**Task**

1. Use a sample nodejs or java application(create it or use any from github)

2. Create EKS cluster with terraform

3. Deploy the application with HA on EKS with Helm charts and expose with NLB(aws-load-balancer-controller)

**Create the EKS Cluster with Terraform**

Prerequisites

1. **AWS CLI installed and configured.**
2. **Terraform installed on your local system.**

Then,

I’ve create the terraform file for eks cluster configuration.

**vim eksclust.tf**

provider "aws" {

region = "us-west-2"

}

resource "aws\_eks\_cluster" "my\_cluster" {

name = "my-eks-cluster"

role\_arn = aws\_iam\_role.eks\_cluster\_role.arn

vpc\_config {

subnet\_ids = ["subnet-05fbd8bfbaa9a072b", "subnet-0fa3f18c62ee6b251"]

security\_group\_ids = ["sg-03bccac3015b87a6a"]

endpoint\_public\_access = true

endpoint\_private\_access = true

}

}

resource "aws\_eks\_node\_group" "my\_node\_group" {

cluster\_name = aws\_eks\_cluster.my\_cluster.name

node\_group\_name = "my-node-group"

node\_role\_arn = aws\_iam\_role.eks\_node\_group\_role.arn

subnet\_ids = ["subnet-05fbd8bfbaa9a072b", "subnet-0fa3f18c62ee6b251"]

scaling\_config {

desired\_size = 2

max\_size = 3

min\_size = 1

}

}

resource "aws\_iam\_role" "eks\_cluster\_role" {

name = "eks\_cluster\_role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Effect = "Allow",

Principal = {

Service = "eks.amazonaws.com"

},

Action = "sts:AssumeRole"

}

]

})

}

resource "aws\_iam\_role" "eks\_node\_group\_role" {

name = "eks\_node\_group\_role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Effect = "Allow",

Principal = {

Service = "ec2.amazonaws.com"

},

Action = "sts:AssumeRole"

}

]

})

}

**Explanation:-**

In this file i’ve written infrastructure.

i.e. i’ve defined

* **Provider Block**: Specifies the AWS provider and the desired region.
* **AWS EKS Cluster Resource**: Defines the EKS cluster, including its name, VPC configuration, and associated IAM role.
* **AWS EKS Node Group Resource**: Specifies the worker nodes for the cluster, including the cluster name, node group name, and scaling configuration.
* **AWS IAM Role Resources**: Defines IAM roles for the EKS cluster and node group.

**Now, Deploy the Node.js Application with a High Availability on EKS using Helm chart and expose it with Network Load Blancer.**

Prerequisites

* Helm installed on your local machine.
* Install docker
* Then, Dockerized Node.js application and packaged to be deployed on Kubernetes
* Now, to install Helm i need a sign-in key of helm. Then, update our packages and install Helm.

curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null

sudo apt-get install apt-transport-https --yes

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list

sudo apt-get update

sudo apt-get install helm

So, Helm installed successfully.

* Install Docker

**apt install docker.io -y**

* Now, Dockerized Node.js application

I have Dockerfile of nodejs application. I’ve create the image of that Dockerfile and push into the image Repository i.e. on DockerHub.

**docker build -t imagejs .**

I push the image on dockerhub. So, firstly we need to tag that image with our github Repository Then we push image on dockerhub.

**docker tag imagejs prodrepo/imagejs**

**docker push prodrepo/imagejs**

Image pushed to dockerhub successfully.

* Then, we create the nodejs chart by helm.

**helm create nodejs**

**cd nodejs**

Chart.yaml charts templates values.yaml

* In values.yaml file oh nodejs-chart we need to do some change i.e. repository of image, tags, and the port no. of service. So, as per the changes, the image will be changed.

vim values.yml

**replicaCount: 3**

**image:**

**repository:imagejs**

**tag: “latest”**

**port: 3000**

Now, inside the template of nodejs chart there is deployment.yml and service.yml. I modify that manifests.

### Deployment Configuration (deployment.yaml):

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-nodejs-app

spec:

replicas: 3

selector:

matchLabels:

app: my-nodejs-app

template:

metadata:

labels:

app: my-nodejs-app

spec:

containers:

- name: my-nodejs-app

image: imagejs:latest

ports:

- containerPort: 3000

### Service Configuration (service.yaml):

apiVersion: v1

kind: Service

metadata:

name: my-nodejs-app

spec:

type: LoadBalancer

selector:

app: my-nodejs-app

ports:

- protocol: TCP

port: 80

targetPort: 3000

Deployment Steps

* Package our helm chart

**helm package my-nodejs-app/**

* Then, I deploy the helm Chart.

Install the helm chart on our EKS Cluster

**helm install my-nodejs-app ./my-nodejs-app-0.1.0.tgz**

* Verify the Deployment

**kubectl get pods**

**kubectl get services**

Now, we can access our application by using NLB’s DNS name or the ip-address.