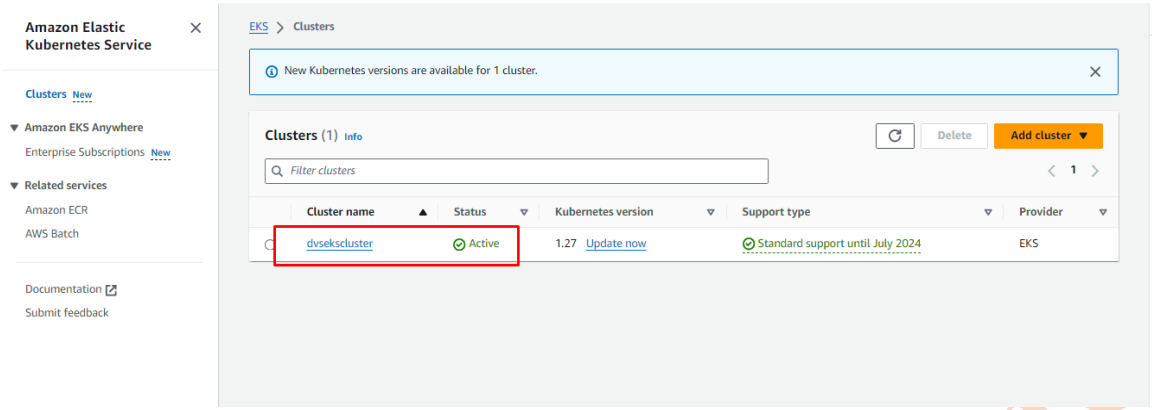
```
root@workstation:~
[root@workstation ~]# eksctl create cluster --name dvsekscluster --region us-east-2 --instance-types t2
.medium --managed --vpc-cidr 192.168.100.0/24 --node-private-networking --version 1.27 --without-nodegr
oup
2023-12-23 07:31:50 [ℹ]  eksctl version 0.166.0
2023-12-23 07:31:50 [ℹ]  using region us-east-2
2023-12-23 07:31:50 [ℹ]  setting availability zones to [us-east-2a us-east-2b us-east-2c]
2023-12-23 07:31:50 [ℹ]  subnets for us-east-2a - public:192.168.100.0/27 private:192.168.100.96/27
2023-12-23 07:31:50 [ℹ]  subnets for us-east-2b - public:192.168.100.32/27 private:192.168.100.128/27
2023-12-23 07:31:50 [ℹ]  subnets for us-east-2c - public:192.168.100.64/27 private:192.168.100.160/27
2023-12-23 07:31:50 [ℹ]  using Kubernetes version 1.27
2023-12-23 07:31:50 [ℹ]  creating EKS cluster "dvsekscluster" in "us-east-2" region with
2023-12-23 07:31:50 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils
 describe-stacks --region=us-east-2 --cluster=dvsekscluster'
2023-12-23 07:31:50 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, private
Access=false} for cluster "dvsekscluster" in "us-east-2"
2023-12-23 07:31:50 [ℹ]  CloudWatch logging will not be enabled for cluster "dvsekscluster" in "us-east
-2"
2023-12-23 07:31:50 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SP
ECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-2 --cluster=dvsekscluster'
2023-12-23 07:31:50 [ℹ]
2 sequential tasks: { create cluster control plane "dvsekscluster", wait for control plane to become re
ady
```

Note: Cluster creation takes 10 mins, please be patient enough and wait for it to come up.

```
2023-12-23 07:36:51 [ℹ]  waiting for CloudFormation stack "eksctl-dvsekscluster-cluster"
2023-12-23 07:37:51 [ℹ]  waiting for CloudFormation stack "eksctl-dvsekscluster-cluster"
2023-12-23 07:38:51 [ℹ]  waiting for CloudFormation stack "eksctl-dvsekscluster-cluster"
2023-12-23 07:39:51 [ℹ]  waiting for CloudFormation stack "eksctl-dvsekscluster-cluster"
2023-12-23 07:40:51 [ℹ]  waiting for CloudFormation stack "eksctl-dvsekscluster-cluster"


2023-12-23 07:42:52 [ℹ]  waiting for the control plane to become ready
2023-12-23 07:42:53 [✔]  saved kubeconfig as "/root/.kube/config"
2023-12-23 07:42:53 [ℹ]  no tasks
2023-12-23 07:42:53 [✔]  all EKS cluster resources for "dvsekscluster" have been created
2023-12-23 07:42:54 [ℹ]  kubectl command should work with "/root/.kube/config", try 'kubectl get nodes'
2023-12-23 07:42:54 [✔]  EKS cluster "dvsekscluster" in "us-east-2" region is ready
```

```
[root@workstation ~]# aws eks list-clusters --region us-east-2
{
    "clusters": [
        "dvsekscluster"
    ]
}
[root@workstation ~]#
```

Now we can see that our cluster is up & running.



Creating custom network:

Update the kubeconfig file with below command to login to the cluster

aws eks update-kubeconfig --region us-east-2 --name "dvseksclusterr"

```
[root@workstation ~]# aws eks update-kubeconfig --region us-east-2 --name "dvseksclusterr"
Added new context arn:aws:eks:us-east-2:439292780570:cluster/dvseksclusterr to /root/.kube/config
[root@workstation ~]#
```

kubectl delete daemonset -n kube-system aws-node
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/tigera-operator.yaml

```
[root@workstation ~]# kubectl delete daemonset -n kube-system aws-node
daemonset.apps "aws-node" deleted
[root@workstation ~]# kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/tigera-operator.yaml
namespace/tigera-operator created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
```

kubectl create -f - <<EOF
kind: Installation
apiVersion: operator.tigera.io/v1
metadata:
name: default
spec:
kubernetesProvider: EKS
cni:
type: Calico
calicoNetwork:
bgp: Disabled

EOF

```
[root@workstation ~]# kubectl create -f - <<EOF
> kind: Installation
> apiVersion: operator.tigera.io/v1
> metadata:
>   name: default
> spec:
>   kubernetesProvider: EKS
>   cni:
>     type: Calico
>   calicoNetwork:
>     bgp: Disabled
> EOF
installation.operator.tigera.io/default created
[root@workstation ~]#
```

Creating nodegroup:

eksctl create nodegroup --cluster "dvseks cluster" --name nodegroup1 --nodes 2 --node-type t2.medium --node-private-networking --managed --max-pods-per-node 200

```
[root@workstation ~]#
[root@workstation ~]# eksctl create nodegroup --cluster "dvseks cluster" --name nodegroup1 --nodes 2 --n
ode-type t2.medium --node-private-networking --managed --max-pods-per-node 200
2023-12-23 07:51:24 [ℹ]  will use version 1.27 for new nodegroup(s) based on control plane version
2023-12-23 07:51:25 [ℹ]  nodegroup "nodegroup1" will use "" [AmazonLinux2/1.27]
2023-12-23 07:51:25 [ℹ]  1 nodegroup (nodegroup1) was included (based on the include/exclude rules)
2023-12-23 07:51:25 [ℹ]  will create a CloudFormation stack for each of 1 managed nodegroups in cluster
 "dvseks cluster"
2023-12-23 07:51:25 [ℹ]
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "nodegrou
p1" } }
}
2023-12-23 07:51:25 [ℹ]  checking cluster stack for missing resources
2023-12-23 07:51:25 [ℹ]  cluster stack has all required resources
2023-12-23 07:51:25 [ℹ]  building managed nodegroup stack "eksctl-dvseks cluster-nodegroup-nodegroup1"
2023-12-23 07:51:25 [ℹ]  deploying stack "eksctl-dvseks cluster-nodegroup-nodegroup1"
2023-12-23 07:51:25 [ℹ]  waiting for CloudFormation stack "eksctl-dvseks cluster-nodegroup-nodegroup1"
```

Here we are going to create the worker nodes which we call it as node group in eks cluster with two nodes of hardware type t2.medium.

Final verification:

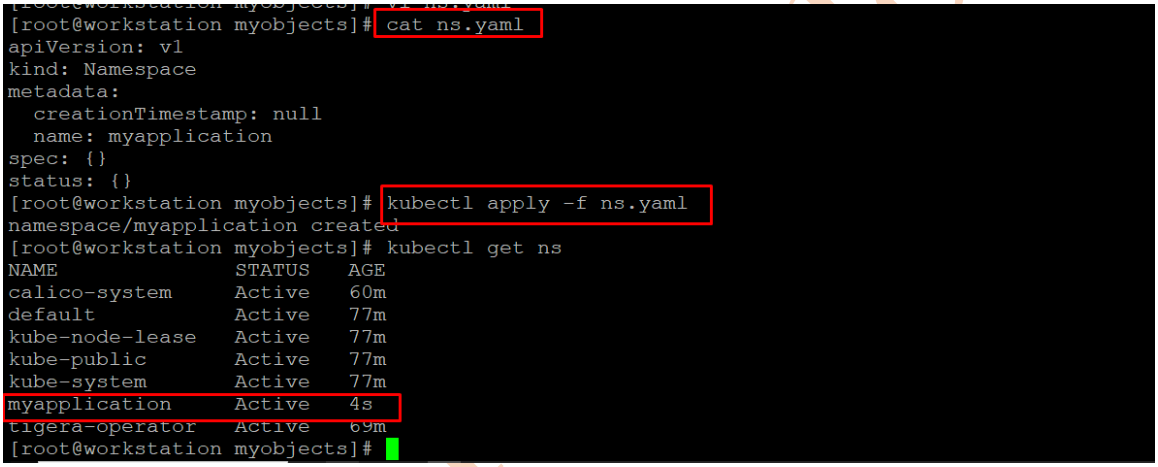
```
[root@workstation ~]# kubectl get nodes
NAME                                           STATUS   ROLES    AGE   VERSION
ip-192-168-100-110.us-east-2.compute.internal   Ready    <none>   50m   v1.27.7-eks-e71965b
ip-192-168-100-173.us-east-2.compute.internal   Ready    <none>   50m   v1.27.7-eks-e71965b
[root@workstation ~]# kubectl get ns
NAME              STATUS   AGE
calico-system     Active   50m
default           Active   67m
kube-node-lease   Active   67m
kube-public       Active   67m
kube-system       Active   67m
tigera-operator   Active   58m
[root@workstation ~]#
```

4. Working With Cluster

Namespace Creation:

vi ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: null
  name: myapplication
spec: {}
status: {}
```



Pod Creation:

vi pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mypod
  name: mypod
  namespace: myapplication
spec:
  containers:
  - image: nginx
```

```
    name: mypod
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

```
[root@workstation myobjects]# cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mypod
  name: mypod
  namespace: myapplication
spec:
  containers:
  - image: nginx
    name: mypod
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
[root@workstation myobjects]# kubectl apply -f pod.yaml -n myapplication
pod/mypod created
[root@workstation myobjects]# kubectl get pods -n myapplication
NAME    READY   STATUS    RESTARTS   AGE
mypod   1/1     Running   0          10s
[root@workstation myobjects]#
```

Deployment creation:

vi deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
```

```
    creationTimestamp: null
    labels:
      app: mydeployment
  spec:
    containers:
    - image: nginx
      name: nginx
      resources: {}
status: {}
```

```
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

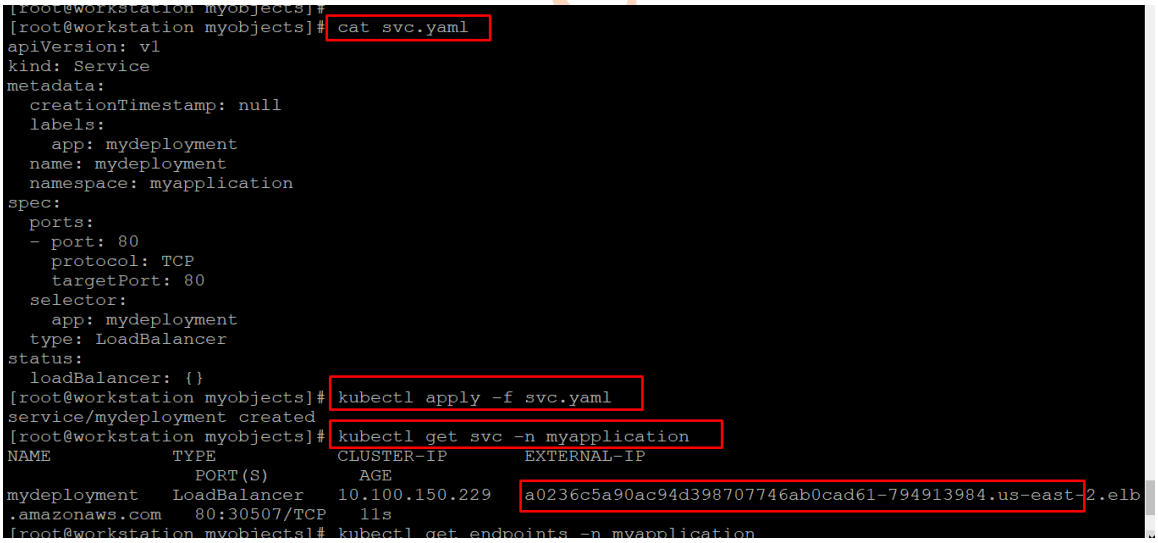
```
[root@workstation myobjects]# kubectl apply -f deploy.yaml
deployment.apps/mydeployment created
[root@workstation myobjects]# kubectl get deploy -n myapplication
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
mydeployment   1/1     1            1           10s
[root@workstation myobjects]#
```

```
root@workstation:~/myobjects
[root@workstation myobjects]# kubectl get pods -n myapplication
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-8496ddbc68-2zxxs   1/1     Running   0          66s
mypod                           1/1     Running   0          5m34s
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-8496ddbc68-2zxxs   1/1     Running   0          75s
[root@workstation myobjects]#
```

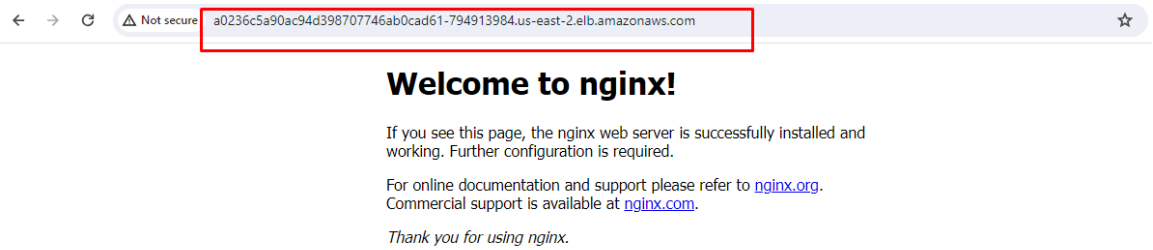
Service creation:

vi svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: mydeployment
  type: LoadBalancer
status:
  loadBalancer: {}
```



We can able to see that our deployment got exposed as service of type loadbalancer. We got out LB url which we can use it for accessing our application from browser.



If you want to see the pods behind this service then you can execute the below command

```
.us-east-2.compute.internal   <none>           <none>
[root@workstation myobjects]# kubectl get endpoints -n myapplication
NAME           ENDPOINTS           AGE
mydeployment   172.16.249.67:80    3m46s
[root@workstation myobjects]# kubectl get pods -n myapplication -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP              NODE
                          NOMINATED NODE   READINESS GATES
mydeployment-8496ddbc68-2zxxs   1/1     Running   0          10m   172.16.249.67   ip-192-168-100-173.u
s-east-2.compute.internal   <none>           <none>
mypod                           1/1     Running   0          14m   172.16.249.66   ip-192-168-100-173.u
s-east-2.compute.internal   <none>           <none>
[root@workstation myobjects]#
```

Increasing Replica Count:

If you want to run more than one pod then you can increase the replica count in the below yaml

```
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

Before changes:

```
[root@workstation myobjects]# kubectl get po -n myapplication -l app=mydeployment
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-8496ddbc68-kbn65   1/1     Running   0          16h
```

We can see before we apply the yaml no.of pods for this deployment is 2.

Post changes:

```
[root@workstation myobjects]# kubectl apply -f deploy.yaml
deployment.apps/mydeployment configured
[root@workstation myobjects]# kubectl get po -n myapplication -l app=mydeployment
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-8496ddbc68-chxdc   1/1     Running   0          7s
mydeployment-8496ddbc68-kbn65   1/1     Running   0          16h
[root@workstation myobjects]#
```

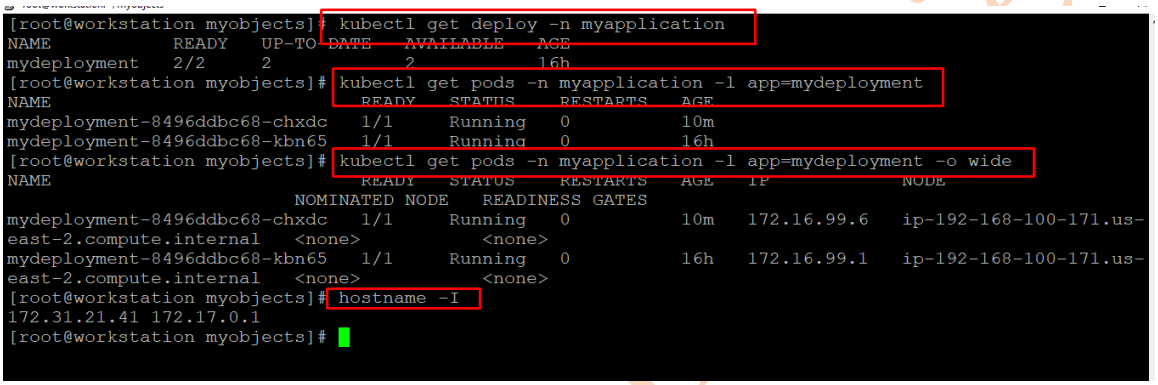
Overall Commands:

```
root@workstation:~/myobjects
[root@workstation myobjects]# kubectl get ns
NAME              STATUS   AGE
calico-system     Active   17h
default           Active   18h
kube-node-lease   Active   18h
kube-public       Active   18h
kube-system       Active   18h
myapplication     Active   16h
tigera-operator   Active   17h
[root@workstation myobjects]# kubectl get deploy -n myapplication
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
mydeployment   2/2     2            2           16h
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-8496ddbc68-chxdc   1/1     Running   0          2m9s
mydeployment-8496ddbc68-kbn65   1/1     Running   0          16h
[root@workstation myobjects]# kubectl get svc -n myapplication
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP
                 PORT(S)         AGE
mydeployment   LoadBalancer   10.100.150.229   a0236c5a90ac94d398707746ab0cad61-794913984.us-east-2.elb
.amazonaws.com   80:30507/TCP    16h
[root@workstation myobjects]# kubectl get endpoints -n myapplication
NAME           ENDPOINTS                         AGE
mydeployment   172.16.99.1:80,172.16.99.6:80   16h
[root@workstation myobjects]#
```

5. Working with Service

Service acts like as a loadbalancer on top of all the pods of an expose deployment. We have three types of services in kubernetes. They are clusterip,nodeport & loadbalancer

1. **ClusterIP: (This is a default service type, which helps communication between microservices internal to the cluster)**



Above screenshot give us an overview of no of pods running for a deployment and their ipaddresses. If you observer properly our host ipaddress(workstation) is not in the range of the cluster ipaddress hence we need an intermediate container which help us to test the application. Hence we are going to use busbox container.

vi busybox.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox1
  labels:
    app: busybox1
  namespace: myapplication
spec:
  containers:
  - image: radial/busyboxplus:curl
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
  restartPolicy: Always
```

```
[root@workstation myobjects]# cat busybox.yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox1
  labels:
    app: busybox1
  namespace: myapplication
spec:
  containers:
  - image: radial/busyboxplus:curl
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
  restartPolicy: Always
[root@workstation myobjects]# kubectl apply -f busybox.yaml
pod/busybox1 created
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP             NODE
                          NOMINATED NODE   READINESS GATES
mydeployment-8496ddbc68-chxdc   1/1     Running   0          38m   172.16.99.6    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
mydeployment-8496ddbc68-kbn65   1/1     Running   0          17h   172.16.99.1    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=busybox1
NAME       READY   STATUS    RESTARTS   AGE
busybox1   1/1     Running   0          22s
[root@workstation myobjects]#
```

Let's login to the busybox pod and check the application status of our deployment pod

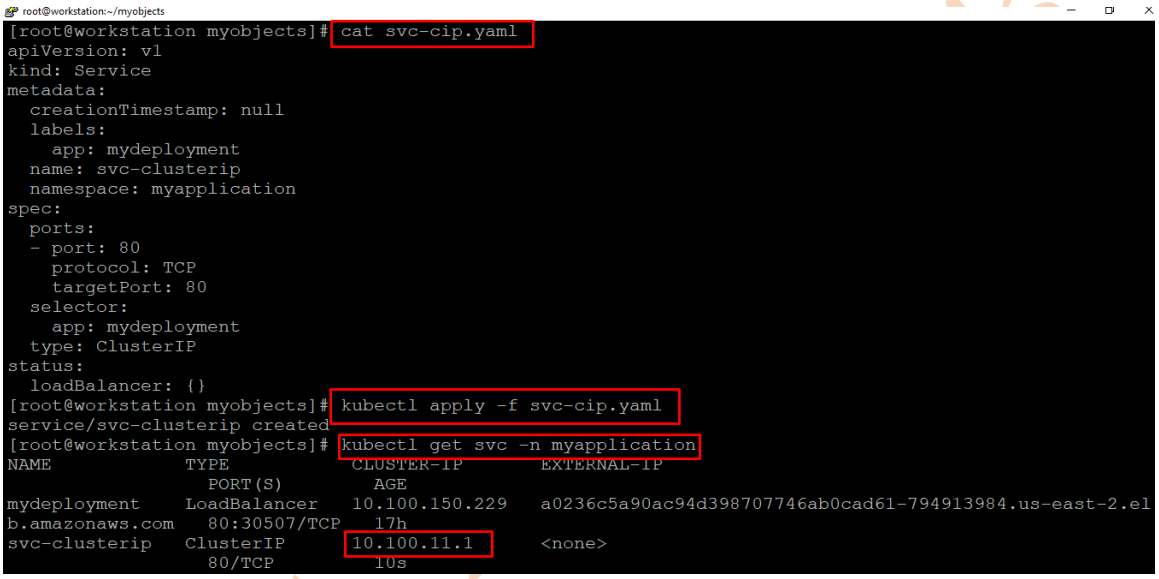
kubectl exec -it busybox1 -n myapplication /bin/sh

```
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP             NODE
                          NOMINATED NODE   READINESS GATES
mydeployment-8496ddbc68-chxdc   1/1     Running   0          38m   172.16.99.6    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
mydeployment-8496ddbc68-kbn65   1/1     Running   0          17h   172.16.99.1    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=busybox1
NAME       READY   STATUS    RESTARTS   AGE
busybox1   1/1     Running   0          22s
[root@workstation myobjects]# kubectl exec -it busybox1 -n myapplication /bin/sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [P
OD] -- [COMMAND] instead.
/bin/sh: shopt: not found
[ root@busybox1:/ ]$ curl http://172.16.99.6
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

Let's test the application with the service of type clusterip using below.

vi svc-cip.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null

```
  labels:
    app: mydeployment
  name: svc-clusterip
  namespace: myapplication
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: mydeployment
  type: ClusterIP
```



Verification:

```
service/svc-clusterip created
[root@workstation myobjects]# kubectl get svc -n myapplication
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
                  PORT(S)          AGE
mydeployment    LoadBalancer   10.100.150.229   a0236c5a90ac94d398707746ab0cad61-794913984.us-east-2.el
b.amazonaws.com   80:30507/TCP   17h
svc-clusterip   ClusterIP      10.100.11.1      <none>
                  80/TCP           10s
[root@workstation myobjects]# kubectl exec -it busybox1 -n myapplication /bin/sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [P
OD] -- [COMMAND] instead.
/bin/sh: shopt: not found
[ root@busybox1:/ ]$ curl http://10.100.11.1
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

From the above we can conclude that we can able to hit the application via service of type clusterip

2. **Nodeport:**

```
mydeployment   3/3     3            3           24h
[ec2-user@master ~]$ kubectl expose deploy dvsbatch3 --name=nodeport-service --target-port=80 --port 8080 --type NodePort
service/nodeport-service exposed
[ec2-user@master ~]$ kubectl get svc
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
dvsbatch3-service  ClusterIP   10.110.222.174   <none>        8080/TCP         13m
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP          24h
myservice          ClusterIP   10.108.174.113   <none>        80/TCP           24h
nodeport-service   NodePort    10.98.235.212    <none>        8080:30964/TCP   4s
[ec2-user@master ~]$
```

vi svc-np.yaml

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: svc-nodeport
  namespace: myapplication
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: mydeployment
```

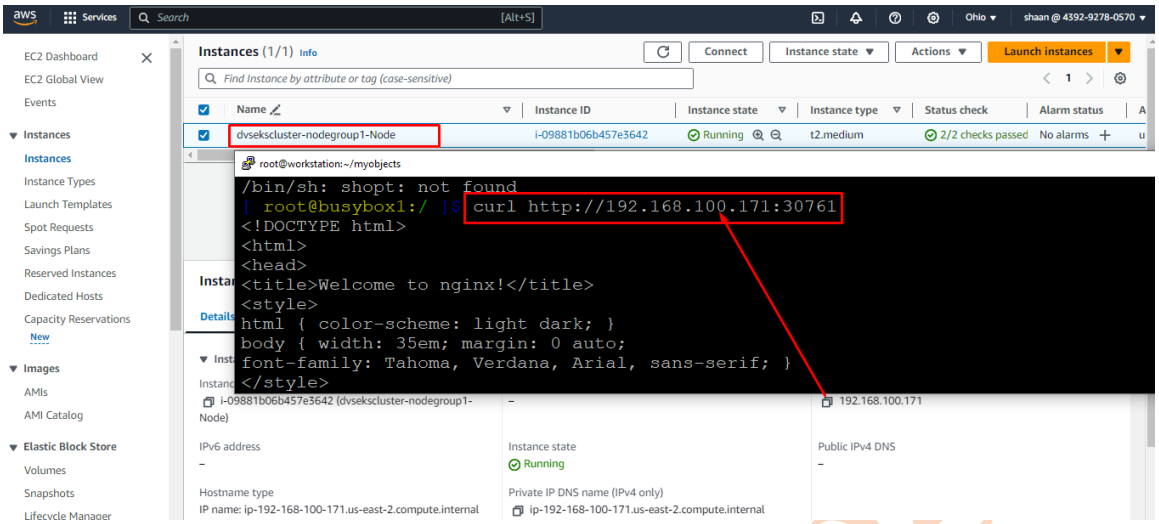
type: **NodePort**

```
[root@workstation myobjects]# cat svc-np.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: svc-nodeport
  namespace: myapplication
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: mydeployment
  type: NodePort
status:
  loadBalancer: {}
[root@workstation myobjects]# kubectl apply -f svc-np.yaml
service/svc-nodeport created
[root@workstation myobjects]# kubectl get svc -n myapplication
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
                  PORT(S)          AGE
mydeployment    LoadBalancer   10.100.150.229   a0236c5a90ac94d398707746ab0cad61-794913984.us-east-2.el
b.amazonaws.com   80:30507/TCP   17h
svc-clusterip   ClusterIP      10.100.11.1      <none>
                  80/TCP           4m19s
svc-nodeport    NodePort       10.100.112.18    <none>
                  80:30761/TCP   8s
[root@workstation myobjects]#
```

Verification:

```
root@workstation:~/myobjects
[root@workstation myobjects]# kubectl get svc -n myapplication
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
                  PORT(S)          AGE
mydeployment    LoadBalancer   10.100.150.229   a0236c5a90ac94d398707746ab0cad61-794913984.us-east-2.el
b.amazonaws.com   80:30507/TCP   17h
svc-clusterip   ClusterIP      10.100.11.1      <none>
                  80/TCP           4m19s
svc-nodeport    NodePort       10.100.112.18    <none>
                  80:30761/TCP   8s
[root@workstation myobjects]# kubectl exec -it busybox1 -n myapplication /bin/sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [P
OD] -- [COMMAND] instead.
/bin/sh: shopt: not found
[ root@busybox1:/ ]$ curl http://192.168.100.171:30761
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
```

Here 192.168.100.171 is the ipaddress of our workernode. You can take any worker node ipaddress to test your node port.

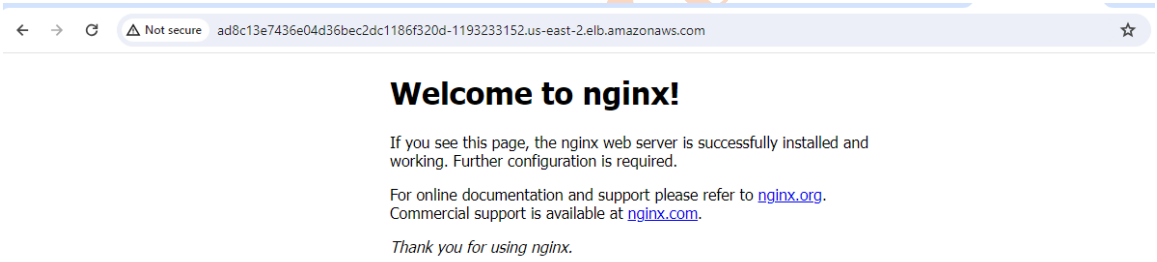


3. Configuring Loadbalancer type:

```
vi svc-lb.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: svc-nodeport
  namespace: myapplication
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: mydeployment
  type: LoadBalancer
status:
  loadBalancer: {}
```

```
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-8496ddbc68-cdldg   1/1     Running   0          54s
mydeployment-8496ddbc68-srbb4   1/1     Running   0          54s
[root@workstation myobjects]# cat svc-lb.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: svc-nodeport
  namespace: myapplication
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: mydeployment
  type: LoadBalancer
status:
  loadBalancer: {}
[root@workstation myobjects]# kubectl apply -f svc-lb.yaml
service/svc-nodeport created
[root@workstation myobjects]# kubectl get svc -n myapplication
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP
                PORT(S)        AGE
svc-nodeport   LoadBalancer   10.100.221.64   ad8c13e7436e04d36bec2dc1186f320d-1193233152.us-east-2.elb
.amazonaws.com   80:31748/TCP   11s
[root@workstation myobjects]#
```

From the above we got our loadbalancer url, just hit above url in the browser



Finally we got our application url and its up & running.

6. Scheduling

1. Labeling the nodes

```
[root@workstation myobjects]# kubectl get nodes
NAME                                           STATUS   ROLES    AGE    VERSION
ip-192-168-100-118.us-east-2.compute.internal   Ready    <none>   10m    v1.27.7-eks-e71965b
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h7m   v1.27.7-eks-e71965b
[root@workstation myobjects]# kubectl get nodes  --show-labels
NAME                                           STATUS   ROLES    AGE    VERSION               LABELS
ip-192-168-100-118.us-east-2.compute.internal   Ready    <none>   10m    v1.27.7-eks-e71965b   alpha.e
sctl.io/cluster-name=dvseksclustr,alpha.eksctl.io/nodegroup-name=nodegroup1,beta.kubernetes.io/arch=a
d64,beta.kubernetes.io/instance-type=t2.medium,beta.kubernetes.io/os=linux,eks.amazonaws.com/capacityT
pe=ON_DEMAND,eks.amazonaws.com/nodegroup-image=ami-02d77f9fdc5a964ea,eks.amazonaws.com/nodegroup=nodeg
oup1,eks.amazonaws.com/sourceLaunchTemplateId=lt-0f742ac99001ba24f,eks.amazonaws.com/sourceLaunchTempl
teVersion=1,failure-domain.beta.kubernetes.io/region=us-east-2,failure-domain.beta.kubernetes.io/zone=
s-east-2a,k8s.io/cloud-provider-aws=c9047fecf124ac306f038a64fec27c52,kubernetes.io/arch=amd64,kubernet
s.io/hostname=ip-192-168-100-118.us-east-2.compute.internal,kubernetes.io/os=linux,node.kubernetes.io/
nstance-type=t2.medium,topology.kubernetes.io/region=us-east-2,topology.kubernetes.io/zone=us-east-2a
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h7m   v1.27.7-eks-e71965b   alpha.e
sctl.io/cluster-name=dvseksclustr,alpha.eksctl.io/nodegroup-name=nodegroup1,beta.kubernetes.io/arch=a
d64,beta.kubernetes.io/instance-type=t2.medium,beta.kubernetes.io/os=linux,eks.amazonaws.com/capacityT
pe=ON_DEMAND,eks.amazonaws.com/nodegroup-image=ami-02d77f9fdc5a964ea,eks.amazonaws.com/nodegroup=nodeg
oup1,eks.amazonaws.com/sourceLaunchTemplateId=lt-0f742ac99001ba24f,eks.amazonaws.com/sourceLaunchTempl
teVersion=1,failure-domain.beta.kubernetes.io/region=us-east-2,failure-domain.beta.kubernetes.io/zone=
s-east-2c,k8s.io/cloud-provider-aws=c9047fecf124ac306f038a64fec27c52,kubernetes.io/arch=amd64,kubernet
s.io/hostname=ip-192-168-100-171.us-east-2.compute.internal,kubernetes.io/os=linux,node.kubernetes.io/
nstance-type=t2.medium,topology.kubernetes.io/region=us-east-2,topology.kubernetes.io/zone=us-east-2c
[root@workstation myobjects]#
```

Adding label to a node:

```
^[[ANo resources found
[root@workstation myobjects]# kubectl get nodes -l env=dev
No resources found
[root@workstation myobjects]# kubectl label node ip-192-168-100-171.us-east-2.compute.internal env=dev
node/ip-192-168-100-171.us-east-2.compute.internal labeled
[root@workstation myobjects]# kubectl get nodes -l env=dev
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   4h52m   v1.27.7-eks-e71965b
[root@workstation myobjects]#
```

Remove label from a node:

```
[root@workstation myobjects]# kubectl get nodes -l env=dev
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   4h54m   v1.27.7-eks-e71965b
[root@workstation myobjects]# kubectl label node ip-192-168-100-171.us-east-2.compute.internal env-
node/ip-192-168-100-171.us-east-2.compute.internal unlabeled
[root@workstation myobjects]# kubectl get nodes -l env=dev
No resources found
[root@workstation myobjects]#
```

2 . Nodename:

If you want to schedule pods in only one node then you can use the below

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
```

```
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
      nodeName: ip-192-168-100-118.us-east-2.compute.internal
status: {}
```

```
[root@workstation myobjects]# kubectl apply -f deploy.yaml
deployment.apps/mydeployment configured
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
NAME                            READY   STATUS    RESTARTS   AGE    IP              NODE
                          NOMINATED NODE   READINESS GATES
mydeployment-868b9c8fd6-h6c4m   1/1     Running   0          2m2s   172.16.71.67    ip-192-168-100-118.u
s-east-2.compute.internal   <none>           <none>
mydeployment-868b9c8fd6-pcttz   1/1     Running   0          2m7s   172.16.71.66    ip-192-168-100-118.u
s-east-2.compute.internal   <none>           <none>
[root@workstation myobjects]#
```

Let's increase the replica count & see if the pods get schedule in different nodes or same node.

```
root@workstation:~/myobjects
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 4
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
      nodeName: ip-192-168-100-118.us-east-2.compute.internal
status: {}
[root@workstation myobjects]# kubectl apply -f deploy.yaml
deployment.apps/mydeployment configured
[root@workstation myobjects]#
```

```
root@workstation:~/myobjects
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP              NODE
                       NOMINATED NODE   READINESS GATES
mydeployment-868b9c8fd6-948p2   1/1     Running   0          17s     172.16.71.68    ip-192-168-100-118.
us-east-2.compute.internal   <none>           <none>
mydeployment-868b9c8fd6-h6c4m   1/1     Running   0          3m24s   172.16.71.67    ip-192-168-100-118.
us-east-2.compute.internal   <none>           <none>
mydeployment-868b9c8fd6-kvqhl   1/1     Running   0          17s     172.16.71.69    ip-192-168-100-118.
us-east-2.compute.internal   <none>           <none>
mydeployment-868b9c8fd6-pcttz   1/1     Running   0          3m29s   172.16.71.66    ip-192-168-100-118.
us-east-2.compute.internal   <none>           <none>
[root@workstation myobjects]#
```

From the above output we can conclude that all the 4 pods are running in the same node.

2. NodeSelector

```
[root@workstation myobjects]# kubectl get nodes
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-118.us-east-2.compute.internal   Ready    <none>   22m     v1.27.7-eks-e71965b
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h19m   v1.27.7-eks-e71965b
[root@workstation myobjects]# kubectl get nodes -l env=dev
No resources found
[root@workstation myobjects]# kubectl label node ip-192-168-100-171.us-east-2.compute.internal env=dev
node/ip-192-168-100-171.us-east-2.compute.internal labeled
[root@workstation myobjects]# kubectl get nodes -l env=dev
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h20m   v1.27.7-eks-e71965b
[root@workstation myobjects]#
```

Let's create the deployment with nodeselector.

```
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
      nodeSelector:
        env: dev
status: {}
```

```
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
      nodeSelector:
        env: dev
status: {}
[root@workstation myobjects]# kubectl apply -f deploy.yaml
deployment.apps/mydeployment configured
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
```

From the above we can see that our deployment is going to run on the nodes who is having label as "env=dev".

```
[root@workstation myobjects]# kubectl get nodes -l env=dev
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h29m   v1.27.7-eks-e71965b
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP              NODE
                       NOMINATED NODE   READINESS GATES
mydeployment-665d88877-lp27x    1/1     Running   0          2m16s   172.16.99.16    ip-192-168-100-171.u
s-east-2.compute.internal   <none>           <none>
mydeployment-665d88877-mhzlk    1/1     Running   0          2m14s   172.16.99.17    ip-192-168-100-171.u
s-east-2.compute.internal   <none>           <none>
[root@workstation myobjects]#
```

Now let's give label as env=dev to other node & then increase the replica count to 4 & check where the new pods going to get schedule.

```
[root@workstation myobjects]# kubectl get nodes
kNAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-118.us-east-2.compute.internal   Ready    <none>   39m     v1.27.7-eks-e71965b
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h35m   v1.27.7-eks-e71965b
[root@workstation myobjects]# kubectl get nodes -l env=dev
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h36m   v1.27.7-eks-e71965b
[root@workstation myobjects]# kubectl label node ip-192-168-100-118.us-east-2.compute.internal env=dev
node/ip-192-168-100-118.us-east-2.compute.internal labeled
[root@workstation myobjects]# kubectl get nodes -l env=dev
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-118.us-east-2.compute.internal   Ready    <none>   39m     v1.27.7-eks-e71965b
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   6h36m   v1.27.7-eks-e71965b
[root@workstation myobjects]#
```

```
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 4
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
      nodeSelector:
        env: dev
```

```
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP              NODE
                        NOMINATED NODE   READINESS GATES
mydeployment-665d88877-lp27x    1/1     Running   0          10m   172.16.99.16    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
mydeployment-665d88877-mhzlk    1/1     Running   0          10m   172.16.99.17    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
[root@workstation myobjects]# kubectl apply -f deploy.yaml
deployment.apps/mydeployment configured
[root@workstation myobjects]# kubectl get pods -n myapplication -l app=mydeployment -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP              NODE
                        NOMINATED NODE   READINESS GATES
mydeployment-665d88877-bcpwh    1/1     Running   0          3s    172.16.71.71    ip-192-168-100-118.us-
east-2.compute.internal   <none>           <none>
mydeployment-665d88877-lp27x    1/1     Running   0          10m   172.16.99.16    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
mydeployment-665d88877-mhzlk    1/1     Running   0          10m   172.16.99.17    ip-192-168-100-171.us-
east-2.compute.internal   <none>           <none>
mydeployment-665d88877-n5qbj    1/1     Running   0          3s    172.16.71.70    ip-192-168-100-118.us-
east-2.compute.internal   <none>           <none>
[root@workstation myobjects]#
```

From the above we can conclude that pod's are getting scheduled in the node where we have label set as "env=dev"

3. Resource Management:

```
[ec2-user@master kubernetes-installation]$ cat redis.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: redis
  name: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: redis
    spec:
      containers:
      - image: redis:3.2-alpine
        name: redis
        resources:
          limits:
            cpu: 1
            memory: 200Mi
          requests:
            cpu: 80m
            memory: 20Mi
[ec2-user@master kubernetes-installation]$ kubectl apply -f redis.yaml
deployment.apps/redis configured
[ec2-user@master kubernetes-installation]$
```

```
[ec2-user@master kubernetes-installation]$ kubectl get pods -l app=redis -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP          NODE      NOMINATED NODE   READINESS GATES
redis-749c46d5bd-5nfqf   1/1     Running   0          79s   10.36.0.1   worker2   <none>           <none>
[ec2-user@master kubernetes-installation]$ kubectl get pods -l app=redis -o wide -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: Pod
  metadata:
    creationTimestamp: "2020-08-12T02:52:46Z"
```

```
    uid: 57de7091-67e6-4590-8af0-b6491a1239d2
  resourceVersion: "51103"
  selfLink: /api/v1/namespaces/default/pods/redis-749c46d5bd-5nfqf
  uid: ed6cd0fa-881a-40a3-9466-8aac3d4b39b6
spec:
  containers:
  - image: redis:3.2-alpine
    imagePullPolicy: IfNotPresent
    name: redis
    resources:
      limits:
        cpu: "1"
        memory: 200Mi
      requests:
        cpu: 80m
        memory: 20Mi
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-c9tfm
      readOnly: true
```

4. DaemonSets and Manually Scheduled Pods :

```
[ec2-user@master kubernetes-installation]$ cat deamon.yaml
apiVersion: apps/v1
kind: "DaemonSet"
metadata:
  labels:
    app: nginx
    ssd: "true"
  name: nginx-fast-storage
spec:
  selector:
    matchLabels:
      app: nginx
      ssd: "true"
  template:
    metadata:
      labels:
        app: nginx
        ssd: "true"
    spec:
      nodeSelector:
        ssd: "true"
      containers:
        - name: nginx
          image: nginx:1.10.0
[ec2-user@master kubernetes-installation]$ kubectl create -f deamon.yaml
daemonset.apps/nginx-fast-storage created
```

```
daemonset.apps/nginx-fast-storage created
[ec2-user@master kubernetes-installation]$ kubectl get ds
NAME                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
nginx-fast-storage   0         0         0       0            0           ssd=true        9s
[ec2-user@master kubernetes-installation]$ kubectl get nodes -l ssd=true
No resources found in default namespace.
[ec2-user@master kubernetes-installation]$ kubectl label node worker1 ssd=true
node/worker1 labeled
[ec2-user@master kubernetes-installation]$ kubectl get nodes -l ssd=true
NAME      STATUS   ROLES    AGE   VERSION
worker1   Ready    <none>   2d    v1.18.6
[ec2-user@master kubernetes-installation]$ kubectl get ds
NAME                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
nginx-fast-storage   1         1         1       1            1           ssd=true        64s
[ec2-user@master kubernetes-installation]$ kubectl get pods -l ssd=true
NAME                       READY   STATUS    RESTARTS   AGE
nginx-fast-storage-d7gkq   1/1     Running   0          43s
[ec2-user@master kubernetes-installation]$ kubectl get pods -l ssd=true -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP           NODE      NOMINATED NODE   READINESS GATES
nginx-fast-storage-d7gkq   1/1     Running   0          50s   10.44.0.4    worker1   <none>           <none>
[ec2-user@master kubernetes-installation]$ kubectl get nodes -l ssd=true
NAME      STATUS   ROLES    AGE   VERSION
worker1   Ready    <none>   2d    v1.18.6
```

```
[ec2-user@master kubernetes-installation]$ kubectl label node worker2 ssd=true
node/worker2 labeled
[ec2-user@master kubernetes-installation]$ kubectl get pods -l ssd=true -o wide
NAME                        READY   STATUS              RESTARTS   AGE    IP          NODE      NOMINATED NODE   READINESS
GATES
nginx-fast-storage-d7gkq    1/1     Running             0          110s   10.44.0.4   worker1   <none>           <none>
nginx-fast-storage-w98tw    0/1     ContainerCreating   0          8s     <none>      worker2   <none>           <none>
[ec2-user@master kubernetes-installation]$ kubectl get ds
NAME                 DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
nginx-fast-storage   2         2         2       2            2           ssd=true        2m52s
[ec2-user@master kubernetes-installation]$ kubectl lable node worker2 ssd-
```

```
[ec2-user@master kubernetes-installation]$ kubectl label node worker2 ssd-
node/worker2 labeled
[ec2-user@master kubernetes-installation]$ kubectl get pods -l ssd=true -o wide
NAME                        READY   STATUS        RESTARTS   AGE     IP          NODE      NOMINATED NODE   READINESS G
nginx-fast-storage-d7gkq    1/1     Running       0          2m42s   10.44.0.4   worker1   <none>           <none>
nginx-fast-storage-w98tw    0/1     Terminating   0          60s     10.36.0.2   worker2   <none>           <none>
[ec2-user@master kubernetes-installation]$
```

5. Displaying Scheduler Events:

```
This section helps us more in checking the events happening at the scheduler level.

Listing the scheduler pod.
[root@master1 ~]# kubectl get pods -n kube-system|grep -i schedul
kube-scheduler-master1            1/1     Running   1          19h

Checking the logs

[root@master1 ~]# kubectl logs kube-scheduler-master1 -f -n kube-system


To get all the events in the project
kubectl get events -n <project name>

To Delete all the pods in the project/namespace

kubectl delete pods --all -n mynginx

Watch events as they are appearing in real time:

kubectl get events -w

The location of a systemd service scheduler pod:

/var/log/kube-scheduler.log
```

7. Application lifecycle management

Installing and configuring docker in the workstation:

yum install docker -y
systemctl enable docker
systemctl restart docker

```
[root@workstation ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Sat 2023-12-16 02:29:10 UTC; 20min ago
     Docs: https://docs.docker.com
  Process: 3800 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCES
S)
  Process: 3799 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
 Main PID: 3803 (dockerd)
    Tasks: 7
   Memory: 20.3M
   CGroup: /system.slice/docker.service
           └─3803 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-u...

Dec 16 02:29:09 workstation dockerd[3803]: time="2023-12-16T02:29:09.899484120Z" level=info msg="...rpc
Dec 16 02:29:09 workstation dockerd[3803]: time="2023-12-16T02:29:09.948628977Z" level=warning ms...ht"
Dec 16 02:29:09 workstation dockerd[3803]: time="2023-12-16T02:29:09.949138521Z" level=warning ms...ce"
Dec 16 02:29:09 workstation dockerd[3803]: time="2023-12-16T02:29:09.949654225Z" level=info msg="...t."
Dec 16 02:29:10 workstation dockerd[3803]: time="2023-12-16T02:29:10.186262647Z" level=info msg="...ss"
Dec 16 02:29:10 workstation dockerd[3803]: time="2023-12-16T02:29:10.232813777Z" level=info msg="...e."
Dec 16 02:29:10 workstation dockerd[3803]: time="2023-12-16T02:29:10.249123824Z" level=info msg="....25
Dec 16 02:29:10 workstation dockerd[3803]: time="2023-12-16T02:29:10.249612112Z" level=info msg="...on"
Dec 16 02:29:10 workstation systemd[1]: Started Docker Application Container Engine.
Dec 16 02:29:10 workstation dockerd[3803]: time="2023-12-16T02:29:10.276199375Z" level=info msg="...ck"
Hint: Some lines were ellipsized, use -l to show in full.
[root@workstation ~]#
```

Creating a custom image & pushing it to Dockerhub

```
[ec2-user@master ~]$ cat Dockerfile
FROM httpd
RUN echo "Hi I am V1" > /usr/local/apache2/htdocs/index.html
[ec2-user@master ~]$ sudo docker build -t "shan5a6/myimage:v1" .
Sending build context to Docker daemon 3.639 MB
Step 1/2 : FROM httpd
 ---> a6ea92c35c43
Step 2/2 : RUN echo "Hi I am V1" > /usr/local/apache2/htdocs/index.html
 ---> Using cache
 ---> baaf3cf68032
Successfully built baaf3cf68032
[ec2-user@master ~]$ sudo docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub
.docker.com to create one.
Username (shan5a6): shan5a6
Password:
Login Succeeded
[ec2-user@master ~]$ docker push shan5a6/myimage:v1
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fv
ar%2Frun%2Fdocker.sock/v1.26/images/shan5a6/myimage/push?tag=v1: dial unix /var/run/docker.sock: connect: permission denie
d
[ec2-user@master ~]$ sudo docker push shan5a6/myimage:v1
The push refers to a repository [docker.io/shan5a6/myimage]
7cc39fb2281c: Pushed
88b680b1fdfc: Mounted from library/httpd
843c3701e622: Mounted from library/httpd
3ba8a4f66ba2: Mounted from library/httpd
c865989f86f7: Mounted from library/httpd
d0f104dc0a1f: Mounted from library/httpd
v1: digest: sha256:59fe38c4d1180e7ffb7c5ebec68e2e702ae8243e568fcb428a049ffefaebd377 size: 1574
[ec2-user@master ~]$
```

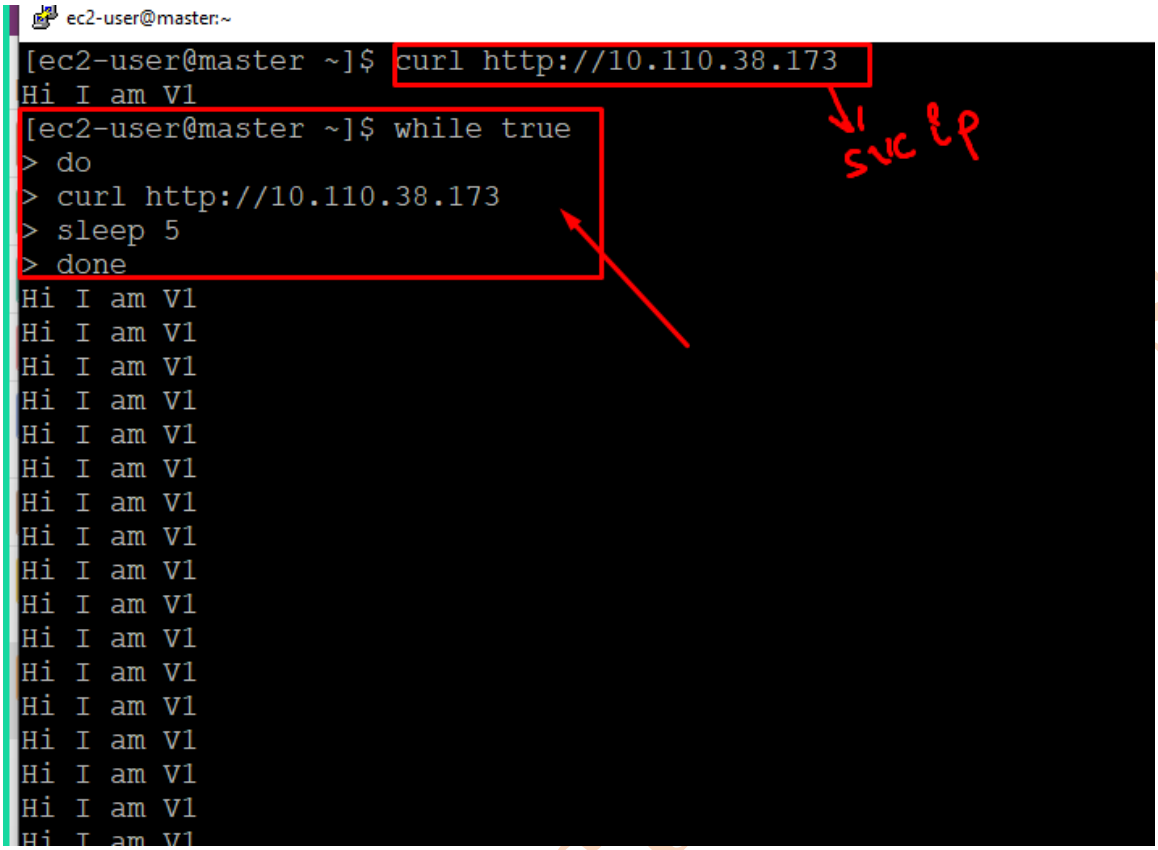
Creating deployment with custom image:

Note: Make sure you are using "kubectl create -f deploy.yaml --record" for creating the deployment

```
[ec2-user@master ~]$ cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mydeployment
  template:
    metadata:
      name: mydeployment
      labels:
        app: mydeployment
    spec:
      containers:
      - image: shan5a6/myimage:v1
        name: app
[ec2-user@master ~]$ kubectl apply -f  deploy.yaml
deployment.apps/mydeployment created
[ec2-user@master ~]$ kubectl get deploy
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
mydeployment   0/3     3            0           5s
[ec2-user@master ~]$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-598689ff9f-bl6m6   1/1     Running   0          11s
mydeployment-598689ff9f-h2gz9   1/1     Running   0          11s
mydeployment-598689ff9f-q8m29   1/1     Running   0          11s
```

Creating service and verifying the current application status:

```
[ec2-user@master ~]$ kubectl create -f deploy.yaml --record
deployment.apps/mydeployment created
[ec2-user@master ~]$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-598689ff9f-kjdkj   1/1     Running   0          49s
mydeployment-598689ff9f-v9xb4   1/1     Running   0          49s
mydeployment-598689ff9f-z5wp6   1/1     Running   0          49s
nginx-fast-storage-x8r98        1/1     Running   0          10m
[ec2-user@master ~]$ kubectl get -o yaml deploy mydeployment|grep -i image
                f:image: {}
                f:imagePullPolicy: {}
      - image: shan5a6/myimage:v1
        imagePullPolicy: IfNotPresent
[ec2-user@master ~]$ kubectl expose deploy mydeployment --port 80 --target-port 80 --name mydeploysvc
service/mydeploysvc exposed
[ec2-user@master ~]$ kubectl describe svc mydeploysvc
Name:              mydeploysvc
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=mydeployment
Type:              ClusterIP
IP:                10.110.38.173
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.36.0.1:80,10.36.0.2:80,10.44.0.3:80
Session Affinity:  None
Events:            <none>
[ec2-user@master ~]$
```



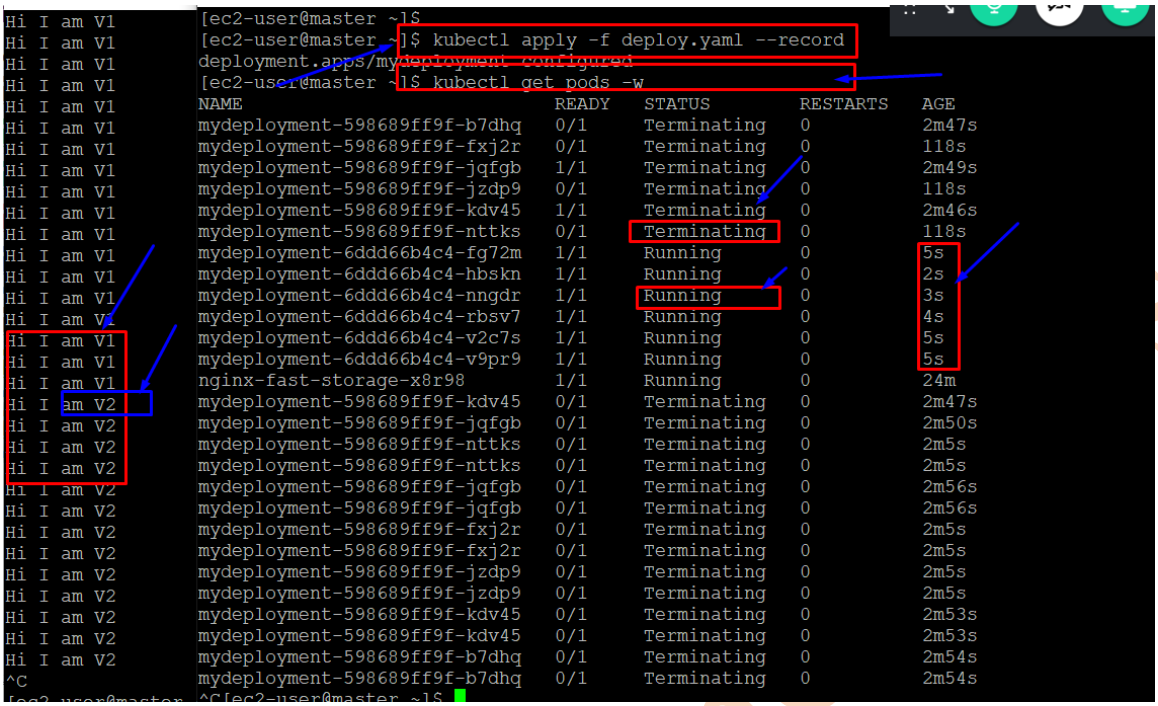
Note: If you are getting loadbalncer url then use the same for hitting and displaying application status. In this case

Creating customized image for the new release:

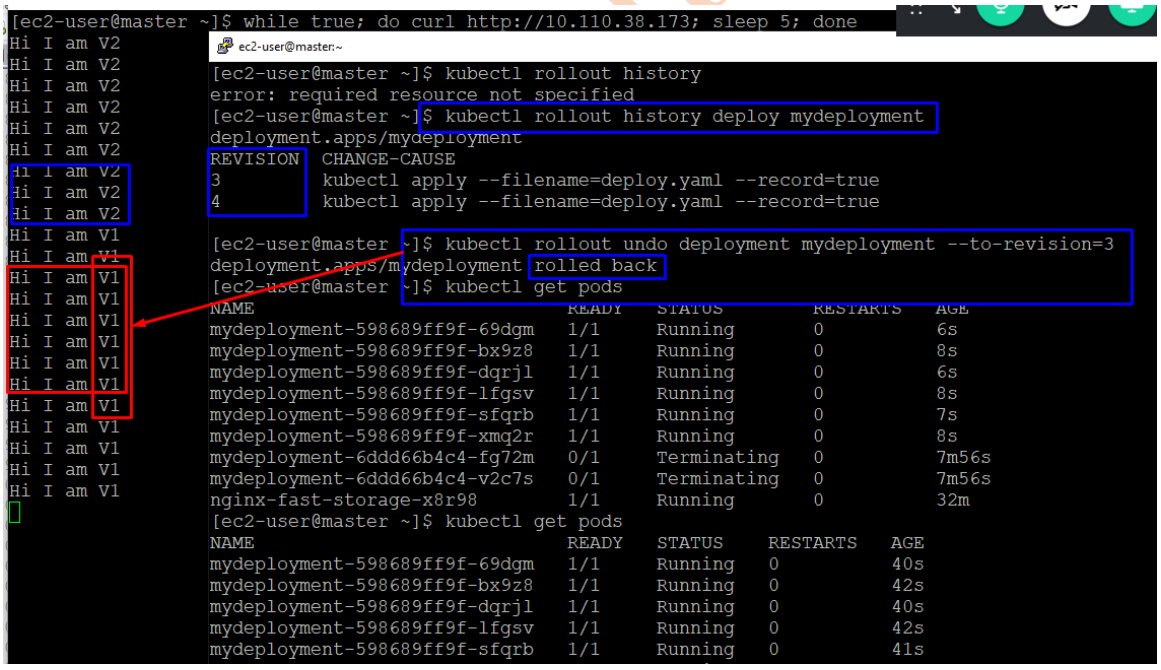
```
[ec2-user@master ~]$ cat Dockerfile
FROM httpd
RUN echo "Hi I am V2" > /usr/local/apache2/htdocs/index.html
[ec2-user@master ~]$ sudo docker build -t "shan5a6/myimage:v2" .
Sending build context to Docker daemon  3.64 MB
Step 1/2 : FROM httpd
 ---> a6ea92c35c43
Step 2/2 : RUN echo "Hi I am V2" > /usr/local/apache2/htdocs/index.html
 ---> Running in 12bd4bfa1388

 ---> 0a1e938f1efe
Removing intermediate container 12bd4bfa1388
Successfully built 0a1e938f1efe
[ec2-user@master ~]$ sudo docker push shan5a6/myimage:v2
The push refers to a repository [docker.io/shan5a6/myimage]
817fa498a6a9: Pushed
88b680b1fdfc: Layer already exists
843c3701e622: Layer already exists
3ba8a4f66ba2: Layer already exists
c865989f86f7: Layer already exists
d0f104dc0a1f: Layer already exists
v2: digest: sha256:e959e13896dde4781b9613fb3ebfce94176365231264c475459eec6031b0a9cc si
[ec2-user@master ~]$
```

```
[ec2-user@master ~]$ kubectl scale deploy mydeployment --replicas=6
deployment.apps/mydeployment scaled
[ec2-user@master ~]$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
mydeployment-598689ff9f-b7dhq   1/1     Running   0          51s
mydeployment-598689ff9f-fxj2r   1/1     Running   0          2s
mydeployment-598689ff9f-jqfgb   1/1     Running   0          53s
mydeployment-598689ff9f-jzdp9   1/1     Running   0          2s
mydeployment-598689ff9f-kdv45   1/1     Running   0          50s
mydeployment-598689ff9f-nttks   1/1     Running   0          2s
nginx-fast-storage-x8r98        1/1     Running   0          22m
[ec2-user@master ~]$ vi deploy.yaml
[ec2-user@master ~]$ cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeployment
spec:
  replicas: 6
  selector:
    matchLabels:
      app: mydeployment
  template:
    metadata:
      name: mydeployment
      labels:
        app: mydeployment
    spec:
      containers:
      - image: shan5a6/myimage:v2
        name: app
[ec2-user@master ~]$
```

Take two terminals and monitor the deployment changes like below:



Rollback to the previous version:



8. Configuring an Application for High Availability and Scale

1. Configmaps :

Creating configmap:

```
[ec2-user@master ~]$ cat configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: appconfig
data:
  key1: value1
  key2: value2
[ec2-user@master ~]$ kubectl create -f configmap.yaml
configmap/appconfig created
[ec2-user@master ~]$ kubectl describe configmap appconfig
Name:         appconfig
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
key1:
----
value1
key2:
----
value2
Events:  <none>
[ec2-user@master ~]$
```

Creating pod with configmap

```
[ec2-user@master ~]$
[ec2-user@master ~]$ cat configmap-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
  - name: app-container
    image: busybox:1.28
    command: ['sh', '-c', "echo $(MY_VAR) && sleep 3600"]
    env:
    - name: MY_VAR
      valueFrom:
        configMapKeyRef:
          name: appconfig
          key: key1
[ec2-user@master ~]$ kubectl apply -f configmap-pod.yaml
pod/configmap-pod created
[ec2-user@master ~]$ kubectl get pods|grep -i app-container
[ec2-user@master ~]$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
configmap-pod                 1/1     Running   0          27s
```

Configmap Name

Verifying details inside the pod:

```
[ec2-user@master ~]$ kubectl exec -it configmap-pod sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version.
MAND] instead.
/ # env|grep -i MYVAR
/ # env|grep -i MY
MY_VAR=value1
MYDEPLOYSVC_SERVICE_HOST=10.110.38.173
MYDEPLOYSVC_SERVICE_PORT=80
MYDEPLOYSVC_PORT=tcp://10.110.38.173:80
MYDEPLOYSVC_PORT_80_TCP_ADDR=10.110.38.173
MYDEPLOYSVC_PORT_80_TCP_PORT=80
MYDEPLOYSVC_PORT_80_TCP_PROTO=tcp
MYDEPLOYSVC_PORT_80_TCP=tcp://10.110.38.173:80
[ec2-user@master ~]$ kubectl describe configmap appconfig
Name:         appconfig
Namespace:    default
Labels:       <none>
```

2. Creating a secret

Let's create the secret

```
[ec2-user@master ~]$ cat secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: appsecret
stringData:
  subject: "kubernetes"
  batchno: "3"
[ec2-user@master ~]$ kubectl create -f secret.yaml
secret/appsecret created
[ec2-user@master ~]$ kubectl describe secret appsecret
Name:         appsecret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
batchno:  1 bytes
subject:  10 bytes
[ec2-user@master ~]$ kubectl get -o yaml secret appsecret
apiVersion: v1
data:
  batchno: Mw==
  subject: a3ViZXJuZXRlcw==
kind: Secret
metadata:
  creationTimestamp: "2020-08-13T02:58:09Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
```

Encrypted data

Creating a pod:

```
[ec2-user@master ~]$ cat secret-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
  - name: app-container
    image: busybox
    command: ['sh', '-c', "echo Hello, Kubernetes! && sleep 3600"]
    env:
    - name: MY_SUB
      valueFrom:
        secretKeyRef:
          name: appsecret
          key: subject
[ec2-user@master ~]$ kubectl create -f secret-pod.yaml
pod/secret-pod created
[ec2-user@master ~]$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
configmap-pod                 1/1     Running   0          10m
mydeployment-598689ff9f-69dgm 1/1     Running   0          32m
mydeployment-598689ff9f-bx9z8 1/1     Running   0          32m
mydeployment-598689ff9f-dqrjl 1/1     Running   0          32m
mydeployment-598689ff9f-lfgsv 1/1     Running   0          32m
mydeployment-598689ff9f-sfqrb 1/1     Running   0          32m
mydeployment-598689ff9f-xmq2r 1/1     Running   0          32m
nginx-fast-storage-x8r98      1/1     Running   0          65m
secret-pod                    1/1     Running   0          4s
[ec2-user@master ~]$
```

```
[ec2-user@master ~]$ kubectl exec -it secret-pod sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future
MAND] instead.
/ # env|grep -i my
MY_SUB=kubernetes
MYDEPLOYSVC_SERVICE_HOST=10.110.38.173
MYDEPLOYSVC_PORT=tcp://10.110.38.173:80
```

Encrypting & decrypting the secrets:

```
[ec2-user@master ~]$ echo "dvsbatch3"|base64
ZHZzYmF0Y2gzCg==
[ec2-user@master ~]$ echo "ZHZzYmF0Y2gzCg=="|base64 -d
dvsbatch3
[ec2-user@master ~]$
```

9. Storage

In Eks we can make use of aws ebs volume by default in the cluster we will not have the storage class we have to enable it. Please do execute the below to enable ebs storage class in eks cluster.

Storage Management::
https://www.stacksimplify.com/aws-eks/kubernetes-storage/install-aws-ebs-csi-driver-on-aws-eks-for-persistent-storage/

Ref: Git Link
https://github.com/kubernetes-sigs/aws-ebs-csi-driver.git

POLICY_ARN=$(aws iam list-policies --query 'Policies[?PolicyName==`AmazonEBSCSIDriverPolicy`].Arn' --output text)
NODE_ROLE_NAME=$(kubectl get configmap aws-auth -n kube-system -o yaml |grep rolearn|awk '{print $2}'|awk -F'/' '{print $2}')
aws iam attach-role-policy --policy-arn $POLICY_ARN --role-name $NODE_ROLE_NAME

```
[root@workstation myobjects]# POLICY_ARN=$(aws iam list-policies --query 'Policies[?PolicyName==`Amazon
EBSCSIDriverPolicy`].Arn' --output text)
NODE_ROLE_NAME=$(kubectl get configmap aws-auth -n kube-system  -o yaml |grep rolearn|awk '{print $2}'|
awk -F'/' '{print $2}')
aws iam attach-role-policy  --policy-arn $POLICY_ARN --role-name $NODE_ROLE_NAME
[root@workstation myobjects]# NODE_ROLE_NAME=$(kubectl get configmap aws-auth -n kube-system  -o yaml |
grep rolearn|awk '{print $2}'|awk -F'/' '{print $2}')
[root@workstation myobjects]# aws iam attach-role-policy  --policy-arn $POLICY_ARN --role-name $NODE_RO
LE_NAME
[root@workstation myobjects]#
[root@workstation myobjects]# echo $POLICY_ARN
arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
[root@workstation myobjects]# echo $NODE_ROLE_NAME
eksctl-dvseks cluster-nodegroup-nod-NodeInstanceRole-PByQJHwQfDyk
[root@workstation myobjects]#
```

Above commands attach the AmazonEBSCSIDriverPolicy to the node role "eksctl-dvseks cluster-nodegroup-nod-NodeInstanceRole-PByQJHwQfDyk"

Install EBS CSI Driver
kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/overlays/stable/?ref=master"

```
eksctl-dvseksccluster-nodegroup-nod-NodeInstanceRole-PBtyQJHwQfDyk
[root@workstation myobjects]# kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/ku
bernetes/overlays/stable/?ref=master"
serviceaccount/ebs-csi-controller-sa created
serviceaccount/ebs-csi-node-sa created
role.rbac.authorization.k8s.io/ebs-csi-leases-role created
clusterrole.rbac.authorization.k8s.io/ebs-csi-node-role created
clusterrole.rbac.authorization.k8s.io/ebs-external-attacher-role created
clusterrole.rbac.authorization.k8s.io/ebs-external-provisioner-role created
clusterrole.rbac.authorization.k8s.io/ebs-external-resizer-role created
clusterrole.rbac.authorization.k8s.io/ebs-external-snapshotter-role created

rolebinding.rbac.authorization.k8s.io/ebs-csi-leases-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/ebs-csi-attacher-binding created
clusterrolebinding.rbac.authorization.k8s.io/ebs-csi-node-getter-binding created
clusterrolebinding.rbac.authorization.k8s.io/ebs-csi-provisioner-binding created
```

kubectl create -f - <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
EOF

```
[root@workstation myobjects]# kubectl create -f - <<EOF
> apiVersion: storage.k8s.io/v1
> kind: StorageClass
> metadata:
>   name: ebs-sc
> provisioner: ebs.csi.aws.com
> volumeBindingMode: WaitForFirstConsumer
> EOF
storageclass.storage.k8s.io/ebs-sc created
[root@workstation myobjects]# kubectl get sc
NAME            PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION
GE
ebs-sc          ebs.csi.aws.com         Delete          WaitForFirstConsumer   false
s
gp2 (default)   kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   false
4h
[root@workstation myobjects]#
```

Now from the above we conclude that storage class ebs-sc is created let's create the PVC and then the pod then verify if we are getting the pod with storage or not.

creating a persistent-volume claim:

kubectl create -f - <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2
  resources:

```
    requests:
      storage: 4Gi
EOF
```

```
[root@workstation myobjects]# kubectl create -f - <<EOF
> apiVersion: v1
> kind: PersistentVolumeClaim
> metadata:
>   name: ebs-claim
> spec:
>   accessModes:
>     - ReadWriteOnce
>   storageClassName: gp2
>   resources:
>     requests:
>       storage: 4Gi
> EOF

persistentvolumeclaim/ebs-claim created
[root@workstation myobjects]#
[root@workstation myobjects]# kubectl get pvc
NAME        STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
ebs-claim   Pending                                       gp2            7s
[root@workstation myobjects]# kubectl get pv
No resources found
[root@workstation myobjects]#
```

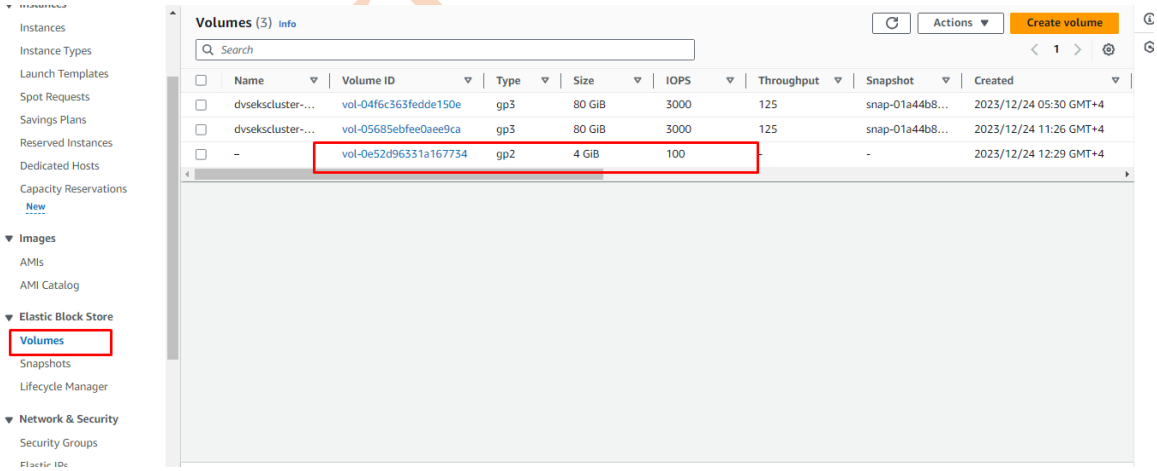
Creating the consumer to consume the pod:
pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: centos
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 5; done"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: ebs-claim
```

```
[root@workstation myobjects]# cat pod-with-pv.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: centos
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 5; done"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: ebs-claim

[root@workstation myobjects]# kubectl apply -f pod-with-pv.yaml
pod/app created
[root@workstation myobjects]# kubectl get pod
NAME    READY    STATUS     RESTARTS    AGE
app     1/1      Running    0           26s
```

Once we got our pod up & running we can see that we have our pv & pvc got created & bounded

```
[root@workstation myobjects]# kubectl get pod
NAME    READY    STATUS     RESTARTS    AGE
app     1/1      Running    0           26s
[root@workstation myobjects]# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
            STORAGECLASS   REASON   AGE
pvc-02fb9cb7-2ecd-47a5-b65c-e930e1c32d19   4Gi        RWO            Delete           Bound    default/
ebs-claim   gp2                     28s
[root@workstation myobjects]# kubectl get pvc
NAME         STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
  AGE
ebs-claim    Bound    pvc-02fb9cb7-2ecd-47a5-b65c-e930e1c32d19   4Gi        RWO            gp2
  105s
[root@workstation myobjects]#
```

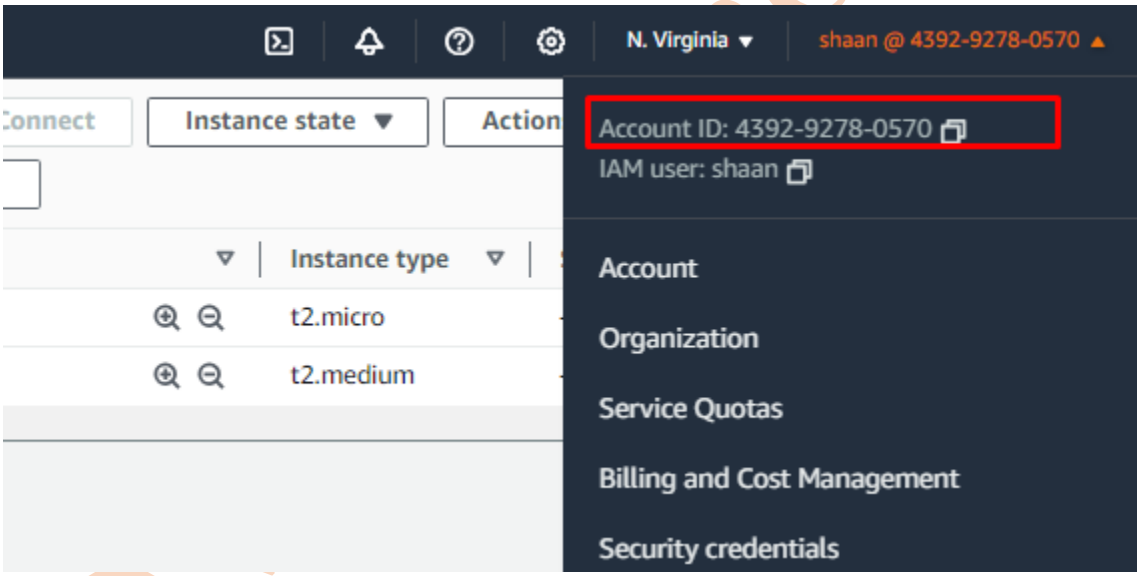


10 Cluster autoscalling:

Note: Make sure that you are changing the nodegroup ASG node type to atleast t2.medium otherwise node scheduler pod will notcome up .

```
[root@workstation myobjects]# aws eks list-clusters
{
    "clusters": [
        "dvseksclustер"
    ]
}
```

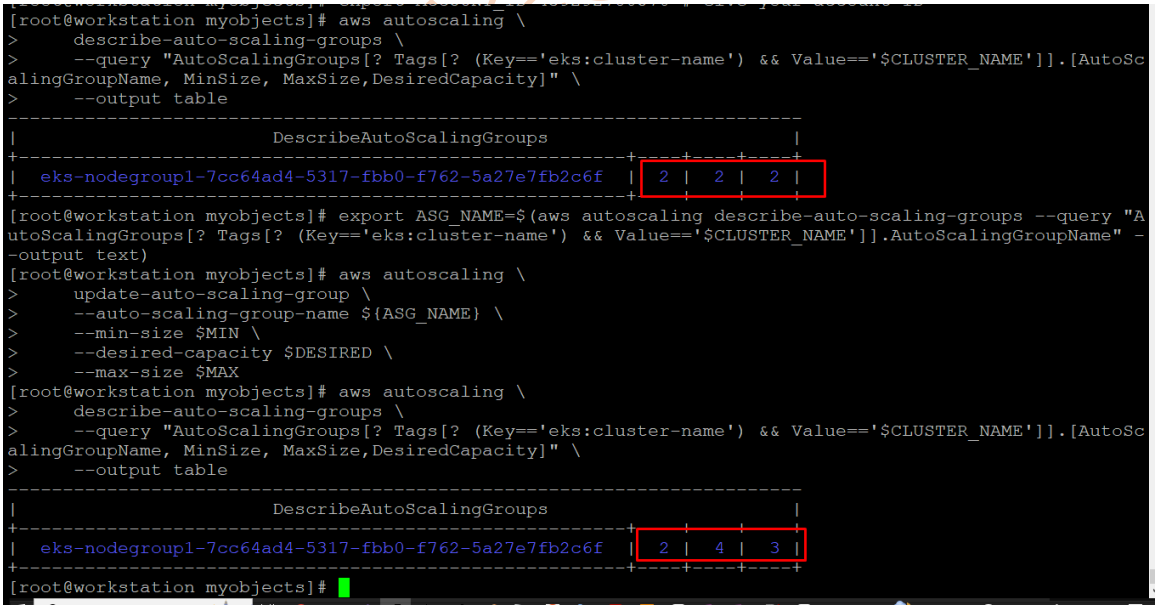
export CLUSTER_NAME="dvseksclustер" # GIVE YOUR CLUSTER NAME
export MIN=2
export MAX=4
export DESIRED=3
export ACCOUNT_ID=439292780570 # Give your account ID




```
[root@workstation myobjects]# export CLUSTER_NAME="dvseksclustер" # GIVE YOUR CLUSTER NAME
[root@workstation myobjects]# export MIN=2
[root@workstation myobjects]# export MAX=4
[root@workstation myobjects]# export DESIRED=3
[root@workstation myobjects]#
[root@workstation myobjects]# export ACCOUNT_ID=439292780570 # Give your account ID
[root@workstation myobjects]#
```

aws autoscaling \
describe-auto-scaling-groups \

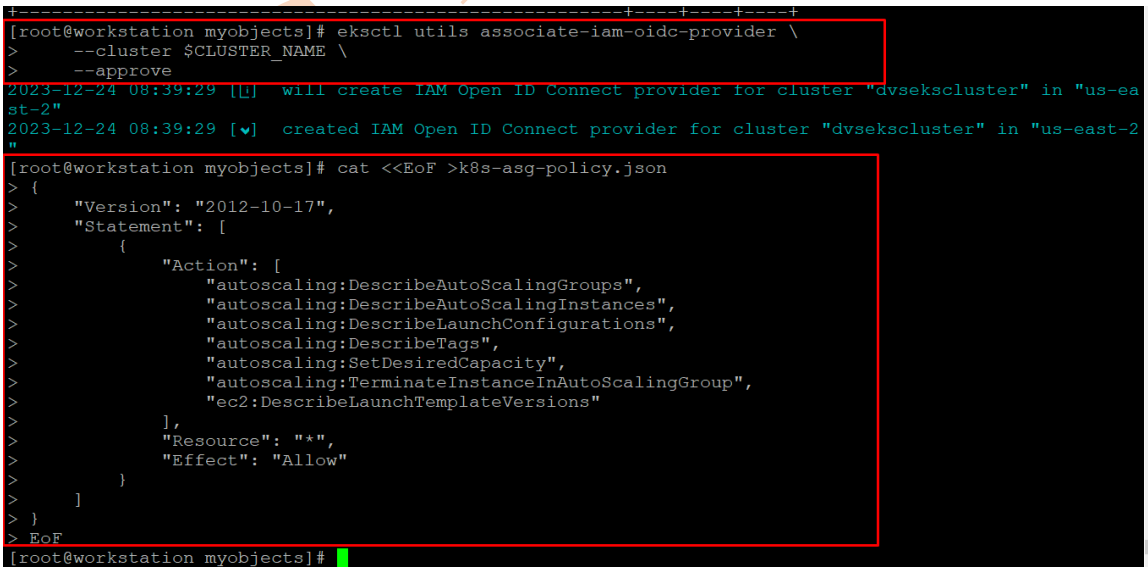
```
   --query "AutoScalingGroups[? Tags[? (Key=='eks:cluster-name') &&
Value=='$CLUSTER_NAME']].[AutoScalingGroupName, MinSize,
MaxSize,DesiredCapacity]" \
   --output table
export ASG_NAME=$(aws autoscaling describe-auto-scaling-groups --query
"AutoScalingGroups[? Tags[? (Key=='eks:cluster-name') &&
Value=='$CLUSTER_NAME']].AutoScalingGroupName" --output text)

# increase max capacity up to 4
aws autoscaling \
   update-auto-scaling-group \
   --auto-scaling-group-name ${ASG_NAME} \
   --min-size $MIN \
   --desired-capacity $DESIRED \
   --max-size $MAX

# Check new values
aws autoscaling \
   describe-auto-scaling-groups \
   --query "AutoScalingGroups[? Tags[? (Key=='eks:cluster-name') &&
Value=='$CLUSTER_NAME']].[AutoScalingGroupName, MinSize,
MaxSize,DesiredCapacity]" \
   --output table
```



Enabling IAM roles for service accounts on your cluster
eksctl utils associate-iam-oidc-provider \
  --cluster $CLUSTER_NAME \
  --approve

Creating an IAM policy for your service account that will allow your CA pod to interact with the autoscaling groups.

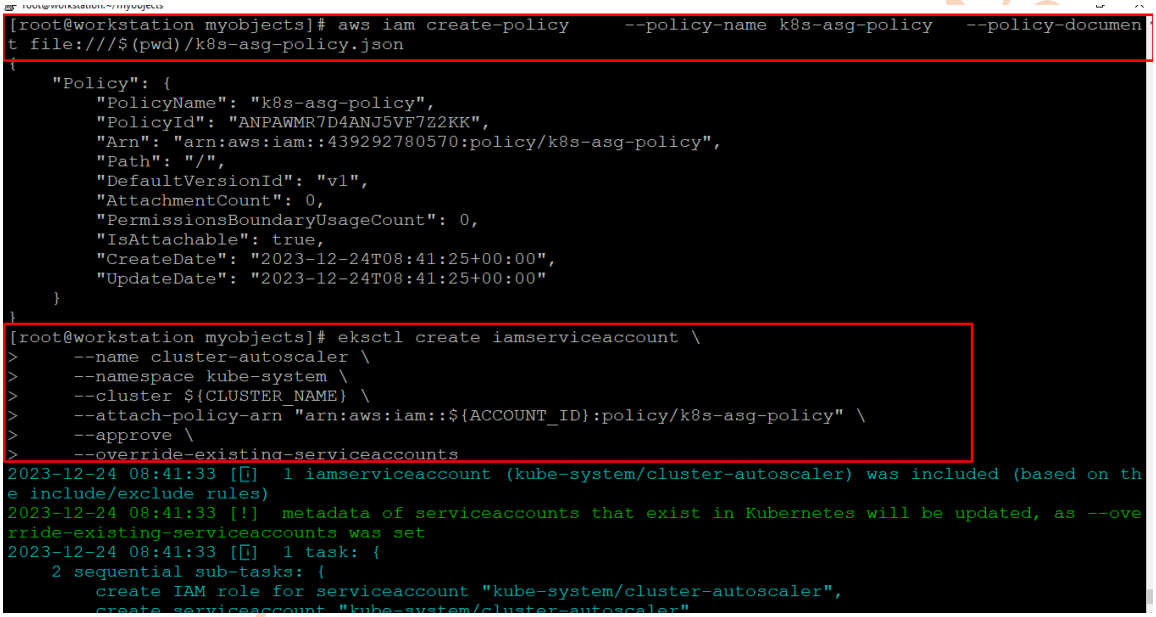
```
cat <<EoF >k8s-asg-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeTags",
        "autoscaling:SetDesiredCapacity",
        "autoscaling:TerminateInstanceInAutoScalingGroup",
        "ec2:DescribeLaunchTemplateVersions"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
EoF
```



aws iam create-policy --policy-name k8s-asg-policy --policy-document file:///$(pwd)/k8s-asg-policy.json

eksctl create iamserviceaccount \
--name cluster-autoscaler \
--namespace kube-system \
--cluster ${CLUSTER_NAME} \
--attach-policy-arn "arn:aws:iam::${ACCOUNT_ID}:policy/k8s-asg-policy" \
--approve \
--override-existing-serviceaccounts

kubectl -n kube-system describe sa cluster-autoscaler



kubectl apply -f https://www.eksworkshop.com/beginner/080_scaling/deploy_ca.files/cluster-autoscaler-autodiscover.yaml

export AUTOSCALER_VERSION="1.21.2"
kubectl -n kube-system \
set image deployment.apps/cluster-autoscaler \
cluster-autoscaler=us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-autoscaler:v${AUTOSCALER_VERSION}

```
[root@workstation myobjects]# kubectl apply -f https://www.eksworkshop.com/beginner/080_scaling/deploy_
ca.files/cluster-autoscaler-autodiscover.yaml

export AUTOSCALER_VERSION="1.21.2"
kubectl -n kube-system \
    set image deployment.apps/cluster-autoscaler \
    cluster-autoscaler=us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-autoscaler:v${AUTOSCALER_VERSIO
N}
clusterrole.rbac.authorization.k8s.io/cluster-autoscaler created
role.rbac.authorization.k8s.io/cluster-autoscaler created
clusterrolebinding.rbac.authorization.k8s.io/cluster-autoscaler created
rolebinding.rbac.authorization.k8s.io/cluster-autoscaler created
deployment.apps/cluster-autoscaler created
[root@workstation myobjects]#
[root@workstation myobjects]# export AUTOSCALER_VERSION="1.21.2"
[root@workstation myobjects]# kubectl -n kube-system \
>     set image deployment.apps/cluster-autoscaler \
>     cluster-autoscaler=us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-autoscaler:v${AUTOSCALER_VERS
ION}
deployment.apps/cluster-autoscaler image updated
[root@workstation myobjects]#
```

Make sure that you are changing the deployment & update the nodegroup vaules
kubectl edit deploy cluster-autoscaler -n kube-system

Edit the below section in the deployment
- --node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/dvsbatch3-eks-2{{ update this one as per your cluster name }}

Before:

```
deployment.apps/cluster-autoscaler image updated
[root@workstation myobjects]# kubectl edit deploy cluster-autoscaler -n kube-system
        prometheus.io/port: "8085"
        prometheus.io/scrape: "true"
      creationTimestamp: null
      labels:
        app: cluster-autoscaler
    spec:
      containers:
      - command:
        - ./cluster-autoscaler
        - --v=4
        - --stderrthreshold=info
        - --cloud-provider=aws
        - --skip-nodes-with-local-storage=false
        - --expander=least-waste
        - --node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscal
er/eksworkshop-eksctl
        - --balance-similar-node-groups
        - --skip-nodes-with-system-pods=false
        image: us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-autoscaler:v1.21.2
        imagePullPolicy: Always
        name: cluster-autoscaler
```

After:

```
[root@workstation myobjects]# kubectl edit deploy cluster-autoscaler -n kube-system
        prometheus.io/port: "8085"
        prometheus.io/scrape: "true"
      creationTimestamp: null
      labels:
        app: cluster-autoscaler
    spec:
      containers:
      - command:
        - ./cluster-autoscaler
        - --v=4
        - --stderrthreshold=info
        - --cloud-provider=aws
        - --skip-nodes-with-local-storage=false
        - --expander=least-waste
        - --node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscal
er/dvseksclusters
        - --balance-similar-node-groups
        - --skip-nodes-with-system-pods=false
        image: us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-autoscaler:v1.21.2
        imagePullPolicy: Always
        name: cluster-autoscaler
        resources:
          limits:
            cpu: 100m
            memory: 500Mi
          requests:
            cpu: 100m
```

Now we are done with our changes for creating cluster auto scaler

Let's test the setup.

```
[root@workstation myobjects]# kubectl get nodes
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-118.us-east-2.compute.internal   Ready    <none>   83m     v1.27.7-eks-e71965b
ip-192-168-100-157.us-east-2.compute.internal   Ready    <none>   12m     v1.27.7-eks-e71965b
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   7h20m   v1.27.7-eks-e71965b
[root@workstation myobjects]# vi deploy.yaml
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 100
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
```

As of now we have 3 nodes and we are increasing our deployment values to 100 which means we are increasing load to our infrastructure as part of the cluster autoscaller it should scale the servers to max value in our case its 4

apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication

```
spec:
  replicas: 1000
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

```
[root@workstation myobjects]# cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: mydeployment
  name: mydeployment
  namespace: myapplication
spec:
  replicas: 1000
  selector:
    matchLabels:
      app: mydeployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mydeployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
[root@workstation myobjects]# kubectl apply -f deploy.yaml
deployment.apps/mydeployment configured
[root@workstation myobjects]#
```

```
^[[B^C[root@workstation myobjects]# kubectl get nodes
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-100-105.us-east-2.compute.internal   Ready    <none>   11m     v1.27.7-eks-e71965b
ip-192-168-100-118.us-east-2.compute.internal   Ready    <none>   112m    v1.27.7-eks-e71965b
ip-192-168-100-157.us-east-2.compute.internal   Ready    <none>   40m     v1.27.7-eks-e71965b
ip-192-168-100-171.us-east-2.compute.internal   Ready    <none>   7h49m   v1.27.7-eks-e71965b
[root@workstation myobjects]#
```

Hence we can say that cluster auto scaler is working fine.

11. Ingress

https://kubernetes.github.io/ingress-nginx/deploy/#azure

Installing Ingress: (Manually)

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.2/deploy/static/provider/cloud/deploy.yaml

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.3/deploy/static/provider/cloud/deploy.yaml

Make sure that you have below softwares in the workstation

Installing helm:

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

Installing Ingress using helm:

https://docs.microsoft.com/en-us/azure/aks/ingress-basic?tabs=azure-cli

NAMESPACE=ingress-basic

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update

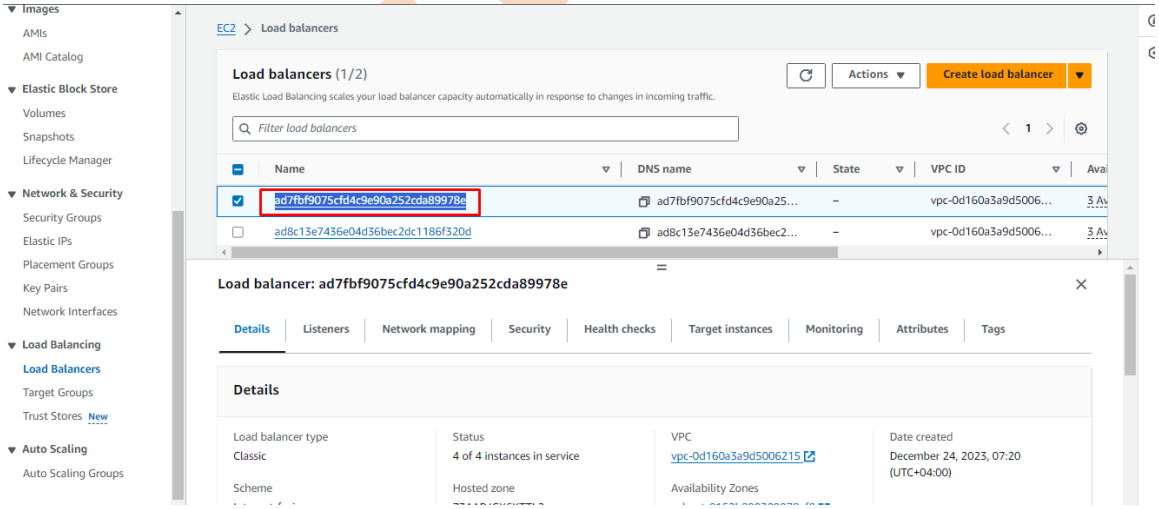
helm install ingress-nginx ingress-nginx/ingress-nginx --create-namespace --namespace $NAMESPACE

```
[root@workstation ~]# NAMESPACE=ingress-basic
[root@workstation ~]#
[root@workstation ~]# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
"ingress-nginx" has been added to your repositories
[root@workstation ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
Update Complete. ⎈Happy Helming!⎈
[root@workstation ~]#
```

```
Update Complete. ⎈Happy Helming!⎈
[root@workstation ~]# helm install ingress-nginx ingress-nginx/ingress-nginx --create-namespace --namespace $NAMESPACE
NAME: ingress-nginx
LAST DEPLOYED: Mon Apr 11 06:47:42 2022
NAMESPACE: ingress-basic
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The ingress-nginx controller has been installed.
It may take a few minutes for the LoadBalancer IP to be available.
You can watch the status by running 'kubectl --namespace ingress-basic get services -o wide -w ingress-nginx-controller'

An example Ingress that makes use of the controller:
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: example
    namespace: foo
  spec:
    ingressClassName: nginx
```

```
replicaset.apps/ingress-nginx-controller-6944641741   1         1         1       6h15m
[root@workstation myobjects]# kubectl get svc -n ingress-basic
NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP
                                      PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.100.144.246   ad7fbf9075cfd4c9e90a252cda89978e-5
4684905.us-east-2.elb.amazonaws.com   80:30929/TCP,443:31381/TCP   6h16m
ingress-nginx-controller-admission   ClusterIP      10.100.149.75    <none>
                                      443/TCP                      6h16m
[root@workstation myobjects]#
```

In the aws console:



Let's configure the application with ingress and test the routing:

Create two deployments:

Create two images & push to dokcerhub:

App1:

Dockerfile

FROM httpd
RUN echo "<h1>Hi Team welcome to deployment1-app1</h1>" > /usr/local/apache2/htdocs/index.html

docker build -t "shan5a6/deployment1-app1:v1.0" .

```
[root@workstation myobjects]# cat Dockerfile
FROM httpd
RUN echo "<h1>Hi Team welcome to deployment1-app1</h1>" > /usr/local/apache2/htdocs/index.html
[root@workstation myobjects]# docker build -t "shan5a6/deployment1-app1:v1.0" .
Sending build context to Docker daemon  9.216kB
Step 1/2 : FROM httpd
latest: Pulling from library/httpd
af107e978371: Pull complete
eba4da411ea0: Pull complete
4f4fb700ef54: Pull complete
ed4d6291a6c2: Pull complete
b42c390e1de9: Pull complete
eafe388a0bb8: Pull complete
Digest: sha256:f0a93744d8006e6f7ee5086c9ddccdcfa33d1091f15269a00547b4c382459c1f
Status: Downloaded newer image for httpd:latest
 ---> 6fd77d7e5eb7
Step 2/2 : RUN echo "<h1>Hi Team welcome to deployment1-app1</h1>" > /usr/local/apache2/htdocs/index.h
ml
 ---> Running in 2ec8c7983979
Removing intermediate container 2ec8c7983979
 ---> 2b4b0bd79968
Successfully built 2b4b0bd79968
```

```
[root@workstation myobjects]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head
over to https://hub.docker.com to create one.
Username: shan5a6
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@workstation myobjects]# docker push shan5a6/deployment1-app1:v1.0
The push refers to repository [docker.io/shan5a6/deployment1-app1]
9af29645bb07: Pushed
8cd4026118f7: Mounted from library/httpd
8c10599774e2: Mounted from library/httpd
87939ee964f4: Mounted from library/httpd
5f70bf18a086: Mounted from library/httpd
1b65777f1238: Mounted from library/httpd
7292cf786aa8: Mounted from library/httpd
v1.0: digest: sha256:4340efe6d3c2f0befbc58a42bdca7df92275eb485e6dc2623b057fa116c1ad40 size: 1779
[root@workstation myobjects]# vi deploy.yaml
```

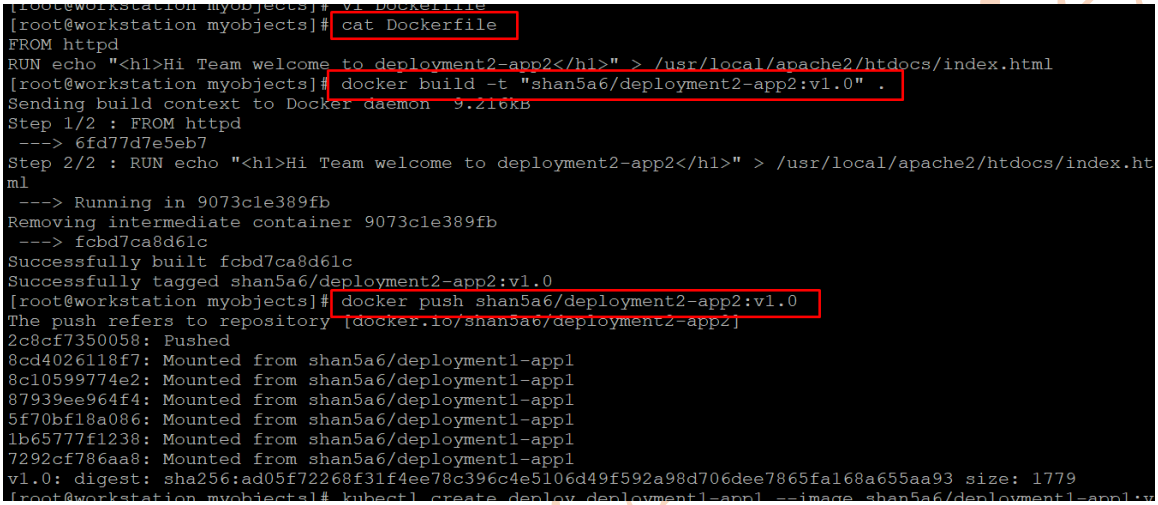
Do the same for App2:

App2:

Dockerfile:

FROM httpd
RUN echo "<h1>Hi Team welcome to deployment2-app2</h1>" > /usr/local/apache2/htdocs/index.html

docker build -t "shan5a6/deployment2-app2:v1.0" .



Create the deployments along with services in javaapp namespace:

kubectl create ns javaapp

App2:

Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: app2-deploy
  name: app2-deploy
  namespace: javaapp
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      app: app2-deploy
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: app2-deploy
    spec:
      containers:
      - image: shan5a6/deployment2-app2:v1.0
        name: app2
        resources: {}
status: {}
```

Service:

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: app2-deploy
  name: app2-deploy
  namespace: javaapp
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: app2-deploy
status:
  loadBalancer: {}
```

Same goes with APP1:

Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: app1-deploy
```

```
  name: app1-deploy
  namespace: javaapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app1-deploy
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: app1-deploy
    spec:
      containers:
      - image: shan5a6/deployment1-app1:v1.01
        name: app1
        resources: {}
status: {}
```

Service:

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: app1-deploy
  name: app1-deploy
  namespace: javaapp
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: app1-deploy
status:
  loadBalancer: {}
```

Let's configure the ingress rules:

[root@workstation ~]# cat app-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: application-access
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /app1
        pathType: Prefix
        backend:
          service:
            name: app1-deploy
            port:
              number: 80
      - path: /app2
        pathType: Prefix
        backend:
          service:
            name: app2-deploy
            port:
              number: 80
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app1-deploy
            port:
              number: 80
```

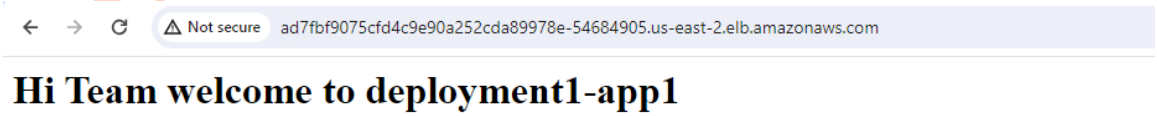
```
[root@workstation myobjects]# cat ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: application-access
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /app1
        pathType: Prefix
        backend:
          service:
            name: deployment1-app1-svc1
            port:
              number: 80
      - path: /app2
        pathType: Prefix
        backend:
          service:
            name: deployment2-app2-svc2
            port:
              number: 80
      - path: /
        pathType: Prefix
        backend:
```

```
[root@workstation myobjects]# kubectl delete -A ValidatingWebhookConfiguration ingress-nginx-admission
validatingwebhookconfiguration.admissionregistration.k8s.io "ingress-nginx-admission" deleted
[root@workstation myobjects]#
[root@workstation myobjects]#
[root@workstation myobjects]# kubectl apply -f ingress.yaml -n javaapp
ingress.networking.k8s.io/application-access created
[root@workstation myobjects]#
[root@workstation myobjects]#
[root@workstation myobjects]# kubectl get ingress -n javaapp
NAME                CLASS   HOSTS   ADDRESS
     PORTS   AGE
application-access   nginx   *       ad7fbf9075cfd4c9e90a252cda89978e-54684905.us-east-2.elb.amazonaws.
com   80      19s
```

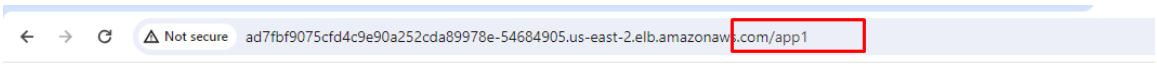
Execute the below if you are facing any issues during ingress definition creation.

kubectl delete -A ValidatingWebhookConfiguration ingress-nginx-admission

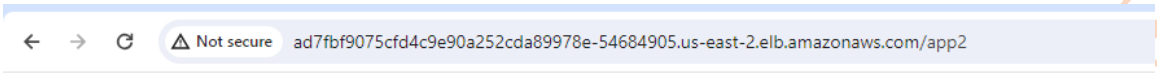
Final verification:



Hi Team welcome to deployment1-app1



Hi Team welcome to deployment1-app1



Hi Team welcome to deployment2-app2