**Provisioning Kubernetes clusters on AWS and Deploying Sample node.js:**

There are three populer ways for creating EKS Cluster:

* AWS web interface
* Eksctl command-line utility
* Using code with a tool Terraform

Here I am, including all the steps which I have done for this sample project:

* Setting Up AWS Account
* Setting up the Network
* VPC
* Public subnet & private subnet
* NAT Gateway
* Internet Gateway
* Routetables
* Created IAM user with Admin Access
* Launched Ec2 instance (Ubuntu 18.04)
* Installed AWS CLI & configured



* Installed eksctl



* Installed kubectl

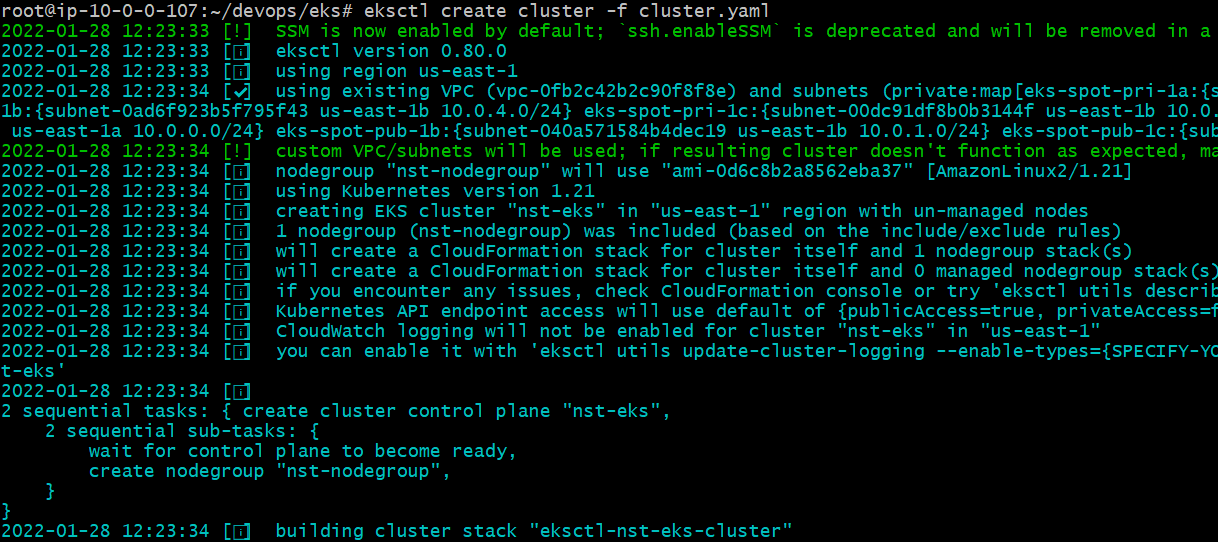


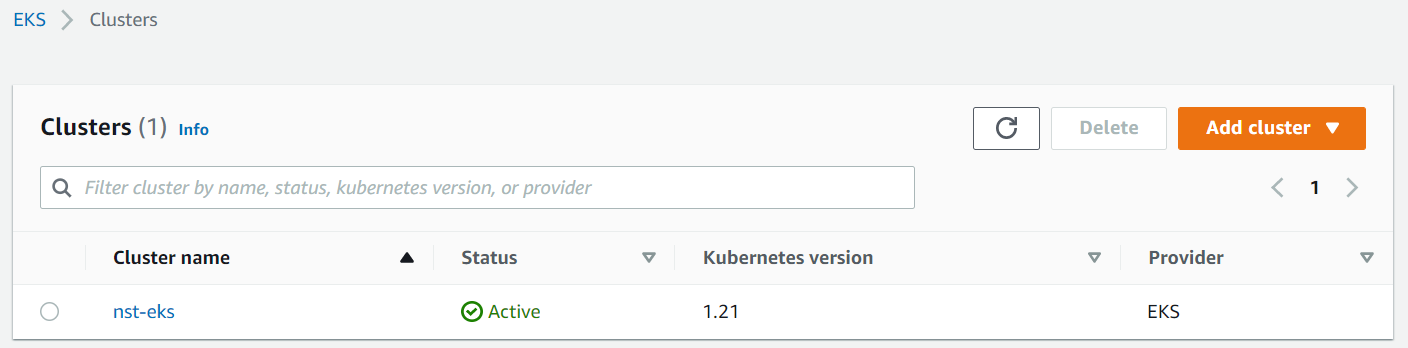
* Installed Docker



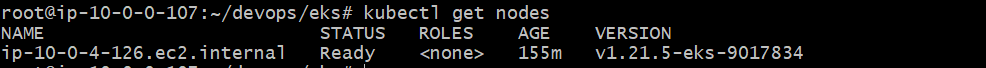
All the required tools I have installed & configured.

Created the cluster using eksctl





Created Nodegroup to deploy the workload:



Attached the required policies to the nodegroup in the yaml file only.

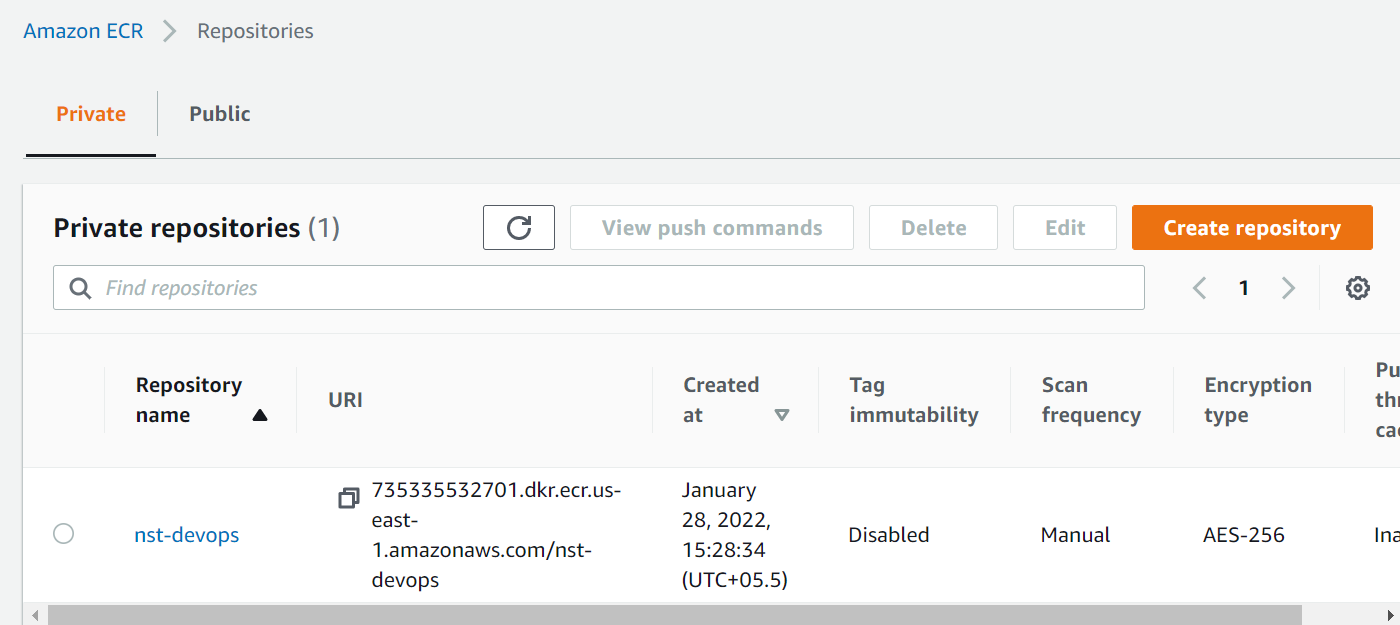
EKS (1.21) Cluster creation is completed

Deploying the Hello world node.js application

* Taken sample helloworld application from official webpage.
* Package
* App.js
* Created the Dockerfile to build the Image



Created the ECR Repo to push & store the Images

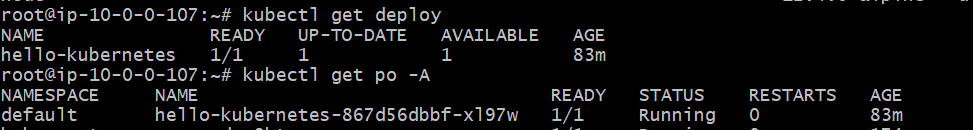


Added Tag to the created image and push the image to ecr

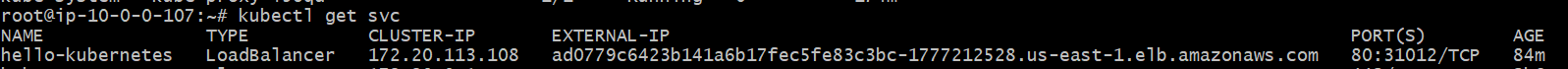


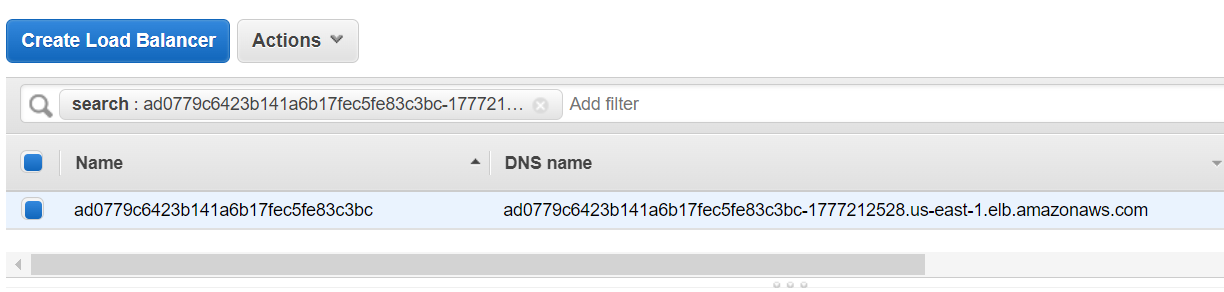
Now we have image ready to deploy into the EKS Cluster

Created the deployment for the image using yaml definition

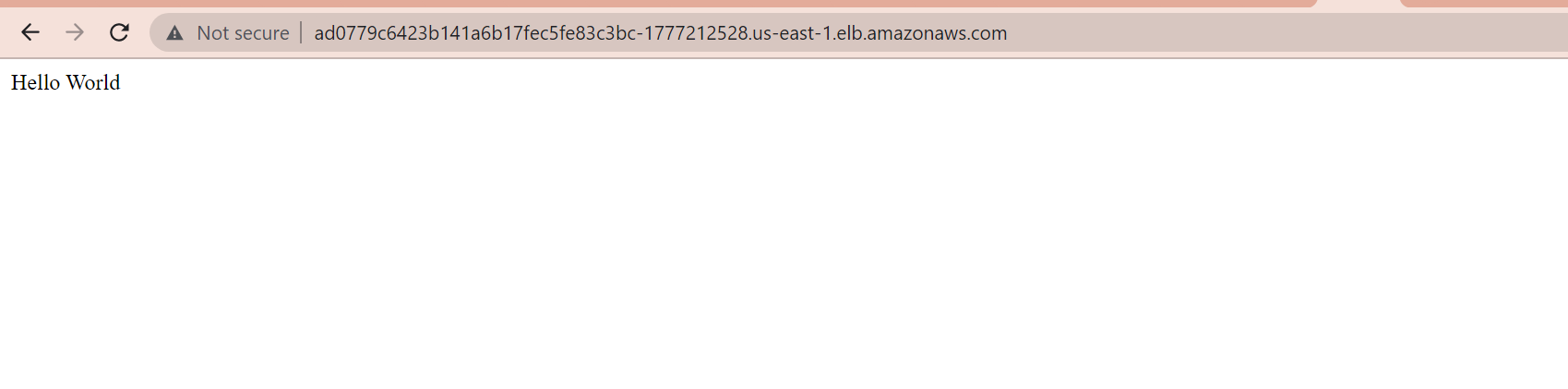


The service pod is up and running

Exposed the service using the Loadbalancer service type 



Finally output on the Browser



* I have uploaded the file which is used for this project into GitHub, please check the node folder

<https://github.com/naresh-ns/awsdevops.git>

<https://github.com/naresh-ns/awsdevops/tree/main/node>

**Autoscaling of the application**:

* we can make service scalable using HPA (Horizontal pod autoscaler), it will adjust the pod count based on the threshold values mentioned.
* We can define the hpa based on the cpu, memory and request count metrics.

**Autoscaling the infrastructure**:

* Using cluster-Autoscaler we can scale the cluster infrastructure, it will adjust the node count based on the usage & allocated resource to the pod.
* It also balances the similar node groups

**Metrics:**

* Metrics server collects resource metrics from kubelet for the use of Autoscaling.

**Secrets:**

Secrets can be store in the cluster using Kubernetes secrets.

It will store the data as key : value pair, can store it in specific attached volume.

I have uploaded the cicd code for the project in GITHUB:

<https://github.com/naresh-ns/awsdevops/tree/main/nodejs>

CICD using Git - Jenkins – Docker – Ecr – HELM – EKS

Using this code, we can manage the complete cicd, whenever there is a code change will deploy it into eks by triggering the Jenkins job.

Also, in the Dev and UAT environment will configure the git poll scm, so whenever the code change in main branch Jenkins job get trigger which deploy the changes into Kubernetes cluster.

Stages include in this pipeline:

* Fetching code from code
* Build the Docker image with new changes
* Login to the ECR
* Push the image to ECR Repository
* Deploy the Application into Kubernetes cluster on the given namespace.

**Monitoring:**

* We can enable the monitoring for the application using different kind of tools (Prometheus, Grafana, ELK, Dynatrace),
* Dynatrace will collect all the metrics through the agents and notifies with alerts whenever the configured threshold reached.