Table of Contents

[Q.1 Starting from an AWS empty environment, explain the \*high level\* steps you would use to setup a Kubernetes cluster (preferably with Terraform), and then deploy 1](#_Toc121874784)

[Step 1 We will install the requisite components 2](#_Toc121874785)

[Step2 create backend for remote state storage (Please clone my repo for follow-up steps) 2](#_Toc121874786)

[Step3 Provision VPC, subnets, nat gateways, igw az etc 2](#_Toc121874787)

[Step4 Create security groups 2](#_Toc121874788)

[step5 Using terraform aws eks module 3](#_Toc121874789)

[Step6 Create the cluster by following steps 3](#_Toc121874790)

[Step6 configure kubectl 4](#_Toc121874791)

[Step7 Create the deployment & access the services 4](#_Toc121874792)

[Q2 For deployment you could explain how a tool such as Gitlab AutoDevOps (https://docs.gitlab.com/ee/topics/autodevops/) or Harness (https://harness.io/kubernetes/) would work, or alternatively explain how the container could be deployed using standard Kubernetes resources. 5](#_Toc121874793)

[Answer: Steps to Deploy container to Kubernetes using standard Kubernetes resources 5](#_Toc121874794)

[Q.3 Also Include high level\* steps to make this accessible from the outside world (eg. DNS) 7](#_Toc121874795)

[Q4, Write a bash script to print a list of all users on the system in the format '$userName $userId $userHomeDirectory' 9](#_Toc121874796)

# Q.1 Starting from an AWS empty environment, explain the \*high level\* steps you would use to setup a Kubernetes cluster (preferably with Terraform), and then deploy

Answer: When we deploy an eks cluster a lot of resources are getting created & deployed depending upon our need. We deploy nearly 56 resources including some WAF & CSI driver setup in our production setup. However basis upon the question I would like to highlight the high level steps here. Some of the very familiar resources which get deployed in cluster are as mentioned below

1. EKS
2. IAM roles
3. IAM roles policies
4. IAM policies attachment
5. IAM OpenID Connect Provider
6. ASG
7. IG
8. SG
9. Subnets
10. Route53

Note: Majority of our task for creating above resources is completed by using following two modules which we are calling in below files

1. [terraform-aws-modules/eks/aws](https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest) (used in eks\_cluster.tf file)
2. [terraform-aws-modules/vpc/aws](https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/latest) (used in vpc.tf file)

However, in my steps I am using below high level steps to create my eks cluster & host the requested image.

Step 1 We will install the requisite components.

1. Create an[*AWS account*](https://portal.aws.amazon.com/billing/signup?nc2=h_ct&src=default&redirect_url=https%3A%2F%2Faws.amazon.com%2Fregistration-confirmation#/start)
2. Install the AWS CLI, [*installed*](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)and [*configured*](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html)
3. [*AWS IAM Authenticator*](https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html)
4. Install the [*Kubernetes CLI*](https://kubernetes.io/docs/tasks/tools/install-kubectl/)*,* also known as kubectl
5. Install the terraform [*instructions how to install the Terraform CLI from the official documentation.*](https://learn.hashicorp.com/tutorials/terraform/install-cli)

## Step2 create backend for remote state storage (Please clone my repo for follow-up steps)

Create S3 bucket & dynamo table to save the state of the eks. We are using our already created bucket however we can either create it from aws or terraform itself. But if we want to create it from terraform itself we should not forget to migrate tfstate file in the same. Our backend code is defined at the location

<https://github.com/My-Infra-Repos/very_basic_terraform_cluster/blob/main/terraform_eks/backend.tf#:~:text=Blame-,terraform%20%7B,%7D,-Footer>

## Step3 Provision VPC, subnets, nat gateways, igw az etc

Next step is to provisions a VPC, subnets, and availability zones using the [AWS VPC Module](https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/2.32.0). We define this code inside vpc.tf file. Please find the file at below location

<https://github.com/My-Infra-Repos/very_basic_terraform_cluster/blob/main/terraform_eks/vpc.tf#:~:text=Blame-,module%20%22vpc%22%20%7B,%7D,-Footer>

We are basically set to create 3 Public subnets, 3 private subnets spawn across 3 different availability zones.

## Step4 Create security groups

We are creating the 2 security groups node-group-1 and node-group-2 for private & public subnets

<https://github.com/My-Infra-Repos/very_basic_terraform_cluster/blob/main/terraform_eks/security-groups.tf#:~:text=Blame-,resource%20%22aws_security_group%22%20%22node_group_one%22%20%7B,%7D,-Footer>

## step5 Using terraform aws eks module

The eks\_managed\_node\_groups parameter will create three nodes across two node groups.

<https://github.com/My-Infra-Repos/very_basic_terraform_cluster/blob/main/terraform_eks/eks-cluster.tf#:~:text=module%20%22eks,%C2%A9%202022%20GitHub>

The outputs file will provide us the following details

1. Cluster Endpoint
2. Cluster id
3. Cluster security group

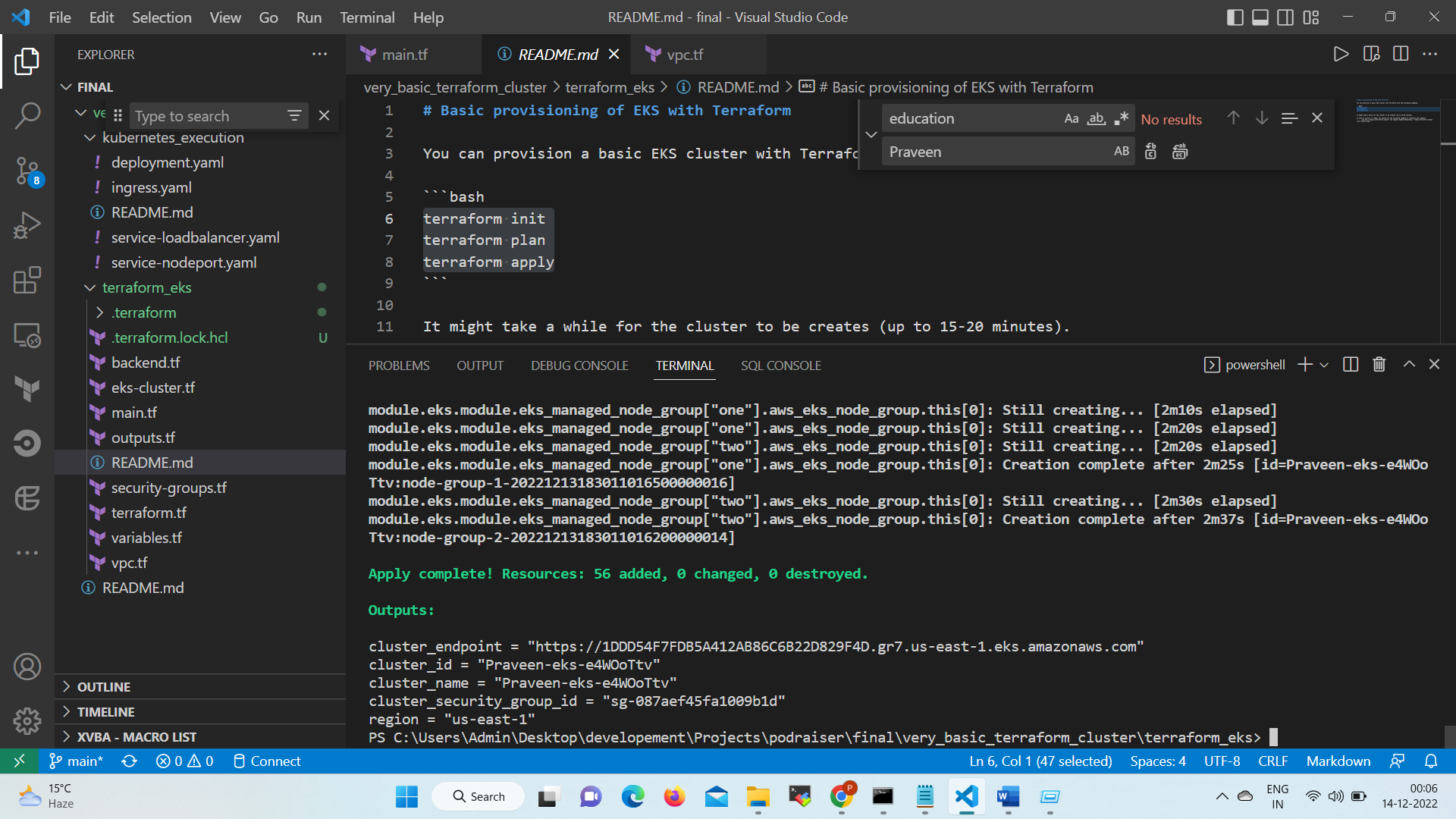
## Step6 Create the cluster by following steps

Clone the repo <https://github.com/My-Infra-Repos/very_basic_terraform_cluster.git>

Navigate to terraform\_eks folder & run the following commands

1. terraform init
2. terraform plan
3. terraform apply

Please fine the final out below once you confirm the cluster creation



## Step6 configure kubectl

Once the cluster is ready we run the following command to configure kubectl

aws eks --region $(terraform output -raw region) update-kubeconfig --name $(terraform output -raw cluster\_name)

You ill get the output similar to below

Added new context arn:aws:eks:us-east-1:613476214421:cluster/Praveen-eks-6OcUQ2ip to C:\Users\Admin\.kube\config

## Step7 Create the deployment & access the services

Navigate to kubernetes\_execution folder from parent directory & run the following commands

kubectl apply -f deployment.yaml

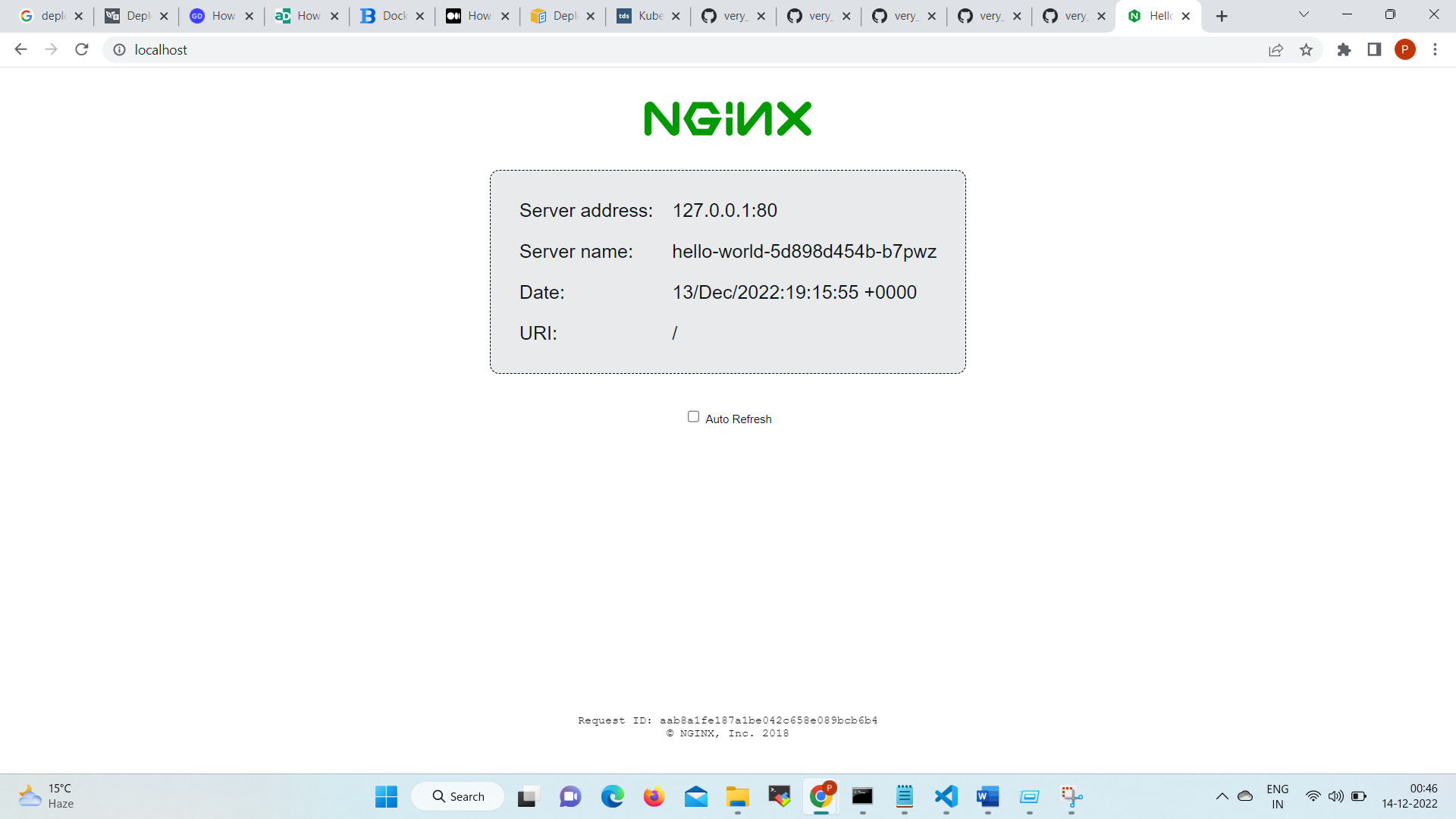
kubectl apply -f service-nodeport.yaml

kubectl apply -f ingress.yaml

We will get our ngnix image accessible to our browser. For a quick check we can also confirm the same with

kubectl port-forward <hello-kubernetes-wfjdz> 80:80

& then opening the localhost in browser we will get the below



# Q2 For deployment you could explain how a tool such as Gitlab AutoDevOps (<https://docs.gitlab.com/ee/topics/autodevops/>) or Harness (<https://harness.io/kubernetes/>) would work, or alternatively explain how the container could be deployed using standard Kubernetes resources.

## Answer: Steps to Deploy container to Kubernetes using standard Kubernetes resources

1. Create a Dockerfile : We need to package our application inside a docker image in order to get the same run inside a pod.
2. Building an Image from Dockerfile: Once the docker file is created build the docker image from docker file (command: docker build -t Tag\_Name). Validate if the Image is created and Listed. Optionally upload to docker Hub to share with the world or upload on
3. Create Manifest file for Kubernetes. We could use 2 separate manifest files or we can specify all in a single file (using --- yaml separator). Please find our deployment manifest files

<https://github.com/My-Infra-Repos/very_basic_terraform_cluster/blob/main/kubernetes_execution/deployment.yaml#:~:text=apiVersion%3A%20apps,containerPort%3A%2080>

We can create the deployment using

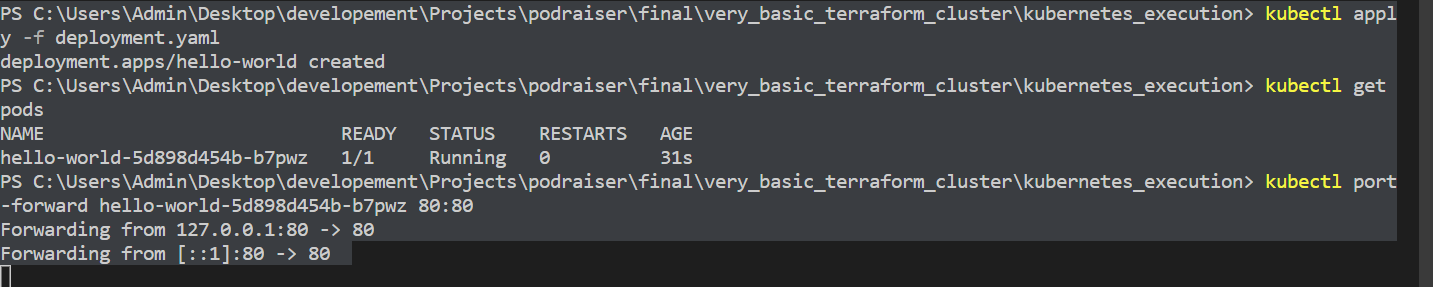
kubectl apply -f deployment.yaml

Once the deployment is created we can run the following command for a quick check

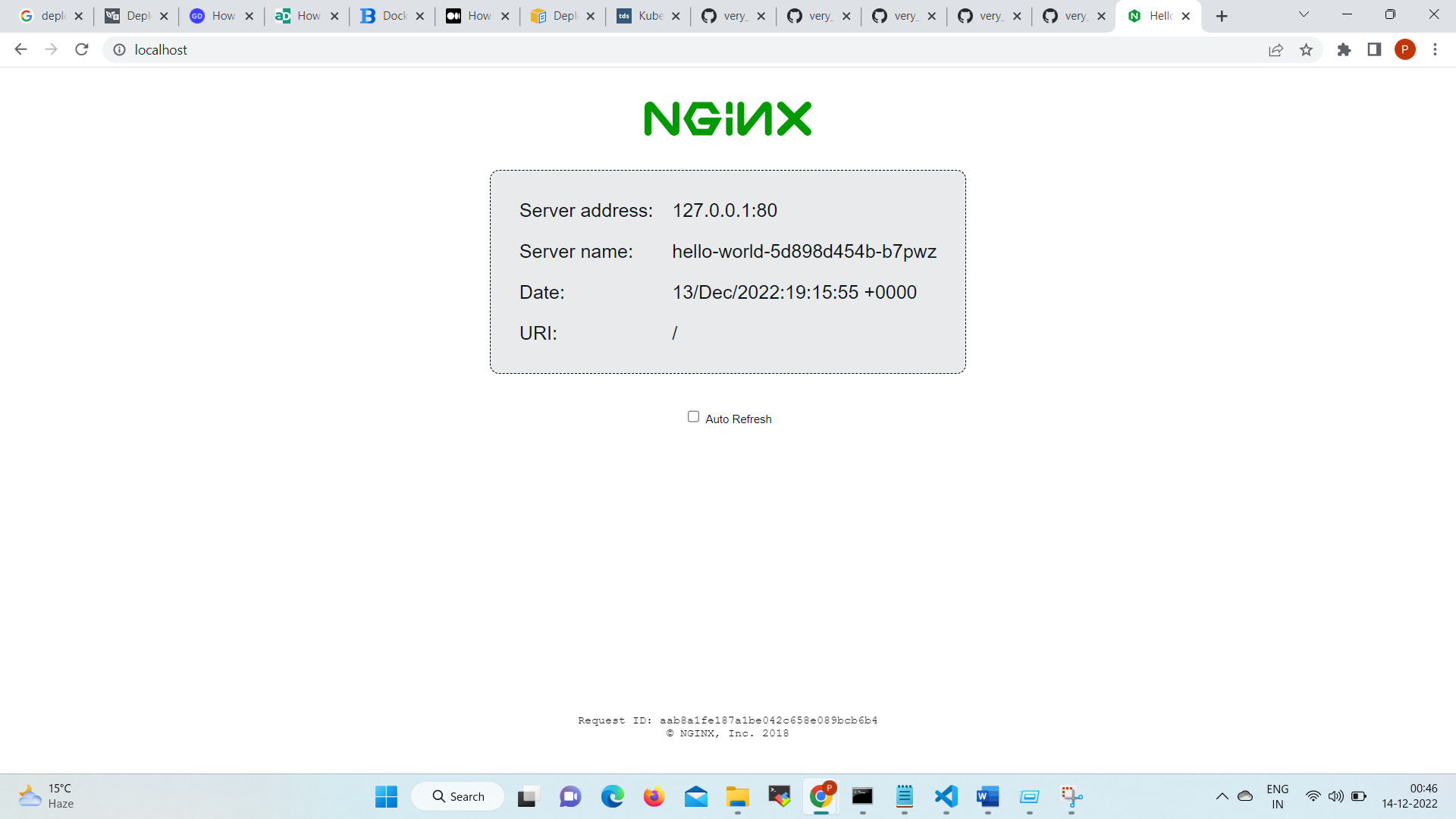
kubectl get pods

Also we can quickly check our application by running

kubectl port-forward <hello-kubernetes-wfjdz> 80:80



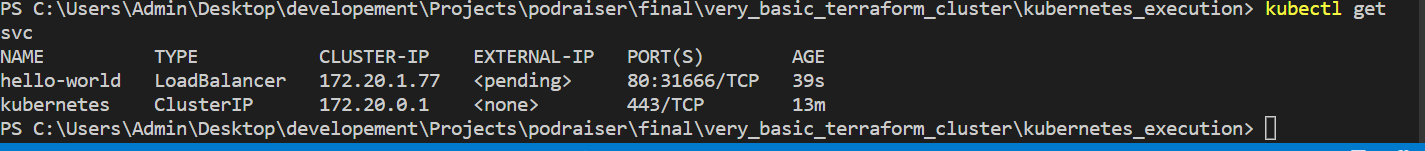
& the accessing it at local host we will get the following screen shot



If we want to route live traffic to the Pod, we could create a load balancer service by applying below file.

<https://github.com/My-Infra-Repos/very_basic_terraform_cluster/blob/main/kubernetes_execution/service-loadbalancer.yaml#:~:text=Blame-,apiVersion%3A%20v1,name%3A%20hello%2Dworld,-Footer>

kubectl apply -f service-loadbalancer.yaml



**As we run the command, AWS provisions a Classic Load Balancer and connects it to your Pod. However It might take a while for the load balancer to be provisioned. Eventually we can describe the Service and retrieve the load balancer's endpoint.**

kubectl describe service hello-world

Note: Now if we visit that URL in our browser, we should see the app live. However there is an issue here. The load balancer that we created serves one service at the time. Also, it has no option to provide intelligent routing based on paths. So if we have multiple services that need to be exposed, we need to create the same amount of load balancers. If we use a Service to type: LoadBalancer for each of them, it will too costly.

1. We can use Ingress to resolve those issue. Ingress has two parts:
2. The first is the Ingress manifest which is the same as Deployment or Service in Kubernetes. This is defined by the kind part in the YAML manifest.
3. The second part is the Ingress controller. This is the actual part that controls the load balancers, so it knows how to serve the requests and forward the data to the Pods.
4. Ingress controller acts as a reverse proxy that routes the traffic to your Pods. Here we have used the ALB Ingress Controller — an Ingress controller that integrates with the Application Load Balancer. PFB link for the same

<https://github.com/My-Infra-Repos/very_basic_terraform_cluster/blob/main/kubernetes_execution/ingress.yaml#:~:text=apiVersion%3A%20extensions,servicePort%3A%2080>

Ingress routes the traffic based on paths, domains, headers, etc., which consolidates multiple endpoints in a single resource that runs inside Kubernetes.

1. Check the newly created POD in Kubernetes DashBoard

# Q.3 Also Include high level\* steps to make this accessible from the outside world (eg. DNS)

Answer: The easiest way to access this service is the use of external DNS (https://github.com/kubernetes-sigs/external-dns) .

However we need a hosted zone inside our account as a pre-requirement which we can use with external-dns.

In order to let [external-dns](https://github.com/kubernetes-sigs/external-dns) make changes to the route53 zone, we need to create an IAM role and attach that to a service account. Once we do terraform apply we can find that it’s bound to a service account called external-dns in the the external-dns namespace.

Now we need to create the same in k8s

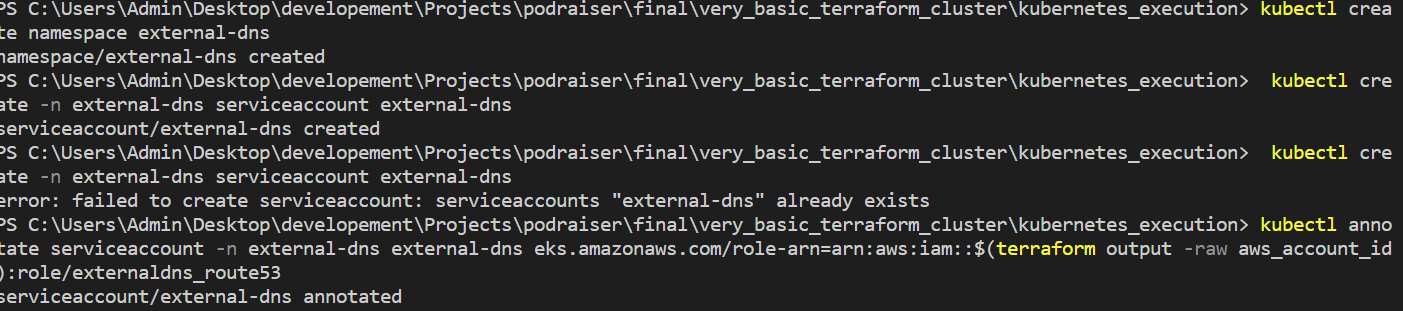
$ kubectl create namespace external-dns

namespace/external-dns created

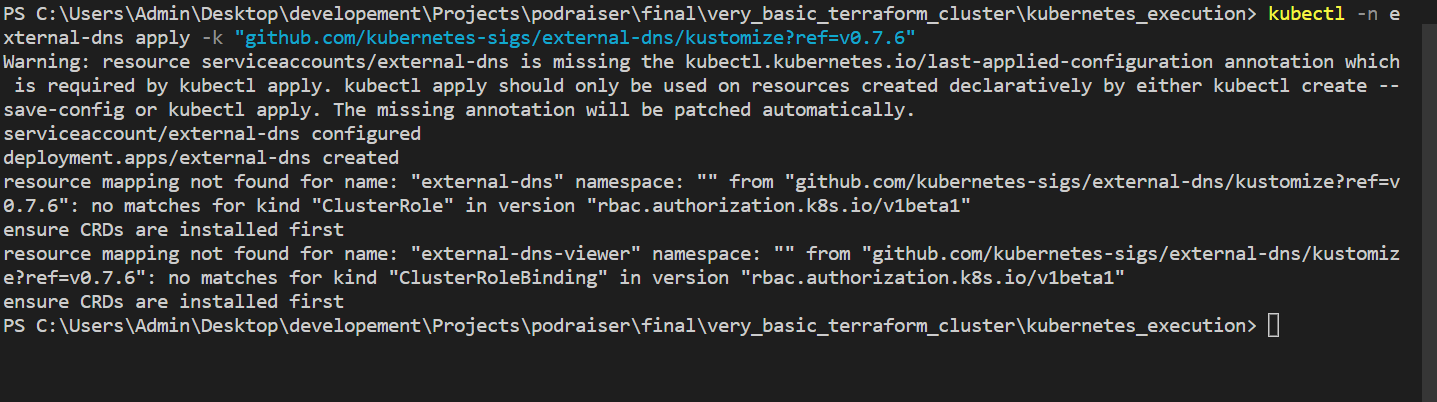
$ kubectl create -n external-dns serviceaccount external-dns

serviceaccount/external-dns created

$ kubectl annotate serviceaccount -n external-dns external-dns eks.amazonaws.com/role-arn=arn:aws:iam::$(terraform output -raw aws\_account\_id):role/externaldns\_route53



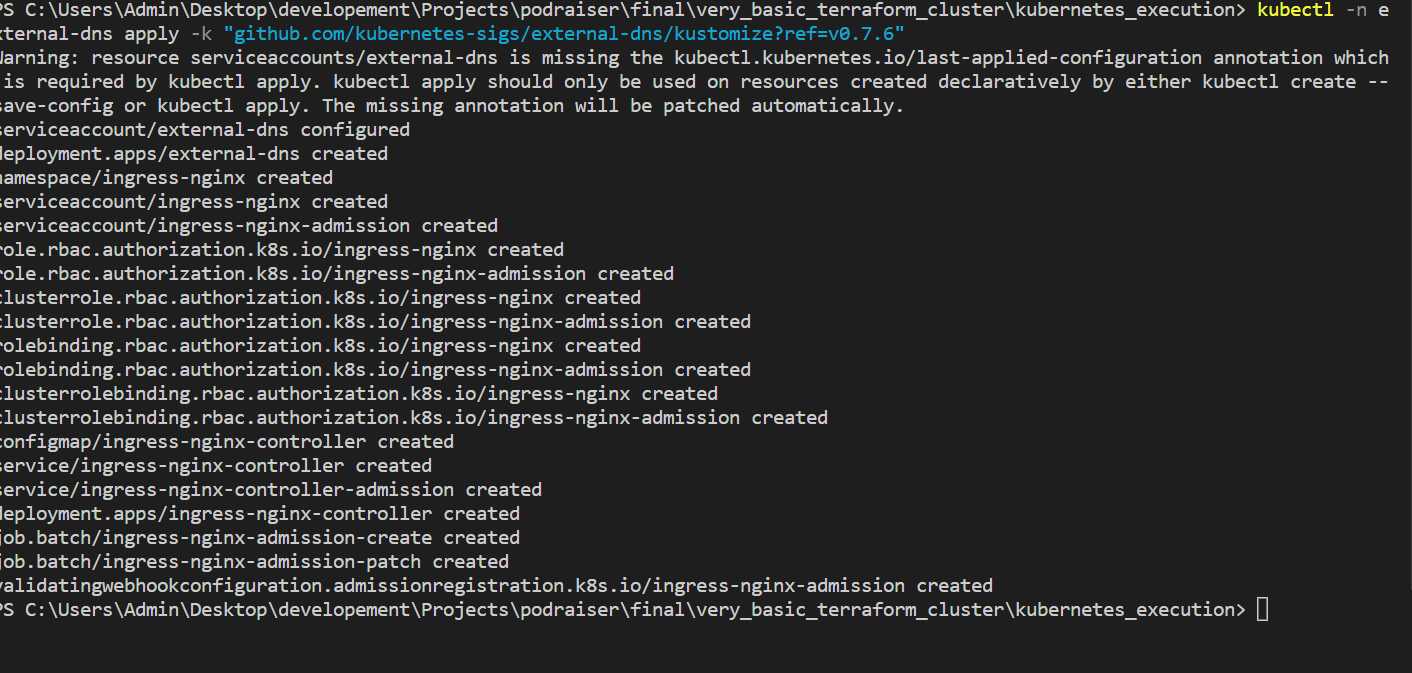
Finally Deploy external-dns:



This will set external-dns to look for hostnames in the ingress configuration and create a record for them that points to the ingress controller’s load balancer.

Now, deploy ingress-nginx. This configuration is set up to request an Amazon Elastic Network Load Balancer and attach it to the ingress controller:

kubectl apply -k "github.com/kubernetes/ingress-nginx.git/deploy/static/provider/aws?ref=controller-v0.44.0"



Sometimes we might need to scale the deployment if we want some resilience. Usually I scale it to 3

kubectl scale -n ingress-nginx --replicas=3 deployment ingress-nginx-controller

Note: I don’t own any dns zone as of now therefore I am unable to share the complete output for the same.

## Q4, Write a bash script to print a list of all users on the system in the format '$userName $userId $userHomeDirectory'

Answer: When we run the below script with sudo permissions it generates the output in file.txt file for the same. The script is tested in Linux environment.

#!/bin/bash

getent passwd | awk -F: '{print $1 " " $3 " " $6}' > file.txt