Kubernetes:
For orchestration, always Kubernetes is better than Docker swarm

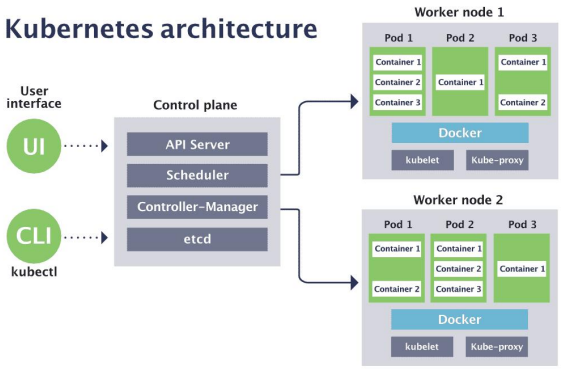
It is free and open-source --> Developed by Google --> GO programming language is used to develop Kubernetes.
Kubernetes is an Orchestration platform --> Used to manage containers (create, start, stop, delete, scale-up, scale-down containers)
It provides framework for managing the complex task of deploying, scaling and operating applications in containers

Advantages:
1. Self-healing: if any container gets crashed, it will be automatically replaced with a new container immediately
2. Auto-scaling: Based on demand, containers count will be increased or decreased
3. Load-balancing: Load will be distributed to all containers equally, which are up and running

Docker vs Kubernetes:
What's the purpose of Docker?
It is for containerization, to containerize the application. Containerization platform. It is for packaging our application code and dependencies as a single unit for the execution is referred as Containerization.
What's the significance of Kubernetes?
It is an orchestration platform. It is for the orchestration purpose. Managing the containers that got created.
https://kubernetes.io/docs/concepts/architecture/

Kubernetes Architecture
--> K8s follows cluster architecture
--> Cluster refers to group of servers (machines, VMs)
--> in K8s cluster, we will have a Control node (Master node) and Worker nodes

K8s Cluster Components:
1. Control Node (Master Node)
   a) In the control node, we have something called as an API server
   b) Scheduler
   c) Controller-Manager
   d) etcd

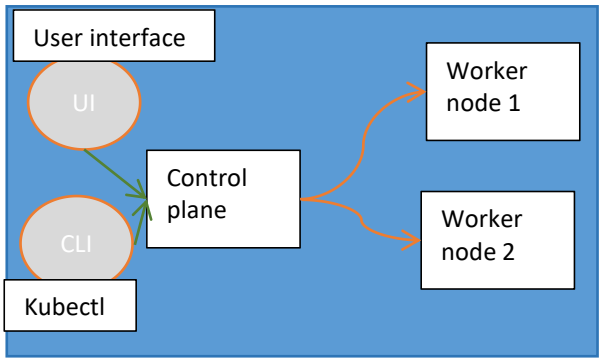


2. Worker Node
   a) Kubelet
   b) Kube proxy
   c) Docker engine
   d) POD

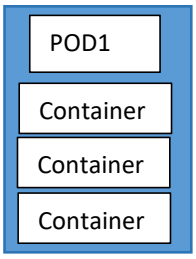
e) Within the POD, we have the Container

Kubernetes Cluster Architecture

Two important parts in Kubernetes Architecture



In Control plane we have 4 components: API server, Scheduler, Controller-manager, etcd
In every Worker node, we have kubelet, kube-proxy, docker, pods (Pod1 will have many containers)



--> To deploy our application using K8s we need to communicate with Control plane (Master node)
--> We usually use KUBECTL (CLI) to communicate with Control plane
--> API server will receive the request given by kubectl and it will store the request with pending status in ETCD
--> ETCD is an internal database of k8s cluster.
--> any pending requests in ETCD will be identified by Scheduler then will schedule tasks in Worker node. Scheduler will identify the Worker node to schedule this pending request with the help of Kubelet. Kubelet is a Node agent, it will maintain information about all Worker node.
Scheduler will go to ETCD, identify pending requests, then it will schedule tasks by identifying the Worker node. Kube proxy provides network for Cluster communication. Controller-manager is used to verify all the tasks are working as per expectations or not. Docker engine will be present in the Worker node. In K8s architecture, will container be directly created under worker node? Containers will be created inside the Pod. All containers will be there inside the Pod only.
--> Scheduler will identify the pending request in ETCD and it will identify Worker node to schedule the task
--> Scheduler will identify Worker node using Kubelet
--> Kubelet is a Node agent, which will maintain all the worker node information
--> Kube proxy will provide network for Cluster communication
--> Controller-manager will verify all the tasks, which have been assigned are working fine as expected or not
--> Docker engine would be present in the Worker node to run Docker container
--> In K8s, Containers will be created inside POD --> POD is the smallest building block that we could create in a K8s cluster
--> Generally in K8s, everything is represented as POD only
--> Note: we don't directly work with containers they stay within Pods

POD:
POD is the smallest building block in the K8s cluster and applications will be deployed as a Pod in K8s.
We can create multiple Pods for one application.
In order to create a Pod, we use a Yaml file (Manifest YML) and in Pod manifest YML we will configure our Docker image
If a Pod is damaged/deleted/crashed, then K8s will create a new Pod (Self-healing).
If an application is running in multiple Pods then K8s will distribute the load to all the running Pods.
This is the concept of Load balancers.
Pods could be increased or decreased automatically based on load (Scalability)

K8s Cluster Setup:
1. Mini Kube --> Single node cluster --> Only for practice
2. Kubeadm cluster --> Self-managed cluster (everything is managed by us only). we are responsible for everything. We are going to create machines, control node etc
3. Provider Managed Cluster --> Ready made cluster --> Provider will take care of everything.
Examples: AWS EKS, Azure, AKS, GCP GKE etc.
Note: Provider-managed clusters they are paid they are chargeable

Practical steps for Kubernetes cluster setup
Step 1: Create EKS management host in AWS

Launch a Linux machine (Ubuntu VM) using AWS EC2 (t2.micro)
Connect to this machine and install Kubectl
Install Kubectl
curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/

sudo apt update && sudo apt install -y unzip

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

Cleanup
rm -rf awscliv2.zip

Verify installation
aws --version

Install eksctl
curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" -o eksctl.tar.gz
tar -xzf eksctl.tar.gz
sudo mv eksctl /usr/local/bin/
eksctl version

Create a new AWS IAM role (using IAM service, select usecase as EC2) and attach to EC2 host

Add below permissions

AdministratorAccess, AmazonEC2FullAccess, AmazonVPCFullAccess, IAMFullAccess

Enter rolename --> Attach created role to EKS management host VM --> Actions --> Security --> Modify IAM user and add created IAM role
Create EKS cluster using eksctl

```
eksctl create cluster --name my-eks-cluster --region  ca-central-1 --node-type t2.medium --zones ca-central-1a,ca-central-1b

 cat /home/ubuntu/.kube/config

ubuntu@ip-172-31-9-165:~$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
ip-192-168-26-249.ca-central-1.compute.internal   Ready    <none>   74m   v1.32.3-eks-473151a
ip-192-168-44-239.ca-central-1.compute.internal   Ready    <none>   74m   v1.32.3-eks-473151a
```

Create two manifest yml file or both Pod and Service in the same file

```
---
apiVersion: v1
kind: Pod
metadata:
   name: javawebapp
   labels:
      app: javawebapp
spec:
   containers:
      - name: javawebappcontainer
        image: hacker123shiva/springbt-in-docker:latest
        ports:
          - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
   name: javawebappsvc
spec:
   type: LoadBalancer
   selector:
      app: javawebapp
   ports:
      - port: 80
        targetPort: 8080
...
```

```
kubectl apply -f k8s-pod-manifest-new.yml
kubectl get pods
kubectl get svc
```

Check if pods and services are up and running
Once service got created, we can see that in EC2 dashboard as well, Loadbalancer got created

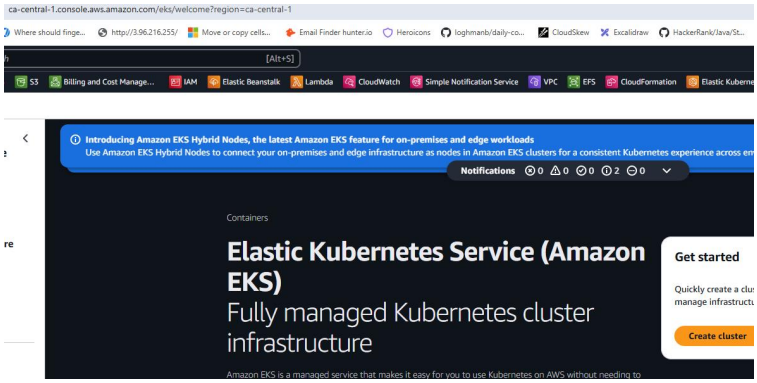
We can access our application using LoadBalancer DNS URL

Delete all resources:
```
kubectl delete all --all
```

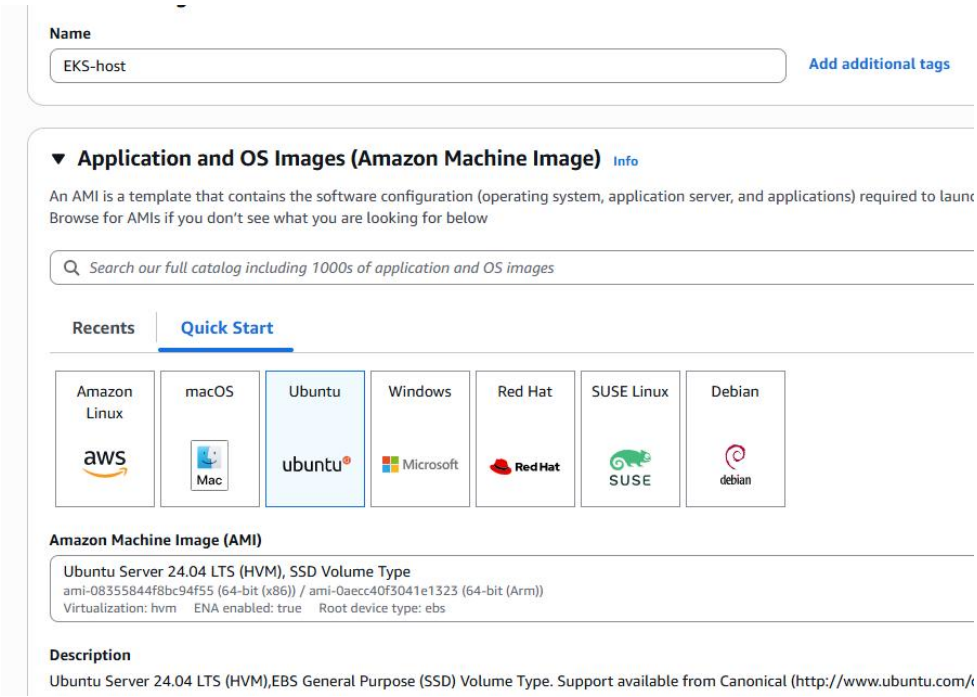
Delete entire cluster:

```
eksctl delete cluster --name my-eks-cluster --region ca-central-1
```

Illustration:



No option to temporarily stop the cluster you have to delete



curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0   2031      0 --:--:-- --:--:-- --:--:--  2029
100 57.3M  100 57.3M    0     0  2259k      0  0:00:25  0:00:25 --:--:-- 2275k
ubuntu@ip-172-31-9-165:~$
```

```
ubuntu@ip-172-31-9-165:~$ kubectl version
Client Version: v1.33.1
Kustomize Version: v5.6.0
The connection to the server localhost:8080 was refused - did you specify the right host or port?
ubuntu@ip-172-31-9-165:~$
```

ubuntu@ip-172-31-9-165:~$ sudo apt update && sudo apt install -y unzip

```
ubuntu@ip-172-31-9-165:~$ sudo apt update && sudo apt install -y unzip
Hit:1 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:5 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:6 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [838 kB]
Get:8 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:9 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:10 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:11 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:12 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
```

ubuntu@ip-172-31-9-165:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 67.2M  100 67.2M    0     0   103M      0 --:--:-- --:--:-- --:--:--  102M
Archive:  awscliv2.zip
   creating: aws/
```

```
ubuntu@ip-172-31-9-165:~$ rm -rf awscliv2.zip
ubuntu@ip-172-31-9-165:~$ ls -l
total 4
drwxr-xr-x 3 ubuntu ubuntu 4096 May 16 18:46 aws
ubuntu@ip-172-31-9-165:~$
```

ubuntu@ip-172-31-9-165:~$ aws --version
aws-cli/2.27.17 Python/3.13.3 Linux/6.8.0-1024-aws exe/x86_64.ubuntu.24

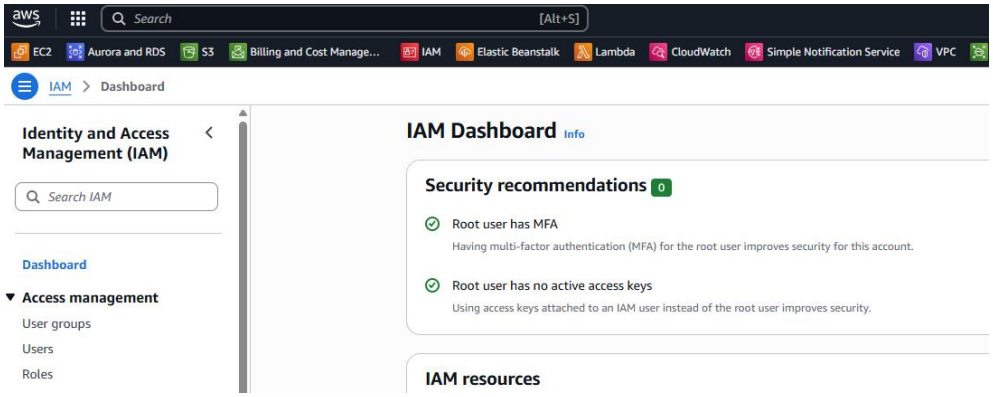
Verify installation

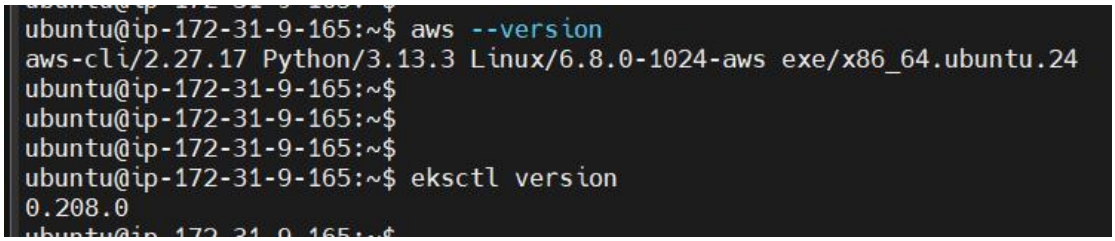
```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ aws --version
aws-cli/2.27.17 Python/3.13.3 Linux/6.8.0-1024-aws exe/x86_64.ubuntu.24
ubuntu@ip-172-31-9-165:~$
```

Install eksctl
ubuntu@ip-172-31-9-165:~$ curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" -o eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ tar -xzf eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ sudo mv eksctl /usr/local/bin/
ubuntu@ip-172-31-9-165:~$ eksctl version
0.208.0

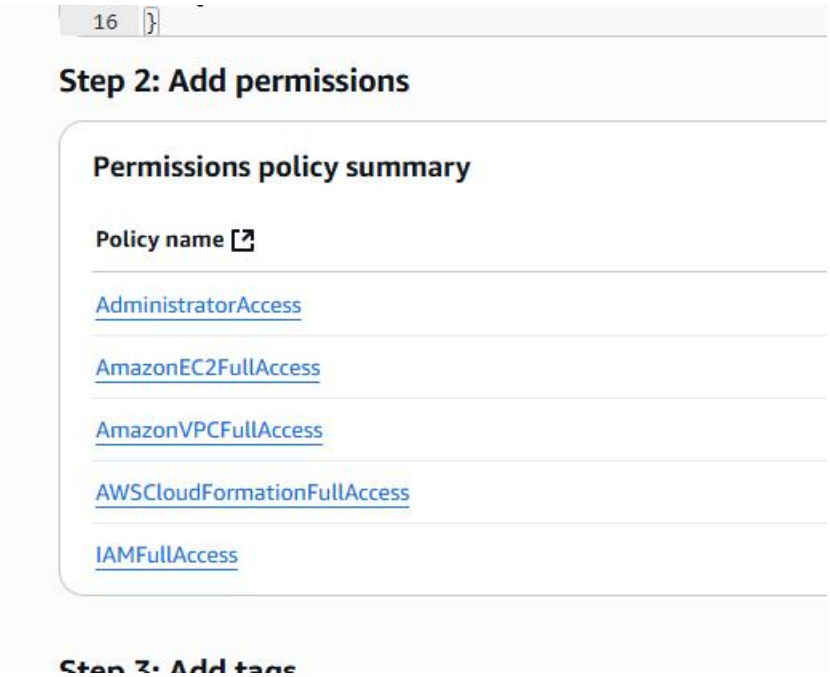
```
ubuntu@ip-172-31-9-165:~$ curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" -o eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ tar -xzf eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ sudo mv eksctl /usr/local/bin/
ubuntu@ip-172-31-9-165:~$ eksctl version
0.208.0
```

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ curl --silent --location "https://github.com/eksctl-io/eksctl/releases/late
r.gz
ubuntu@ip-172-31-9-165:~$ tar -xzf eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ sudo mv eksctl /usr/local/bin/
ubuntu@ip-172-31-9-165:~$ eksctl version
0.208.0
```

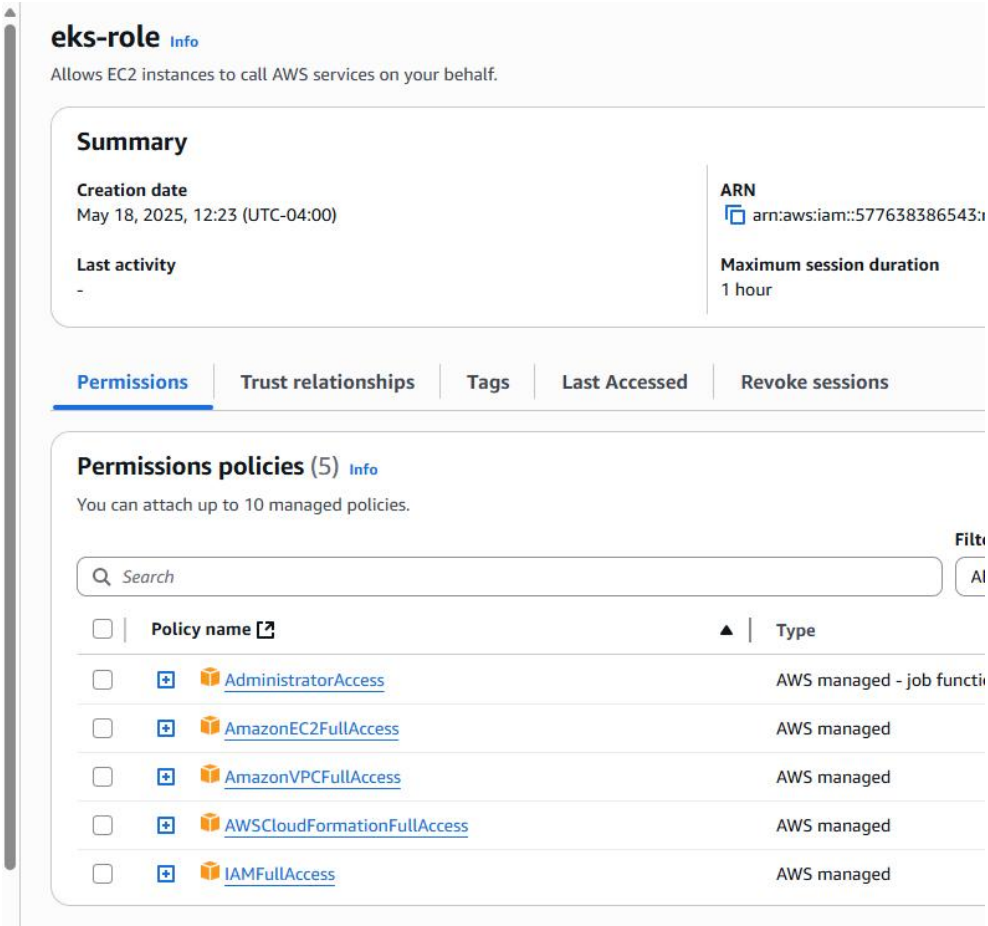




Click Create Role
Add IAMFullAccess, AmazonEC2FullAccess, AmazonVPCFullAccess, CloudFormationFullAccess, AdministratorAccess

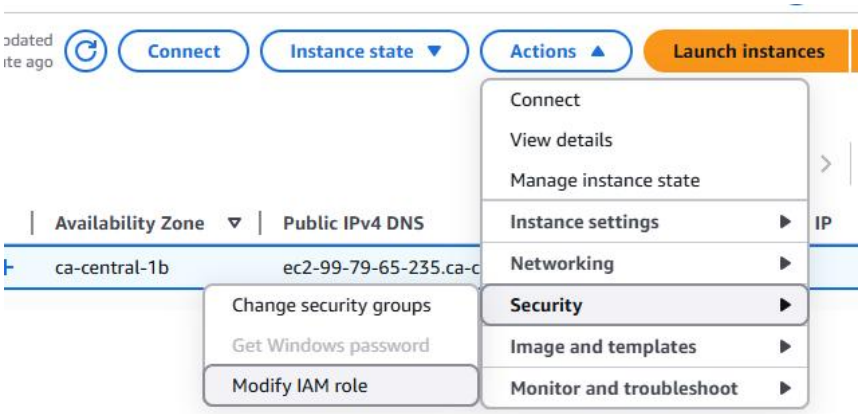


Click Create Role

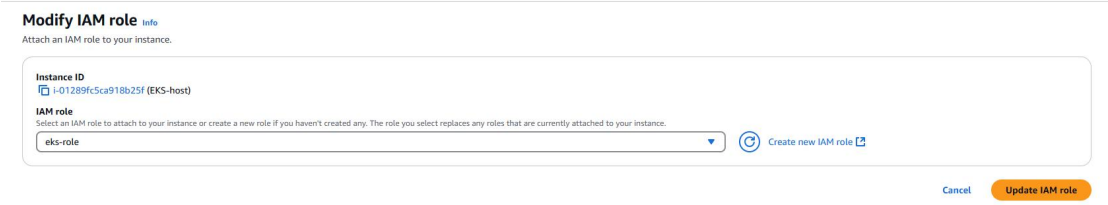


Go back to EC2:

Click on instance
Security --> Modify IAM role

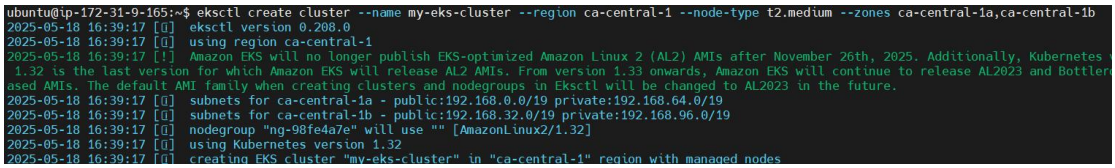


Select eks-role

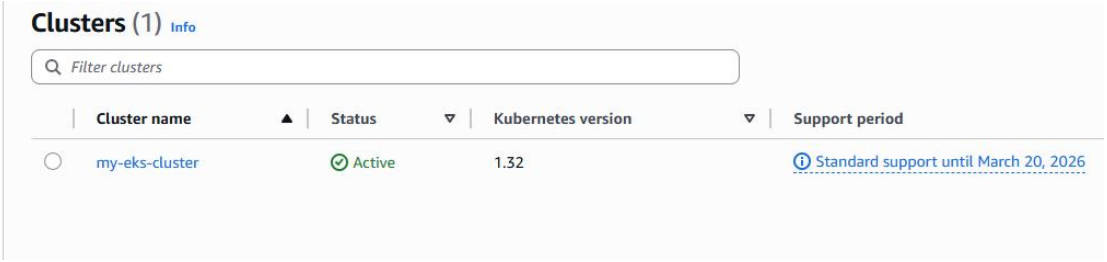


Update IAM role

Go to EC2 and run the command
ubuntu@ip-172-31-9-165:~$ eksctl create cluster --name my-eks-cluster --region ca-central-1 --node-type t2.medium --zones ca-central-1a,ca-central-1b



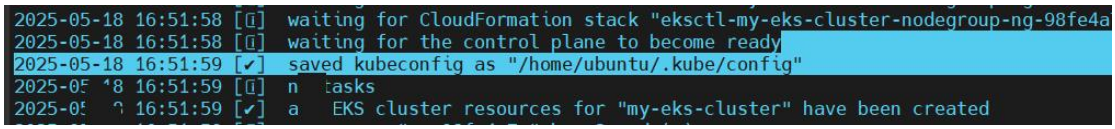
Cluster is created



AWS-managed control plane

Kube config is stored in this location: this is the most important file

2025-05-18 16:51:59 [✔] saved kubeconfig as "/home/ubuntu/.kube/config"



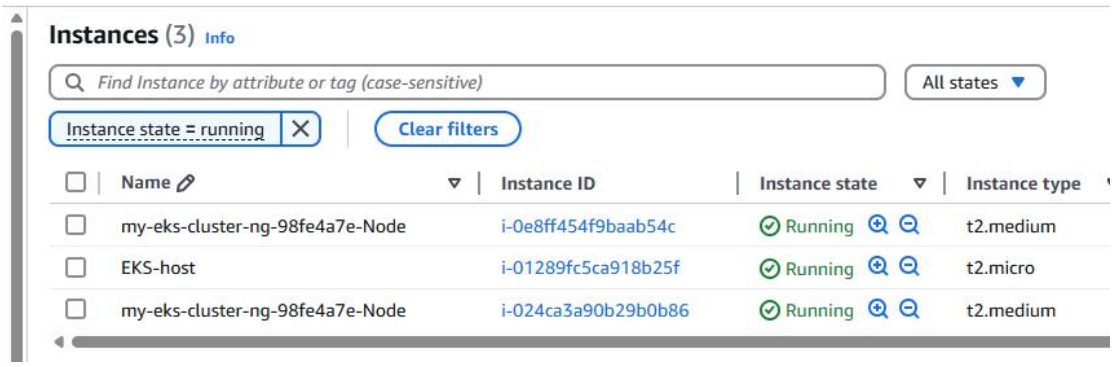
ubuntu@ip-172-31-9-165:~$ cat /home/ubuntu/.kube/config

```
ubuntu@ip-172-31-9-165:~$ cat /home/ubuntu/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tC
hNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TlRBMU1UZ3hOak01TURCYUZ3MHpOVEExTVRZeE5
QkR3QXdnZ0VLCkFvSUJBUURtUkVxSjQrbXRDM0NGVnBWVWF6QlF2RkxnY0c0TFZjUFdsW
8xdUFPT3lzT21vbENGVmlabnVQWApLbU9FSHhGOUJhelc0dGtGUytxbGd2NG1aWkw4TXF
UXFSOHJMVUNuVy8zTkhsOXYxa1E0VE5tMWduRy9COUwKdVR3YlVnd1ZEdnJEZUZhZzR2e
c3QTBxWWc3QWdNQkFBR2pXVEJYTUE0R0ExVWREd0VCL3dRRUF3SUNwREFQCkJnTlZIUk1
akFNZ2dwcmRXSmxjbTVsZEdWek1BMEdDU3FHU0liM0RRRUJDd1VBQTRJQkFRQzY3YnllM
IzClV0UFVsTWZLdlpYRS96TXVDemRtV0xRdCtpWURGQWgwb3ZQeEpFUG1ycEhESVN3bjF
aDlPWkVlbjRoZGlPOApaQmowU2RNWFZuQmxqaENOdU96aS8wOUJKMXp2bEZJMzhLSHlmK
hVVTVVNmlYSTZ0bzZnNVR1ZCtCb0o5U1AKMXkrWThoZFdTS2JxCi0tLS0tRU5EIENFUlR
    server: https://DDD202684213BD7CD318EEFD6EDBD233.gr7.ca-central-1
  name: my-eks-cluster.ca-central-1.eksctl.io
contexts:
```

We created two worker nodes

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl get nodes
NAME                                                STATUS   ROLES    AGE   VERSION
ip-192-168-26-249.ca-central-1.compute.internal     Ready    <none>   74m   v1.32.3-eks-473151a
ip-192-168-44-239.ca-central-1.compute.internal     Ready    <none>   74m   v1.32.3-eks-473151a
ubuntu@ip-172-31-9-165:~$
```

Go to EC2 instances
We have two worker nodes: node1 and node2, automatically created along with EKS-host. These two worker EC2s are managed by Control plane, not by us



Also check on AWS console that cluster and also two new instance worker nodes would be created

ubuntu@ip-172-31-9-165:~$ kubectl get pods
No resources found in default namespace.

Every-time we create a pod IP address will be available

The concept we got to learn is Kubernetes services

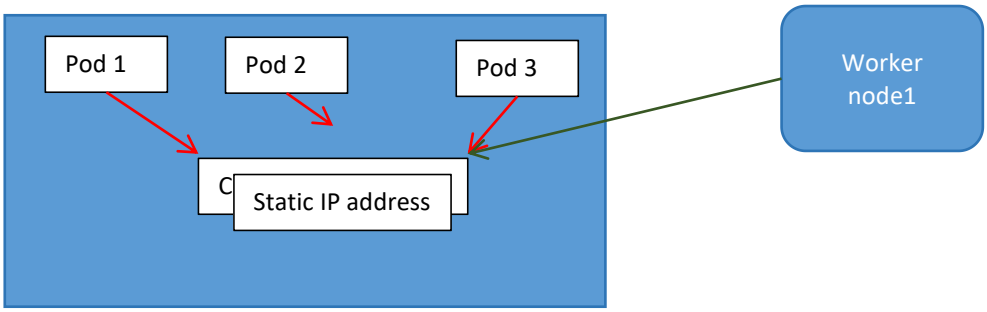
K8S Services:

Service is used to expose PODS --> It is used to expose pods

We have 3 types of services in K8s --> Cluster IP, Node port, Load balancer
Cluster IP

POD is a short-lived object.
If POD is damaged/deleted/crashed then k8s will replace that with a new pod (self-healing)

If POD is destroyed, the IP address with that POD will also be destroyed
When POD is re-created IP will be changed (it is not recommended to access PODs using POD IP)
Cluster IP service is used to link all PODs in single IP (Static IP is fixed)
Cluster IP is a static IP to access the pods
that's where K8s services concept comes into picture



Now it doesn't matter how many times PODs get re-created, they will all be inside Cluster IP service that will have a Static IP address. We are not trying to access those PODs with the IP of those PODs
Using Cluster IP we can access Pods only WITHIN the cluster

Node Port service:
It is used to expose our pods outside the cluster. If you want to expose pods OUTSIDE cluster, then NodePort service is used. If you want to expose pods INSIDE cluster, then ClusterIP service is used
Using NodePort (WorkerNodePort) we can access our app with worker node public IP address. With worker node public IP address we can access only one specific Worker node not all worker nodes. But burden will be increased on a single Worker node so it is not recommended.
--> When we use Worker Node public IP to access our POD then all the requests will go to the same worker node (Burden will be increased on the node).
--> To distribute load to multiple worker nodes we will use Load balancer service

Load balancer service:
Out of three services, to access our application, which service we are going to use?
--> It is used to expose our pods outside cluster using AWS LoadBalancer
--> When we access load balancer URL, requests will be distributed to all the PODs running in all the worker nodes, regardless of how many worker nodes are there

K8S Namespaces:
Grouping all the resources is called as Namespaces in K8s
--> we can group all Frontend ports with all ports of Frontend application, similarly all Backend ports with all ports of Backend application, all database ports together
--> Namespaces are used to group the resources
All Frontend-application-pods --> Frontend-app namespace
All Backend-application-pods --> Backend-app namespace
All Database pods --> database-namespace (one group)

In K8s we use manifest yaml to deploy our application
K8S Manifest YML syntax:
Starts with --- and ends with …

```
---
apiVersion: <version-number>
kind: <resource-type>
```

```
metadata: <name>
spec: <container-info>
…
```

apiVersion: version of the resource you want to create (POD, Service, Namespace etc)
kind: what's the resource you are creating
metadata: more data about what resources you are creating
spec: all the container info will be available in specification

kubectl apply -f <manifest.yml> (to execute Kubernetes manifest yml file)

K8S POD maifest YML

```
---
apiVersion: v1
kind: POD
metadata:
        name: javawebapp
        labels:
          app: javawebapp
spec:
  containers:
     - name: javawebappcontainer
       Image: edydockers/sms-frontend:dev-31
       ports:
          - containerPort: 8080
…
```

```
ubuntu@ip-172-31-9-165:~$ cat k8s-pod-manifest.yml
---
apiVersion: v1
kind: POD
metadata:
   name: javawebapp
   labels:
      app: javawebapp
spec:
   containers:
      - name: javawebappcontainer
        image: edydockers/sms-frontend:dev-31
        ports:
          - containerPort: 8080
…

ubuntu@ip-172-31-9-165:~$ ls -l
total 34148
drwxr-xr-x 3 ubuntu ubuntu     4096 May 16 18:46 aws
-rw-rw-r-- 1 ubuntu ubuntu 34958926 May 17 23:42 eksctl.tar.gz
-rw-rw-r-- 1 ubuntu ubuntu      260 May 18 21:19 k8s-pod-manifest.yml


ubuntu@ip-172-31-9-165:~$ cat k8s-pod-manifest.yml
---
apiVersion: v1
kind: POD
metadata:
```

```
  name: javawebapp
  labels:
    app: javawebapp
spec:
  containers:
    - name: javawebappcontainer
      image: edydockers/sms-frontend:dev-31
      ports:
        - containerPort: 80
…
```

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-manifest.yml

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-manifest.yml
pod/javawebapp created

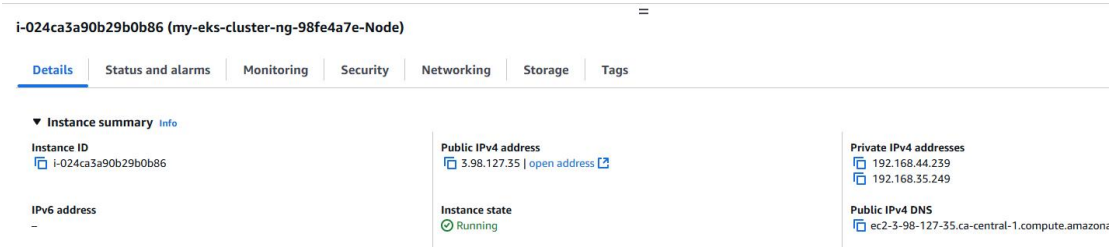
ubuntu@ip-172-31-9-165:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
javawebapp 1/1 Running 0 3m28s

```
ubuntu@ip-172-31-9-165:~$ vi k8s-pod-manifest.yml
ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-manifest.yml
pod/javawebapp created
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
javawebapp    1/1     Running   0          3m28s
ubuntu@ip-172-31-9-165:~$
```

How to get which worker node this pod is deployed?
ubuntu@ip-172-31-9-165:~$ kubectl get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE
NOMINATED NODE READINESS GATES
javawebapp 1/1 Running 0 4m28s 192.168.42.98 ip-192-168-44-239.ca-central-1.compute.internal <none> <none>

```
ubuntu@ip-172-31-9-165:~$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP              NODE
javawebapp    1/1     Running   0          4m28s   192.168.42.98   ip-192-168-44-239.ca-central-1.compute.internal
ubuntu@ip-172-31-9-165:~$
```

that's the EC2



ubuntu@ip-172-31-9-165:~$ kubectl describe pod javawebapp

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl describe pod javawebapp
Name:             javawebapp
Namespace:        default
Priority:         0
Service Account:  default
Node:             ip-192-168-44-239.ca-central-1.compute.internal/192.168.44.239
Start Time:       Sun, 18 May 2025 21:29:51 +0000
Labels:           app=javawebapp
Annotations:      <none>
Status:           Running
IP:               192.168.42.98
IPs:
  IP:  192.168.42.98
Containers:
```

ubuntu@ip-172-31-9-165:~$ kubectl logs javawebapp

```
ubuntu@ip-172-31-9-165:~$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
javawebapp   1/1     Running   0          12m
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl get service
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   4h58m
ubuntu@ip-172-31-9-165:~$
```

ubuntu@ip-172-31-9-165:~$ kubectl get service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 4h58m

K8S Service Manifest YML
--> Service is used to expose pods --> to access application from outside

```
---
apiVersion: v1
kind: Service
metadata:
  name: javawebappsvc
spec:
  type: LoadBalancer
  selector:
    app: javawebapp
      ports:
        - port: 80
          targetPort: 80
…
```

ubuntu@ip-172-31-9-165:~$ cat k8s-service-manifest.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: javawebappsvc
spec:
type: LoadBalancer
selector:
```

```
  app: javawebapp
    ports:
      - port: 80
        targetPort: 80
…
```

```
ubuntu@ip-172-31-9-165:~$ cat k8s-service-manifest.yml
---
apiVersion: v1
kind: Service
metadata:
  name: javawebappsvc
spec:
  type: LoadBalancer
  selector:
    app: javawebapp
  ports:
    - port: 80
      targetPort: 8080
...
```

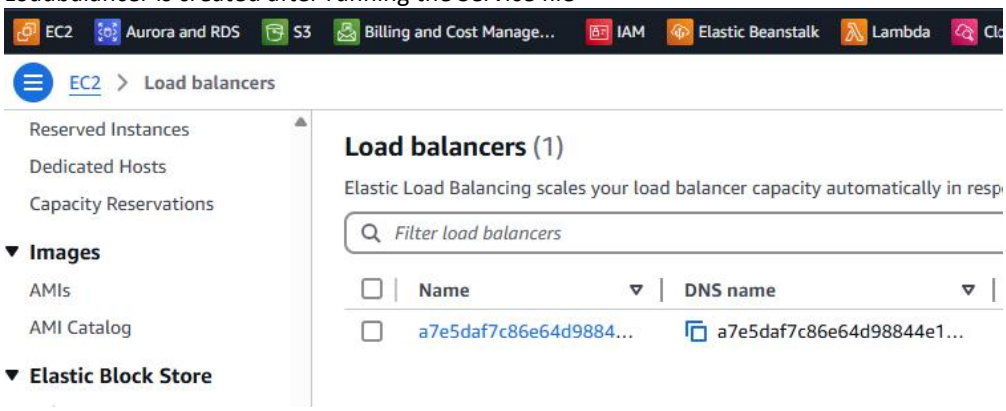
In case of errors while running manifest files: go into vi: type ":set list"

```
ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-service-manifest.yml
service/javawebappsvc created
```

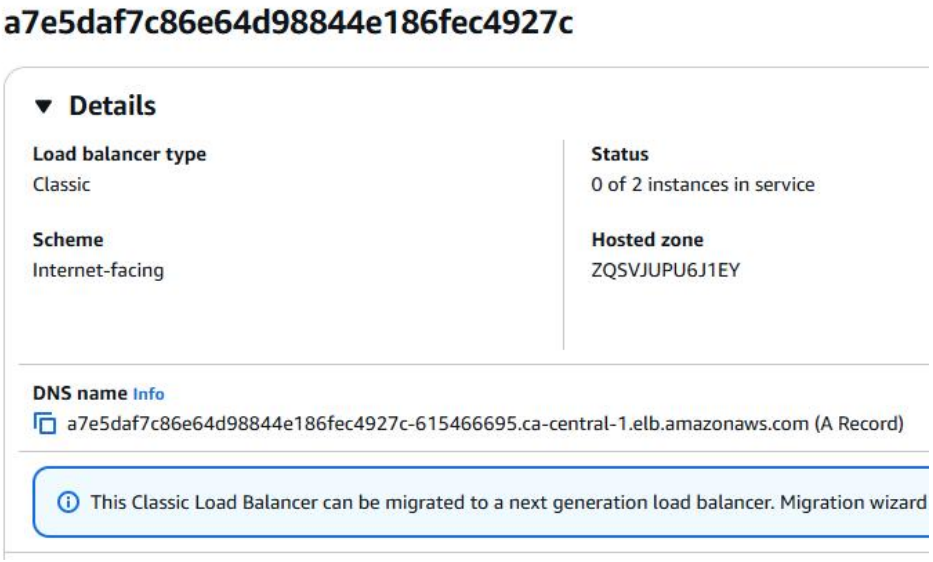
```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ cat k8s-service-manifest.yml
---
apiVersion: v1
kind: Service
metadata:
    name: javawebappsvc
spec:
    type: LoadBalancer
    selector:
        app: javawebapp
    ports:
        - port: 80
          targetPort: 8080
...
ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-service-manifest.yml
service/javawebappsvc created
ubuntu@ip-172-31-9-165:~$
```

```
ubuntu@ip-172-31-9-165:~$ kubectl get service
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP                                                    PORT(S)
AGE
javawebappsvc   LoadBalancer   10.100.134.195   a7e5daf7c86e64d98844e186fec4927c-
615466695.ca-central-1.elb.amazonaws.com   80:31628/TCP   50s
kubernetes      ClusterIP      10.100.0.1       <none>                                                         443/TCP
5h22m
ubuntu@ip-172-31-9-165:~$ kubectl get service
```

Go to EC2 --> Loadbalancer
Loadbalancer is created after running the Service file



Copy DNS name
a7e5daf7c86e64d98844e186fec4927c-615466695.ca-central-1.elb.amazonaws.com



Click on Loadbalancer Security Groups

Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming

Filter load balancers

Name	DNS name	State	VPC ID
a7e5daf7c86e64d9884...	a7e5daf7c86e64d98844e1...	–	vpc-0cd

Load balancer: a7e5daf7c86e64d98844e186fec4927c

Details | Listeners | Network mapping | Security | Health checks | Targ

Security groups (1)

A security group is a set of firewall rules that control the traffic to your load balancer.

Security Group ID	Name	Description
sg-0f7ad66585b61e402	k8s-elb-a7e5d...	Security group for Kubernetes ELB a7

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules Info

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	
sgr-0c9063b9b6fafc196	HTTP	TCP	80	Custom	0.0.0.0/0
sgr-0d9524adcb4ffccda	All TCP	TCP	0 - 65535	Custom	0.0.0.0/0
sgr-0d625aa55801f1964	All TCP	TCP	0 - 65535	Custom	0.0.0.0/0

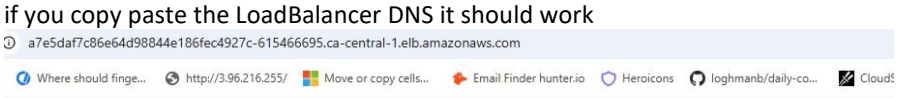
Add rule

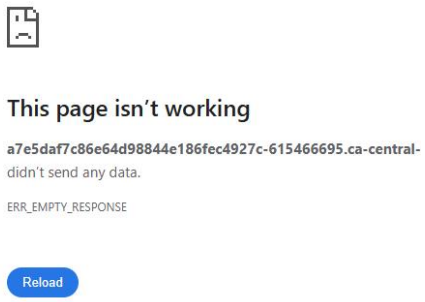
ubuntu@ip-172-31-9-165:~$ kubectl logs javawebapp

```
ubuntu@ip-172-31-9-165:~$ kubectl logs javawebapp
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/05/18 21:29:54 [notice] 1#1: using the "epoll" event method
2025/05/18 21:29:54 [notice] 1#1: nginx/1.27.5
2025/05/18 21:29:54 [notice] 1#1: built by gcc 14.2.0 (Alpine 14.2.0)
2025/05/18 21:29:54 [notice] 1#1: OS: Linux 5.10.236-228.935.amzn2.x86_64
2025/05/18 21:29:54 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/05/18 21:29:54 [notice] 1#1: start worker processes
2025/05/18 21:29:54 [notice] 1#1: start worker process 30
2025/05/18 21:29:54 [notice] 1#1: start worker process 31
ubuntu@ip-172-31-9-165:~$
```

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
javawebapp   1/1     Running   0          52m
ubuntu@ip-172-31-9-165:~$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
ip-192-168-26-249.ca-central-1.compute.internal   Ready    <none>   5h31m   v1.32.3-eks-473151a
ip-192-168-44-239.ca-central-1.compute.internal   Ready    <none>   5h31m   v1.32.3-eks-473151a
ubuntu@ip-172-31-9-165:~$ kubectl get service
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
javawebappsvc   LoadBalancer   10.100.134.195   a7e5daf7c86e64d98844e186fec4927c-615466695.ca-cent
kubernetes      ClusterIP      10.100.0.1       <none>
ubuntu@ip-172-31-9-165:~$
```

if you copy paste the LoadBalancer DNS it should work





Some issue with the docker image but all others are deployed correctly and working fine

How to delete everything?

ubuntu@ip-172-31-9-165:~$ kubectl delete all --all

ubuntu@ip-172-31-9-165:~$ kubectl delete all --all
pod "javawebapp" deleted
service "javawebappsvc" deleted
service "kubernetes" deleted

After delete all everything is gone

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl get pods
No resources found in default namespace.
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl get service
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   2m27s
ubuntu@ip-172-31-9-165:~$
```

This service is internal service and it is for cluster communications and nothing to do with us

ubuntu@ip-172-31-9-165:~$ vi k8s-pod-manifest-new.yml

ubuntu@ip-172-31-9-165:~$ cat k8s-pod-manifest-new.yml

```
---
apiVersion: v1
kind: POD
metadata:
  name: javawebapp
  labels:
    app: javawebapp
spec:
  containers:
    - name: javawebappcontainer
      image: hacker123shiva/springbt-in-docker:latest
      ports:
        - containerPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ cat k8s-pod-manifest-new.yml
---
apiVersion: v1
kind: POD
metadata:
    name: javawebapp
    labels:
        app: javawebapp
spec:
    containers:
        - name: javawebappcontainer
          image: hacker123shiva/springbt-in-docker:latest
          ports:
            - containerPort: 8080
...
ubuntu@ip-172-31-9-165:~$
```

Added both resources and services in the same yml file

ubuntu@ip-172-31-9-165:~$ cat k8s-pod-manifest-new.yml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: javawebapp
  labels:
    app: javawebapp
spec:
```

```
  containers:
    - name: javawebappcontainer
      image: hacker123shiva/springbt-in-docker:latest
      ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: javawebappsvc
spec:
  type: LoadBalancer
  selector:
    app: javawebapp
  ports:
    - port: 80
      targetPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~$ cat k8s-pod-manifest-new.yml
---
apiVersion: v1
kind: Pod
metadata:
    name: javawebapp
    labels:
        app: javawebapp
spec:
    containers:
        - name: javawebappcontainer
          image: hacker123shiva/springbt-in-docker:latest
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
    name: javawebappsvc
spec:
    type: LoadBalancer
    selector:
        app: javawebapp
    ports:
        - port: 80
          targetPort: 8080
...
ubuntu@ip-172-31-9-165:~$
```

```
ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-manifest-new.yml
pod/javawebapp created
service/javawebappsvc created
```

```
ubuntu@ip-172-31-9-165:~$ kubectl get svc
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP
javawebappsvc   LoadBalancer   10.100.58.120   a92497ac034394b9d8900accaacd953e
kubernetes      ClusterIP      10.100.0.1      <none>
ubuntu@ip-172-31-9-165:~$
```

ubuntu@ip-172-31-9-165:~$ kubectl logs javawebapp

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl logs javawebapp

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.2)

2025-05-18T22:40:12.242Z  INFO 1 --- [           main] c.j.f.FirstSpringl
sing Java 17.0.2 with PID 1 (/app/first-springboot-jenkins-0.0.1-SNAPSHO
2025-05-18T22:40:12.246Z  INFO 1 --- [           main] c.j.f.FirstSpringl
efault"
2025-05-18T22:40:13.327Z  INFO 1 --- [           main] o.s.b.w.embedded.
2025-05-18T22:40:13.341Z  INFO 1 --- [           main] o.apache.catalina
2025-05-18T22:40:13.342Z  INFO 1 --- [           main] o.apache.catalina
2025-05-18T22:40:13.474Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat
2025-05-18T22:40:13.475Z  INFO 1 --- [           main] w.s.c.ServletWebS
 ms
2025-05-18T22:40:13.826Z  INFO 1 --- [           main] o.s.b.a.w.s.Welco
2025-05-18T22:40:14.153Z  INFO 1 --- [           main] o.s.b.w.embedded.
2025-05-18T22:40:14.199Z  INFO 1 --- [           main] c.j.f.FirstSpringl
process running for 3.061)
```

Go to EC2 loadbalancer

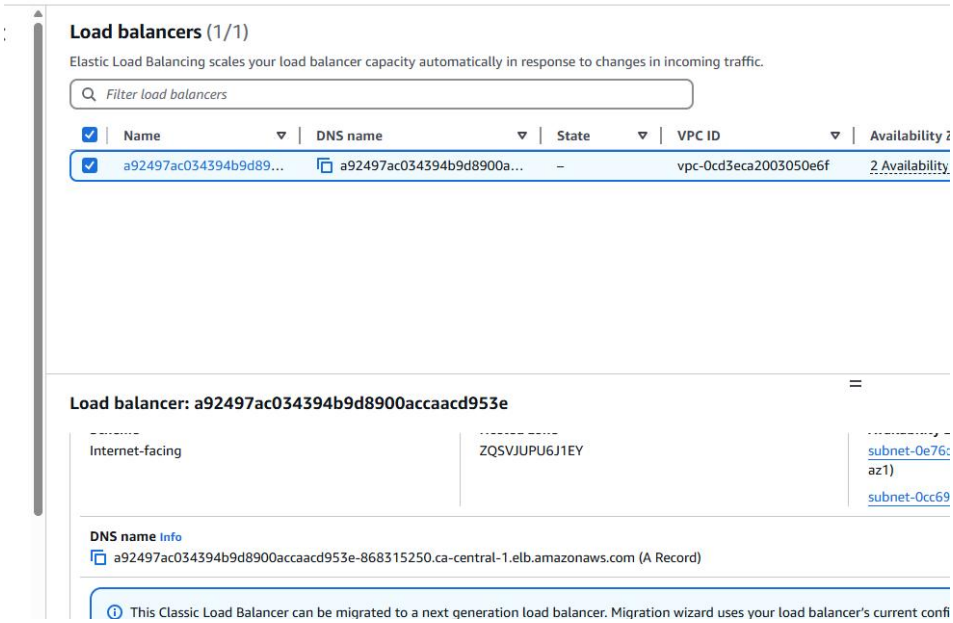
Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Filter load balancers

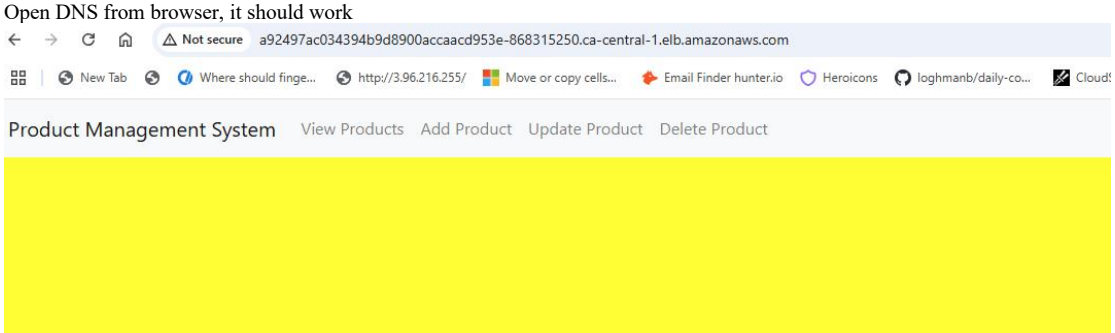
Name	DNS name	State	VPC ID
a7e5daf7c86e64d9884...	a7e5daf7c86e64d98844e1...	–	vpc-0cd3eca2003050e6f

Go to Loadbalancer and copy DNS



a92497ac034394b9d8900accaacd953e-868315250.ca-central-1.elb.amazonaws.com

Open DNS from browser, it should work



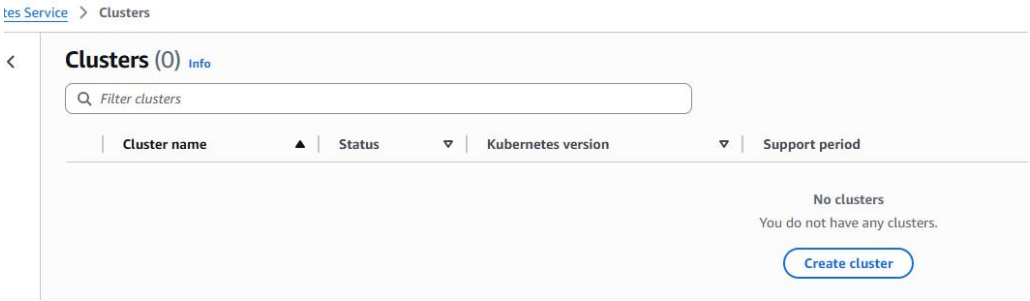

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
javawebapp    1/1     Running   0          9m41s
ubuntu@ip-172-31-9-165:~$ kubectl get nodes
NAME                                          STATUS   ROL
ip-192-168-26-249.ca-central-1.compute.internal   Ready    <no
ip-192-168-44-239.ca-central-1.compute.internal   Ready    <no
ubuntu@ip-172-31-9-165:~$ kubectl get svx
error: the server doesn't have a resource type "svx"
ubuntu@ip-172-31-9-165:~$ kubectl get svc
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP
javawebappsvc   LoadBalancer   10.100.58.120   a92497ac034394b
kubernetes      ClusterIP      10.100.0.1      <none>
ubuntu@ip-172-31-9-165:~$
```

ubuntu@ip-172-31-9-165:~$ kubectl delete all --all
pod "javawebapp" deleted
service "javawebappsvc" deleted
service "kubernetes" deleted

ubuntu@ip-172-31-9-165:~$ eksctl delete cluster --name my-eks-cluster --region ca-central-1

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ eksctl delete cluster --name my-eks-cluster --region ca-central-1
2025-05-18 23:02:13 [ℹ]  deleting EKS cluster "my-eks-cluster"
2025-05-18 23:02:14 [ℹ]  will drain 0 unmanaged nodegroup(s) in cluster "my-eks-cluster"
2025-05-18 23:02:14 [ℹ]  starting parallel draining, max in-flight of 1
2025-05-18 23:02:14 [ℹ]  deleted 0 Fargate profile(s)
2025-05-18 23:02:14 [✓]  kubeconfig has been updated
2025-05-18 23:02:14 [ℹ]  cleaning up AWS load balancers created by Kubernetes objects of Kind Service or Ing
2025-05-18 23:02:14 [ℹ]
2 sequential tasks: { delete nodegroup "ng-98fe4a7e", delete cluster control plane "my-eks-cluster" [async]
}
2025-05-18 23:02:15 [ℹ]  will delete stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:02:15 [ℹ]  waiting for stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e" to get deleted
2025-05-18 23:02:15 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:02:45 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:03:28 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:04:48 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:06:48 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:07:32 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:08:24 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:09:15 [ℹ]  waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-98fe4a7e"
2025-05-18 23:09:15 [ℹ]  will delete stack "eksctl-my-eks-cluster-cluster"
2025-05-18 23:09:15 [✓]  all cluster resources were deleted
ubuntu@ip-172-31-9-165:~$
```

Cluster is gone



PODS
Services (ClusterIP, NodePort, LoadBalancer)
Namespaces
ReplicationController (RS)
ReplicaSet
DaemonSet
StetefulSet
IngressController
HPA
HelmCharts
K8s monitoring (Grafana, Promethues)
EFK stack group setup to monitor app logs

Note: We need a machine where we install kubectl
In the same kubectl Host machine, we need kubectl, awscli and eks cli