For Backend:  
  
Clone dev branch from the Prashant repo.

In our dev branch we don’t have index.js but we have server.js.

Commands:

Run npm install

node server.js

* After run the command we got an error of mongo connectivity, so we need to create .env file in the root and add the PORT and Mongo\_URL.
* After adding the file again run the command “node server.js”.   
   Now you can checkon browser localhost:[Port]. It’s working fine.



Now time to write the docker file for backend

FROM node:18

WORKDIR /app

COPY package.json .

RUN npm install

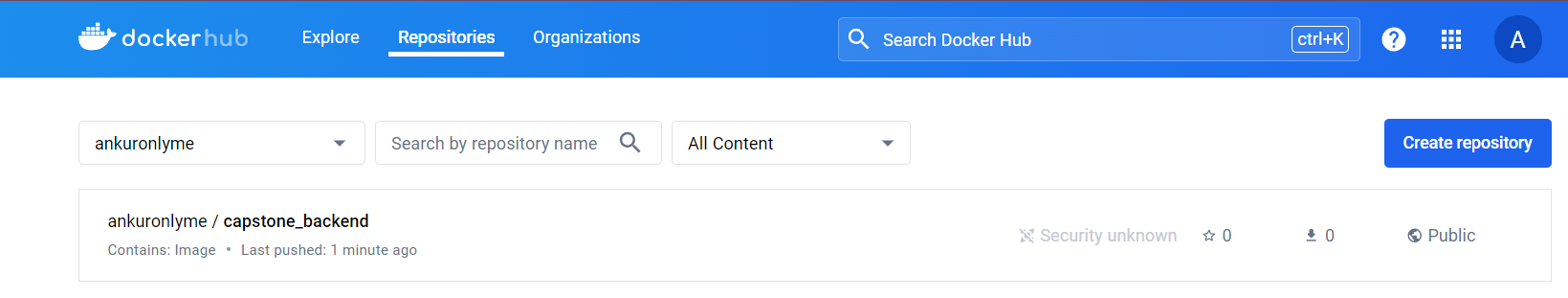
COPY . .

EXPOSE 3001

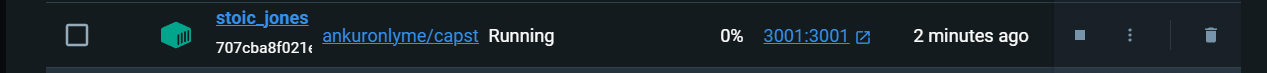
CMD ["node", "server.js"]

After write the docker file, now build the docker image and push it into the docker hub:

* docker build -t lms-backend:latest .
* docker tag lms-backend:latest ankuronlyme/capstone\_backend:v1
* docker push ankuronlyme/capstone\_backend:v1



Make container for backend image:  
  
Run: docker run -dp 3001:3001 -e "PORT=3001" -e "MONGO\_URL=mongodb+srv://TravelMemory:Travel@ankurcluster.h2znnvu.mongodb.net/Travel" -e "GITHUB\_CLIENT\_SECRET=clientpassword" ankuronlyme/capstone\_backend:v1



For Frontend:  
  
Clone angular branch from the Prashant repo.

Commands:

* Run npm install –force (installation of dependencies and packages)
* source ./modify\_quill\_editor.sh
* Run npm start
* Run npm install -g @angular/cli (installation of angular cli)

Check versions of the below:

* node --version
* npm –version
* ng –version
* Run ng build –production (To build your application)
* Run dir dist (to check the location of your build)

Now time to write the docker file for frontend

# Use a Node.js image as the Build Stage

FROM node:18 AS build

# Set the working directory

WORKDIR /app

# Copy package.json and package-lock.json and install dependencies

COPY package\*.json ./

# Installation of all dependencies

RUN npm install --force

RUN npm install -g @angular/cli

# Install dos2unix to convert line endings

RUN apt-get update && apt-get install -y dos2unix

# Copy the script into the Docker image

COPY modify\_quill\_editor.sh .

# Convert line endings to Unix format to remove carriage return characters

RUN dos2unix modify\_quill\_editor.sh

# Make the script executable

RUN chmod +x modify\_quill\_editor.sh

# Execute the script

RUN ./modify\_quill\_editor.sh

# Copy the rest of the application code

COPY . .

# Build the Angular application

RUN npm run build --prod

# Use a lightweight web server to serve the frontend and deployment Process

FROM nginx:alpine

COPY --from=build /app/dist/lms-front-ang /usr/share/nginx/html

# Expose the port on which the frontend will run

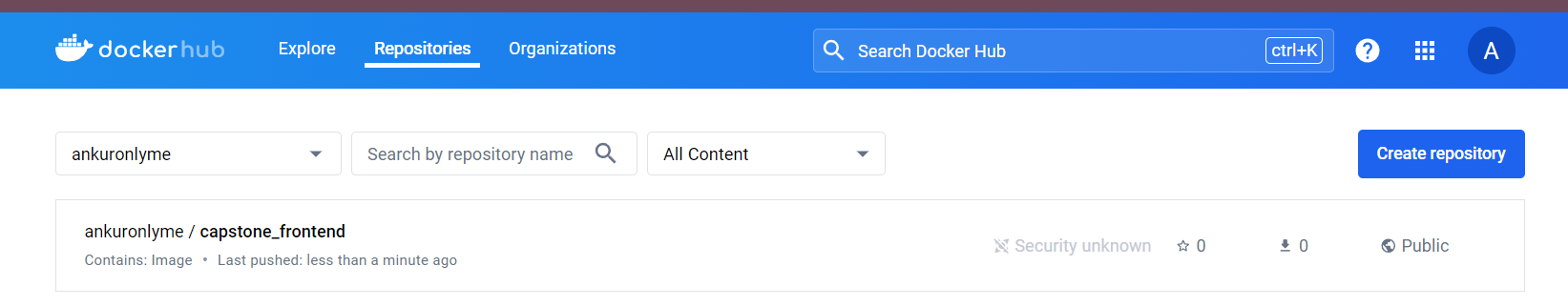
EXPOSE 80

# Start the web server

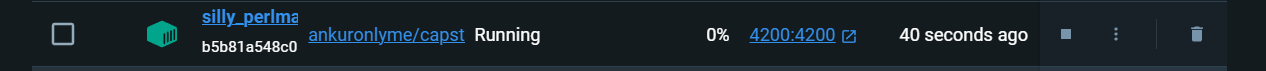
CMD [ "nginx", "-g", "daemon off;" ]

After write the docker file, now build the docker image and push it into the docker hub:

* docker build -t lms-frontend:latest .
* docker tag lms-frontend:latest ankuronlyme/capstone\_frontend:v1
* docker push ankuronlyme/capstone\_frontend:v1



docker run -dp 4200:4200 ankuronlyme/capstone\_frontend:v2



Setup Jenkins Pipeline:

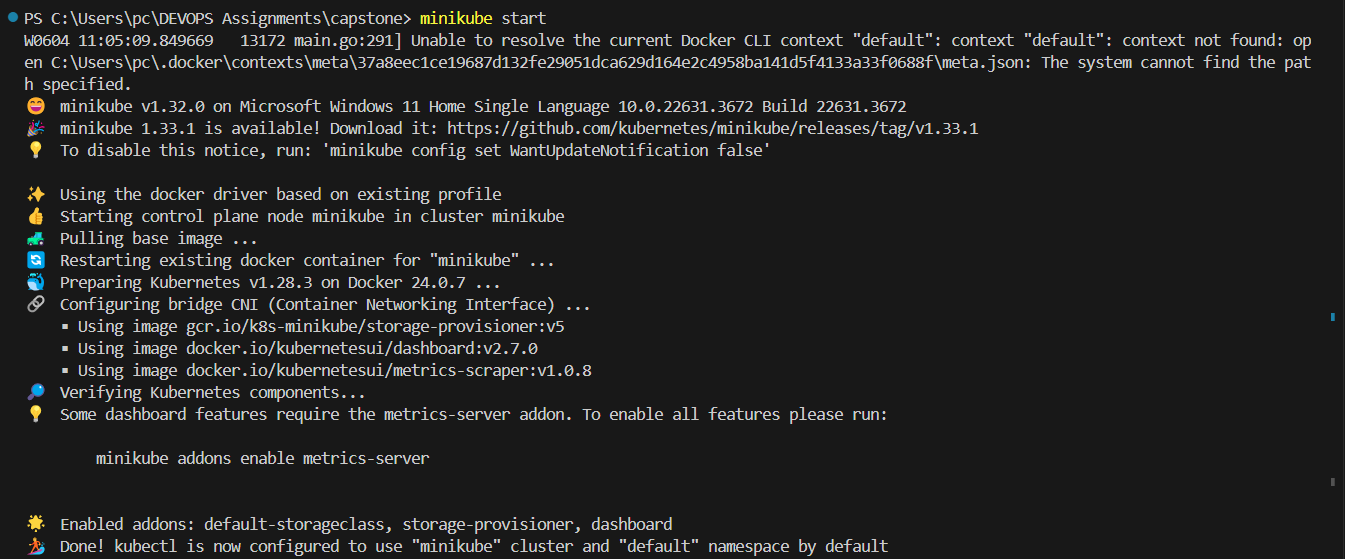
* Install Jenkins and Required Plugins:
* Install Jenkins on your server.
* Install the following plugins:
* Docker
* Kubernetes
* Terraform
* GitHub
* Create Jenkins Pipeline:
* Navigate to Jenkins dashboard and create a new pipeline job.
* Set environment varibles in the pipline
* Write stages to upto docker build images for frontend and backend

Write Kubernetes Files:

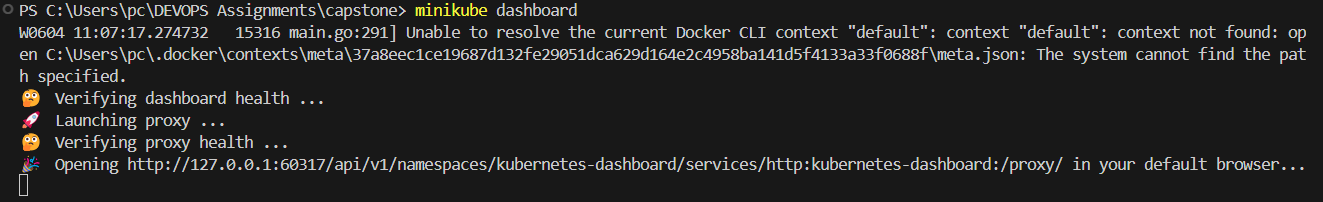
* For frontend
* For backend
* For compiler

Execute Files through Minikube:

Run command: minikube start



Run command: minikube dashboard



Apply Kubernetes Files:

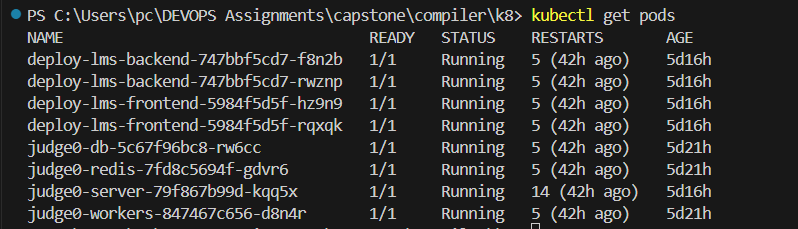
* kubectl apply -f frontend-deployment.yml
* kubectl apply -f backend-deployment.yml
* kubectl apply -f frontend-service.yml
* kubectl apply -f backend-service.yml

For Compiler the sequence to run the command is like:

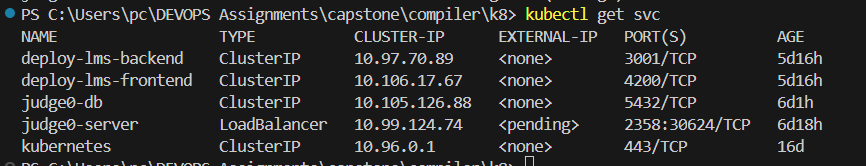
* kubectl apply -f configmap.yaml
* kubectl apply -f postgres-secret.yaml
* kubectl apply -f postgres-pvc.yaml
* kubectl apply -f judge0-db-deployment.yaml
* kubectl apply -f postgres-service.yaml
* kubectl apply -f redis-pvc.yaml
* kubectl apply -f judge0-redis-deployment.yaml
* kubectl apply -f judge0-workers-deployment.yaml
* kubectl apply -f compiler\_service.yaml
* kubectl apply -f judge0-server-deployment.yaml

After excetution of the above files now check the Pods and SCV

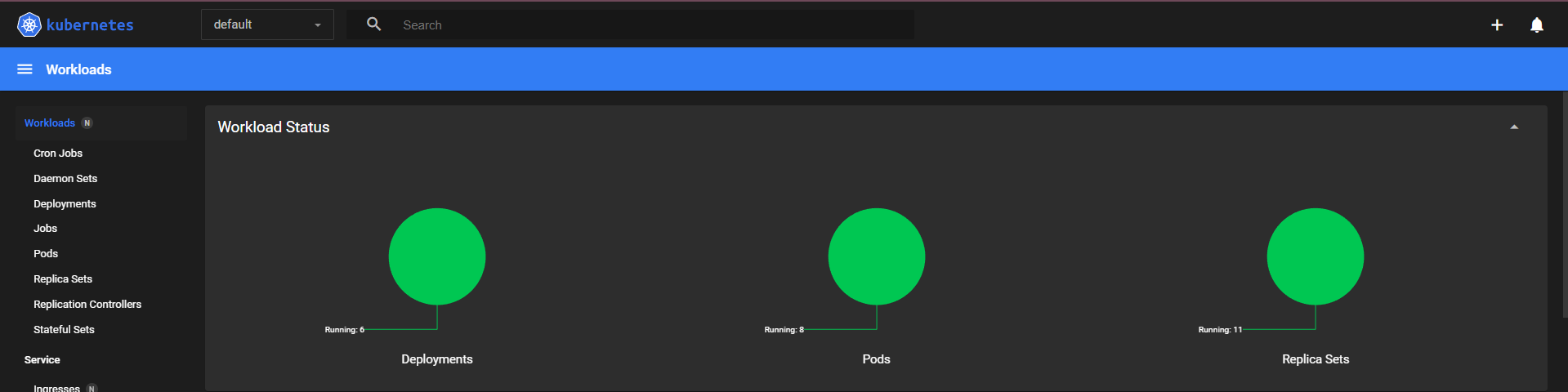
Run Command: kubectl get pods:



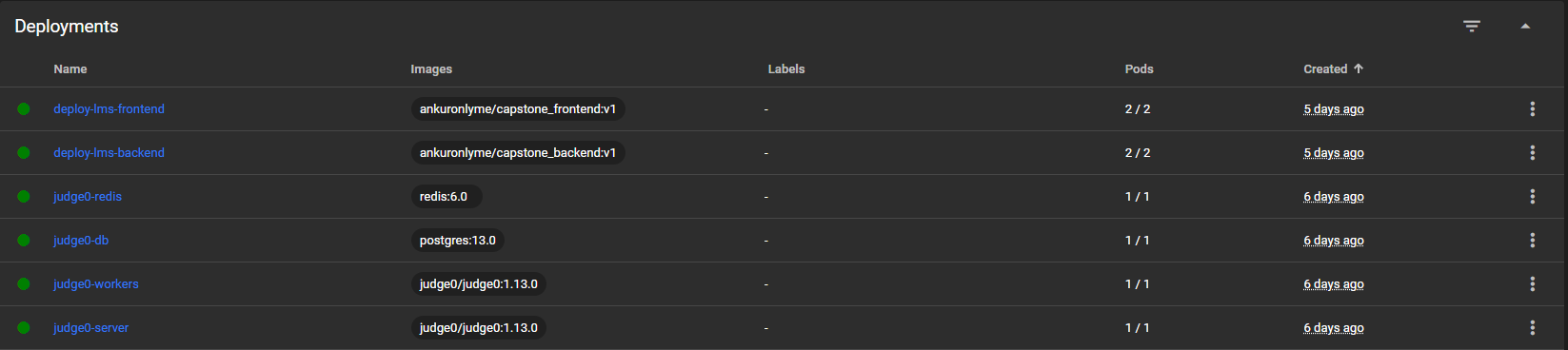
Run command: kubectl get svc:



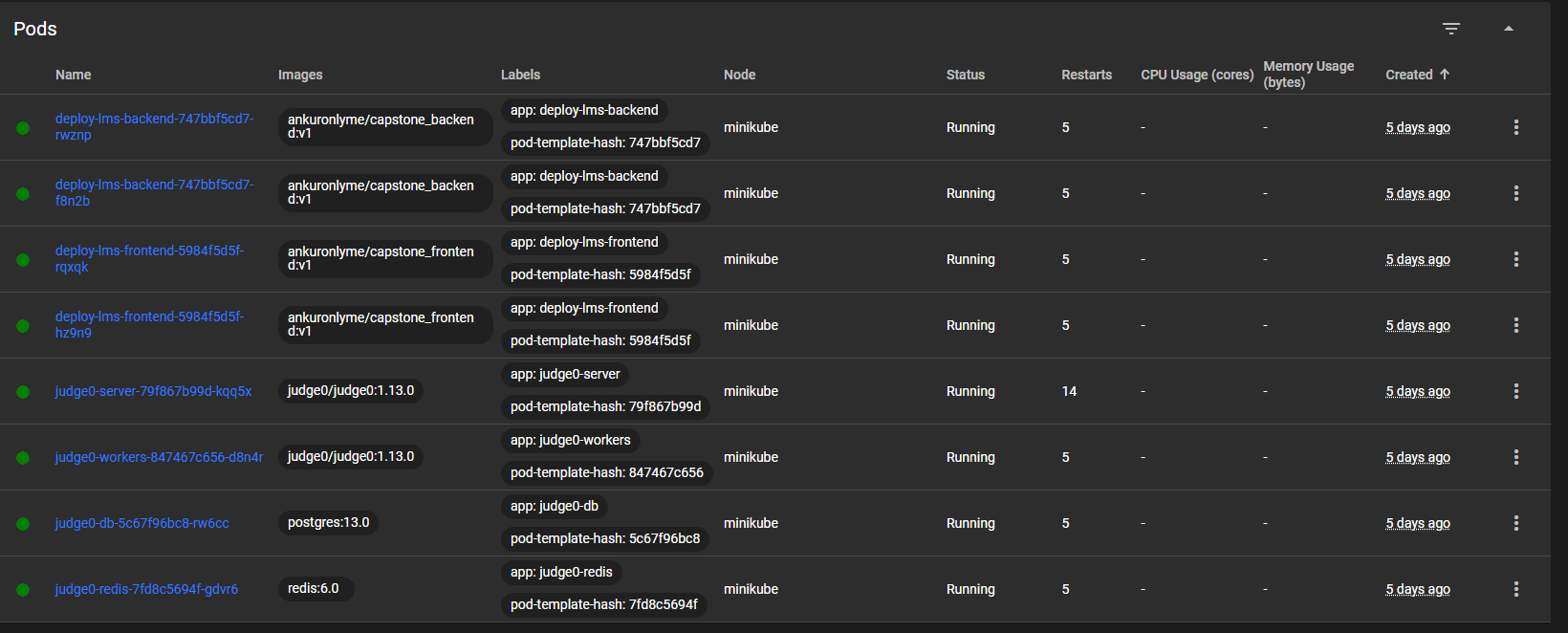
Check On Minikube Dashboard:



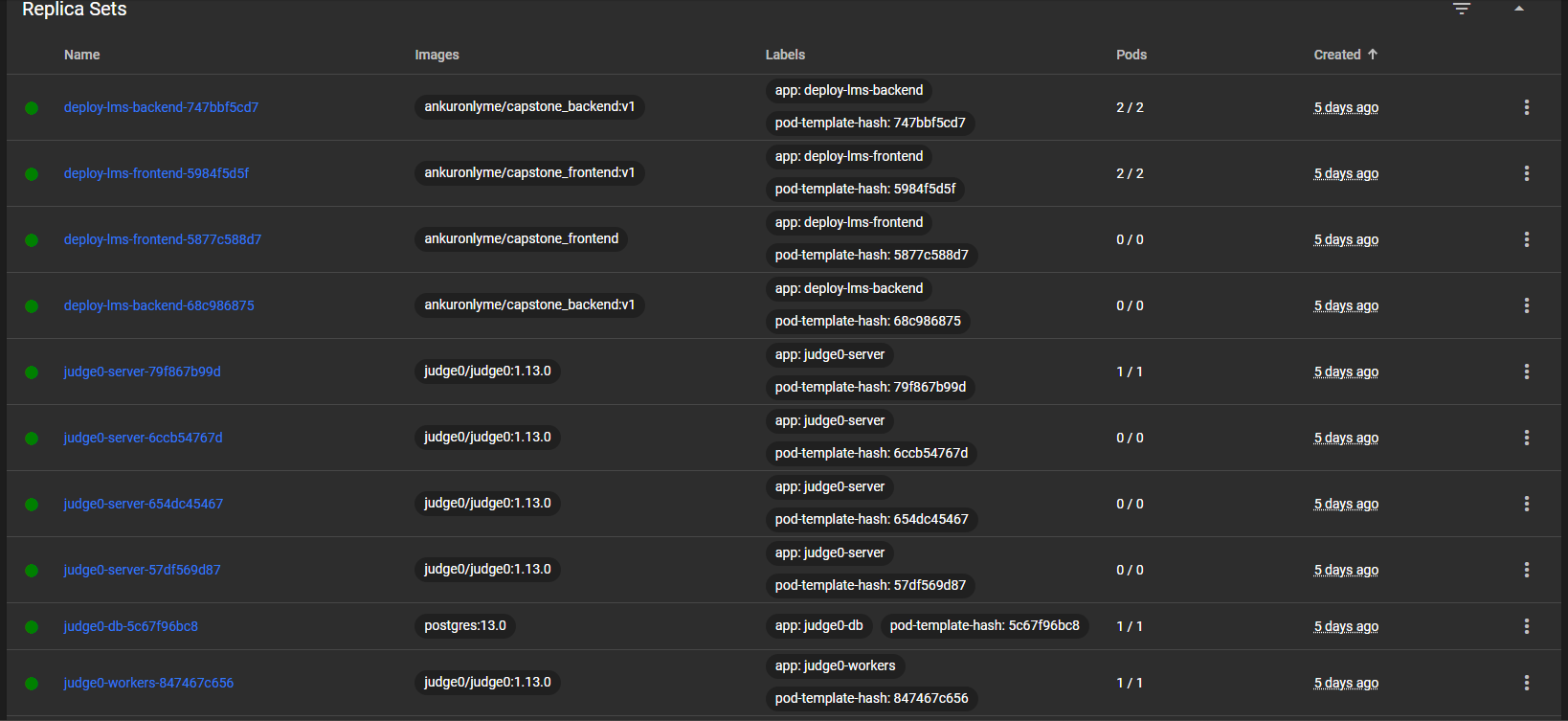
Deployment:



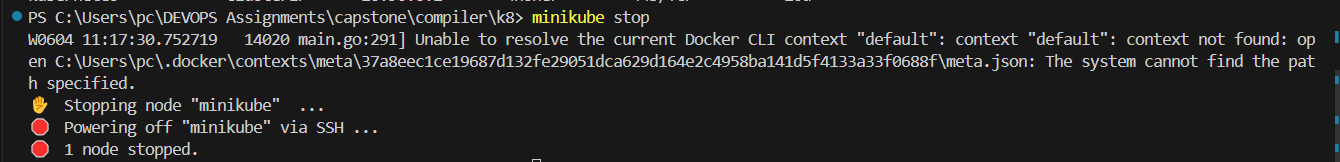
Pods:



Replica Sets:



To stop your minikube excute command: minikube stop



Write Terraform Files:

* Install Terraform
* Terraform Configuration:
* Create file for Terraform configuration: main.tf to manage AWS resources.

Main.tf:

This file sets up your AWS infrastructure using Terraform. Here's what each part does:

**Terraform Configuration:**

#### **Providers and Terraform Block:** Specifies required providers (random and aws) with versions and Terraform minimum version.

terraform {

  required\_providers {

    random = {

      source  = "hashicorp/random"

      version = "3.1.0"  # Specify the version of random provider required

    }

    aws = {

      source  = "hashicorp/aws"

      version = ">= 2.0.0"

    }

  }

  required\_version = ">= 1.1"

}

**AWS Provider Configuration:** Configures the AWS provider with the specified region (ap-south-1).

provider "aws" {

  region = "ap-south-1"  # Replace with your desired AWS region

}

**Fetch Availability Zones:** Fetches availability zones in the configured region.

data "aws\_availability\_zones" "available" {}

**Local Variables:** Stores availability zone names fetched from aws\_availability\_zones data source.

locals {

  availability\_zones = data.aws\_availability\_zones.available.names

}

**IAM Roles:** Fetches existing IAM roles (eks-cluster-role and eks-node-group-role) for EKS.

# Use existing IAM roles for EKS Cluster and Node Group

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

  name = "eks-cluster-role"

}

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

  name = "eks-node-group-role"

}

**IAM Role Policy Attachments:** Attaches required IAM policies   
(AmazonEKSClusterPolicy, AmazonEKSServicePolicy, AmazonEKSWorkerNodePolicy, AmazonEKS\_CNI\_Policy, AmazonEC2ContainerRegistryReadOnly) to their respective roles.

# Attach the required IAM policies to the EKS role

resource "aws\_iam\_role\_policy\_attachment" "eks\_cluster\_AmazonEKSClusterPolicy" {

  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"

  role       = data.aws\_iam\_role.eks\_cluster\_role.name

}

resource "aws\_iam\_role\_policy\_attachment" "eks\_cluster\_AmazonEKSServicePolicy" {

  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"

  role       = data.aws\_iam\_role.eks\_cluster\_role.name

}

# Attach required IAM policies to the Node Group Role

resource "aws\_iam\_role\_policy\_attachment" "eks\_node\_AmazonEKSWorkerNodePolicy" {

  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"

  role       = data.aws\_iam\_role.eks\_node\_group\_role.name

}

resource "aws\_iam\_role\_policy\_attachment" "eks\_node\_AmazonEKS\_CNI\_Policy" {

  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKS\_CNI\_Policy"

  role       = data.aws\_iam\_role.eks\_node\_group\_role.name

}

resource "aws\_iam\_role\_policy\_attachment" "eks\_node\_AmazonEC2ContainerRegistryReadOnly" {

  policy\_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"

  role       = data.aws\_iam\_role.eks\_node\_group\_role.name

}

**VPC and Subnets:**

* Creates a VPC (eks-vpc) with CIDR block 10.0.0.0/16.
* Creates three public subnets (eks-public-subnet-${count.index}) across different availability zones.

# Create the VPC

resource "aws\_vpc" "main" {

  cidr\_block = "10.0.0.0/16"

  tags = {

    Name = "eks-vpc"

  }

}

resource "aws\_subnet" "public" {

  count = 3

  vpc\_id     = aws\_vpc.main.id

  cidr\_block = cidrsubnet(aws\_vpc.main.cidr\_block, 8, count.index)

  availability\_zone = element(local.availability\_zones, count.index)

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "eks-public-subnet-${count.index}"

  }

}

**Internet Gateway and Route Table:**

* Creates an Internet Gateway (eks-gateway) and associates it with the VPC.
* Creates a public route table (eks-public-route-table) with a default route to the Internet Gateway.
* Associates each public subnet with the public route table.

resource "aws\_internet\_gateway" "gw" {

  vpc\_id = aws\_vpc.main.id

  tags = {

    Name = "eks-gateway"

  }

}

resource "aws\_route\_table" "public" {

  vpc\_id = aws\_vpc.main.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.gw.id

  }

  tags = {

    Name = "eks-public-route-table"

  }

}

resource "aws\_route\_table\_association" "public" {

  count          = length(aws\_subnet.public[\*].id)

  subnet\_id      = element(aws\_subnet.public[\*].id, count.index)

  route\_table\_id = aws\_route\_table.public.id

}

**EKS Cluster:**

* Creates an EKS cluster (capstone\_cluster) with the specified IAM role (eks-cluster-role) and VPC configuration.
* Depends on IAM policy attachments for the cluster role.

# Create the EKS cluster

resource "aws\_eks\_cluster" "eks" {

  name     = "capstone\_cluster"

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

  vpc\_config {

    subnet\_ids = aws\_subnet.public[\*].id

  }

  depends\_on = [

    aws\_iam\_role\_policy\_attachment.eks\_cluster\_AmazonEKSClusterPolicy,

    aws\_iam\_role\_policy\_attachment.eks\_cluster\_AmazonEKSServicePolicy,

  ]

}

**EKS Node Group:**

* Creates an EKS node group (lms-node-group) within the EKS cluster.
* Specifies node group configurations like instance types (t3a.large), disk size, and scaling settings (desired\_size, max\_size, min\_size).
* Depends on IAM policy attachments for the node group role.

# Create the EKS Node Group

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

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

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

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

  subnet\_ids      = aws\_subnet.public[\*].id

  scaling\_config {

    desired\_size = 4

    max\_size     = 7

    min\_size     = 4

  }

  instance\_types = ["t3a.large"]

  disk\_size      = 60

  depends\_on = [

    aws\_iam\_role\_policy\_attachment.eks\_node\_AmazonEKSWorkerNodePolicy,

    aws\_iam\_role\_policy\_attachment.eks\_node\_AmazonEKS\_CNI\_Policy,

    aws\_iam\_role\_policy\_attachment.eks\_node\_AmazonEC2ContainerRegistryReadOnly,

  ]

}

**Outputs:**

* Provides outputs for the EKS cluster endpoint (cluster\_endpoint) and security group ID (cluster\_security\_group\_id).

output "cluster\_endpoint" {

  description = "EKS cluster endpoint"

  value       = aws\_eks\_cluster.eks.endpoint

}

output "cluster\_security\_group\_id" {

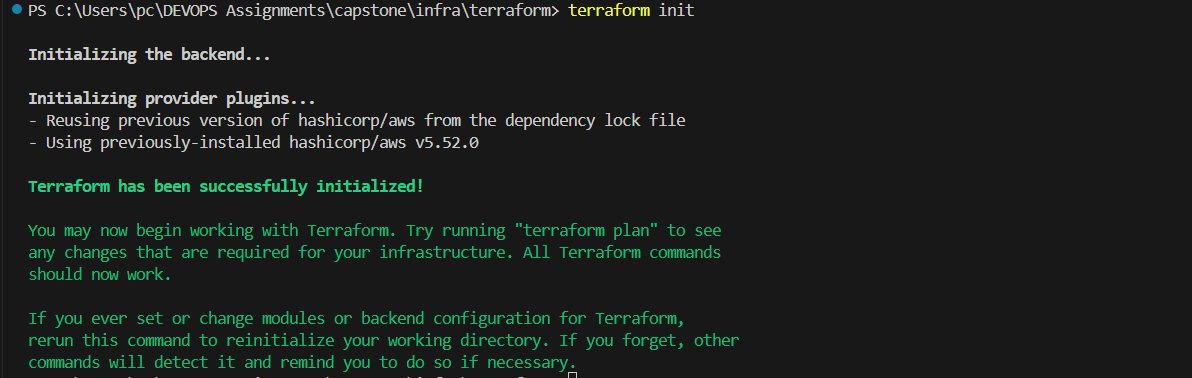
  description = "Security group ID of the EKS cluster"

  value       = aws\_eks\_cluster.eks.vpc\_config[0].cluster\_security\_group\_id

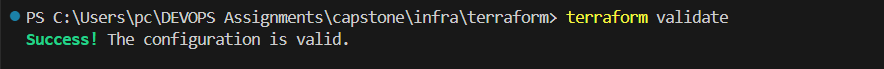
}

After complete the files now time to excute:

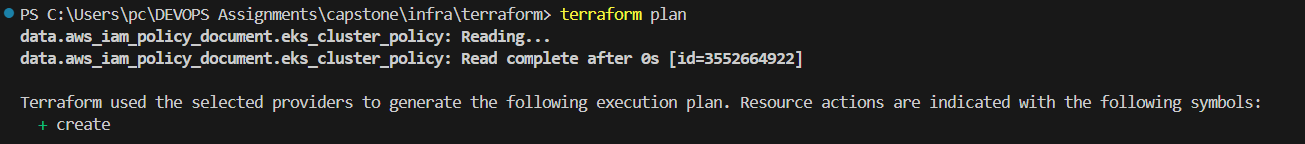
Run command: terraform init



Run command: terraform validate



Run command: terraform plan



Run command: terraform apply

