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

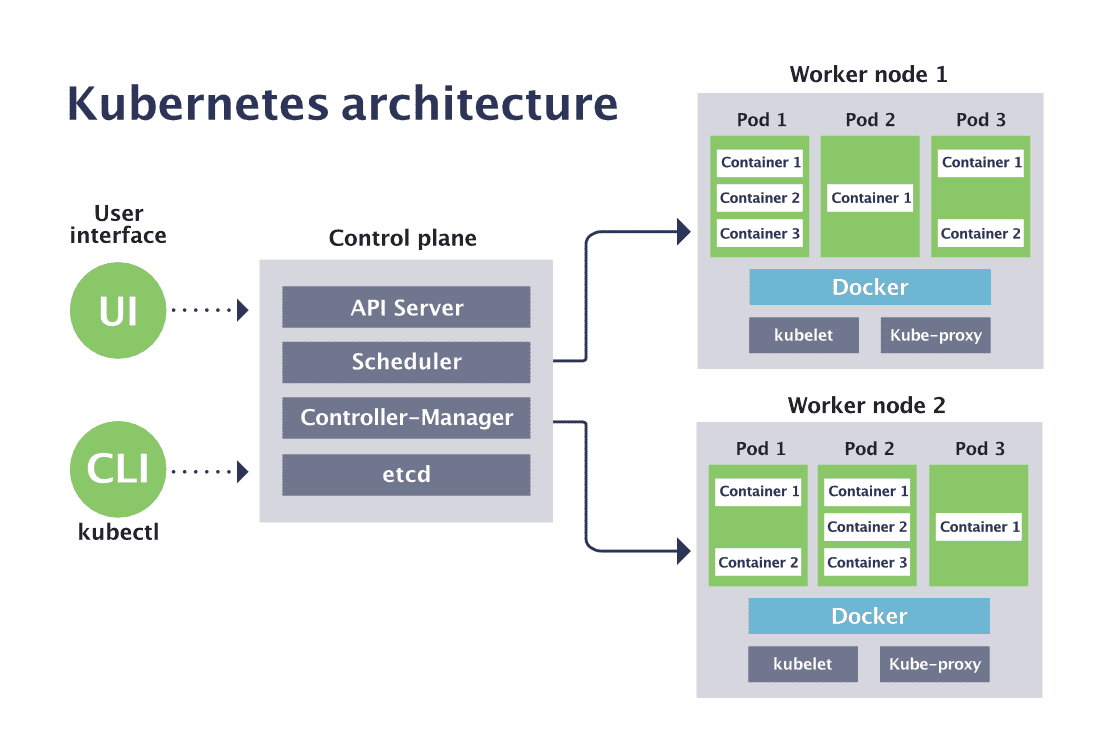
--> K8s follows cluster architecture

--> 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)
   1. In the control node, we have something called as an API server
   2. Scheduler
   3. Controller-Manager
   4. etcd



1. Worker Node
   1. Kubelet
   2. Kube proxy
   3. Docker engine
   4. POD
   5. Within the POD, we have the Container

Kubernetes Cluster Architecture

Two important parts in Kubernetes Architecture

User interface

UI

Worker node 1

Control plane

CLI

Worker node 2

Kubectl

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)

POD1

Container

Container

Container

--> 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

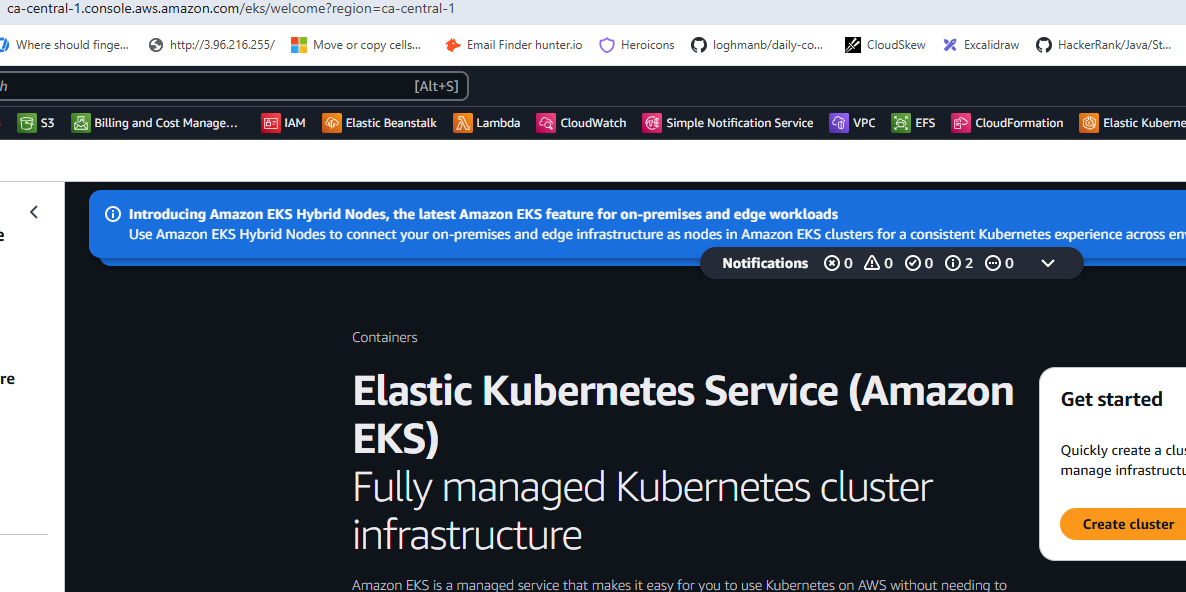
Delete all resources:

kubectl delete all --all

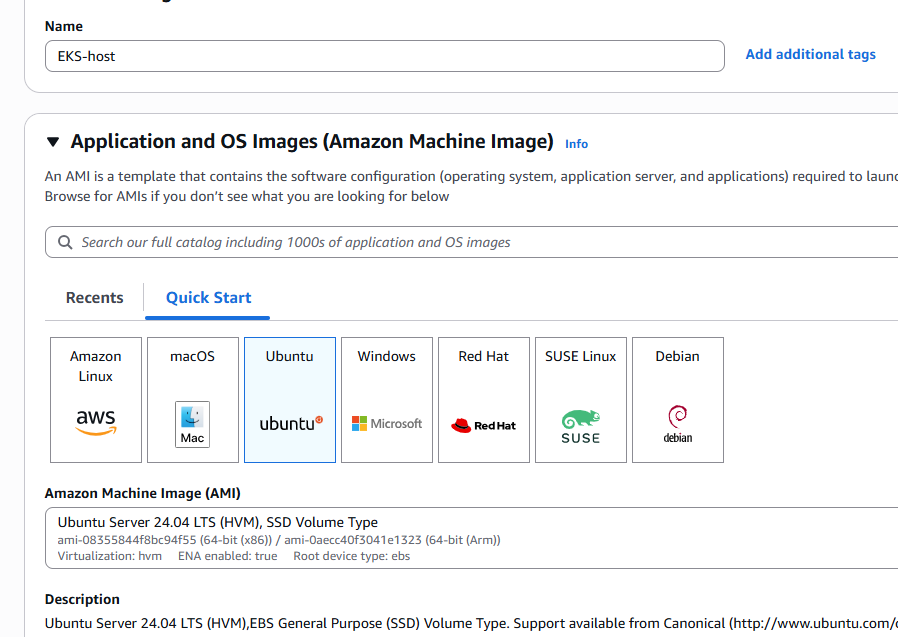
Delete entire cluster:

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

Illustration:



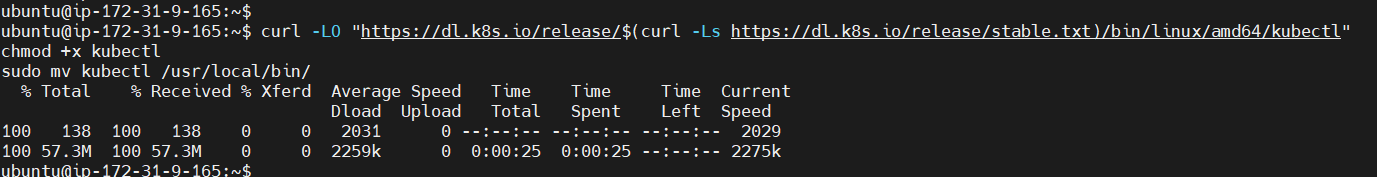
No option to temporarily stop the cluster you have to delete

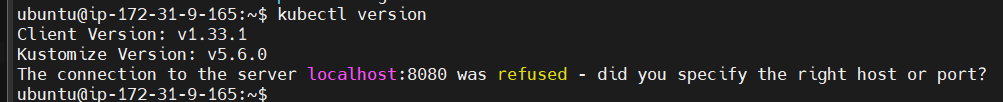


curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

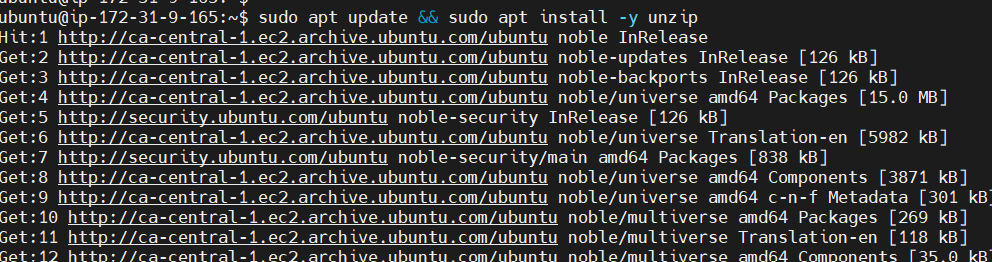
chmod +x kubectl

sudo mv kubectl /usr/local/bin/





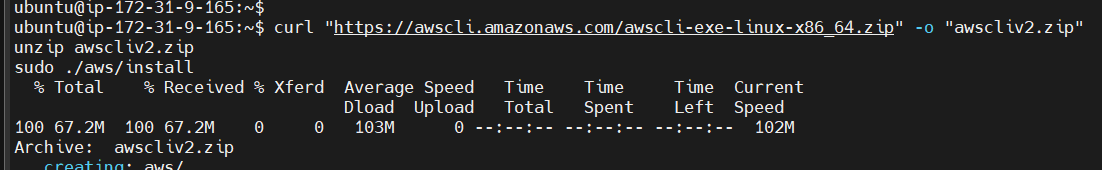
ubuntu@ip-172-31-9-165:~$ sudo apt update && sudo apt install -y unzip

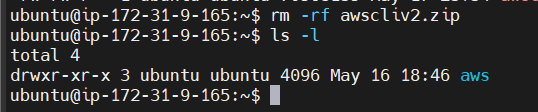


ubuntu@ip-172-31-9-165:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

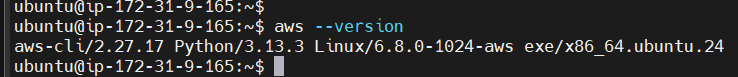




ubuntu@ip-172-31-9-165:~$ aws --version

aws-cli/2.27.17 Python/3.13.3 Linux/6.8.0-1024-aws exe/x86\_64.ubuntu.24

Verify installation



Install eksctl

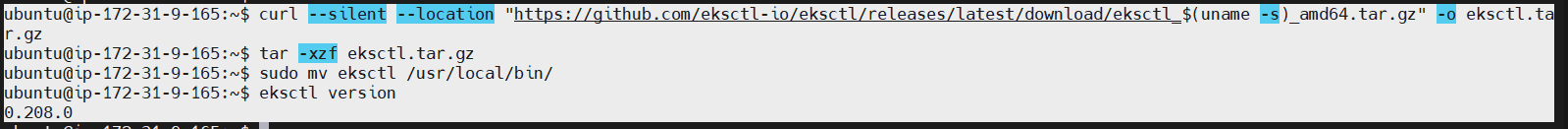
ubuntu@ip-172-31-9-165:~$ curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" -o eksctl.tar.gz

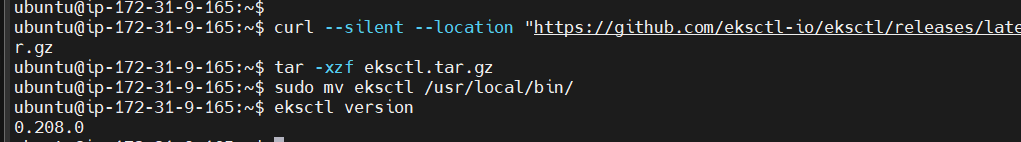
ubuntu@ip-172-31-9-165:~$ tar -xzf eksctl.tar.gz

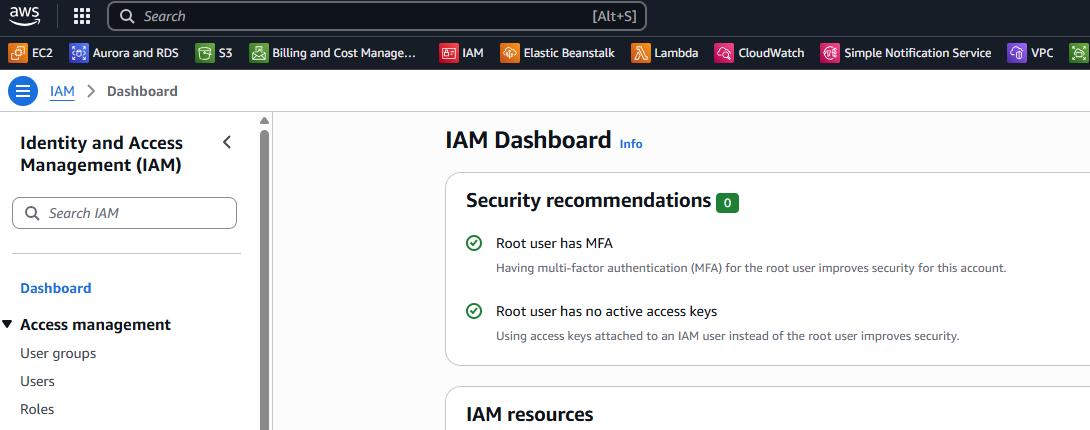
ubuntu@ip-172-31-9-165:~$ sudo mv eksctl /usr/local/bin/

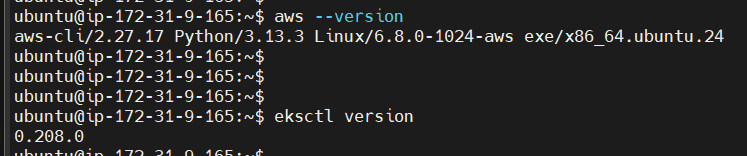
ubuntu@ip-172-31-9-165:~$ eksctl version

0.208.0



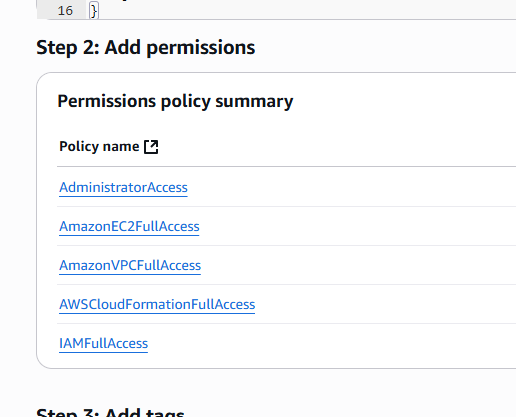




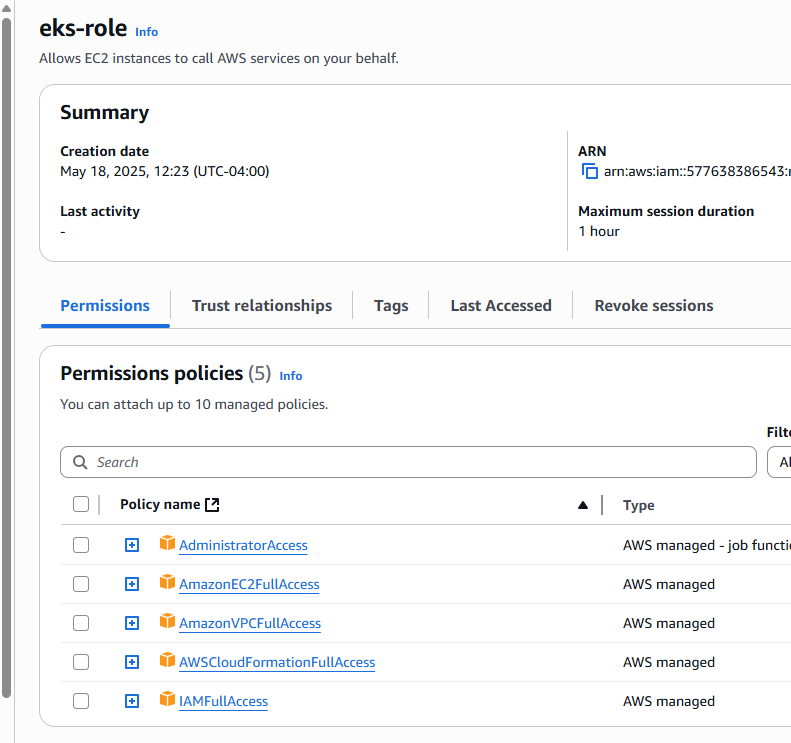


Click Create Role

Add IAMFullAccess, AmazonEC2FullAccess, AmazonVPCFullAccess, CloudFormationFullAccess, AdministratorAccess



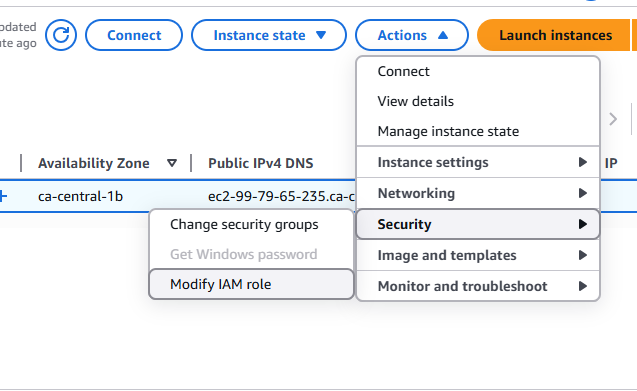
Click Create Role



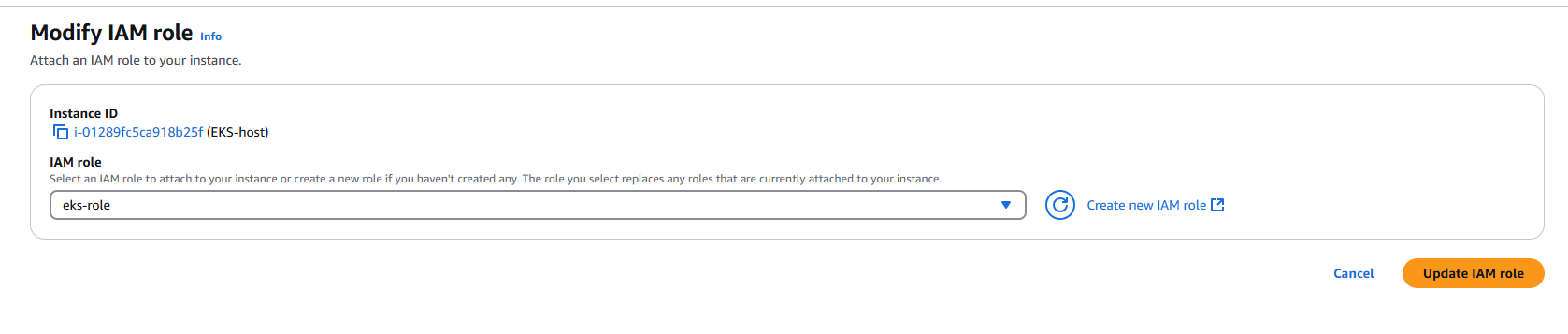
Go back to EC2:

Click on instance

Security --> Modify IAM role



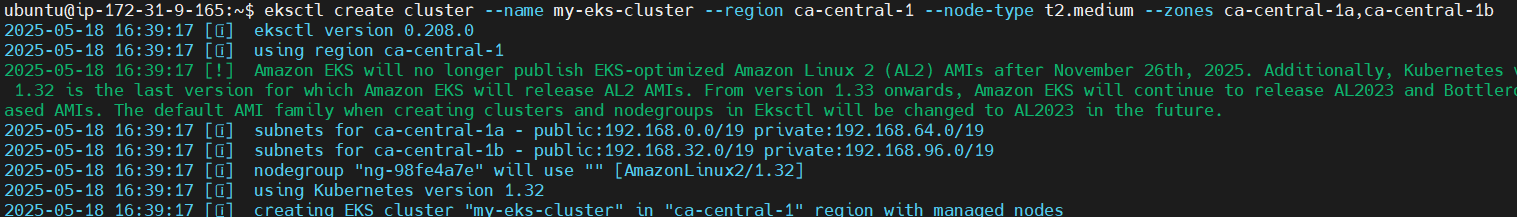
Select eks-role



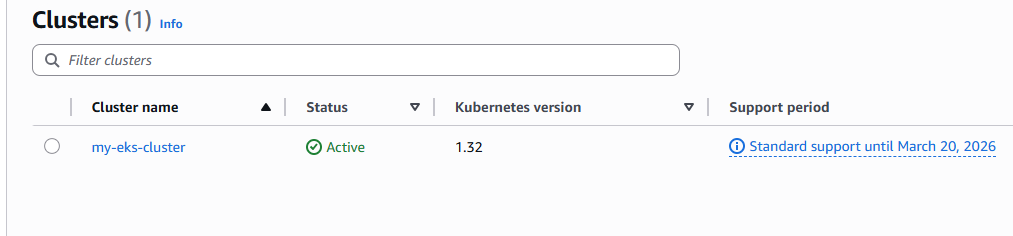
Update IAM role

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



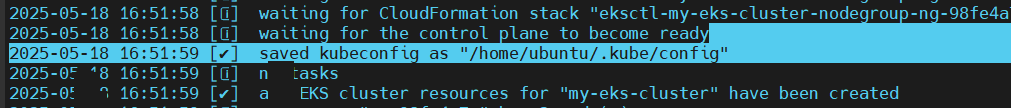
Cluster is created



AWS-managed control plane

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

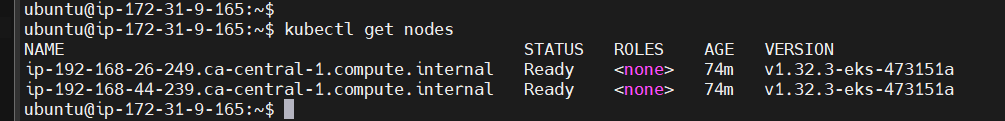
2025-05-18 16:51:59 [✔] saved kubeconfig as "/home/ubuntu/.kube/config"



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

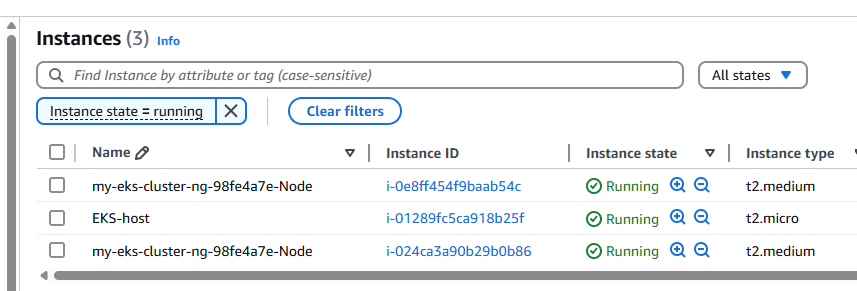


We created two worker nodes



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

Worker node1

Pod 3

Pod 2

Pod 1

Static IP address

Cluster IP service

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

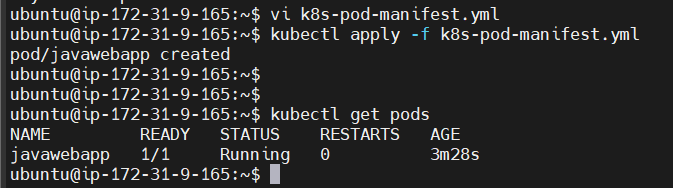
ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-manifest.yml

pod/javawebapp created

ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebapp 1/1 Running 0 3m28s

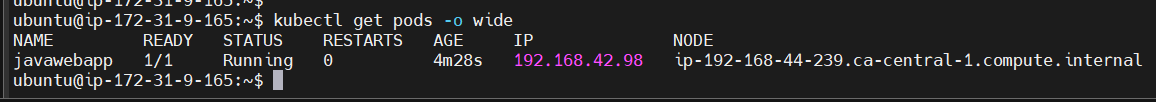


How to get which worker node this pod is deployed?

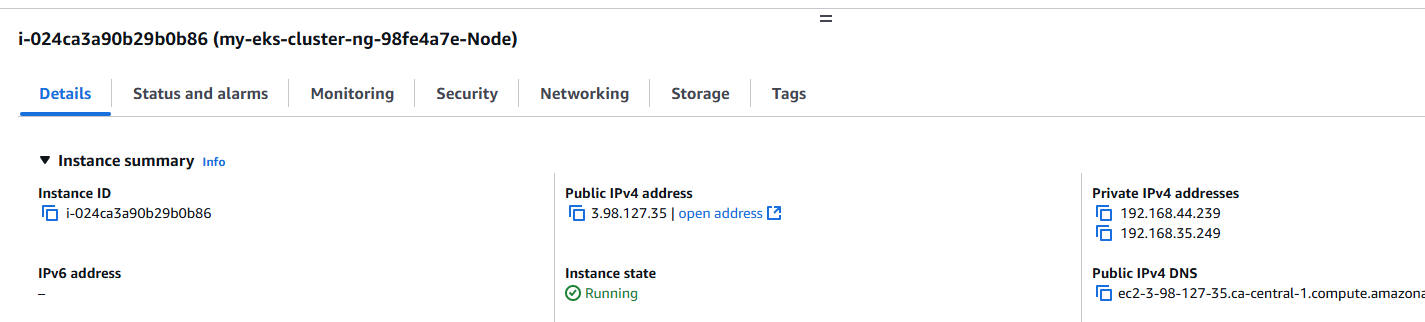
ubuntu@ip-172-31-9-165:~$ kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

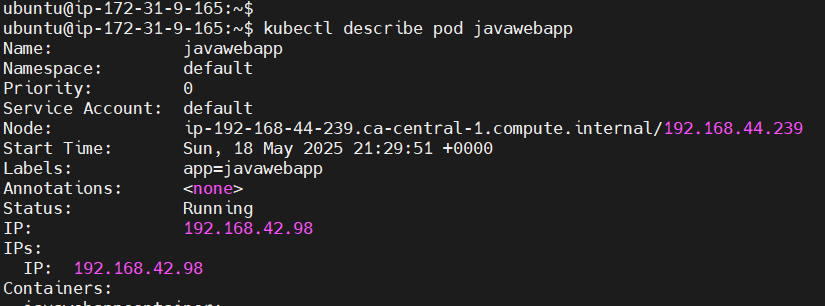
javawebapp 1/1 Running 0 4m28s 192.168.42.98 ip-192-168-44-239.ca-central-1.compute.internal <none> <none>



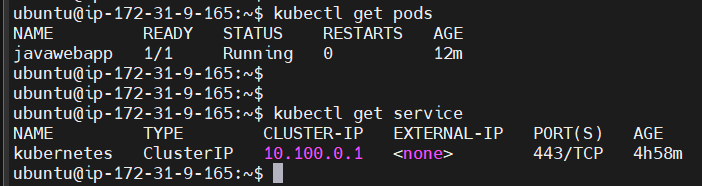
that’s the EC2



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



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



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

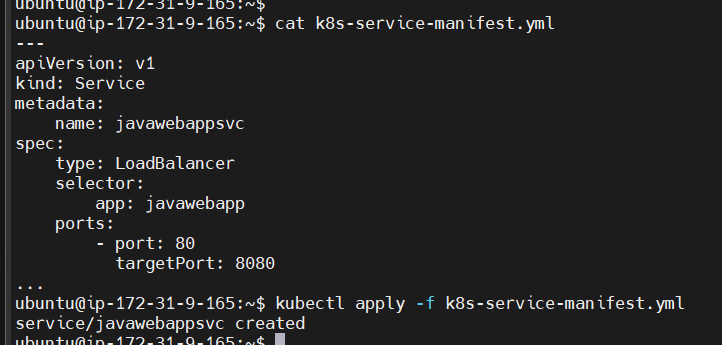
targetPort: 8080

...

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:~$ kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

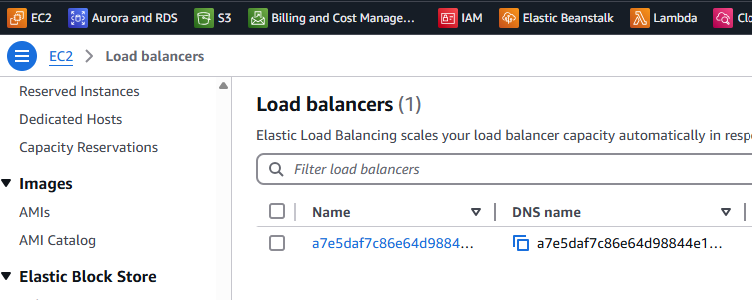
javawebappsvc LoadBalancer 10.100.134.195 a7e5daf7c86e64d98844e186fec4927c-615466695.ca-central-1.elb.amazonaws.com 80:31628/TCP 50s

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 5h22m

ubuntu@ip-172-31-9-165:~$ kubectl get service

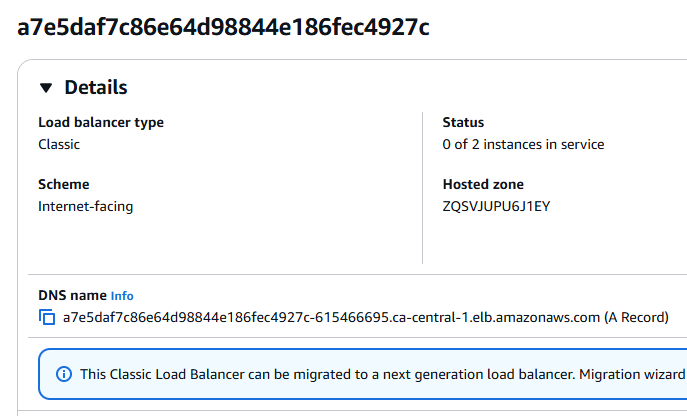
Go to EC2 --> Loadbalancer

Loadbalancer is created after running the Service file

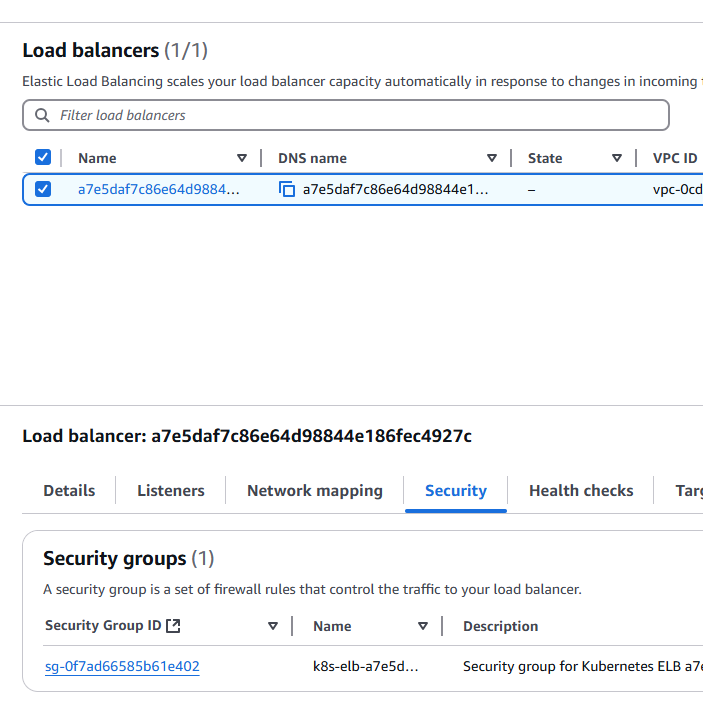


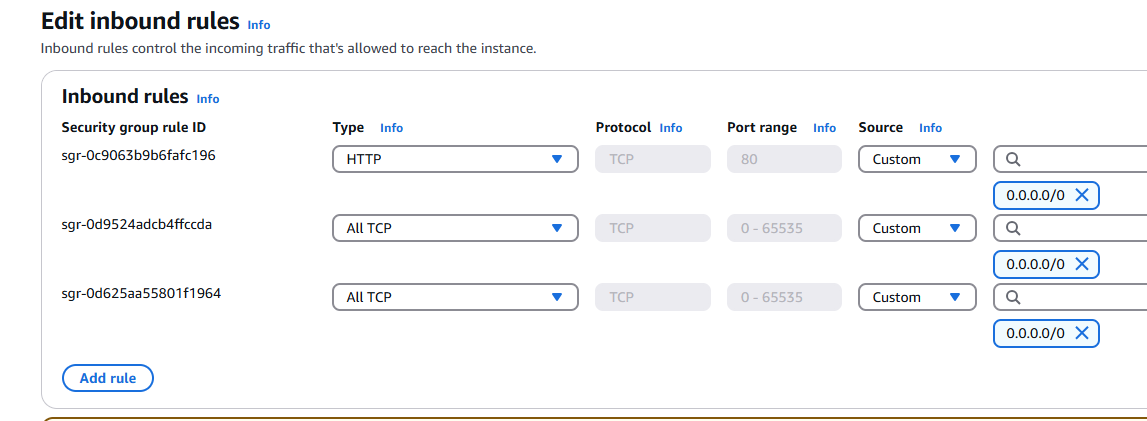
Copy DNS name

a7e5daf7c86e64d98844e186fec4927c-615466695.ca-central-1.elb.amazonaws.com

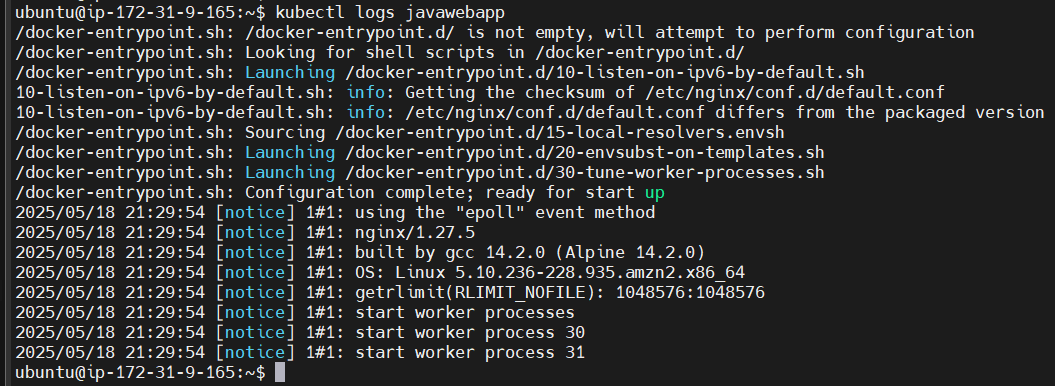


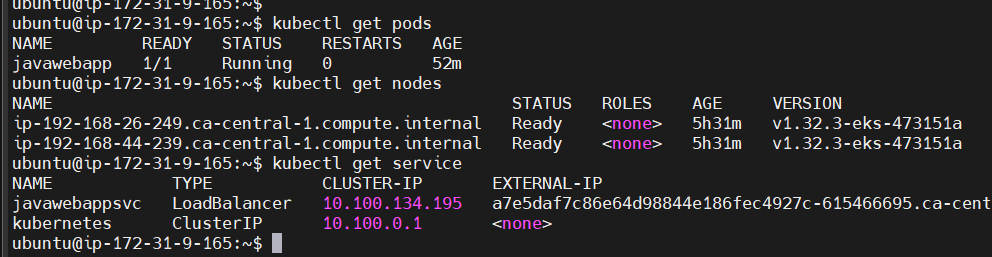
Click on Loadbalancer Security Groups

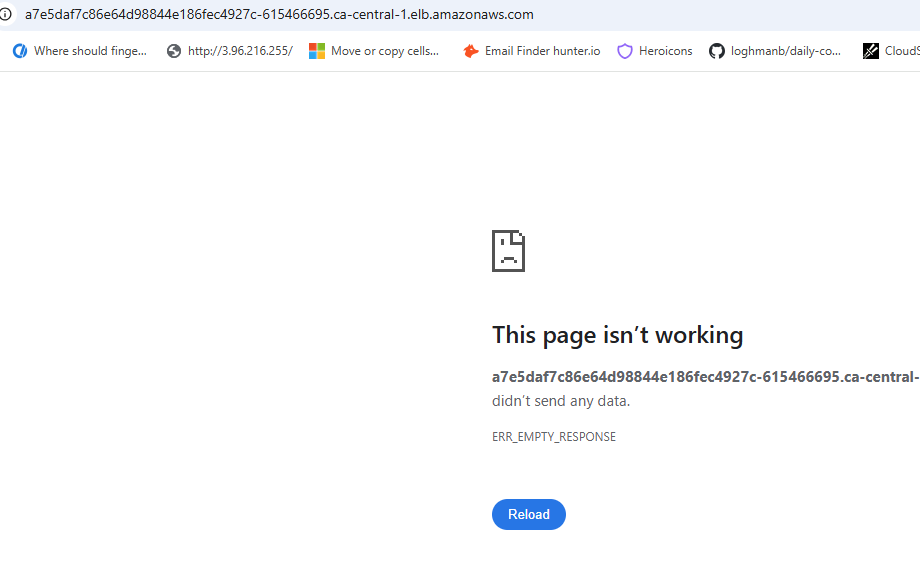




ubuntu@ip-172-31-9-165:~$ kubectl logs javawebapp



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

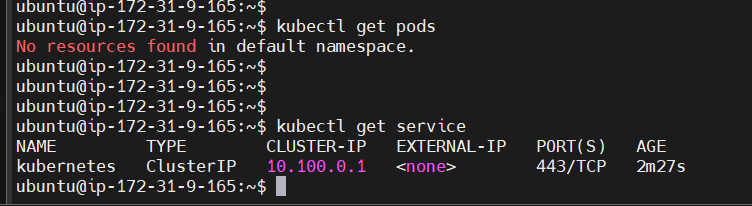
ubuntu@ip-172-31-9-165:~$ kubectl delete all --all

pod "javawebapp" deleted

service "javawebappsvc" deleted

service "kubernetes" deleted

After delete all everything is gone



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:

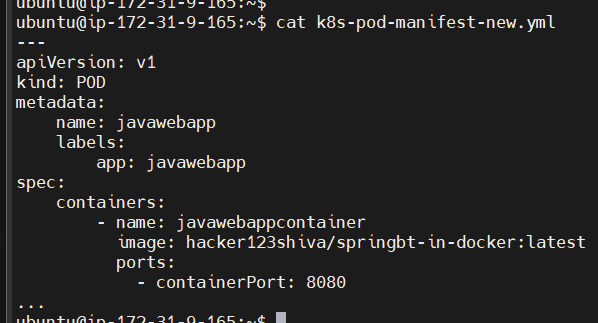
- name: javawebappcontainer

image: hacker123shiva/springbt-in-docker:latest

ports:

- containerPort: 8080

...



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:

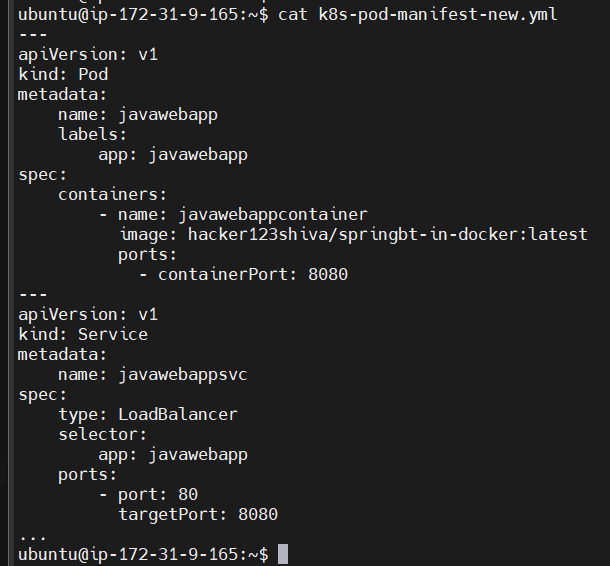
app: javawebapp

ports:

- port: 80

targetPort: 8080

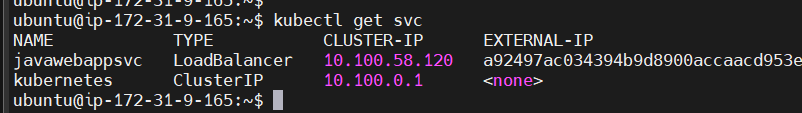
...



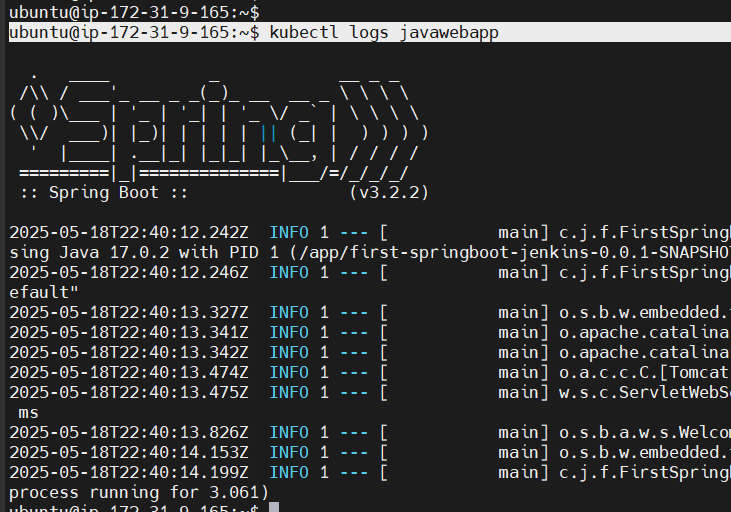
ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-manifest-new.yml

pod/javawebapp created

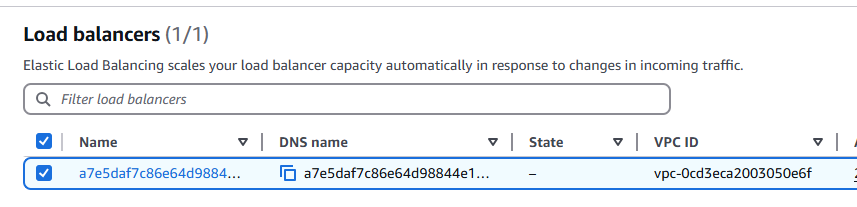
service/javawebappsvc created



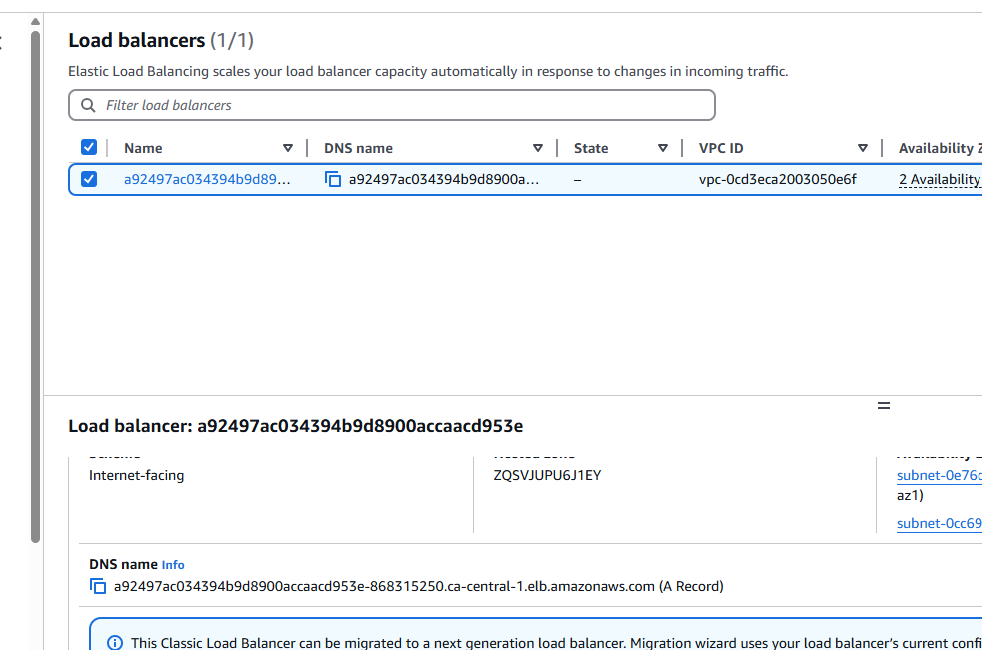
ubuntu@ip-172-31-9-165:~$ kubectl logs javawebapp



Go to EC2 loadbalancer

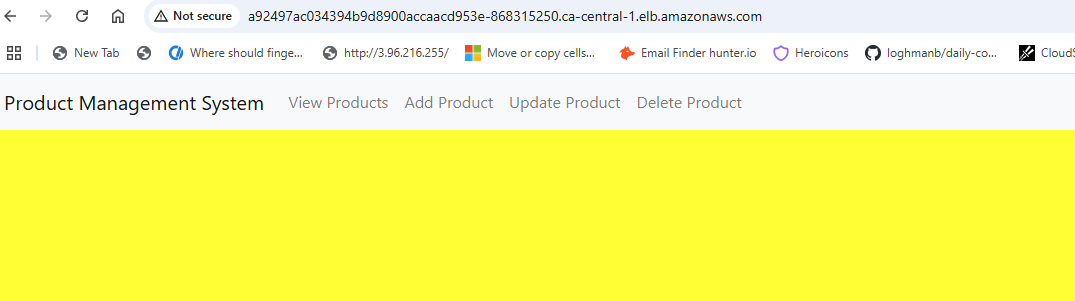


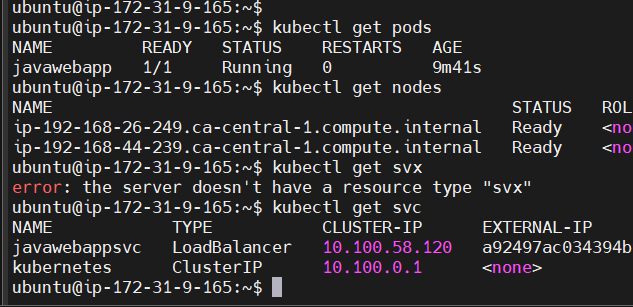
Go to Loadbalancer and copy DNS



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

Open DNS from browser, it should work





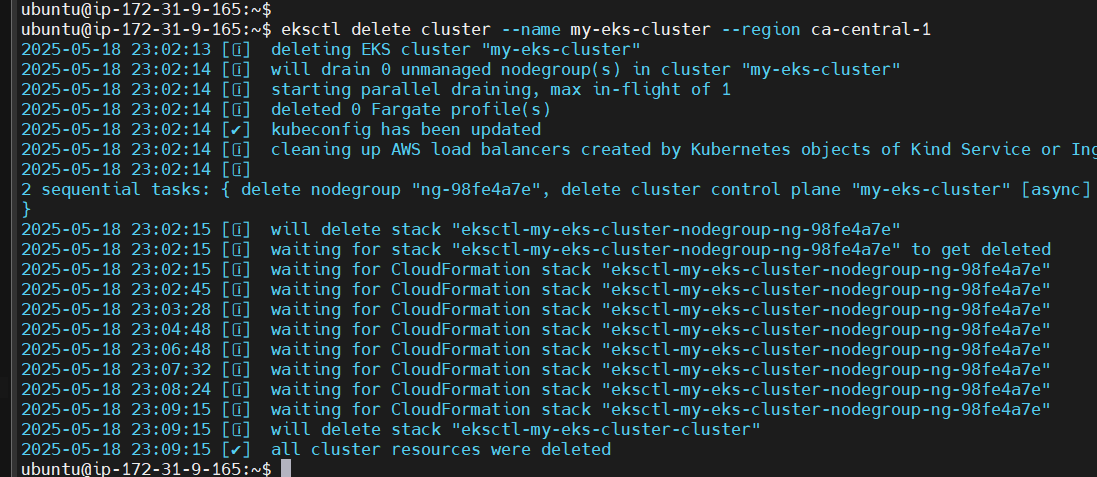
ubuntu@ip-172-31-9-165:~$ kubectl delete all --all

pod "javawebapp" deleted

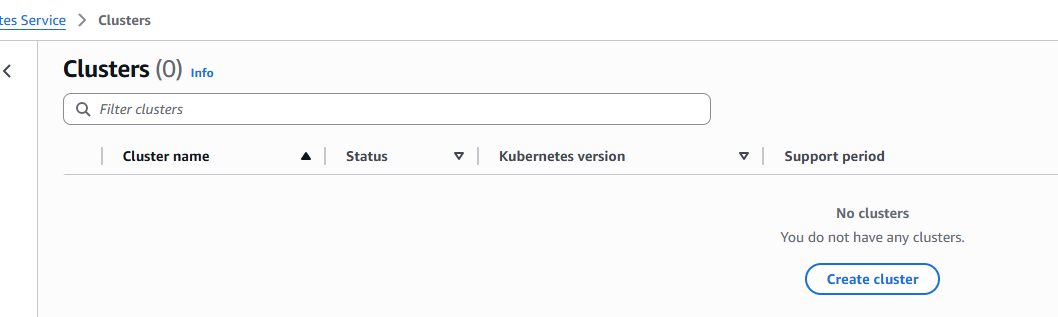
service "javawebappsvc" deleted

service "kubernetes" deleted

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



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

Worker node1

Kubectl  
AWS Cli

Eks Cli

Control plane

AWS IAM Role

Worker node2

EKS Host VM