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

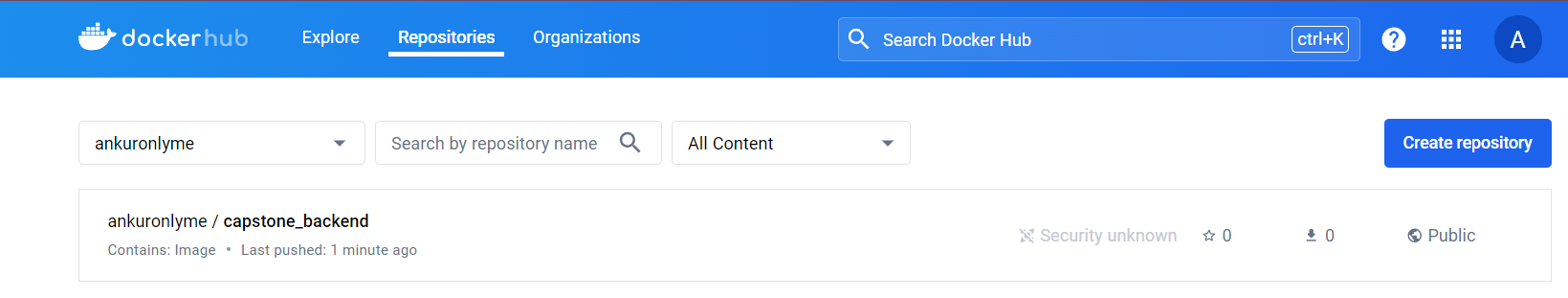
COPY . .

EXPOSE 3001

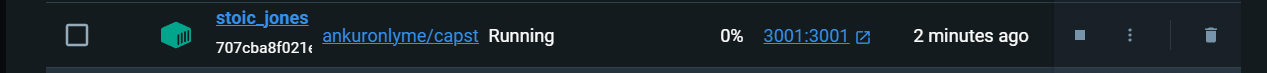
CMD ["node", "server.js"]

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

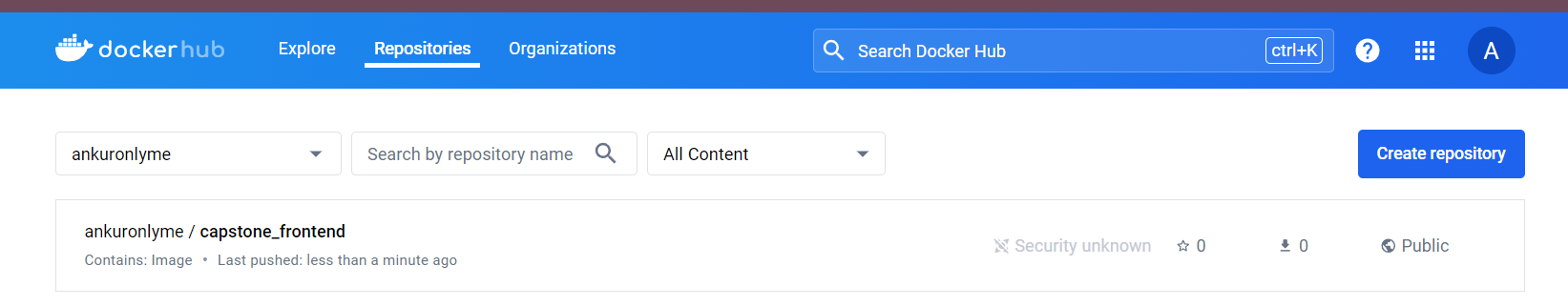
EXPOSE 80

# Start the web server

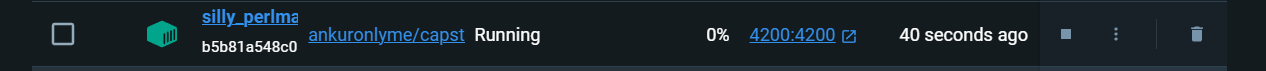
CMD [ "nginx", "-g", "daemon off;" ]

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:

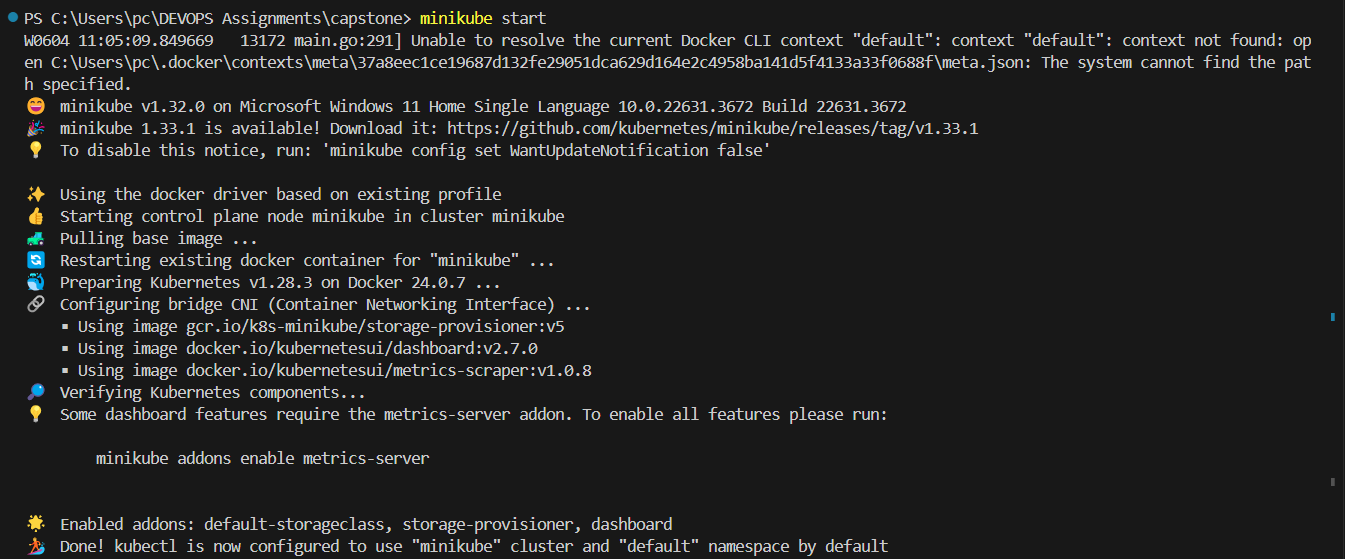
* Install Jenkins and Required Plugins:
* Install Jenkins on your server.
* Install the following plugins:
* Docker
* Kubernetes
* Terraform
* GitHub
* Create Jenkins Pipeline:
* Navigate to Jenkins dashboard and create a new pipeline job.
* Set environment varibles in the pipline
* Write stages to upto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

Write Kubernetes Files:

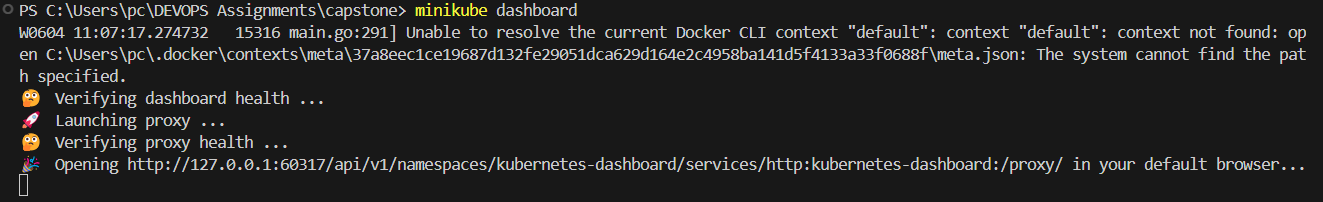
* For frontend
* For backend
* For compiler

Execute Files through Minikube:

Run command: minikube start



Run command: minikube dashboard



Apply Kubernetes Files:

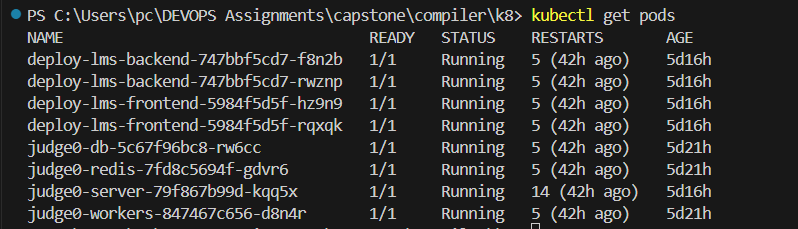
* kubectl apply -f frontend-deployment.yml
* kubectl apply -f backend-deployment.yml
* kubectl apply -f frontend-service.yml
* kubectl apply -f backend-service.yml

For Compiler the sequence to run the command is like:

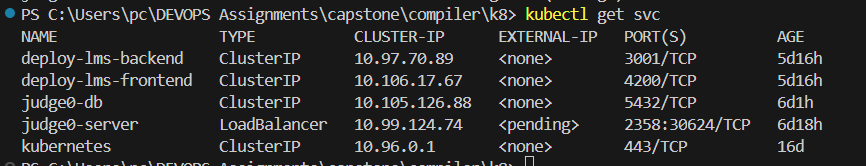
* kubectl apply -f configmap.yaml
* kubectl apply -f postgres-secret.yaml
* kubectl apply -f postgres-pvc.yaml
* kubectl apply -f judge0-db-deployment.yaml
* kubectl apply -f postgres-service.yaml
* kubectl apply -f redis-pvc.yaml
* kubectl apply -f judge0-redis-deployment.yaml
* kubectl apply -f judge0-workers-deployment.yaml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

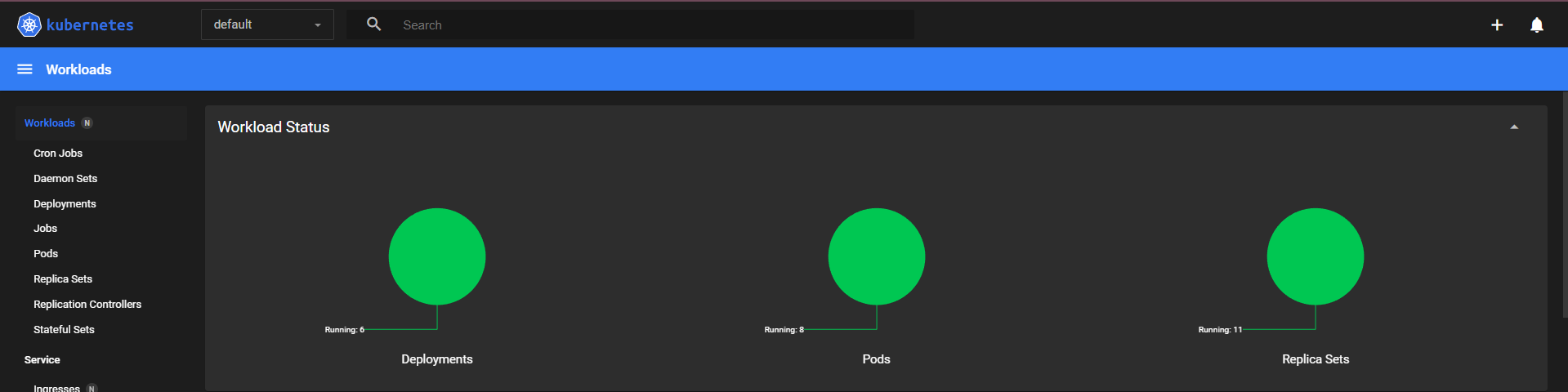
Run Command: kubectl get pods:



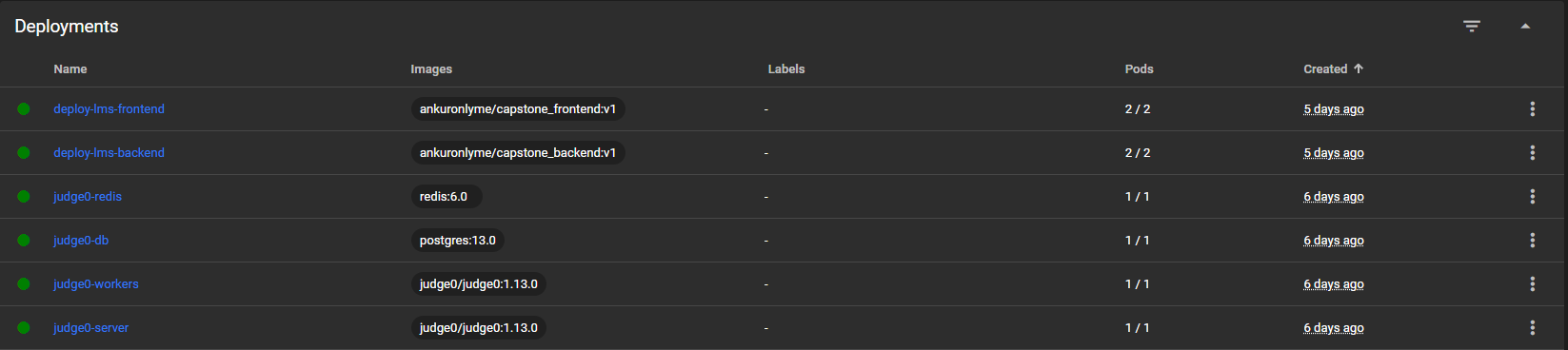
Run command: kubectl get svc:



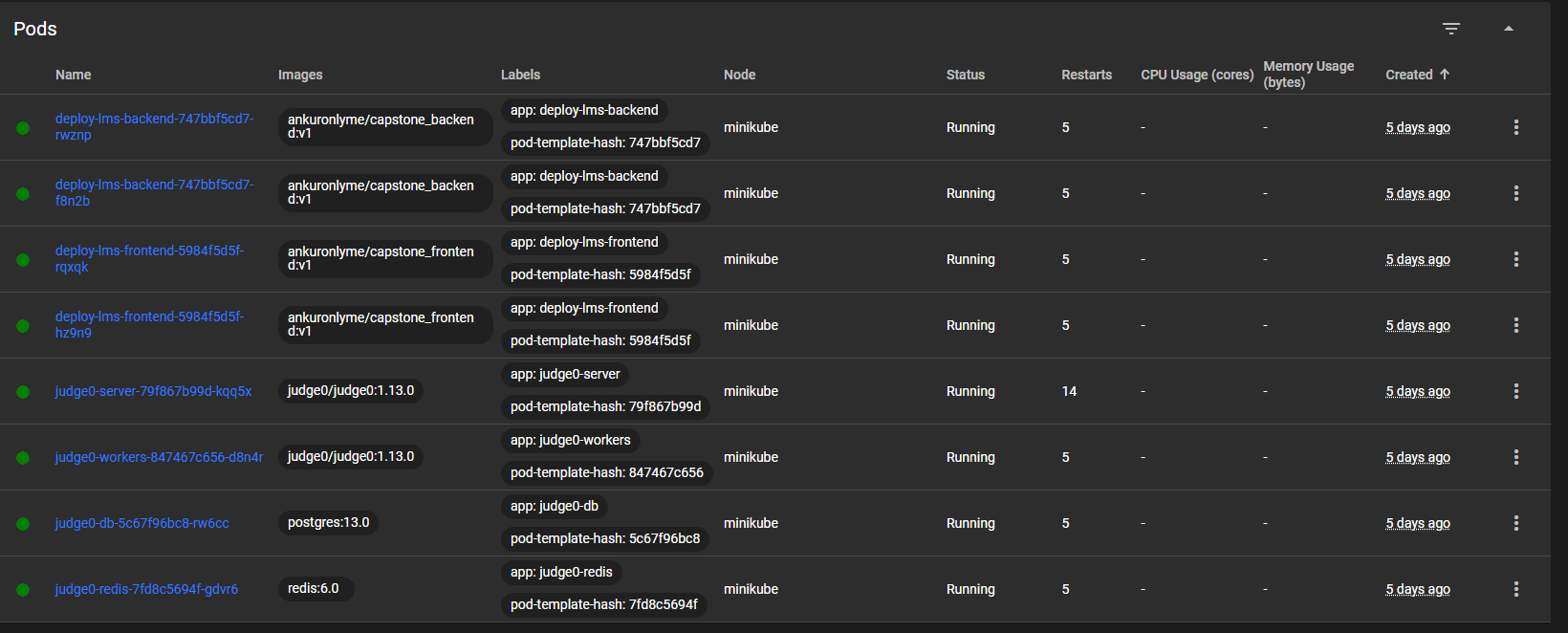
Check On Minikube Dashboard:



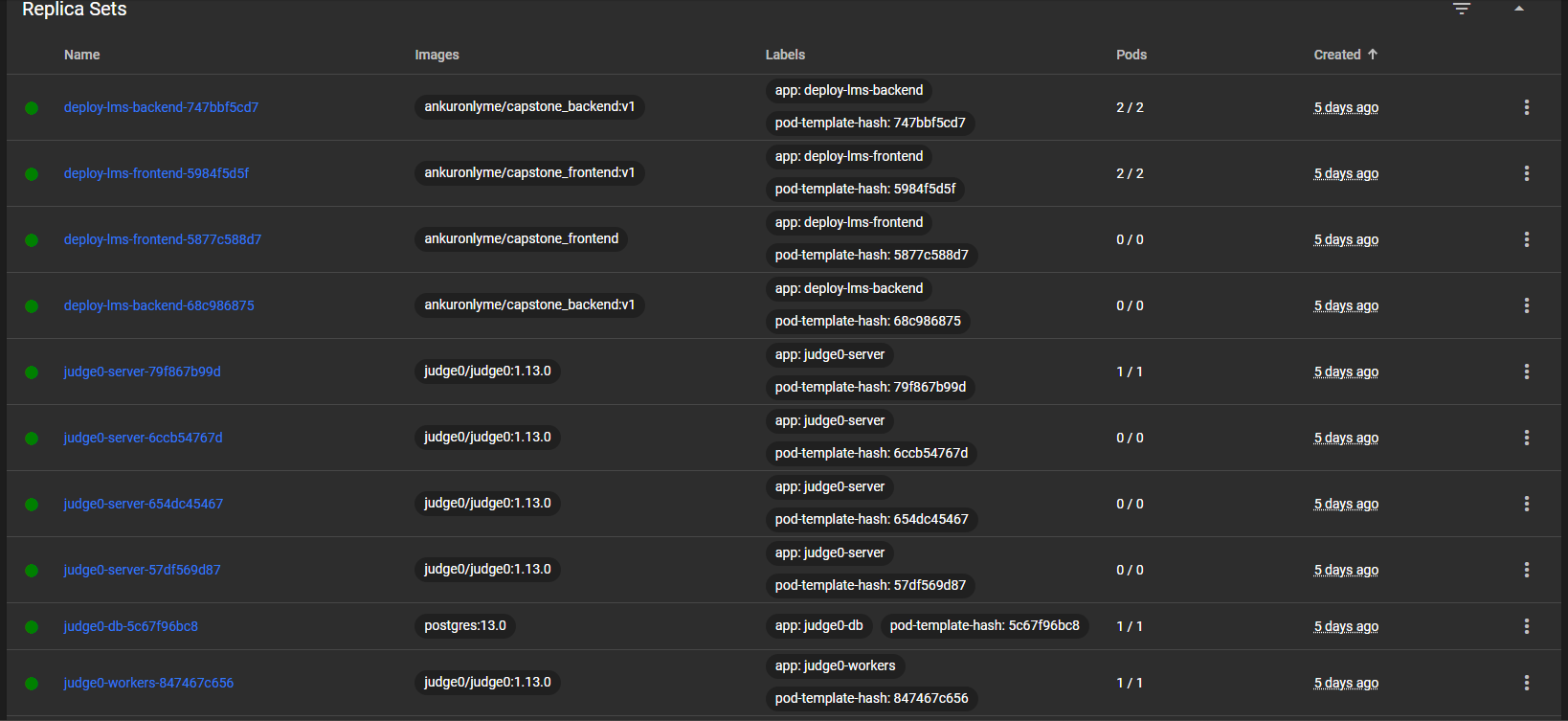
Deployment:



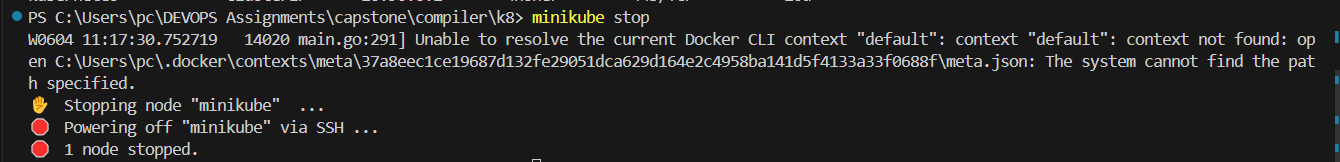
Pods:



Replica Sets:



To stop your minikube excute command: minikube stop



Write Terraform Files:

* Install Terraform
* Terraform Configuration:
* Create three primary files for Terraform configuration: main.tf, variables.tf, and outputs.tf. to manage your AWS resources.

Main.tf:

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

1. **Provider Block**: Tells Terraform you're using AWS and which region to work in.
2. **Creating a VPC**: Makes a private network for your resources to live in.
3. **Making a Public Subnet**: A section of your VPC where resources can talk directly to the internet.
4. **Making a Private Subnet**: Another section of your VPC where resources can't talk directly to the internet.
5. **Internet Gateway**: Connects your VPC to the outside world.
6. **Route Table**: Tells your VPC how to send and receive internet traffic.
7. **Linking Route Table and Public Subnet**: Connects the route table to the public subnet.
8. **Setting Up an EKS Cluster**: Creates a managed Kubernetes service on AWS.
9. **Creating an IAM Role for EKS**: Gives the EKS cluster the permissions it needs.
10. **Adding Policies to IAM Role**: Gives specific permissions to the IAM role.

Output.tf:

This file shows important information after Terraform finishes its work:

1. **VPC ID**: The unique ID of your VPC.
2. **Public Subnet ID**: The unique ID of your public subnet.
3. **Private Subnet ID**: The unique ID of your private subnet.
4. **EKS Cluster ID**: The unique ID of your EKS cluster.
5. **EKS Cluster Endpoint**: The web address where you can access your EKS cluster.

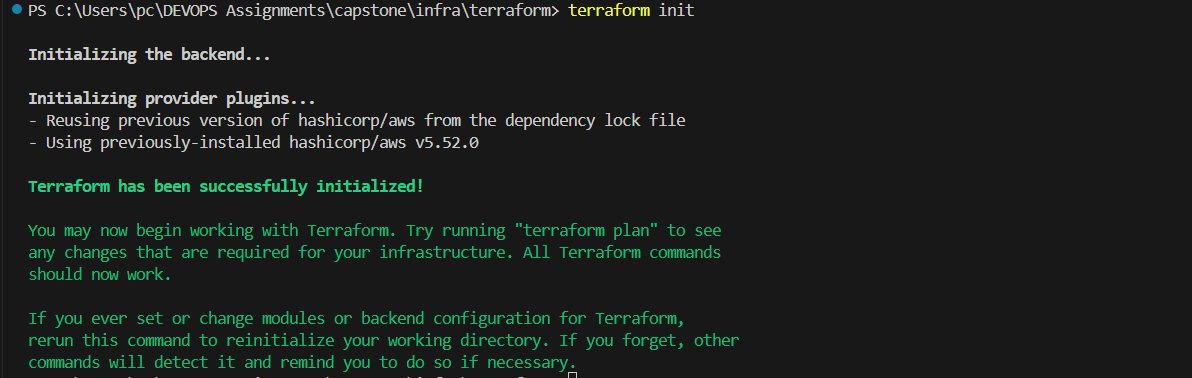
Variables.tf:

This file lets you customize your setup:

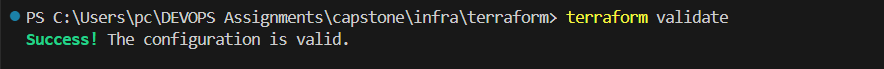
1. **Region**: Which part of the world your resources will be in.
2. **Cluster Name**: The name you'll give your EKS cluster.
3. **Cluster Version**: The version of Kubernetes you want to use.
4. **Environment**: Whether this is for development, testing, or production.
5. **Project**: The name of your project or application.

After complete the files now time to excute:

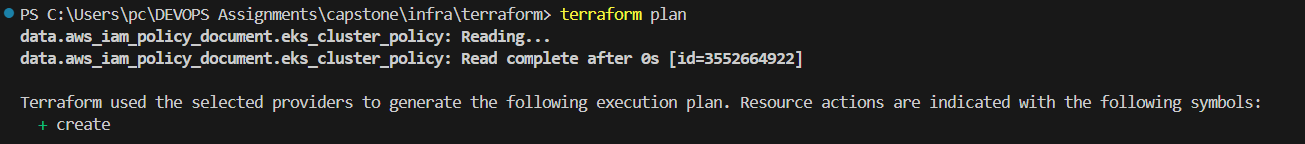
Run command: terraform init



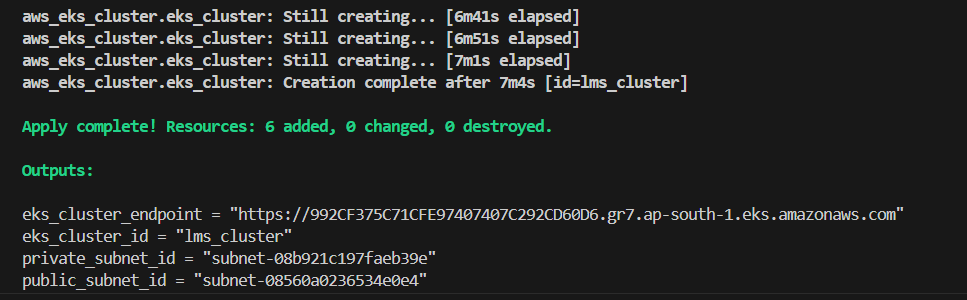
Run command: terraform validate



Run command: terraform plan



Run command: terraform apply



Check in AWS “EKS”

