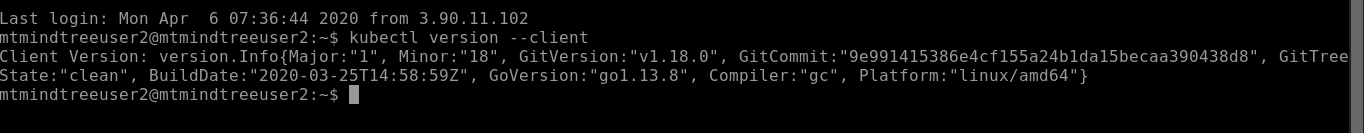
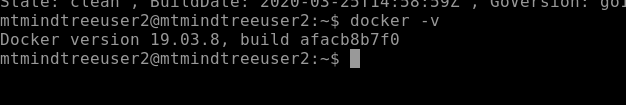
**Project solution for kubernetes 201 project casestudy**

Firstly Install **Docker,kubectl and setup the kubernetes cluster**.

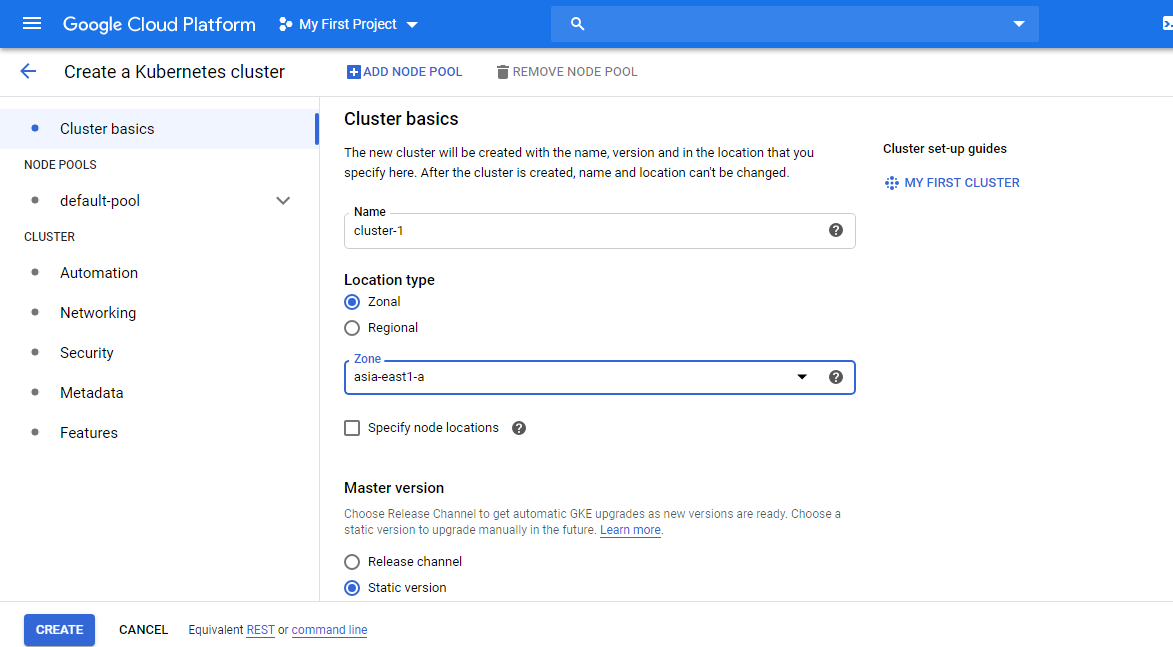
We can set up the kubernetes cluster by **minikube** on our local system,but minikube will let us to create **only 1 node cluster**.

We can also create the Kubernetes clusters on Cloud like **Google cloud platform (GKE) ,AWS (ECS-EKS) or Azure (AKS)**

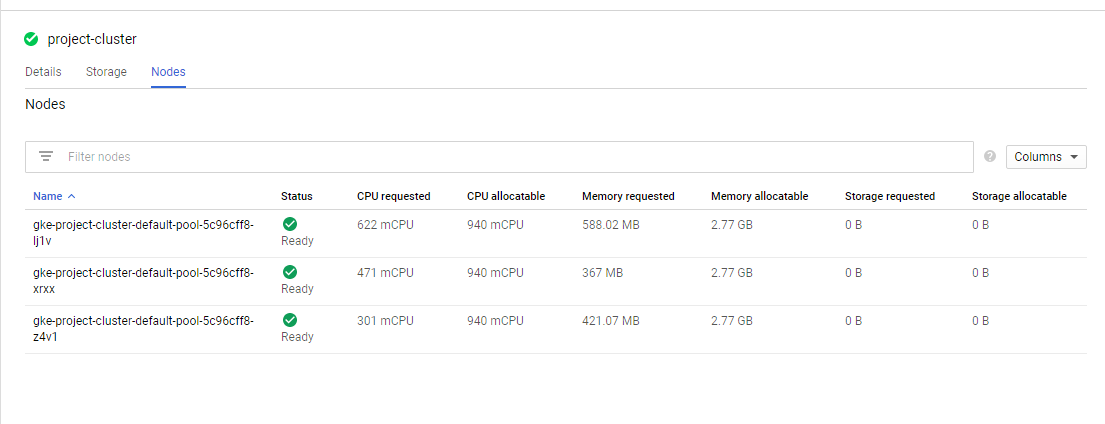




Here, I have created the kubernetes clusters **on Google platform** as follows

I 

Find the cluster details,by default 3 nodes are created .



To connect to these google cloud kubernetes cluster remotely we need to execute few commands in our ubuntu operating system.

First we will **install google could sdk** with the following commands on our ubuntu

Curl <https://sdk.cloud.google.com> |bash

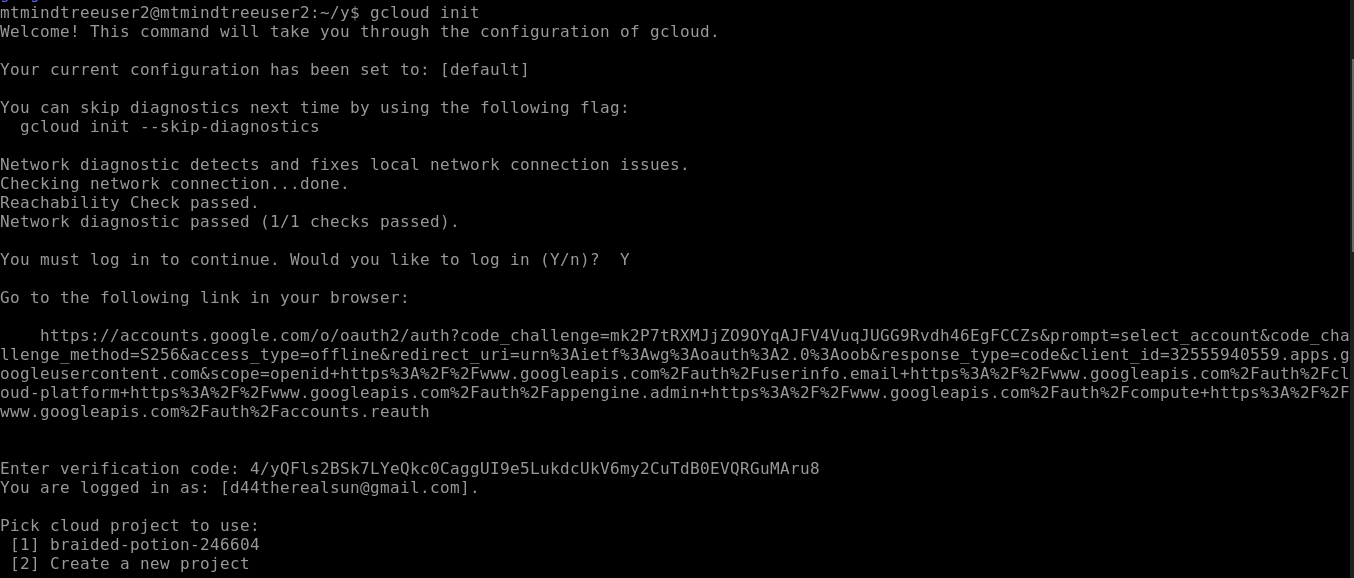
After that

Go to the directory name which we had mentioned during the sdk installation and type exec $SHELL

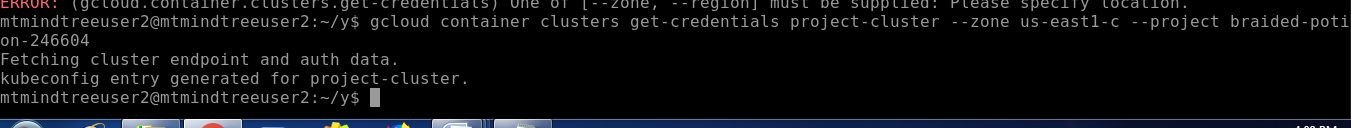
And then initialize it with the command

**gcloud init**

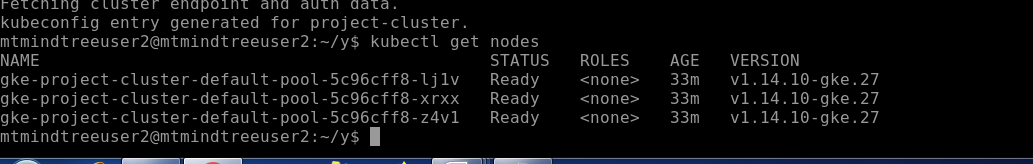
Here we need to provide the login credentials of the google cloud.

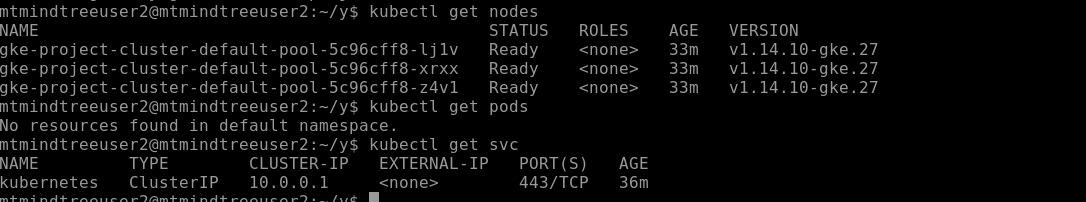


Then we need to provide the cluster name to which we want to connect

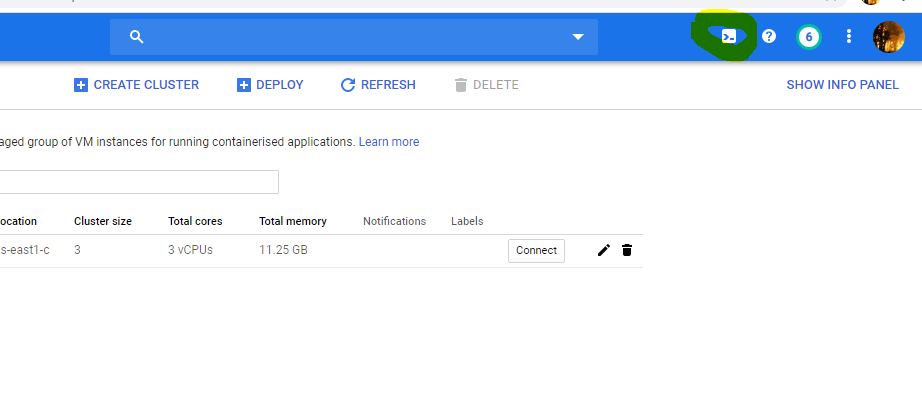


We could see the nodes now in that given cluster.



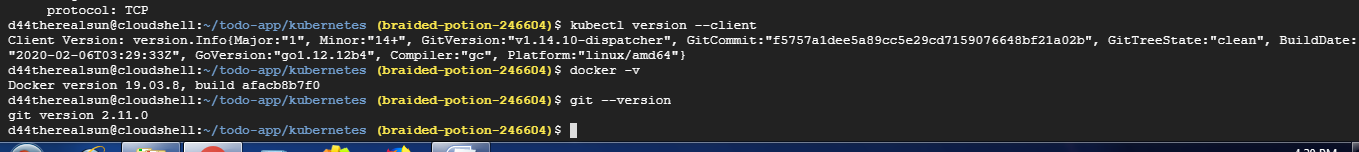


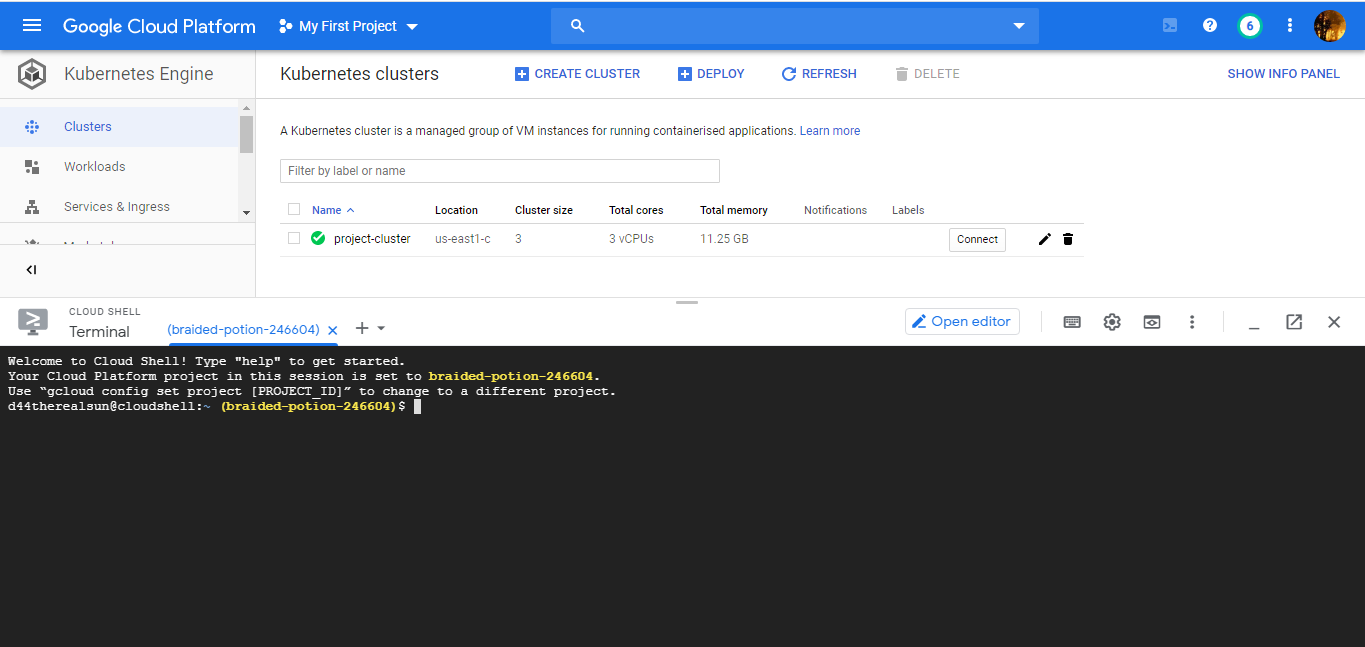
We can also **connect** to these clusters from Gcloud Terminal itself.To do that follow the steps as follows



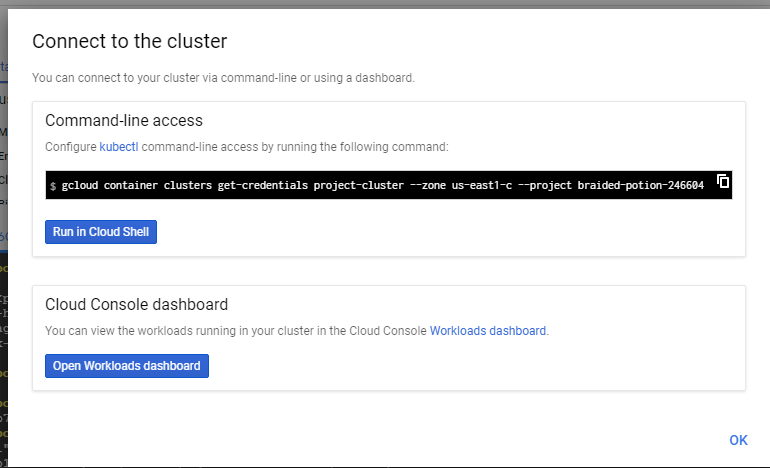
Activate cloud shell

Check whether the required softwares are installed already or not.

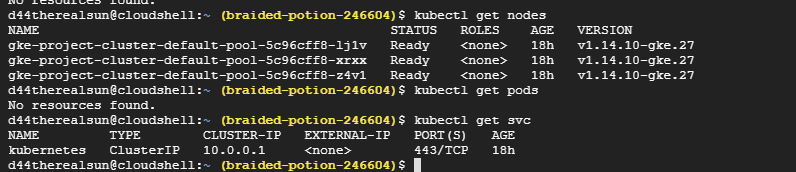




To connect to already created cluster execute the following in cloud shell

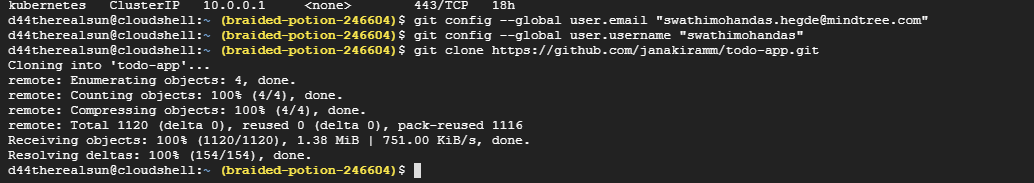


Then check in the cloud shell.

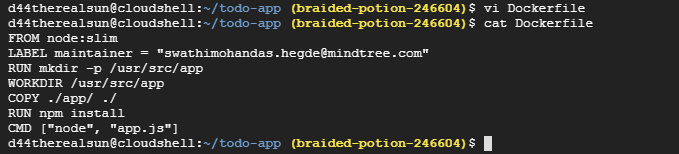


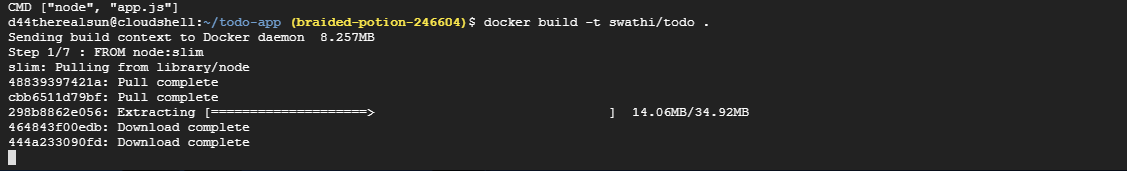
We could see the nodes of the cluster.

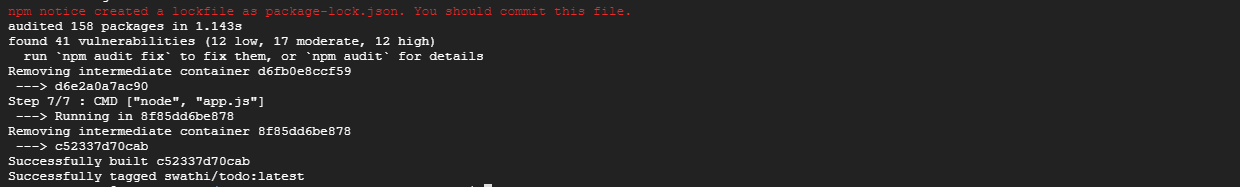
Now we will be connecting to our github repository thru terminal,and we will be cloning the **Microservice based NodeJS** Application source code from the internet.

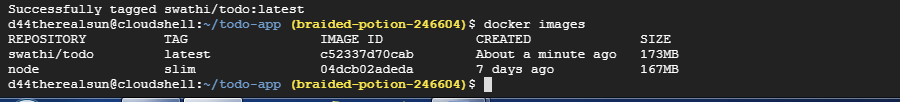


To create docker image of the application we need to run the docker file for that application.





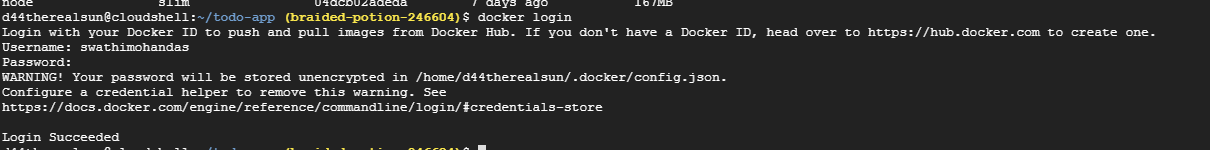




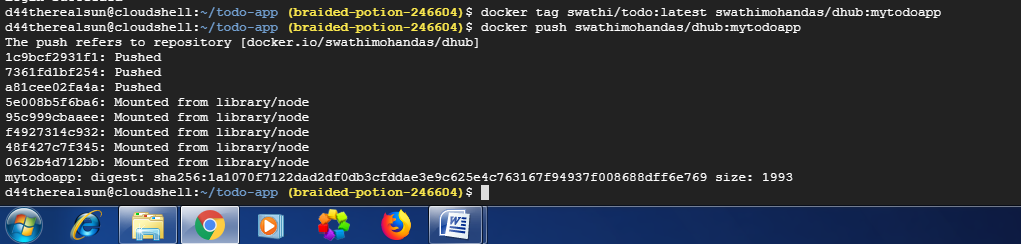
Could see that the image has been build with the name swathi/todo:latest

Now we will be pushing the image to docker hub

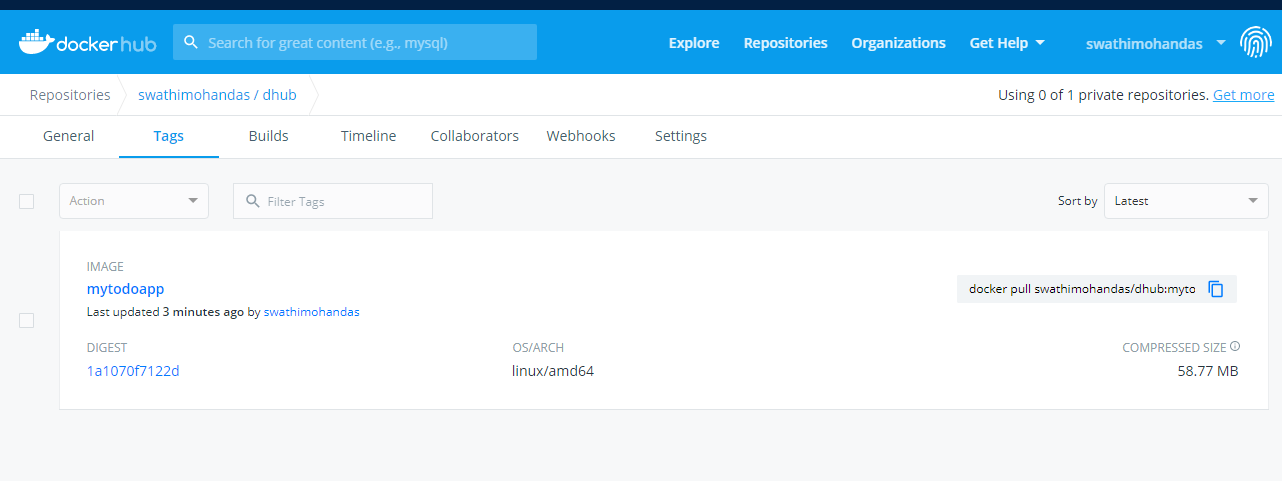
Firstly we will be logging to dockerhub registry from terminal



After the login is succesfull we need to push the docker image to the dockerhub,so that the image created is centrally accessible from anywhere on the internet.



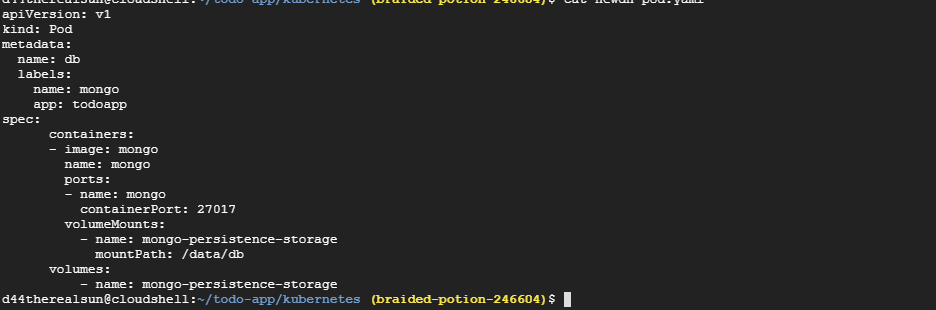
We could see the image in our Docker hub registry as follows

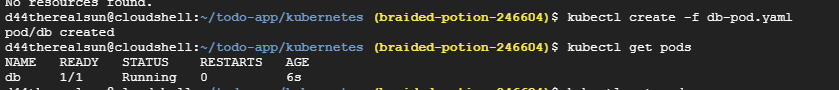


We want our kubernetes cluster to acess the image,so what ever we have build on the dev machine should be made available to kubernetes cluster ,pushing it into the dockerhub registry makes the images centrally accessible.

Now for our **microservice based application we need at database to store the data**

So Now we will be creating the db pod for our **mongodb** i.e first pod of our cluster. For for microservice based application

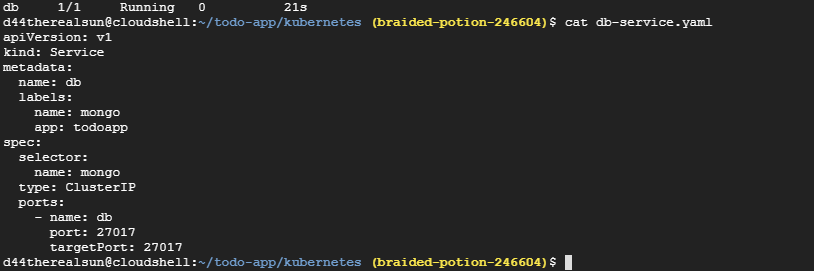


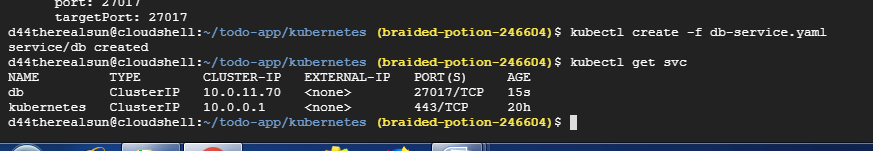


We could see the db pod instances running

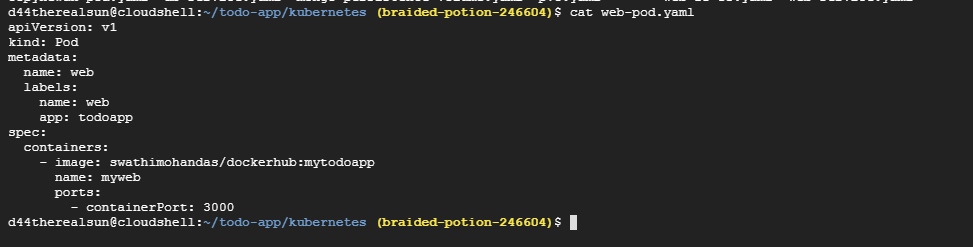
Now we will be creating the service for the db-pod

We don’t want our db service to be exposed to the externally ,so we have set the type as **ClusterIP** so that it will be accessible only within the cluster.

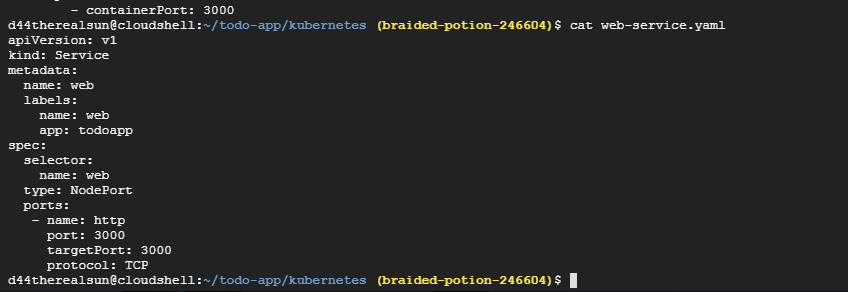


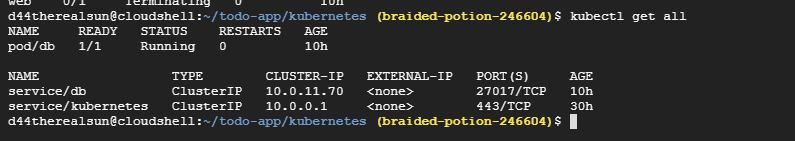


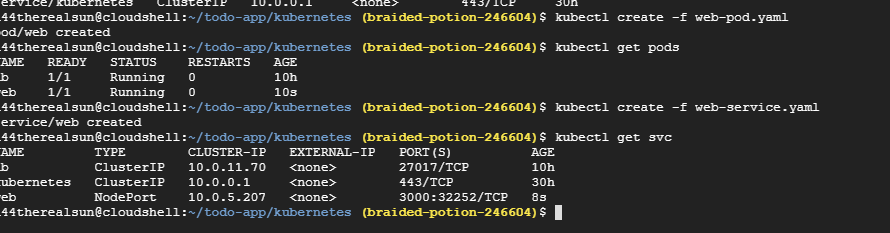
Now we will be creating thepods and services for our web application



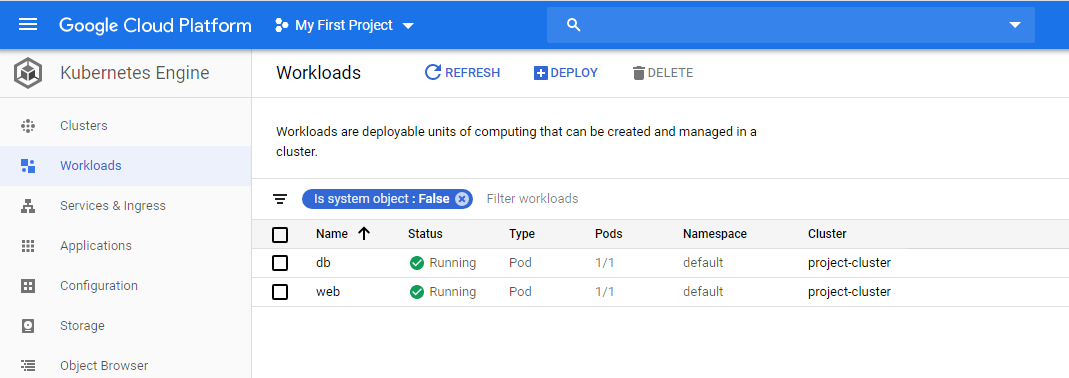
Since we want out application to be accessible externally so we have set the type to **Nodeport**

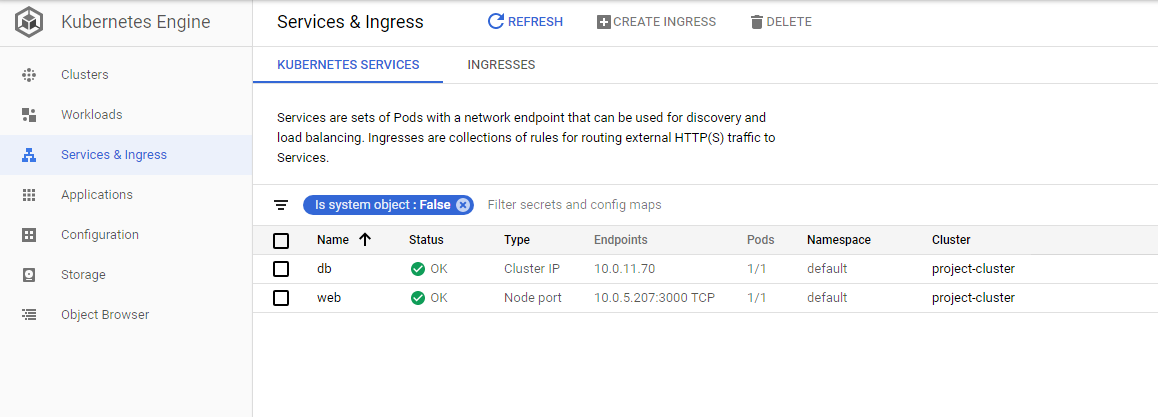






We could see the details of the pods and services in the dashboard as follows

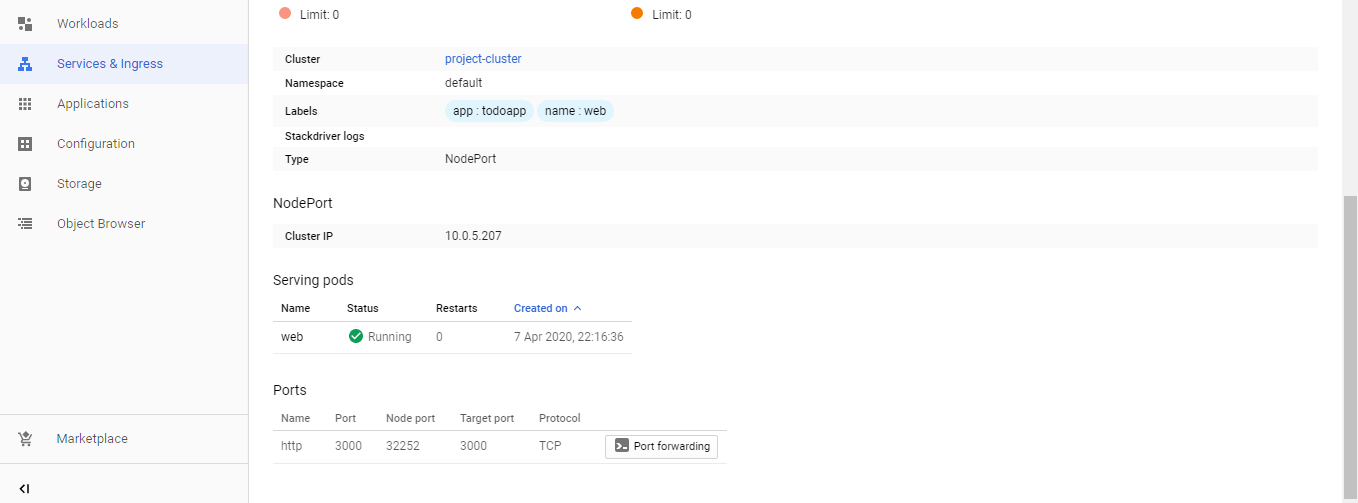




We can get the details of the services and pods.

Please find the service details of web

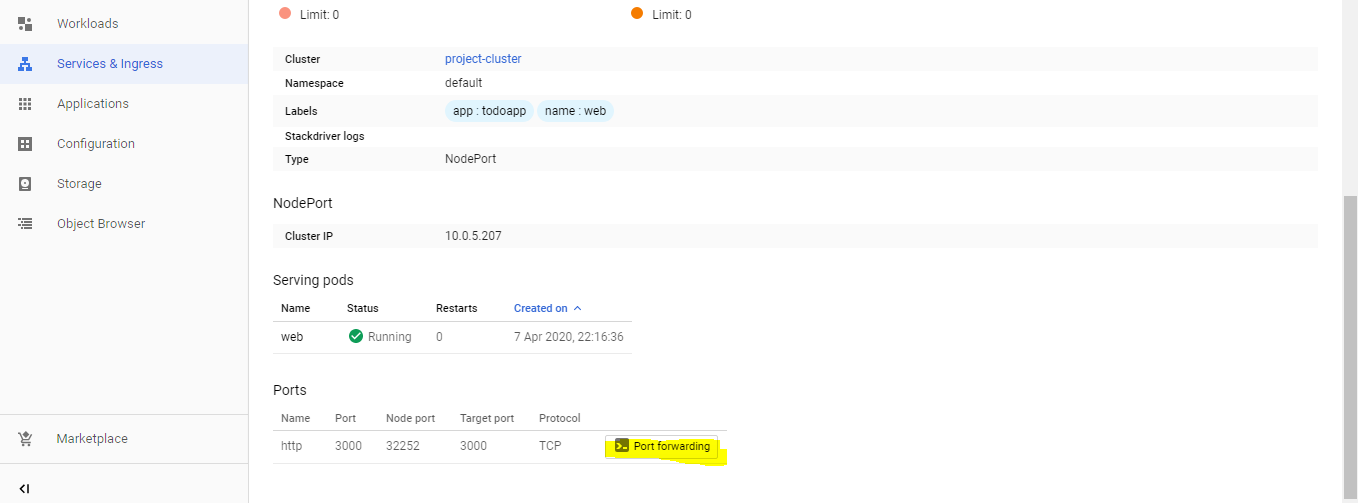


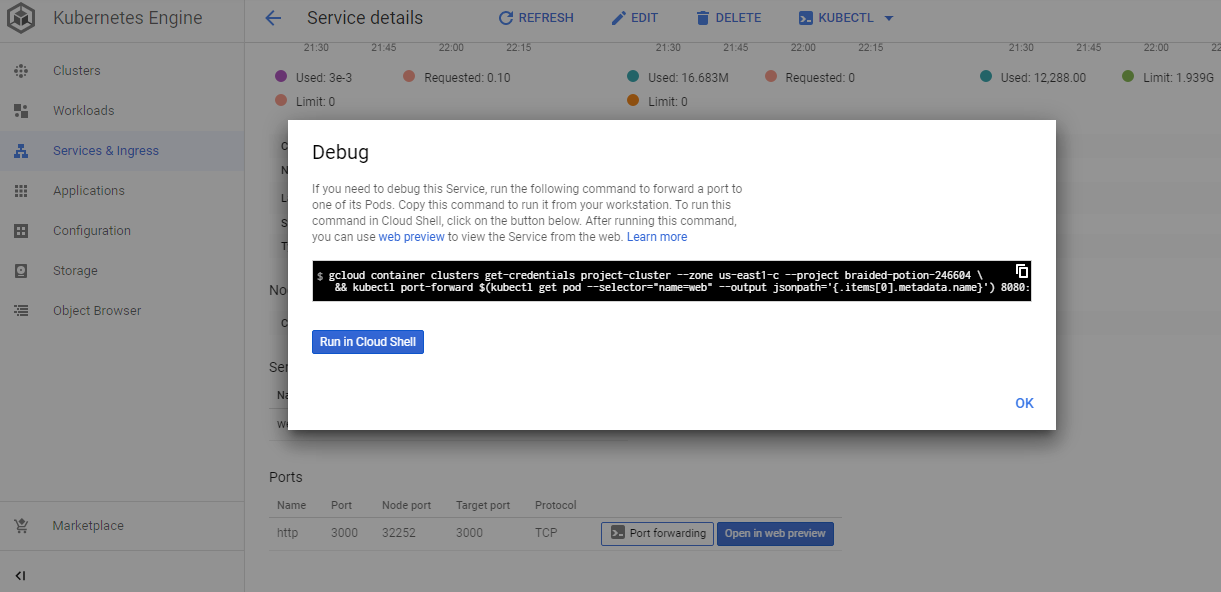


Now we can open and access the user interface for our Microservice based Nodejs application by following ways.

For that we need to access our services of the web

Click on the **port forwarding** as follows.

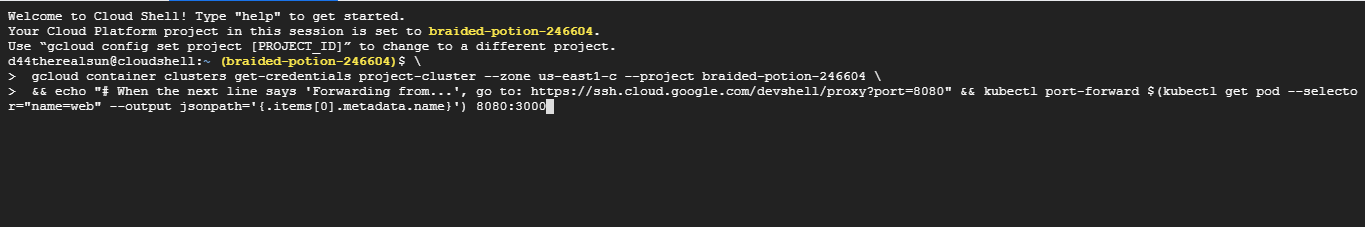


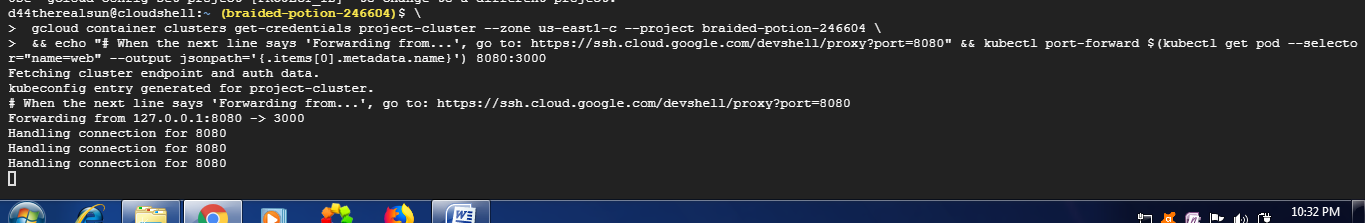


Run it in cloud shell and click on open in web preview button.

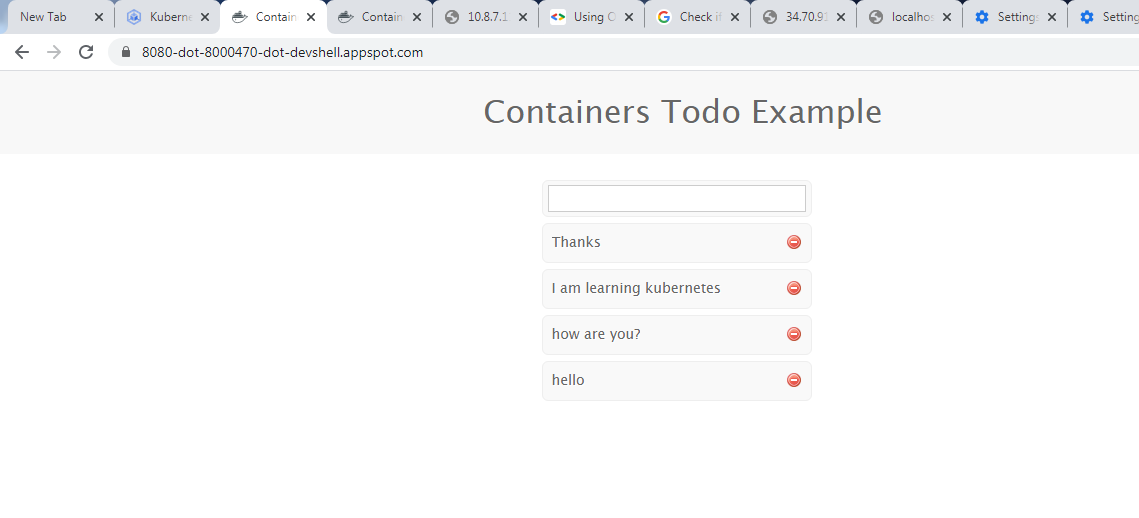
gcloud container clusters get-credentials project-cluster --zone us-east1-c --project braided-potion-246604 \

&& kubectl port-forward $(kubectl get pod --selector="name=web" --output jsonpath='{.items[0].metadata.name}') 8080:3000





Could see the applicationhosted on the web as follows.



Now we have live app deployed as microservice running in kubernetes.

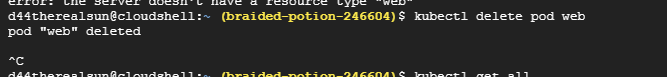
We are running as just one instance.

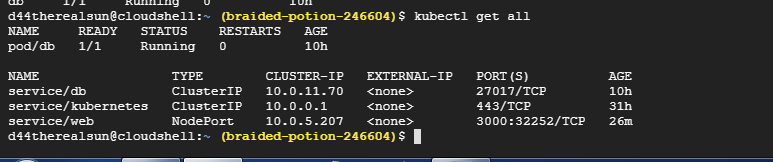
But no application should run only 1 instance it will lead to single point of failure.

So we will be scaling this application.

Kubernetes have primitives called replicaset.so with that we can decide how many number of pods we want to run so that if any of the pod gets failed kubernetes will bring back the another pod.

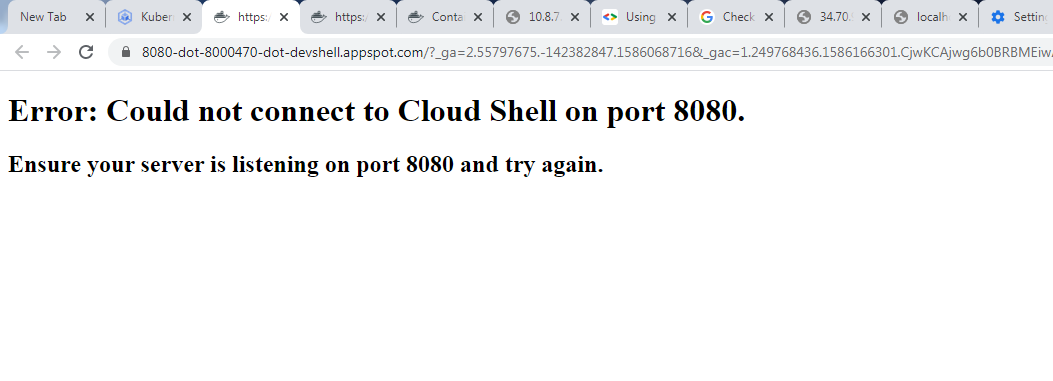
Now if we delete web pod .





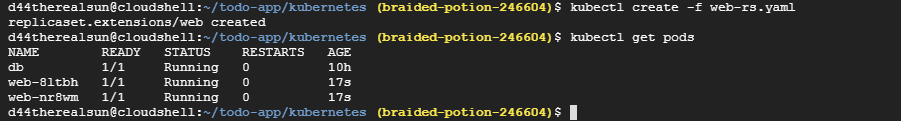
We could see the service is still there it is listening to the port 3000,but we cant access the service due to pod failure.

Could see the below error



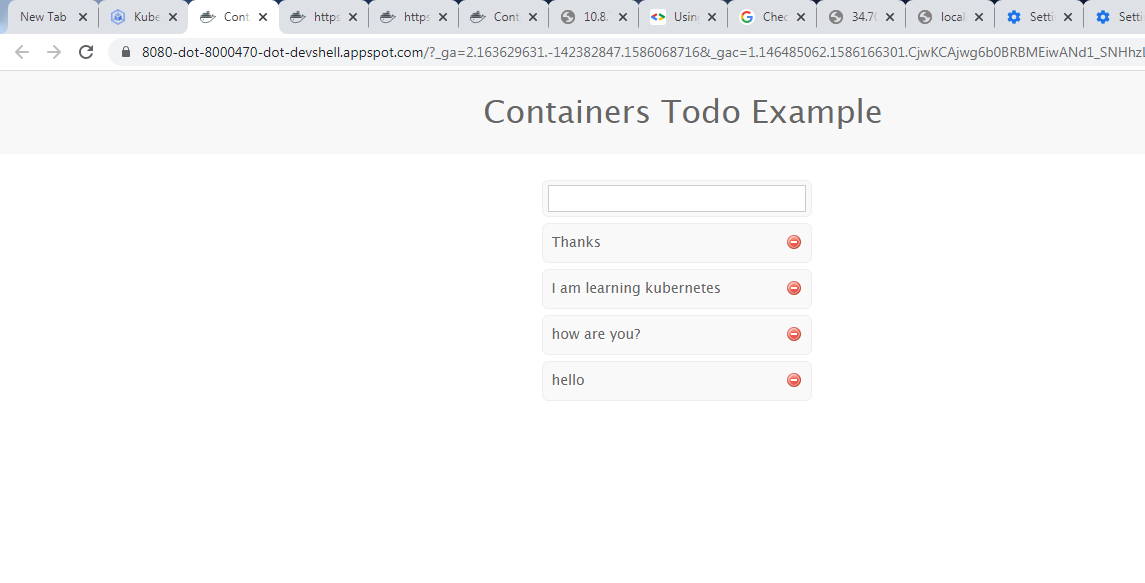
Now we will be running replicaset yaml file



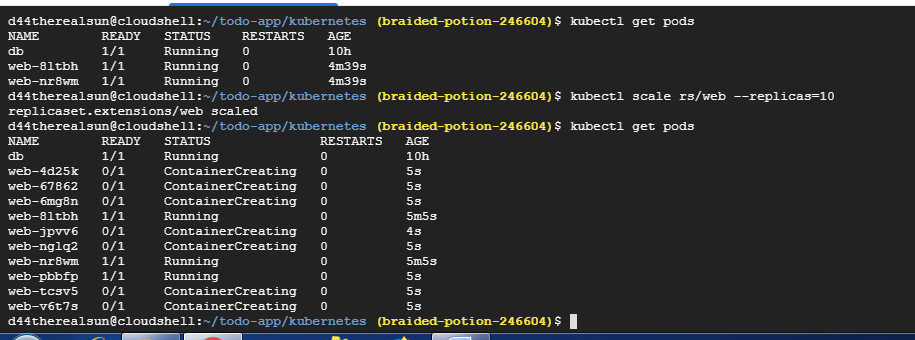


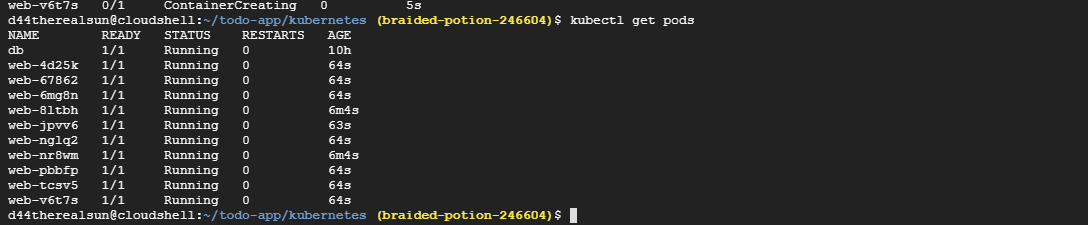
Now we could see two instances of the web pod running.

Also the now we are able to access the application.even after deleting one of the pod



Now we will be scaling the pod as follows

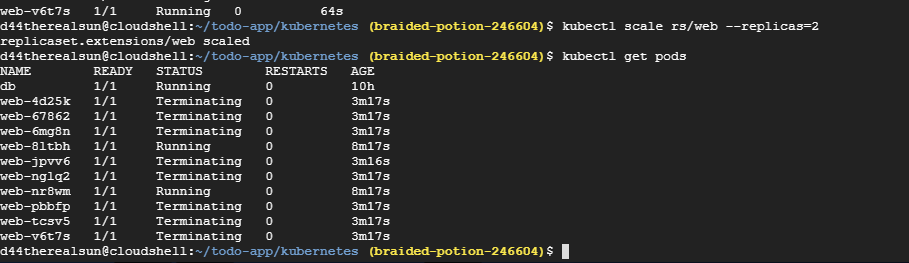


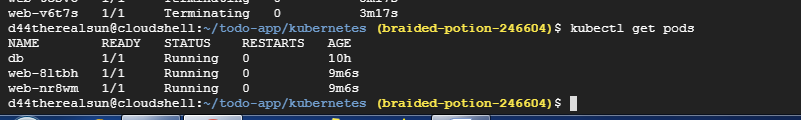


Now 10 web pods instances have started running.

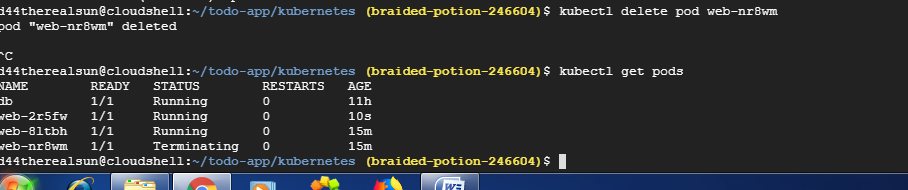
So in this way we can scale the application on the basis of our requirement.

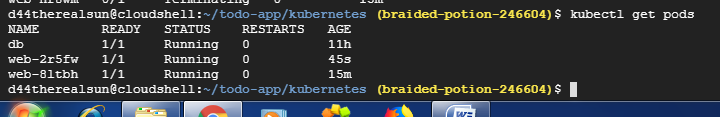
Now we will be scale down to two.





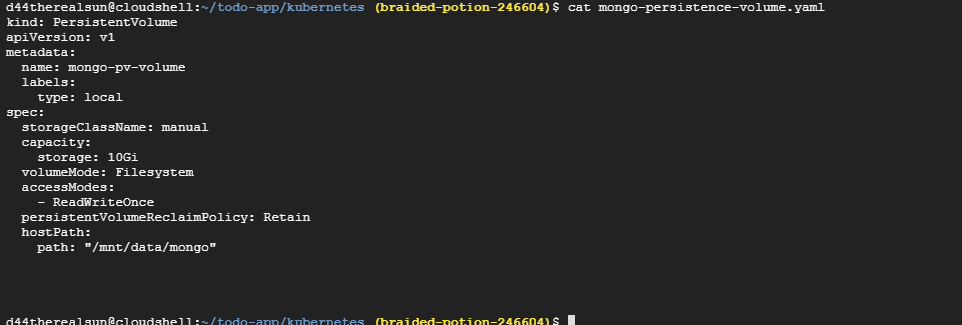
Now if we delete one of the pod as follows,we could see the pod is getting terminated but also could see more pod starts running .So this concept is called autohealing.

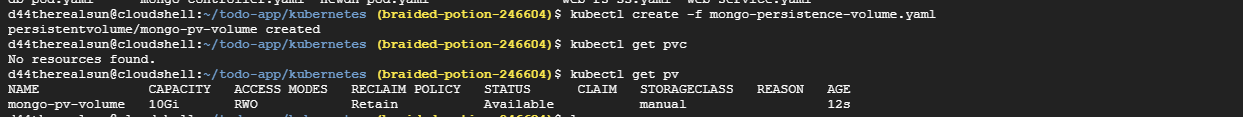




For Data persistene,data in the database must retain so we will be doing data persistence for the db,so that even if the db pod fails data will be retained for the db.

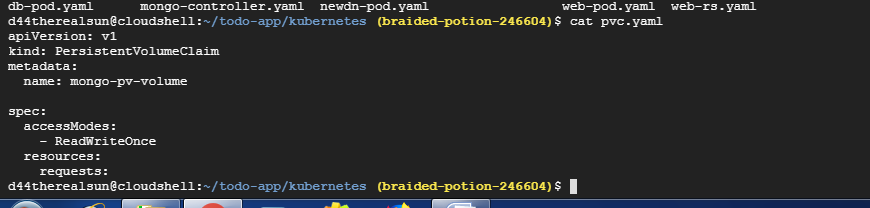
Now we will be creating persistence volume for data storage as follows.

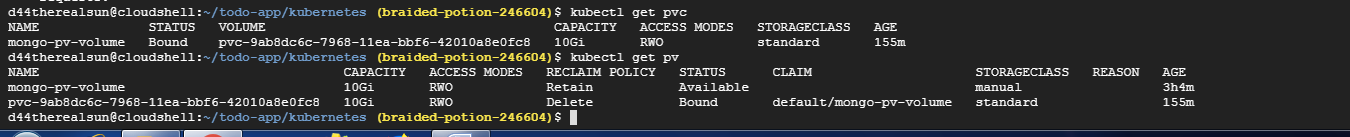




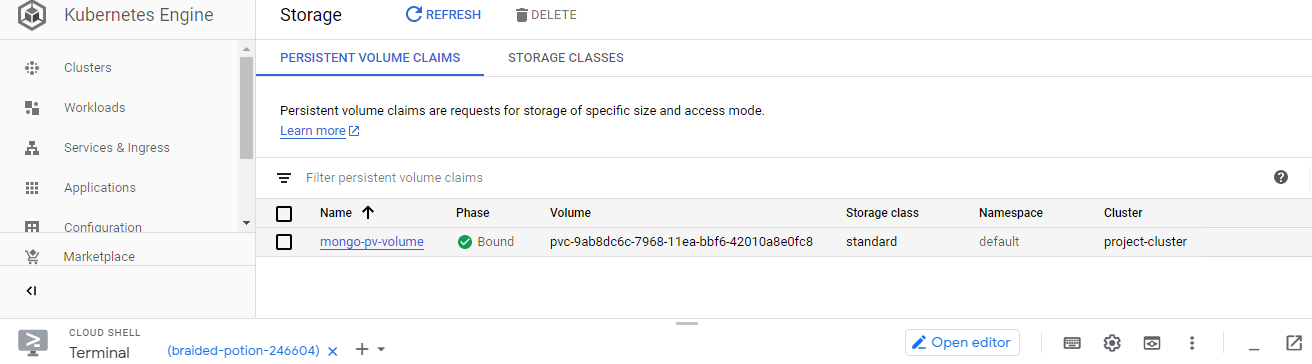
To use this persistence volume we need persistence volume claim

So we will be creating persistence volume claim now

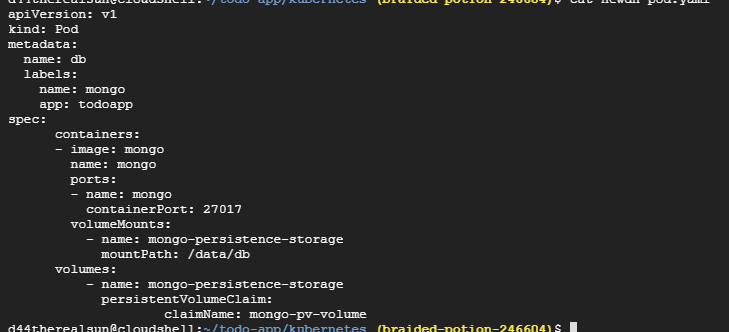


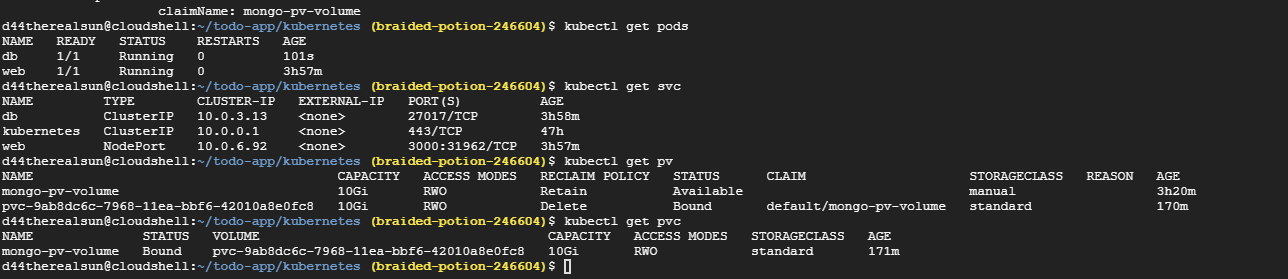


We could see the persistence volume created in dashboard as well



Now we can use this explicity created persistent-volume in our db pod as follows.





So in case of any failure in the pods it can access this persistence volume and restore its data.