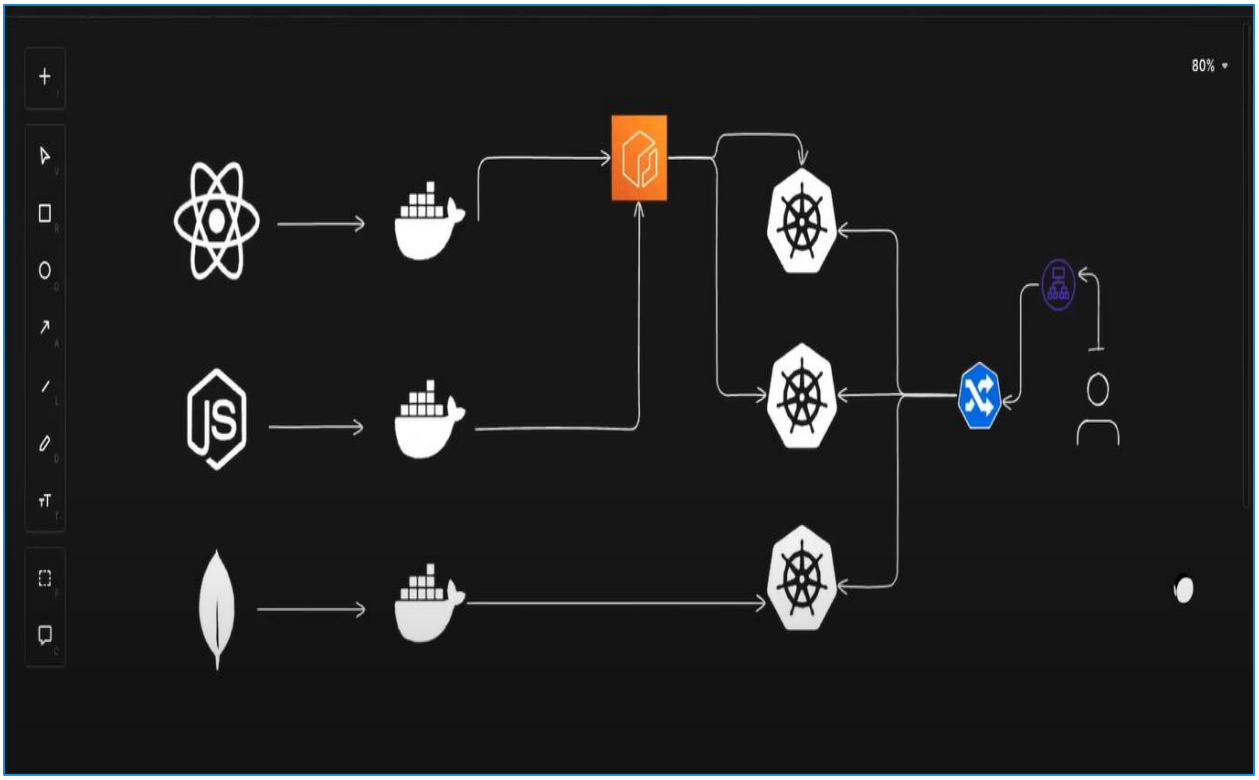
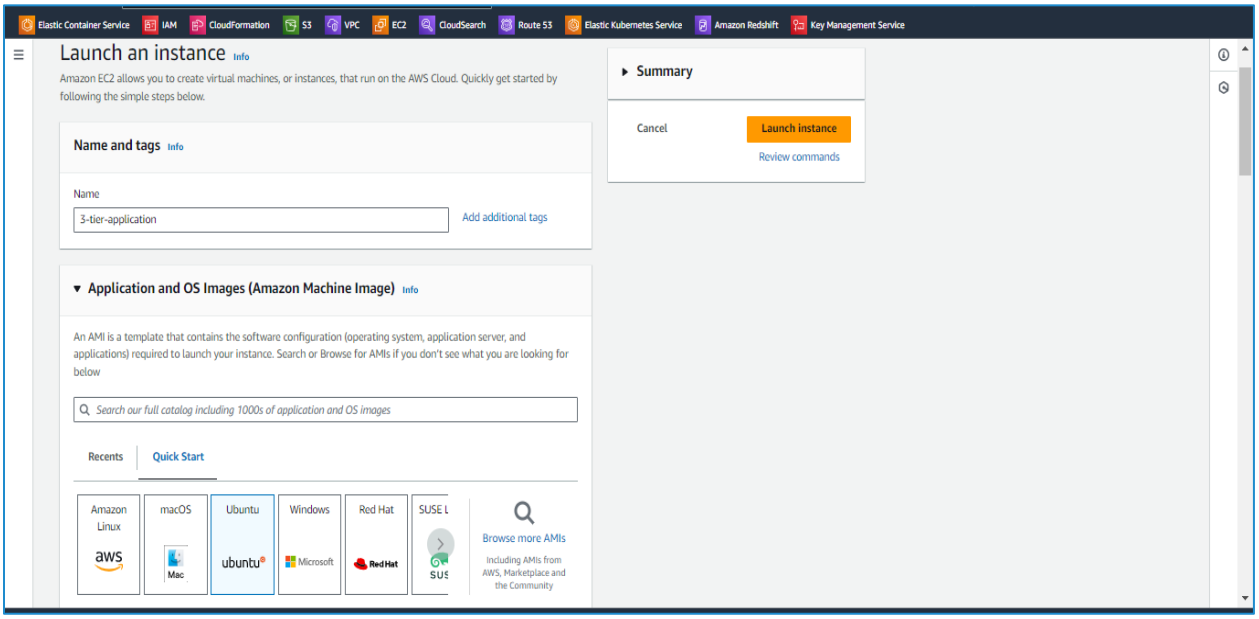
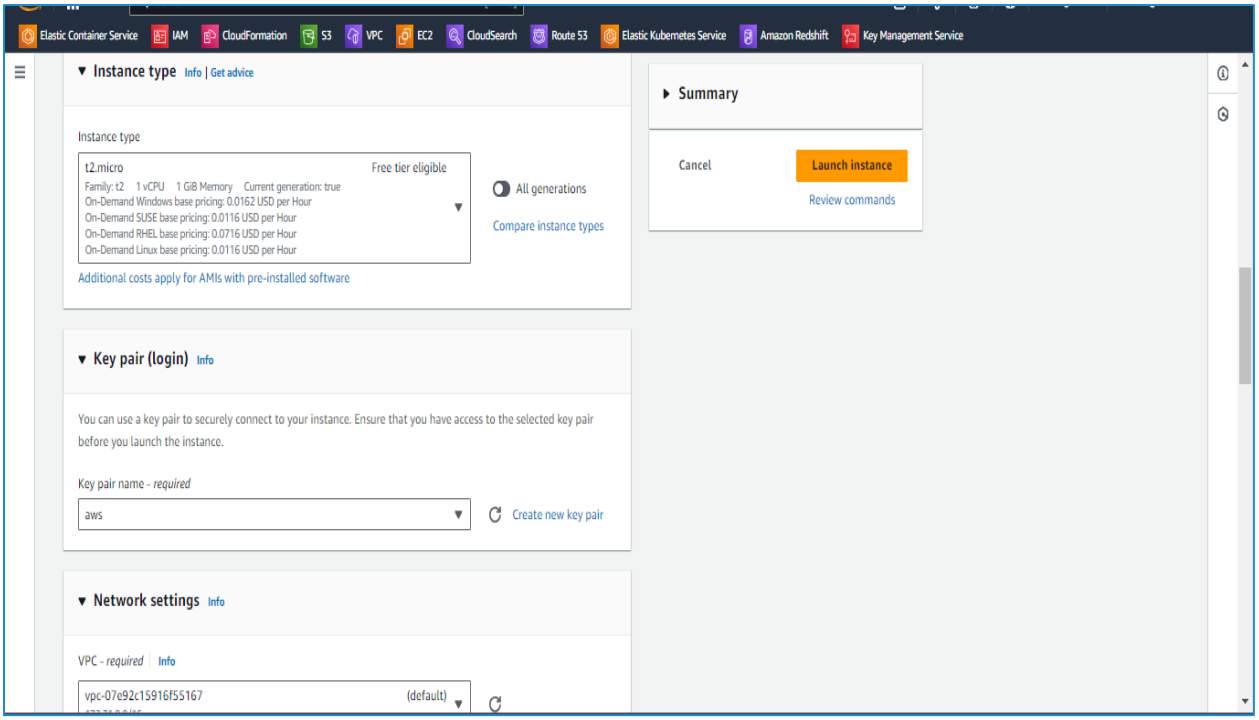
THREE TIER ARCHITECTURE USING DOCKER & K8s



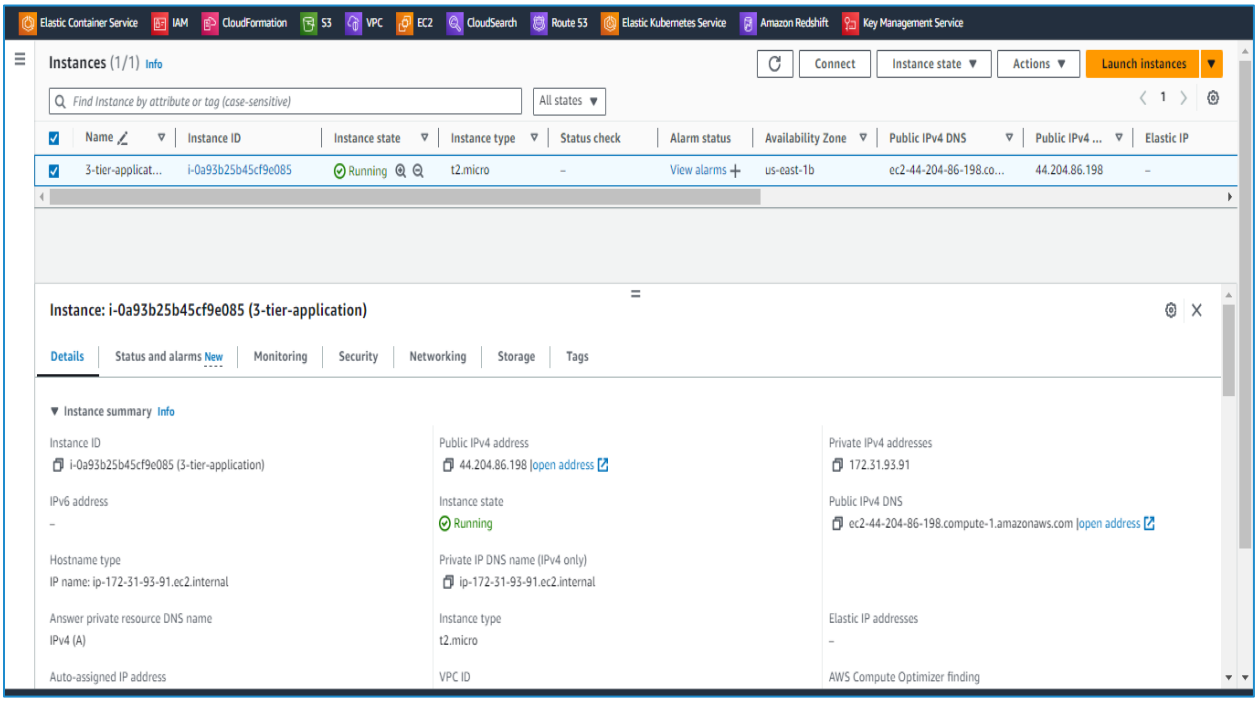
Step1: Launch 1 EC2 instance for creating a docker file



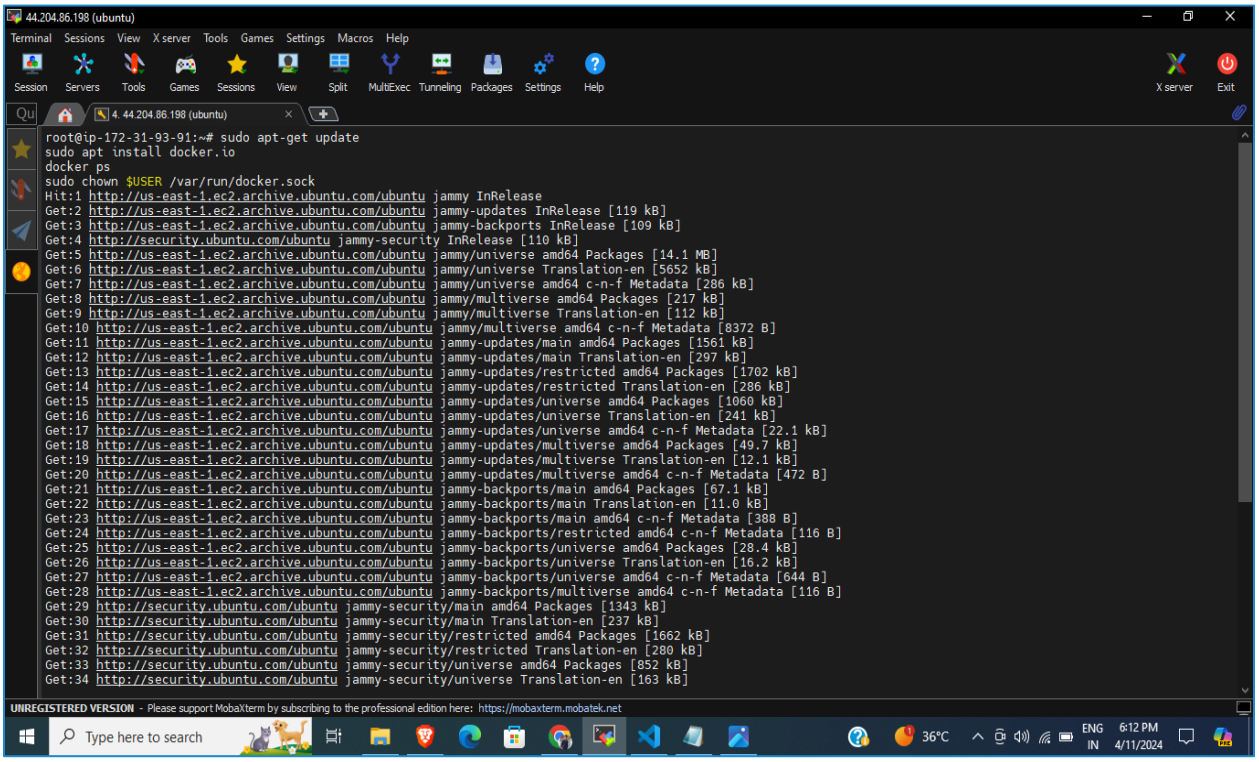
Step2: Select instance type, key pair and VPC



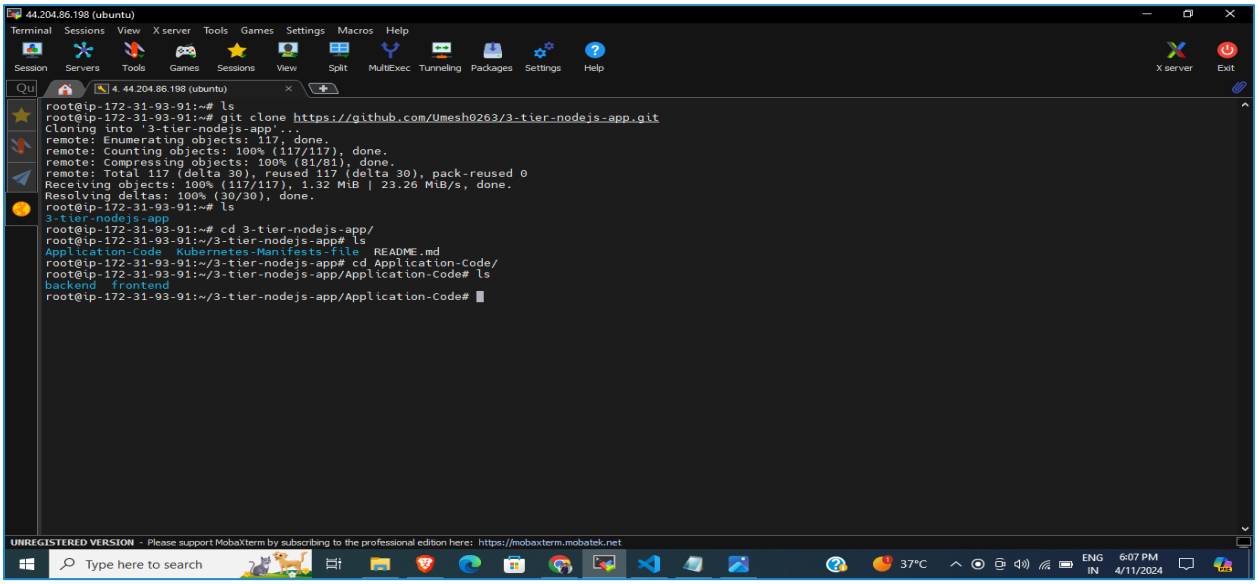
Step 3: Then our instance has been created



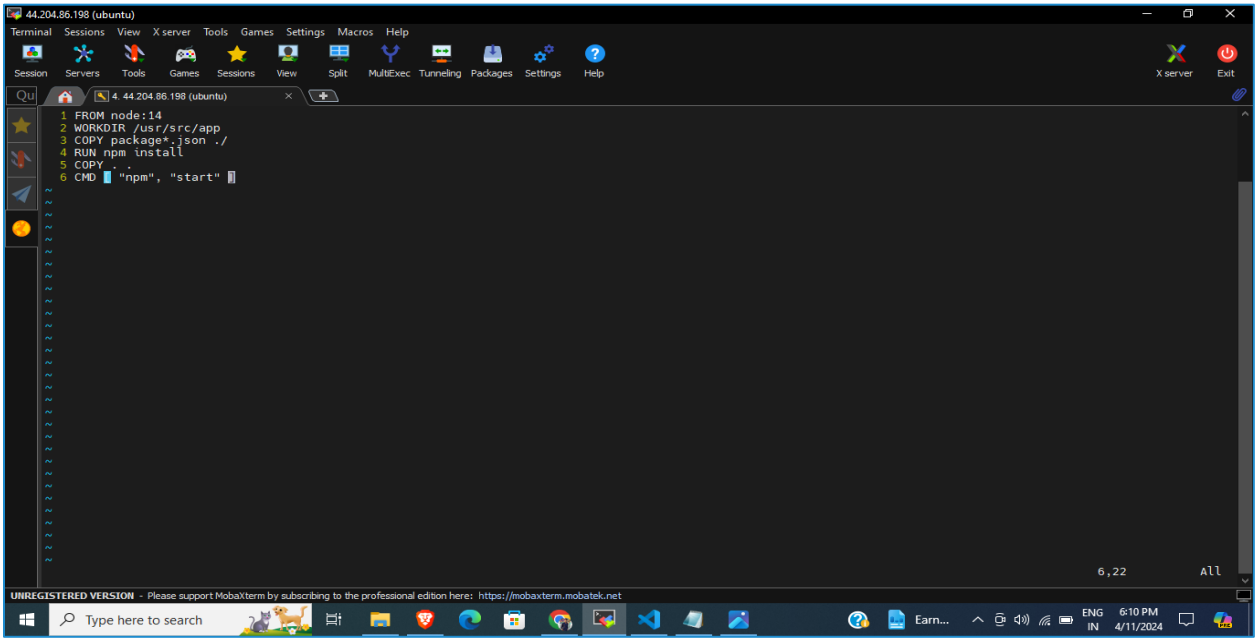
Step 4: install docker on ubuntu machine and start docker daemon service using systemctl start docker command.



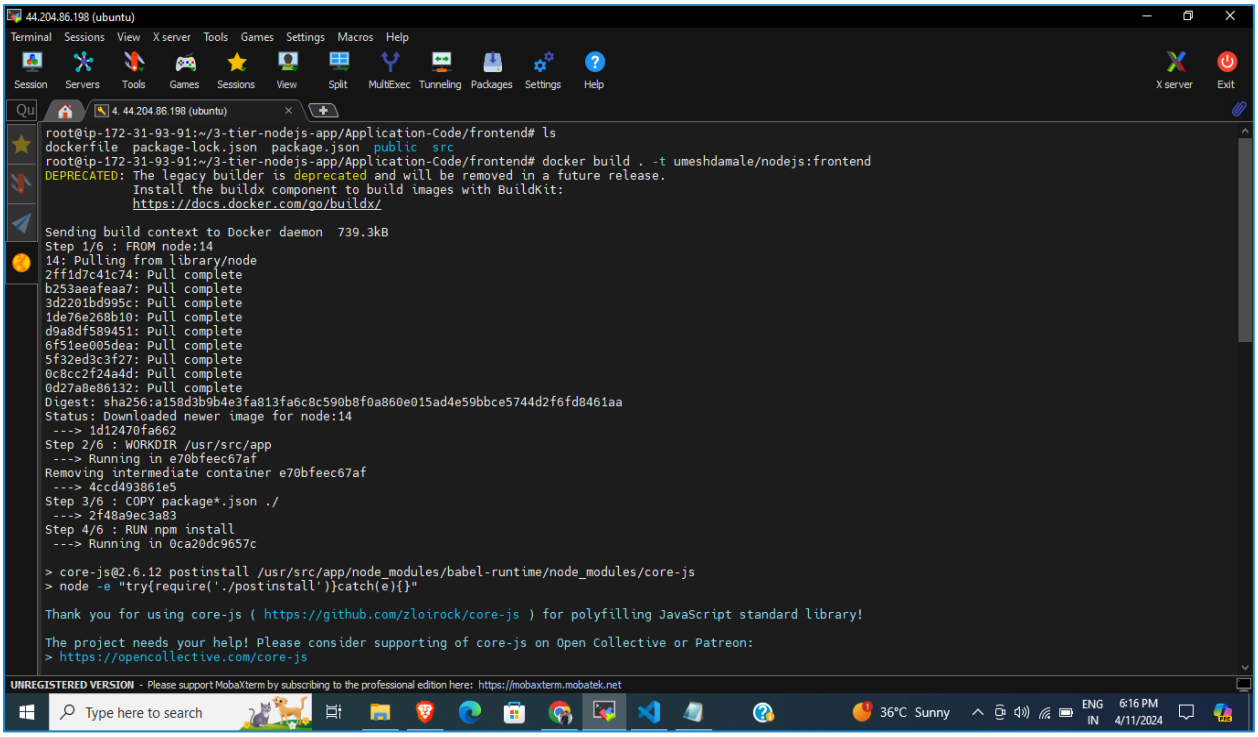
Step 5: Then get the access of instance through ssh and clone the repository of application code.



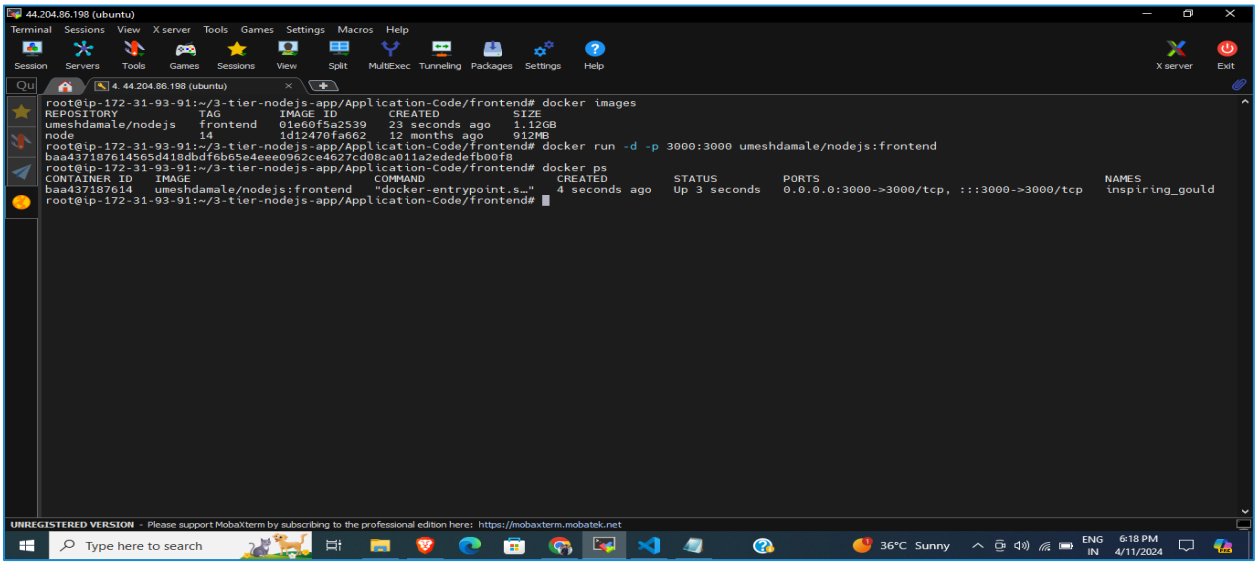
Step 6: Next, create docker file for frontend of the application.



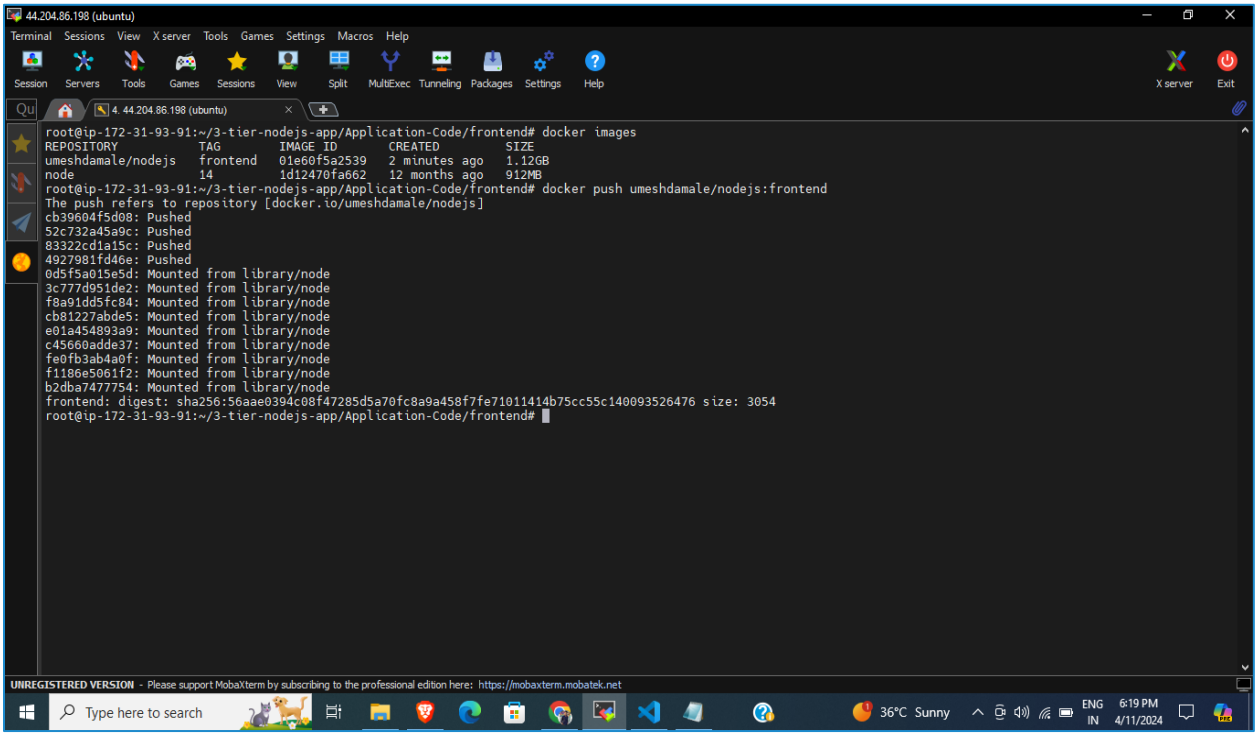
Step 7: Then, build docker file using docker build command & give tag umeshdamale/nodejs:frontend.



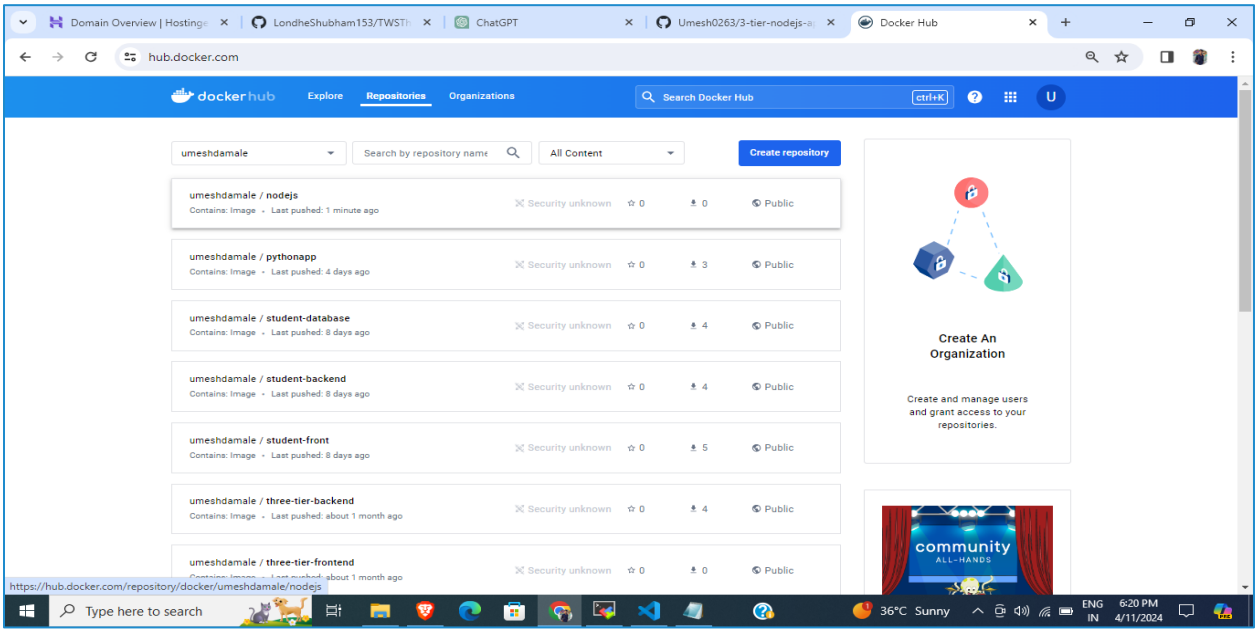
Step 8: Here, Docker image has been created then, run docker images using docker run command on 3000 ports.



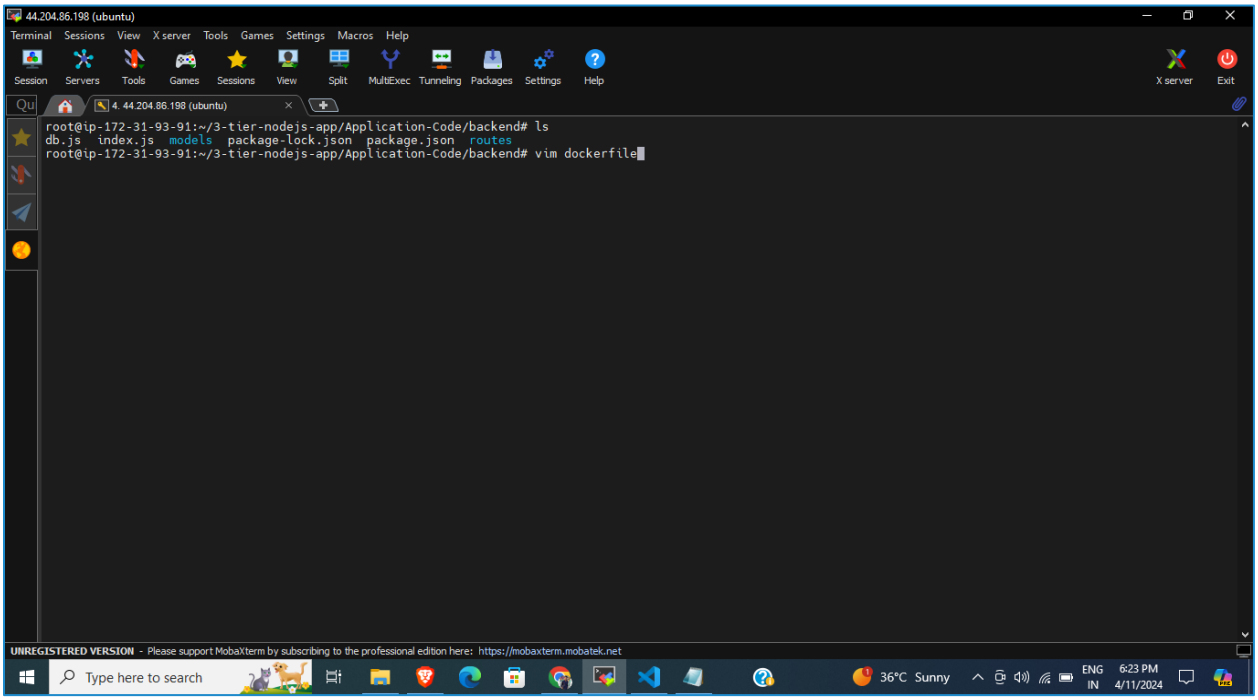
Step 9: Then, push the docker image to docker Hub using docker push command, ensuring that you have logged in first using username and pass.



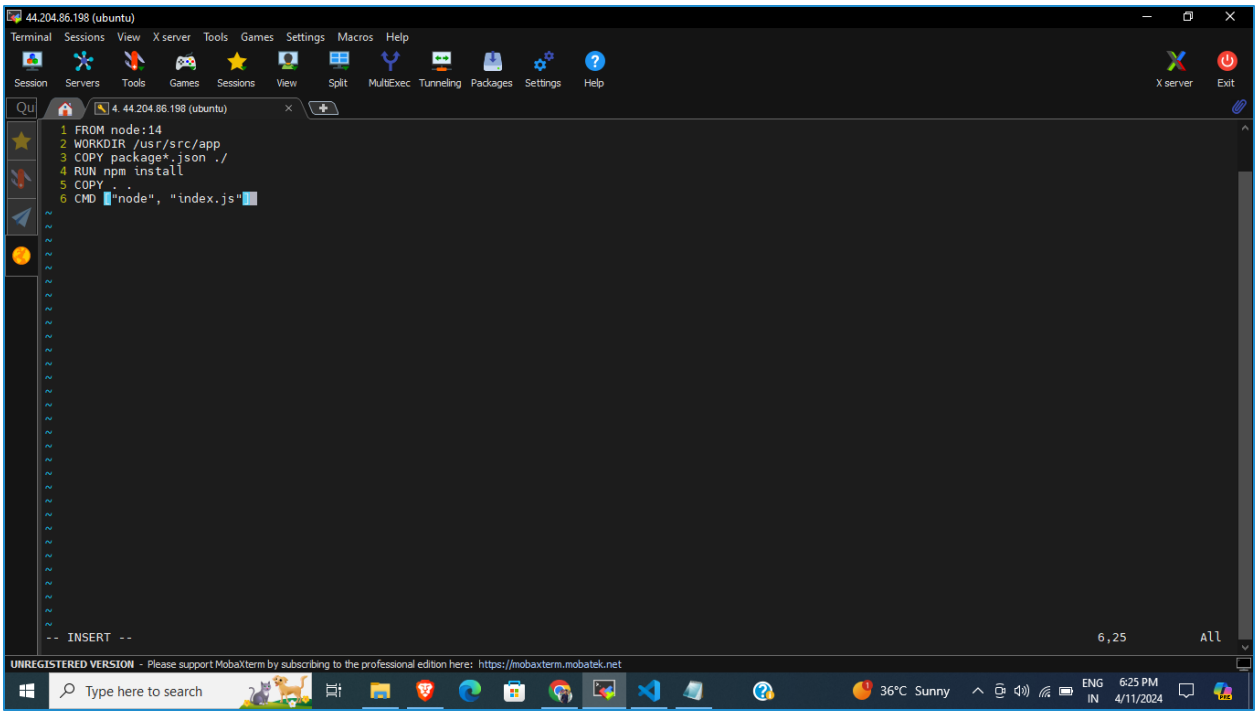
Step 10: Following are the Docker Hub Image of nodejs frontend.



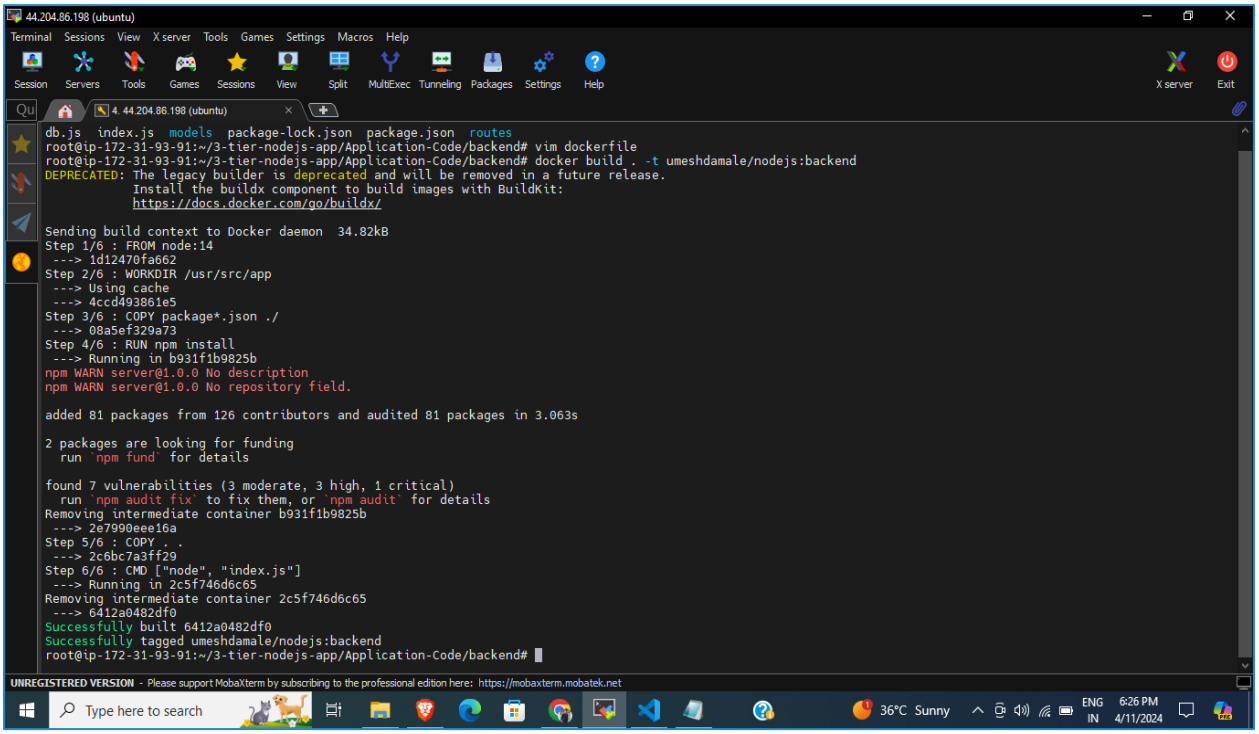
Step 11: Now, create a docker file for the backend app.



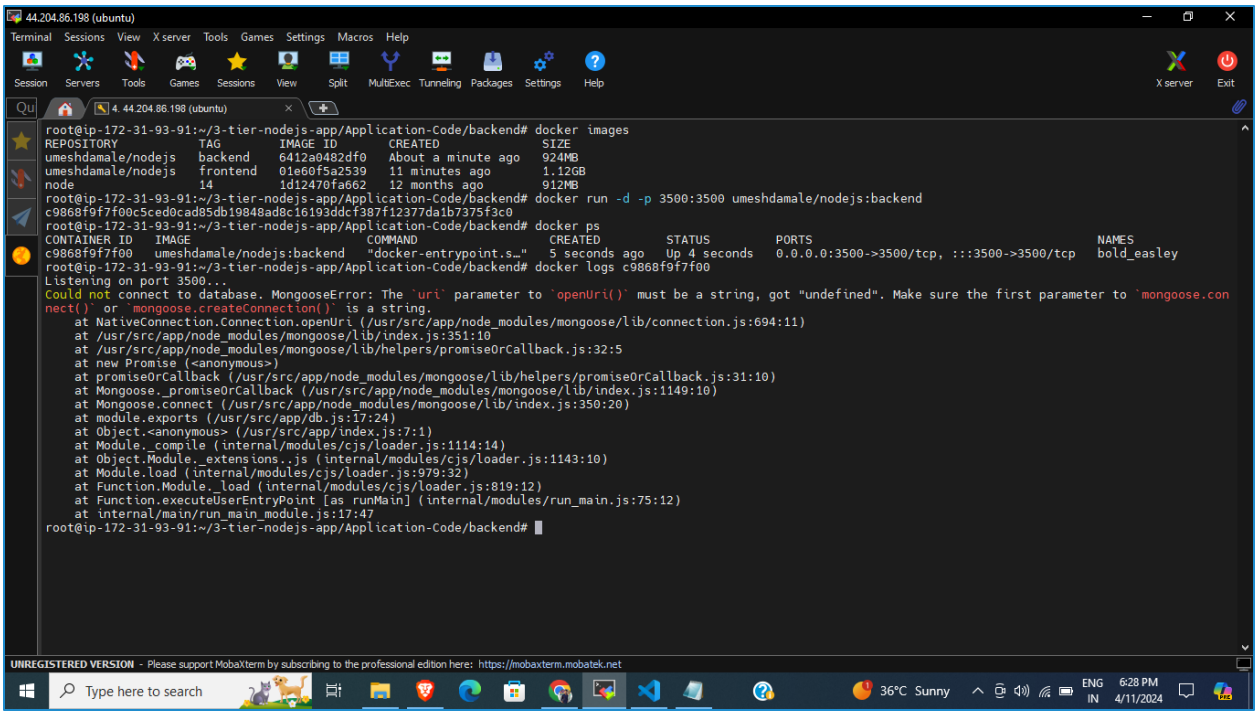
Step 12: following is the docker file for frontend.



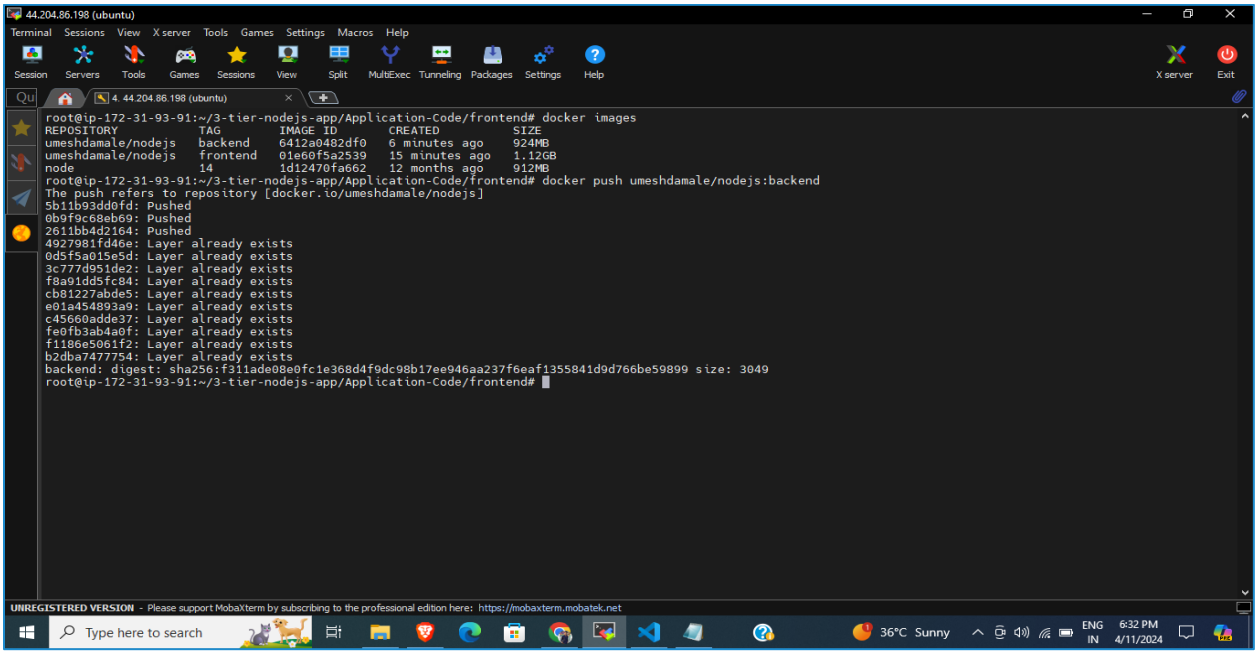
Step 13: Then, build a docker file and give tag.



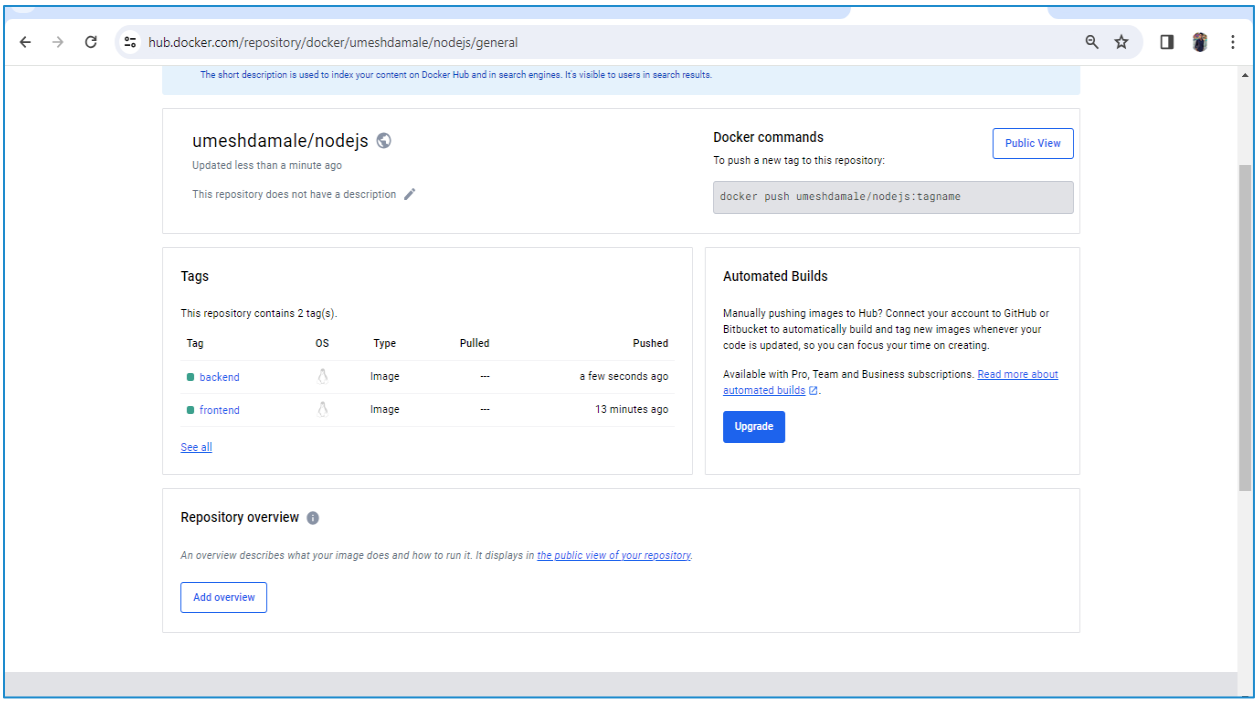
Step 14: Then, run the docker image on 3500 ports using docker run command.



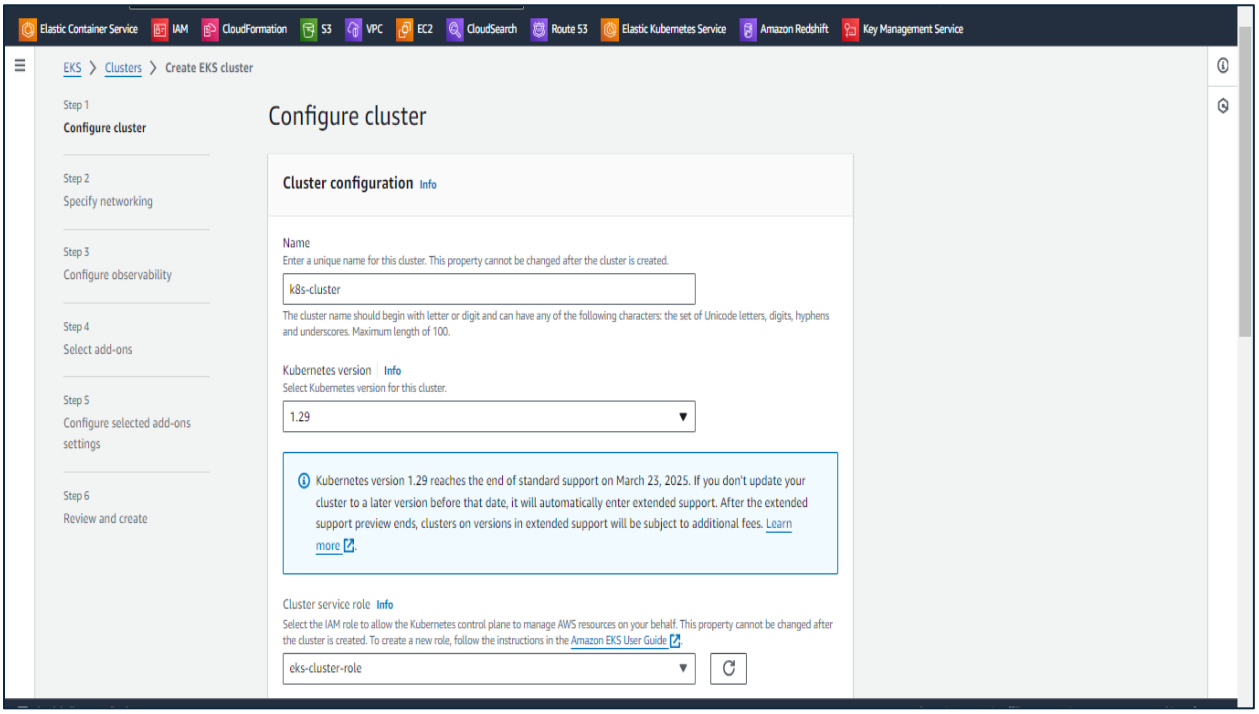
Step 15: Afterward, push the docker image on docker hub.



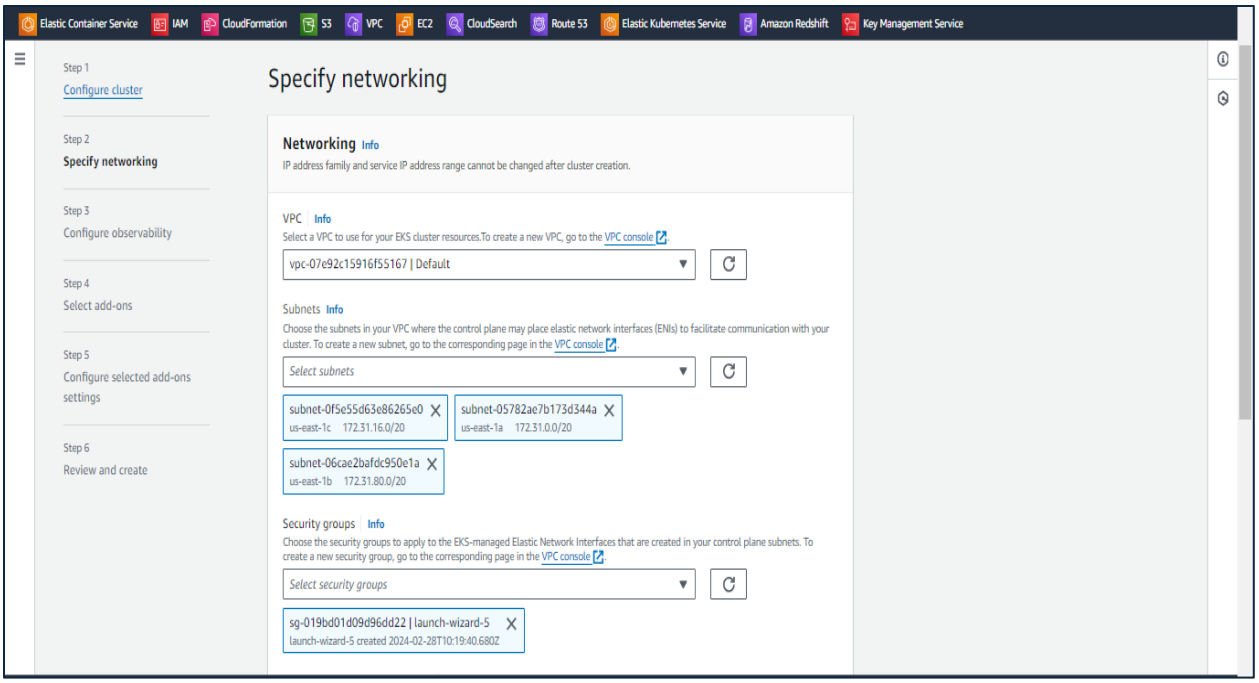
Step 16: Here, the docker image has been pushed to docker hub.



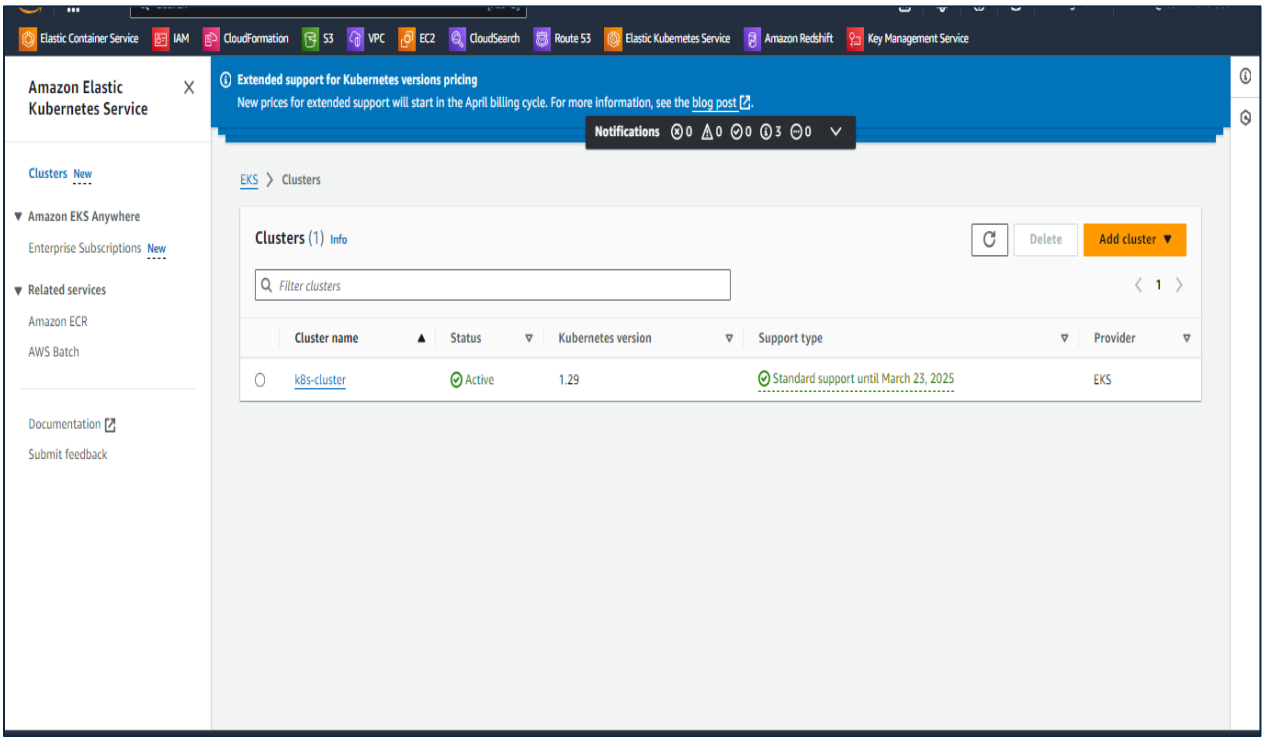
Step 17: Next, create EKs cluster and node group.



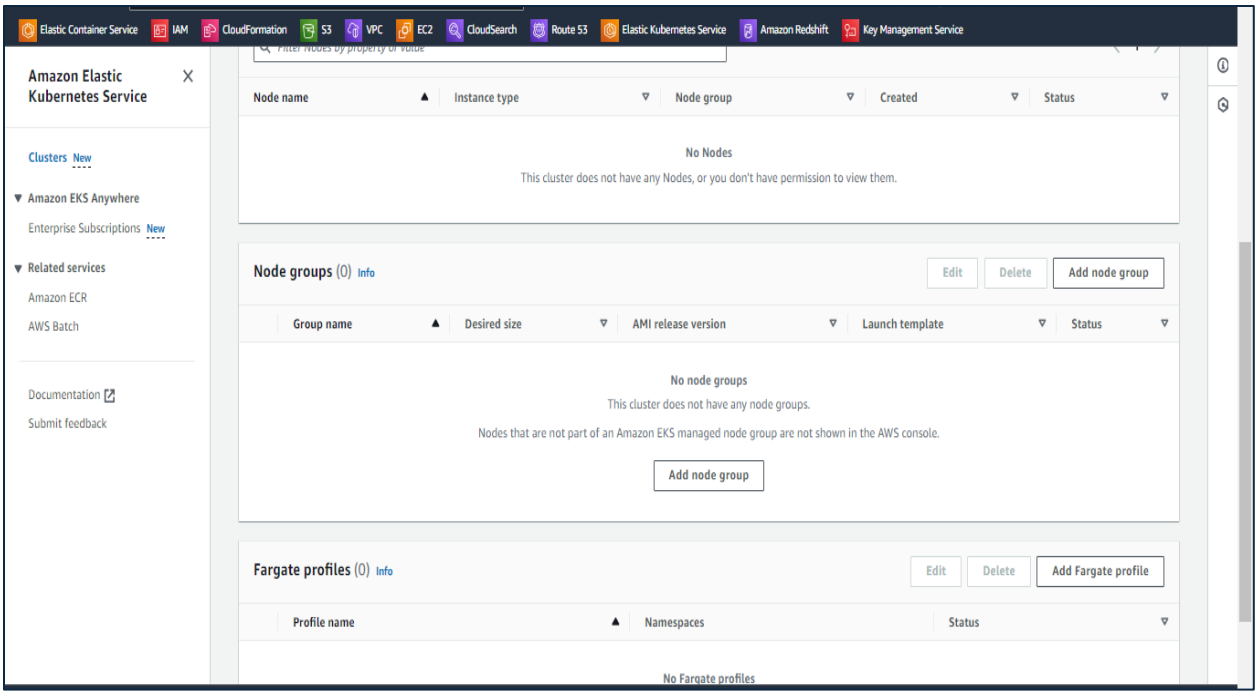
Step 18: Select vpc, subnet, security groups & click on next & click on create a cluster.



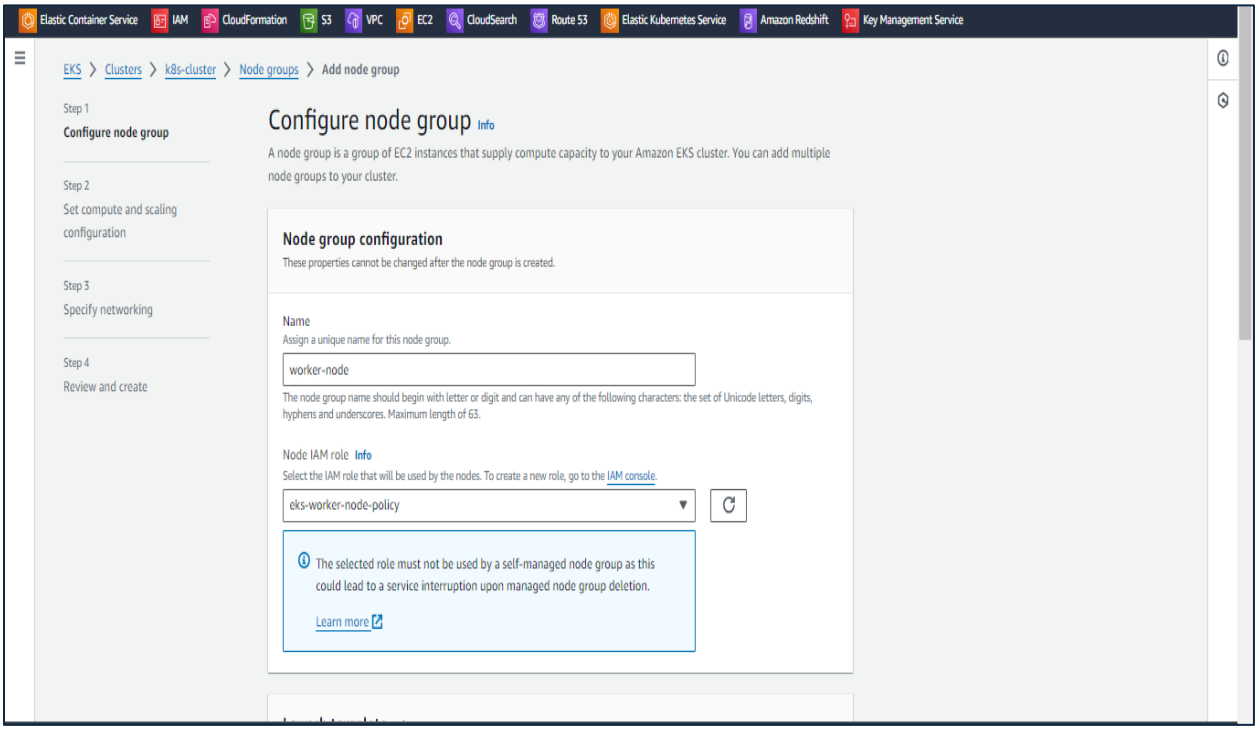
Step 19: Here, our k8s cluster has been created.



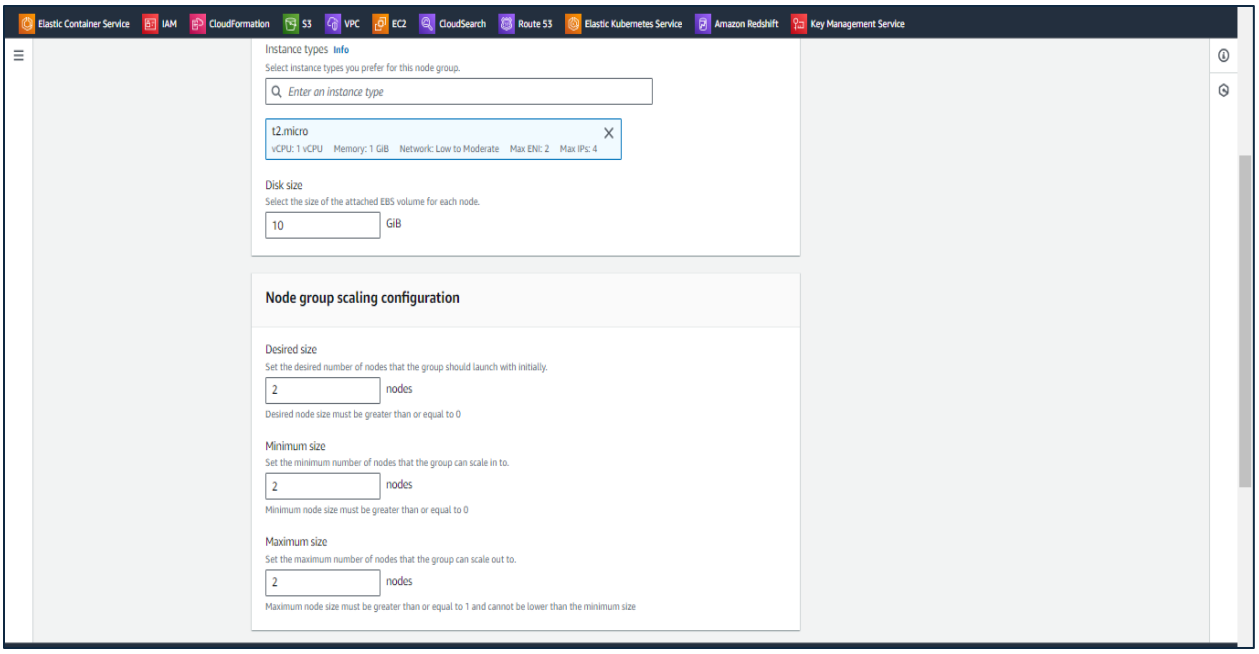
Step 20: After creating a cluster, navigate the compute section & create a node group within a cluster.

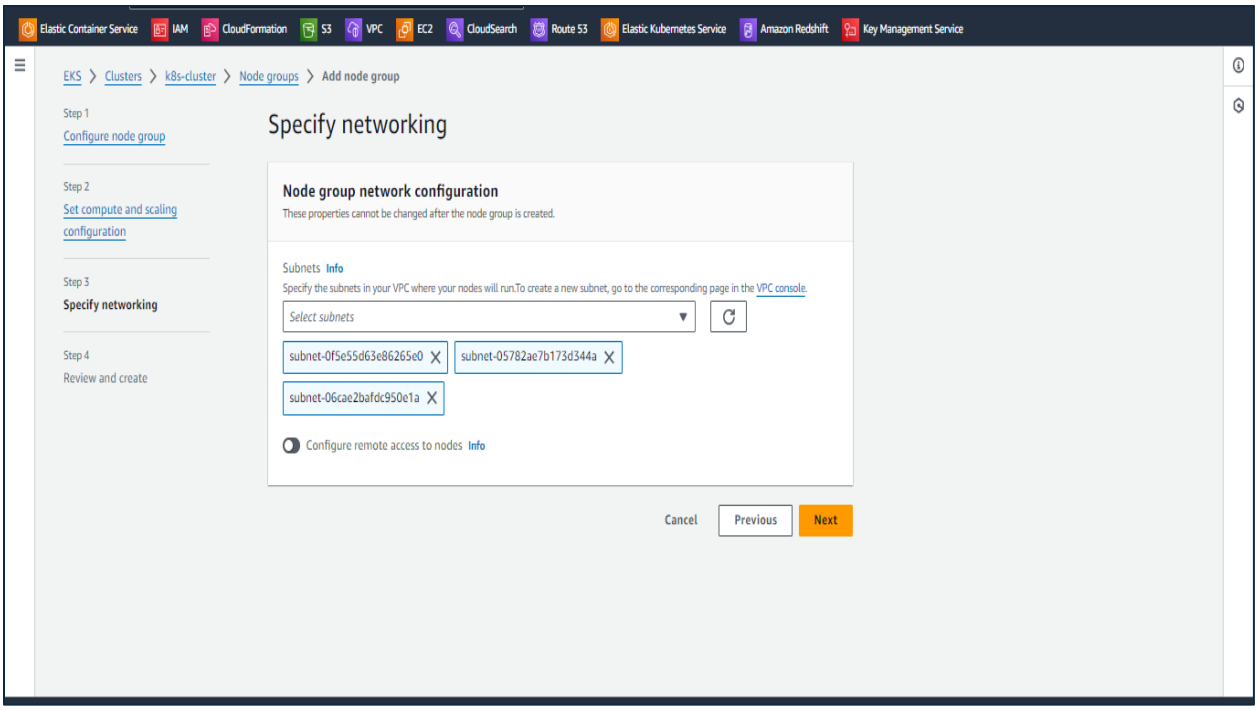


Step 21: Give name & attach role (access for Eks and ec2).

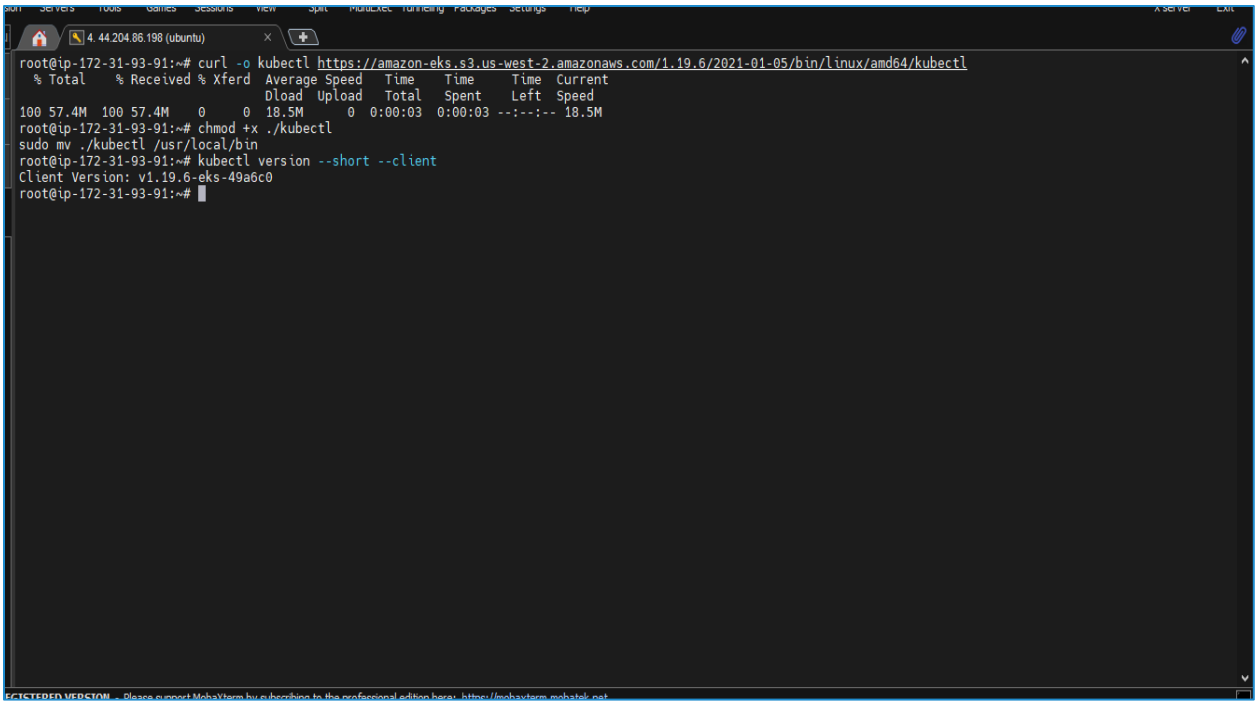


Step 22: Then, select configuration (instance size t3. medium, nodes capacity, subnets) & click on create.

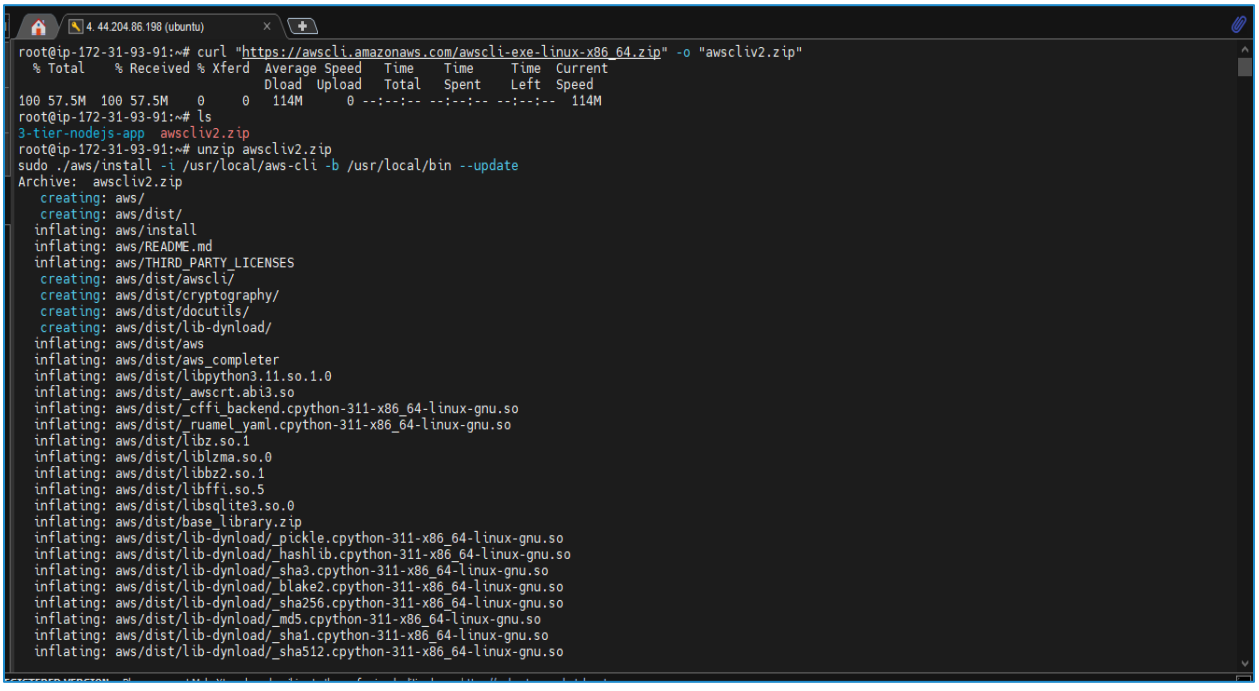




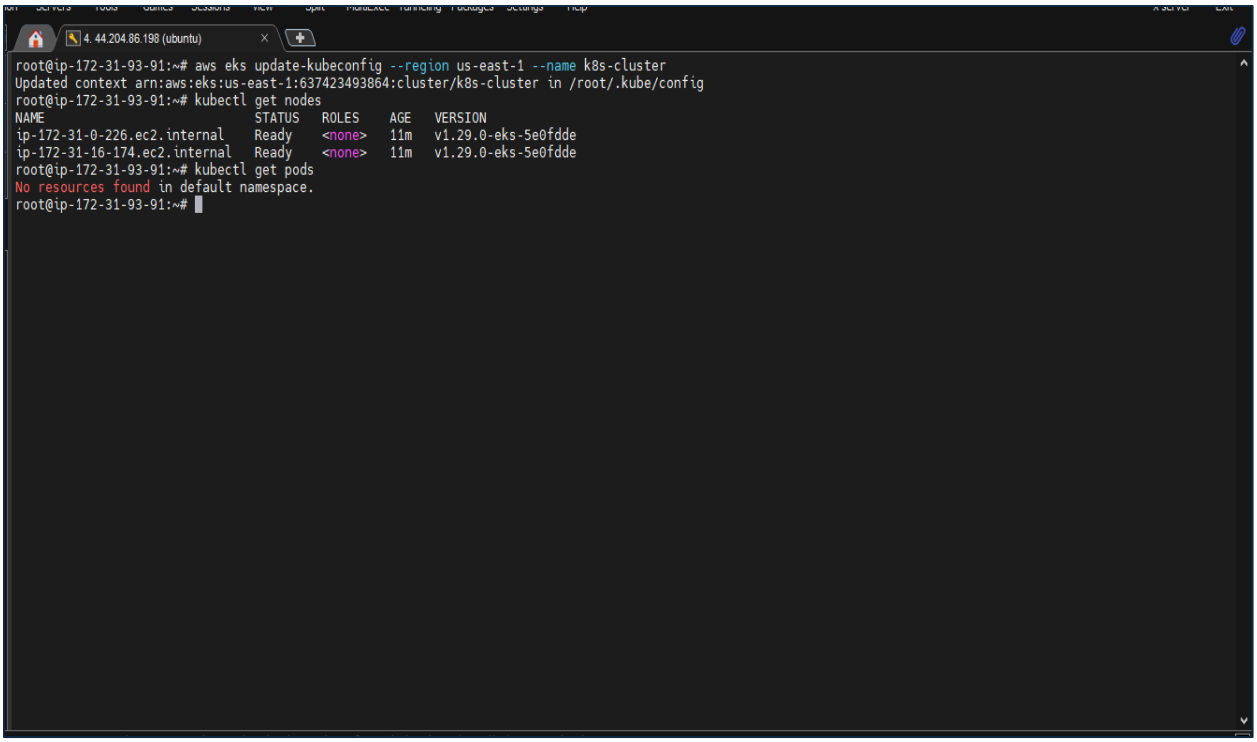
Step 23: Then, install kubectl into our ec2 instance to interact with k8s cluster.



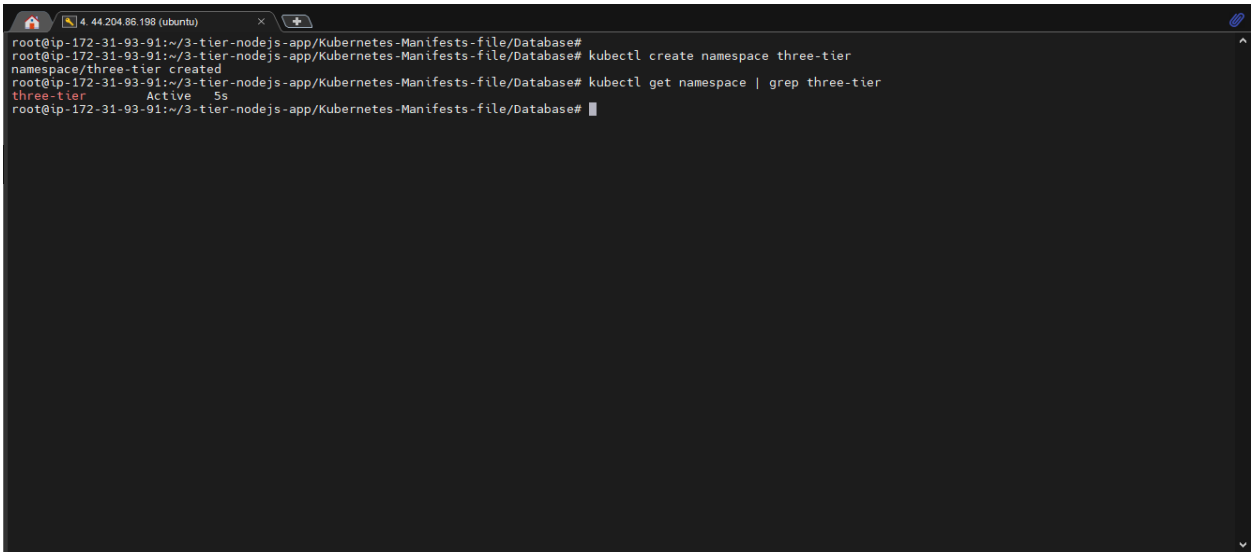
Step 24: next, install awscli and configure using access key and secret key.



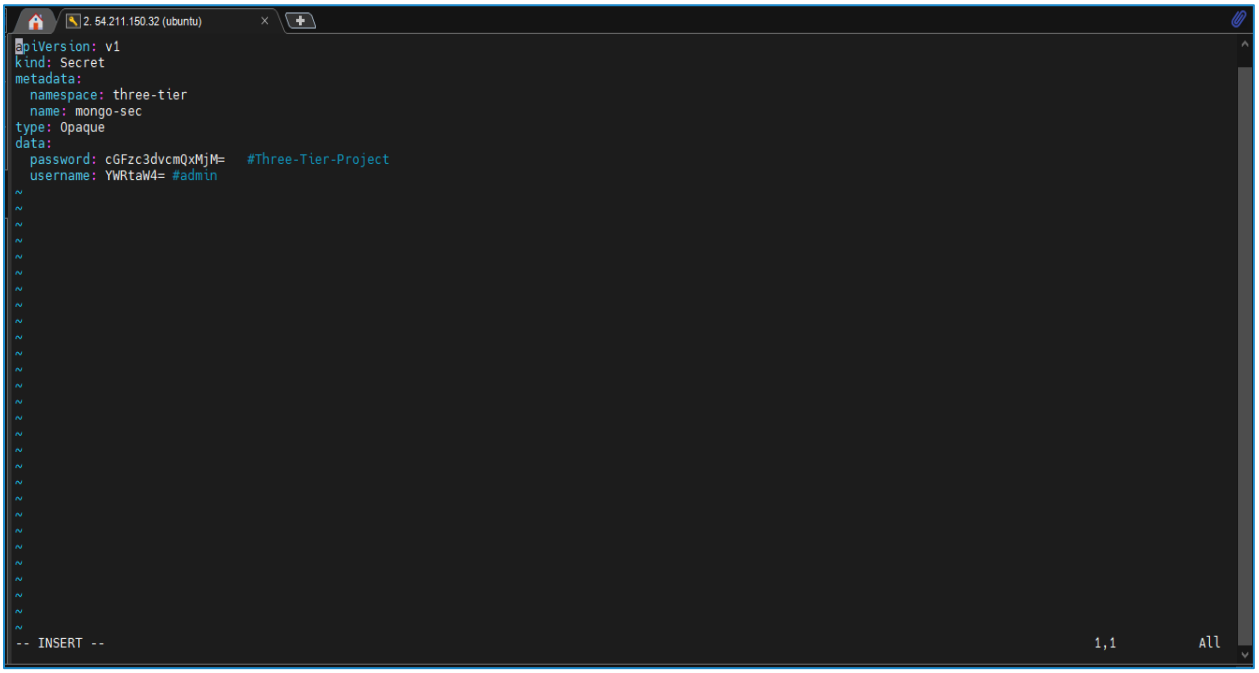
Step 25: connect k8s cluster using aws eks update-kubeconfig –-name cluster-name & check nodes.



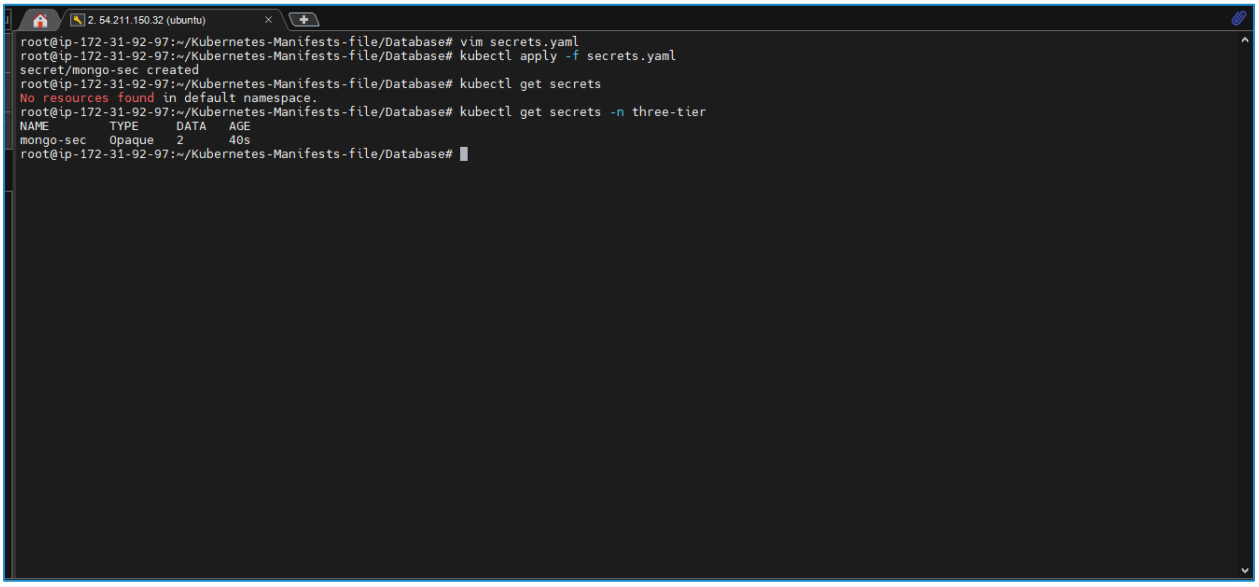
Step 26: then, create a database secret manifest file that contains the secret username and password. So, let's create a three-tier namespace.



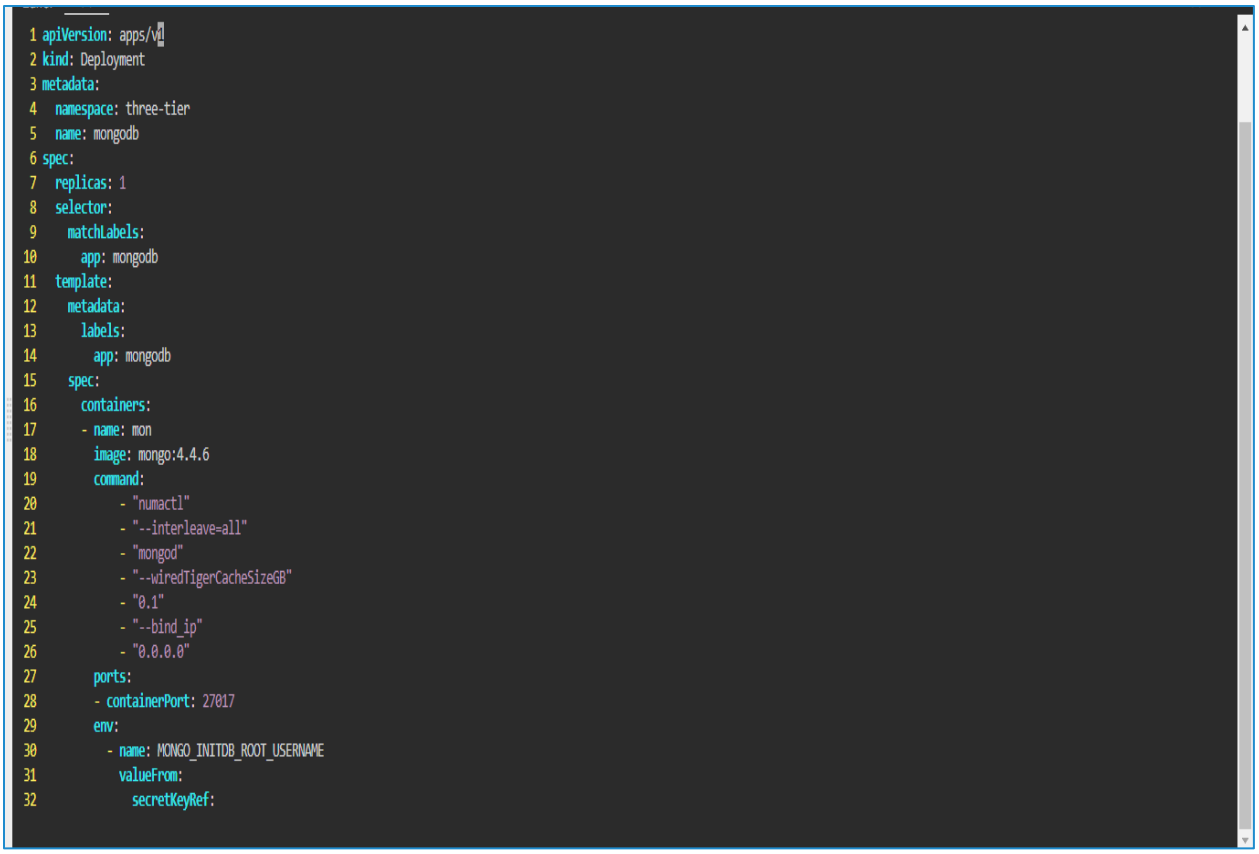
Step 27: then, create a database secrets manifest file that contains the secret username and password.



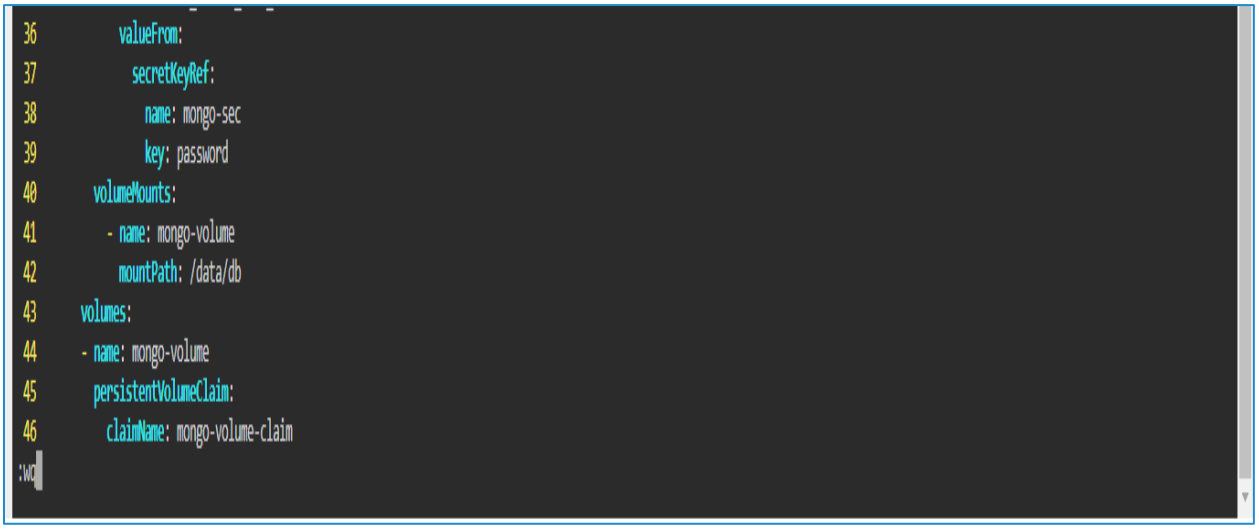
Step 28: After creating a secret file, apply it using the kubectl apply command.



Step 29: Next, create deployment file for database that specifies1 replica for database.

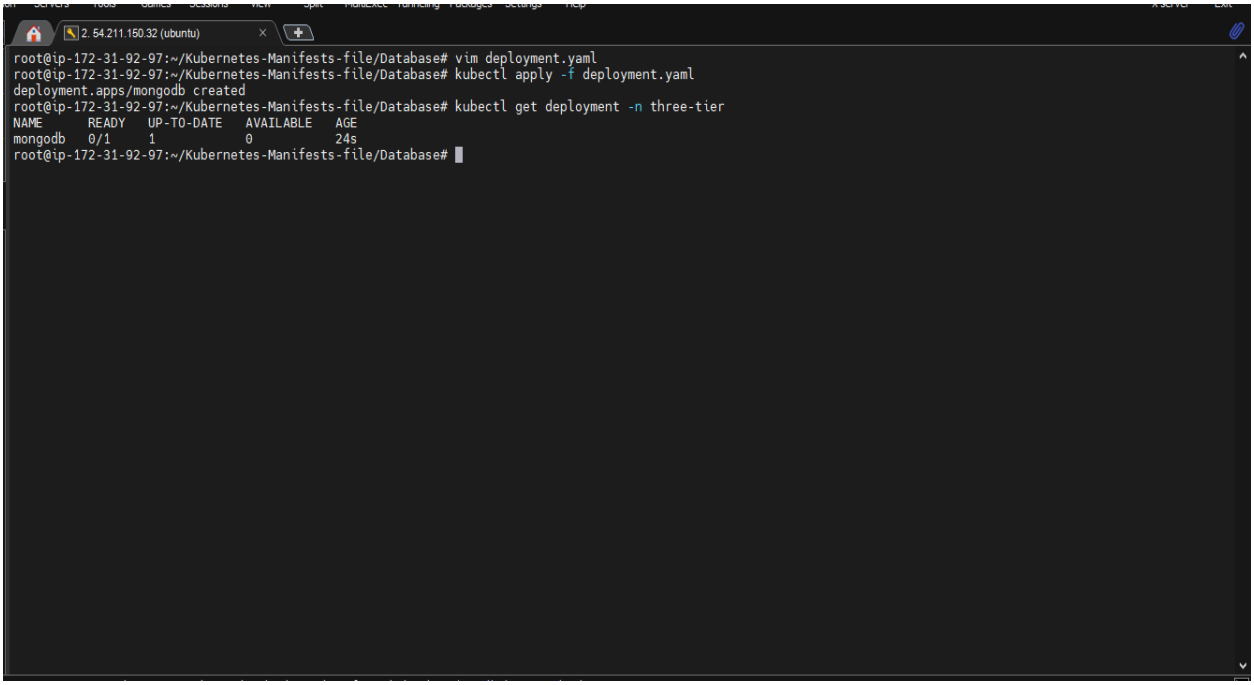


```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: three-tier
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mon
        image: mongo:4.4.6
        command:
            - "numactl"
            - "--interleave=all"
            - "mongod"
            - "--wiredTigerCacheSizeGB"
            - "0.1"
            - "--bind_ip"
            - "0.0.0.0"
        ports:
        - containerPort: 27017
        env:
          - name: MONGO_INITDB_ROOT_USERNAME
            valueFrom:
              secretKeyRef:
```

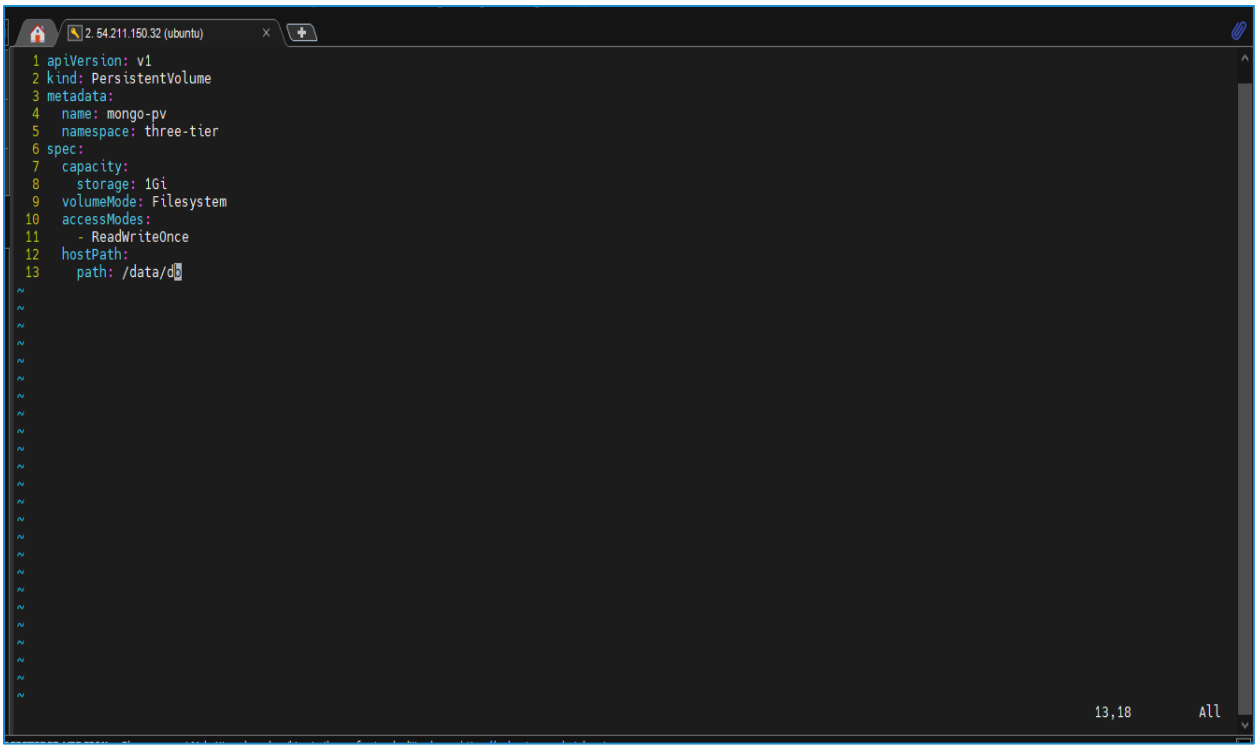


```
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: password
        volumeMounts:
          - name: mongo-volume
            mountPath: /data/db
      volumes:
      - name: mongo-volume
        persistentVolumeClaim:
          claimName: mongo-volume-claim
:wq
```

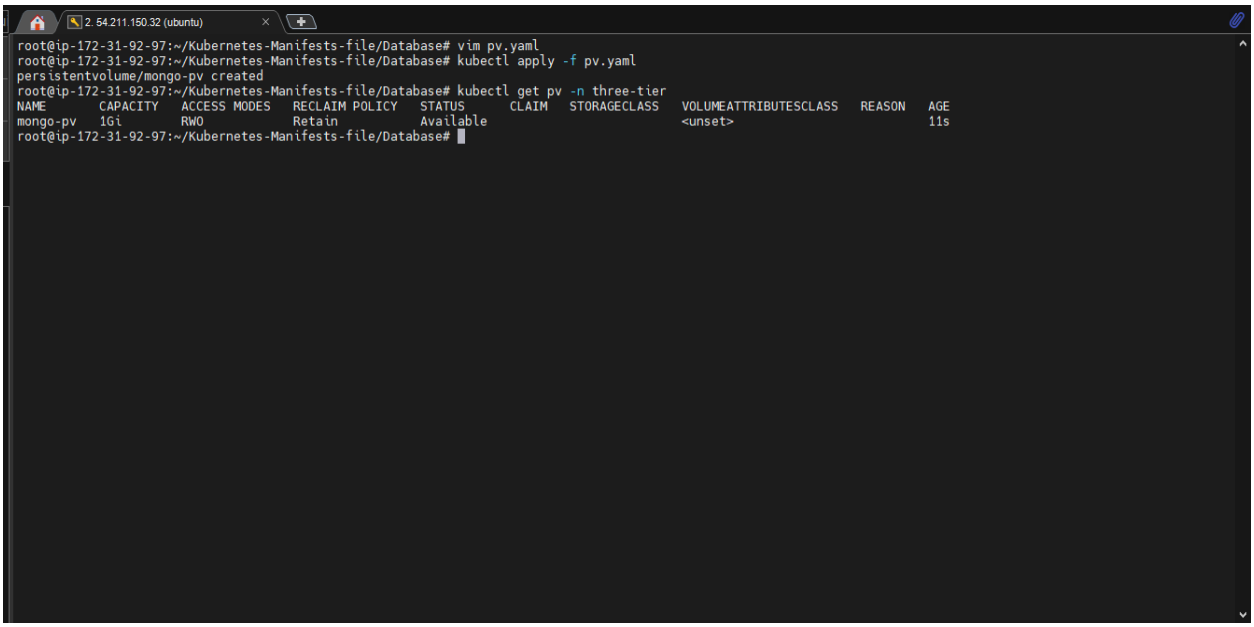
Step 30: Apply deployment manifest file using kubectl apply command.



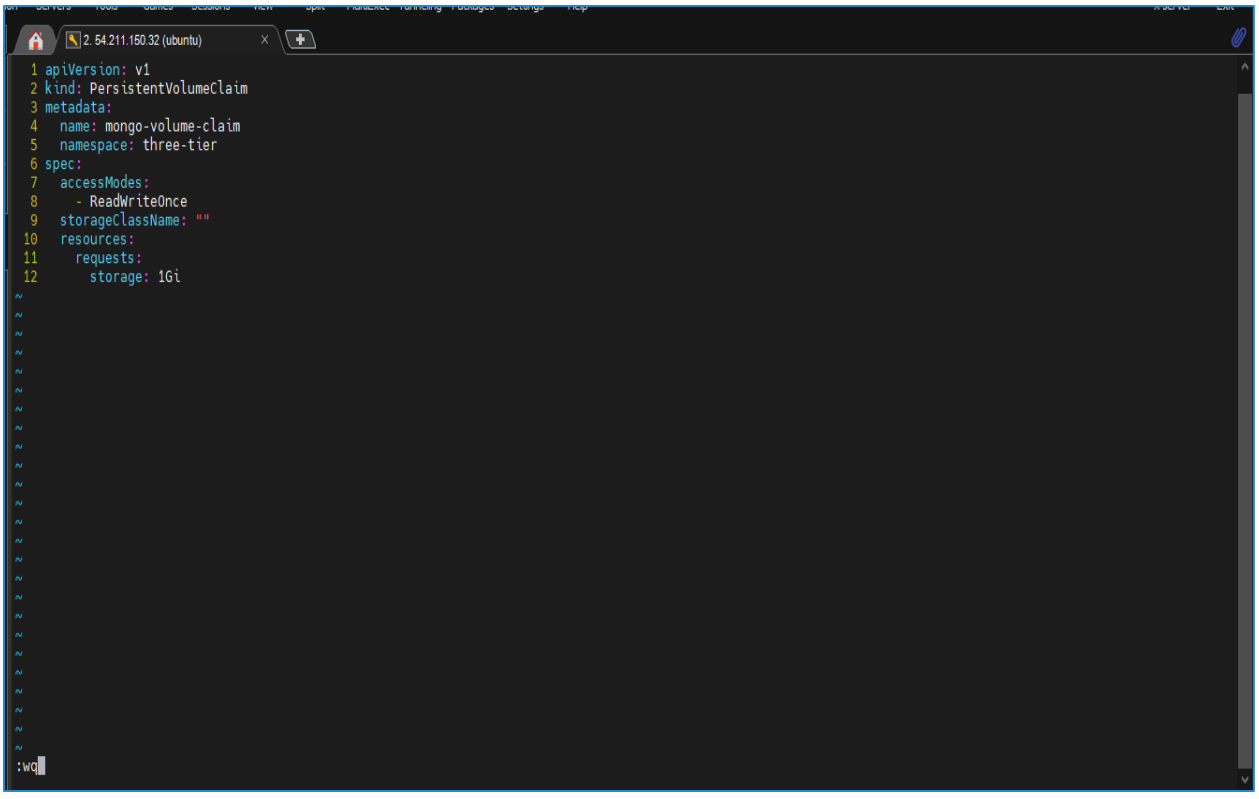
Step 31: next, create a pv volume for the database.



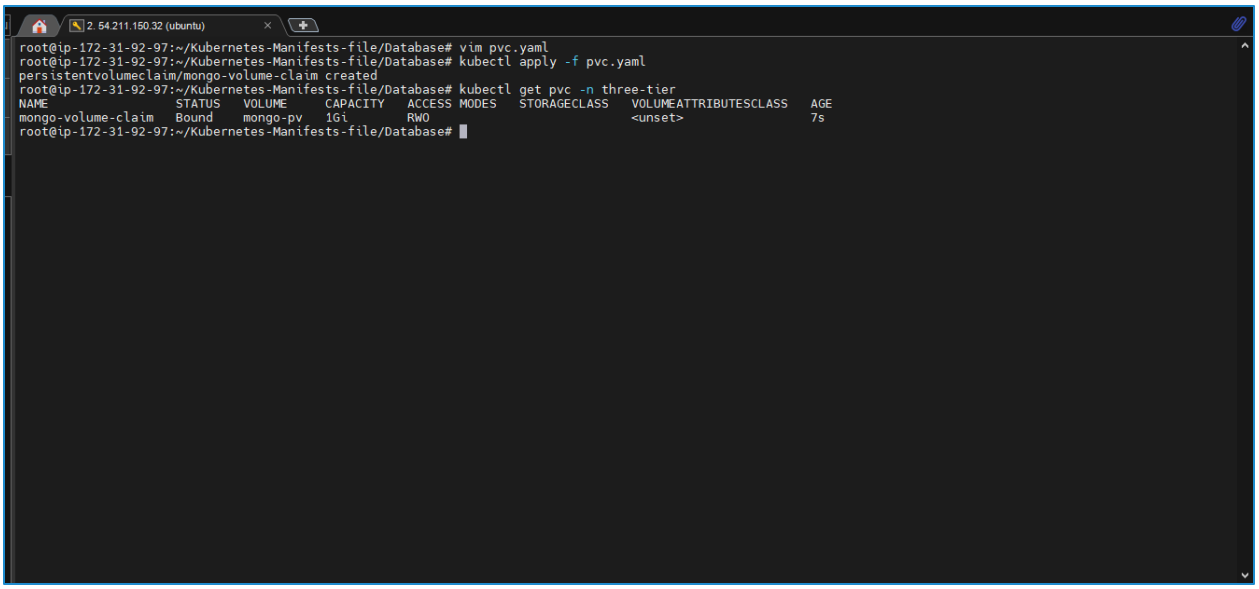
Step 32: PVmanifest file apply using kubectl apply.



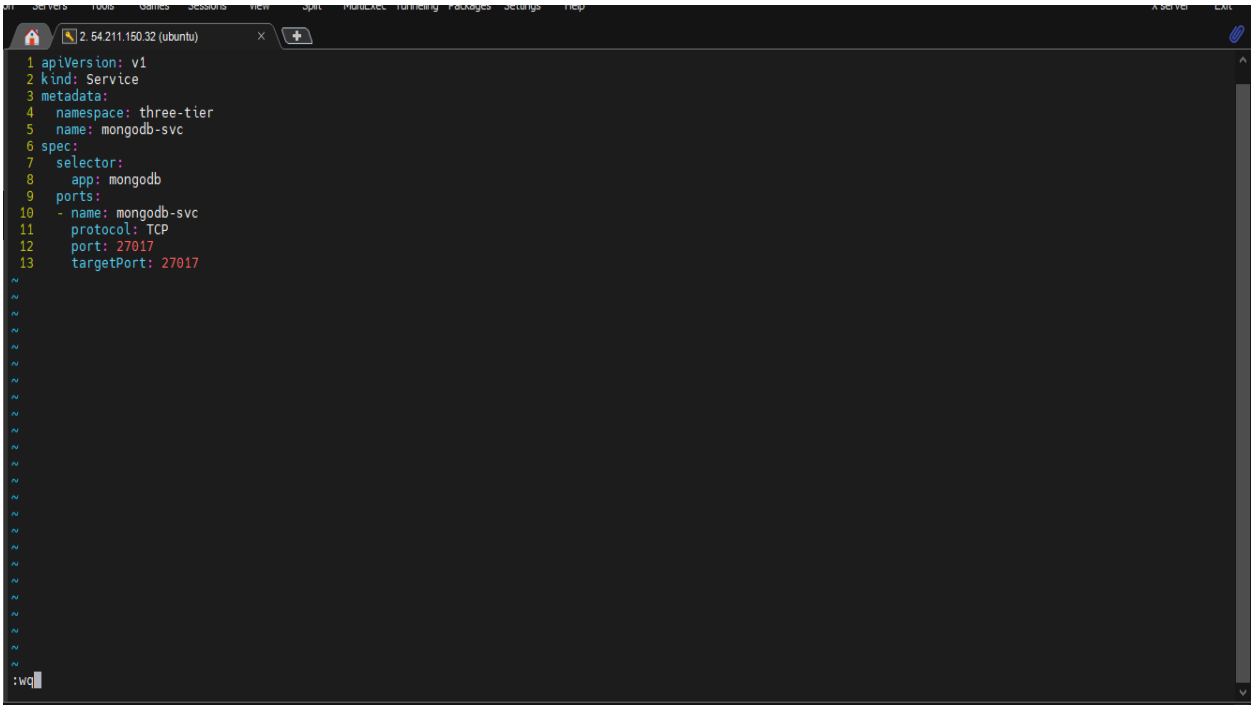
Step 33: Then, create persistent volume claim(pvc) manifest file.



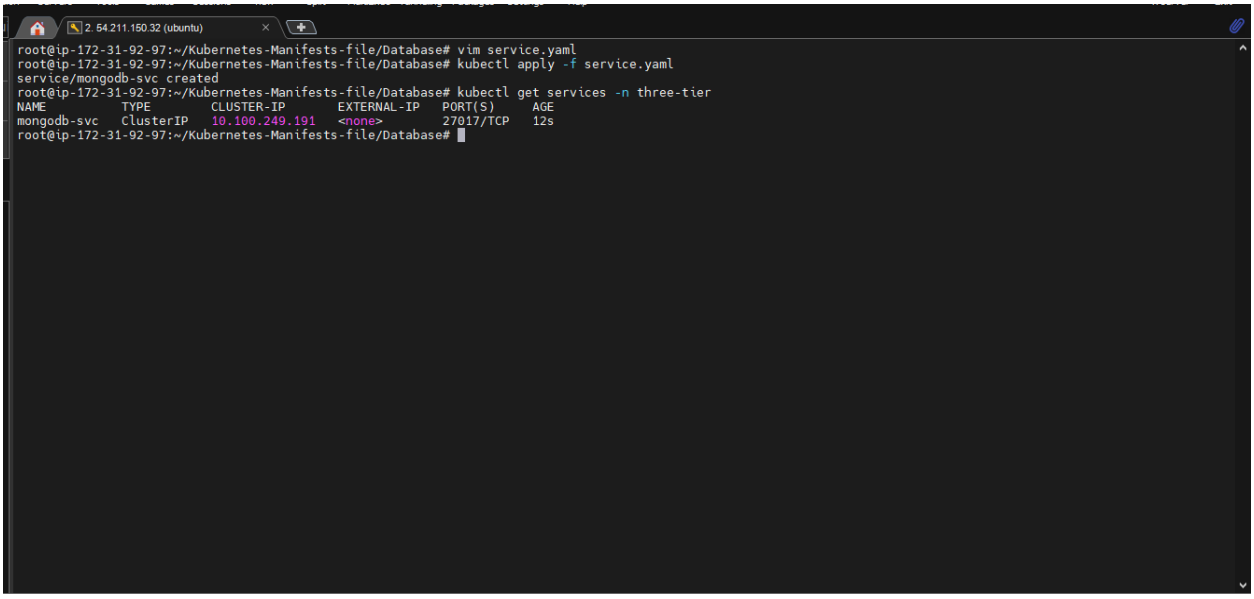
Step 34: Apply pvc manifest file using kubectl apply command.



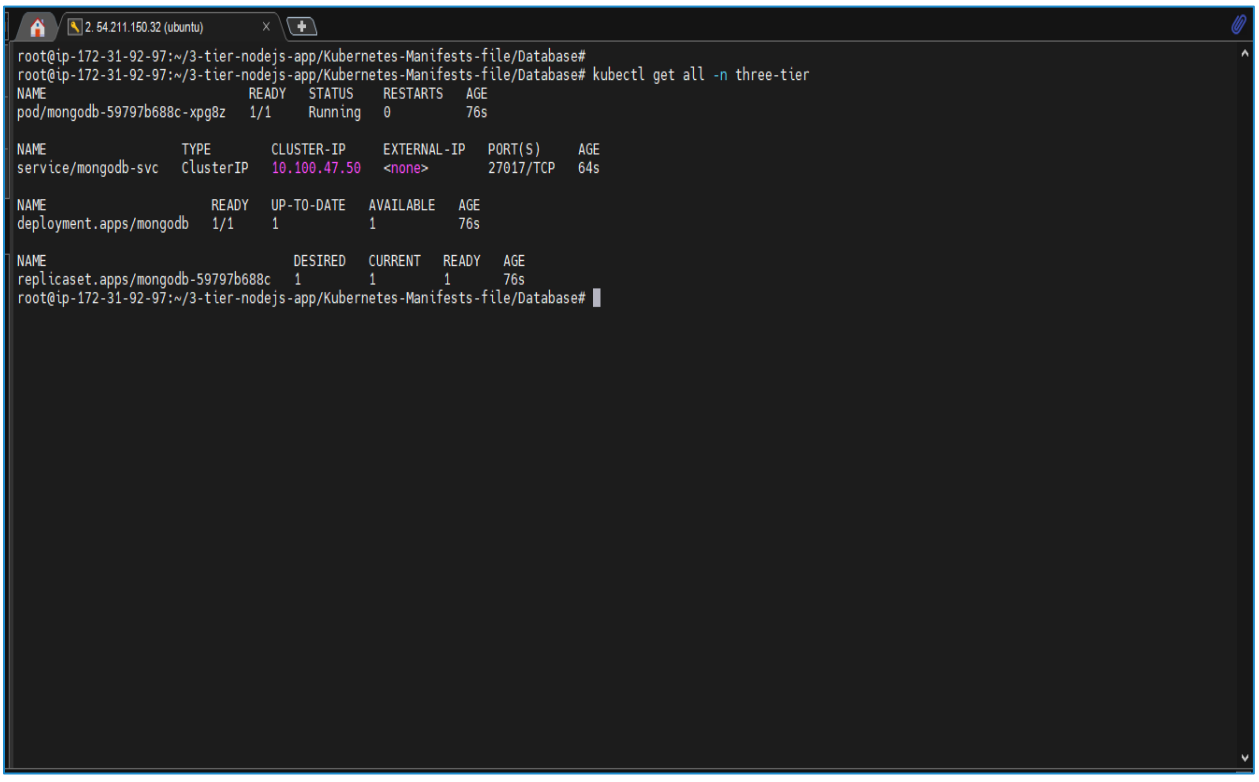
Step 35: Next, create cluster Ip service for the MongoDB database.



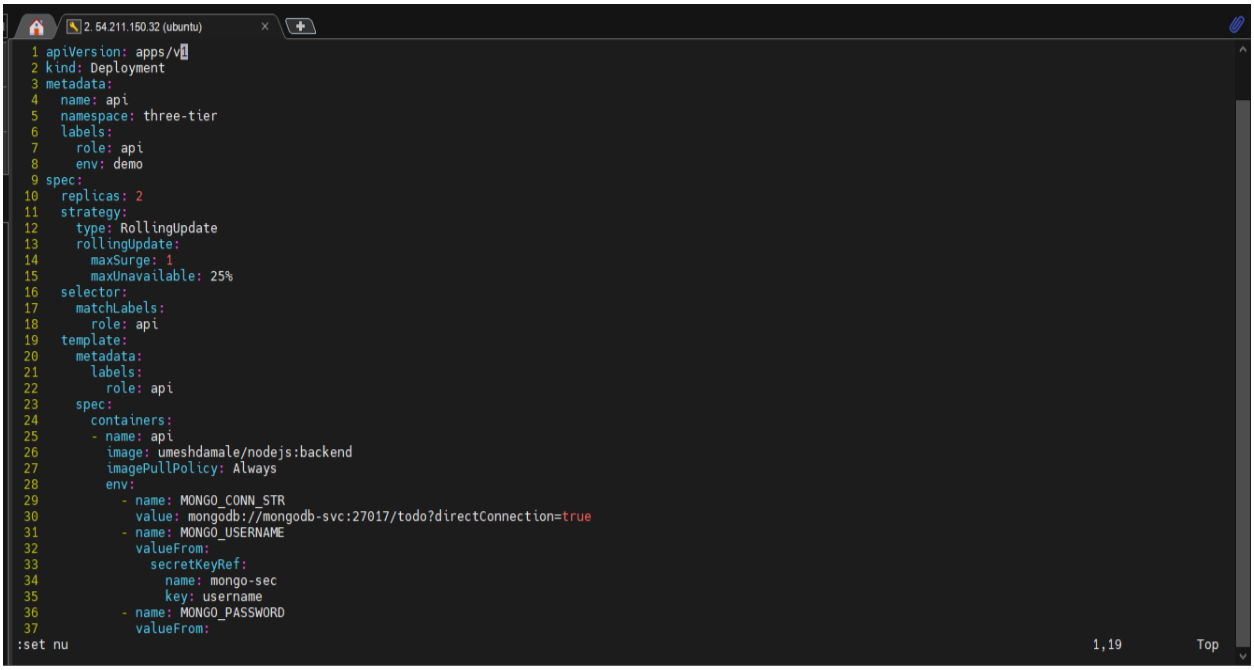
Step 36: Apply service using kubectl apply command.



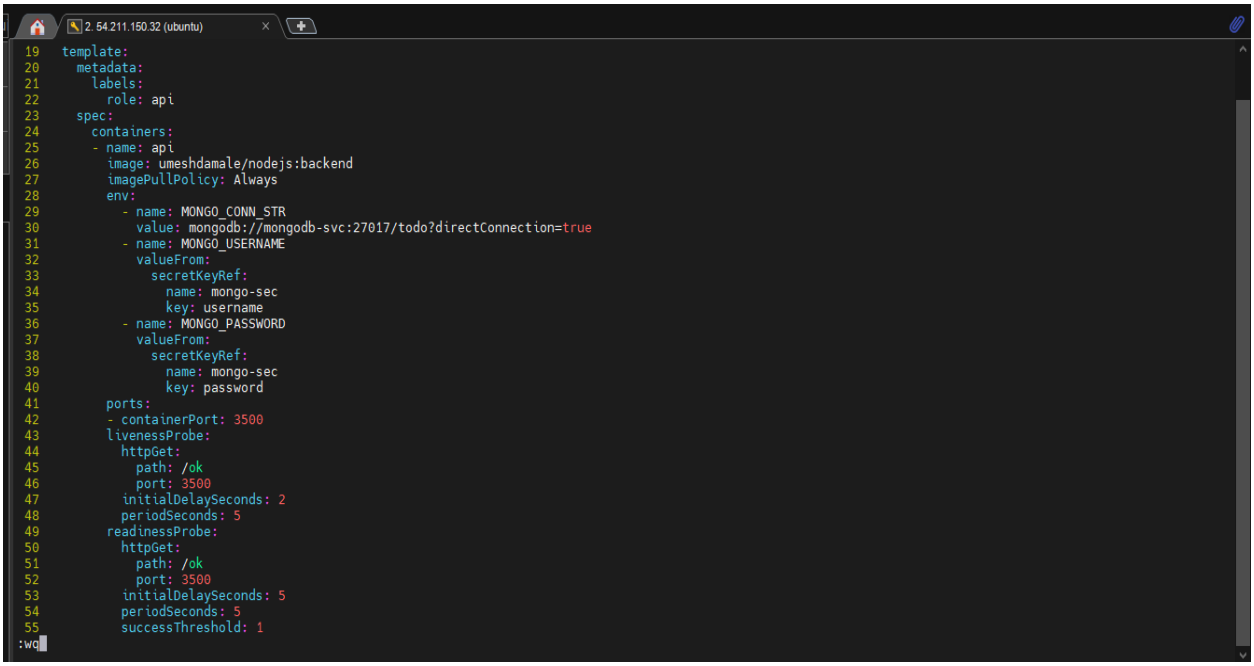
Step 37: Here, all the resources have been created for the MongoDB database.



Step 38: Next, create backend deployment manifest file & that specifies two replicas set.

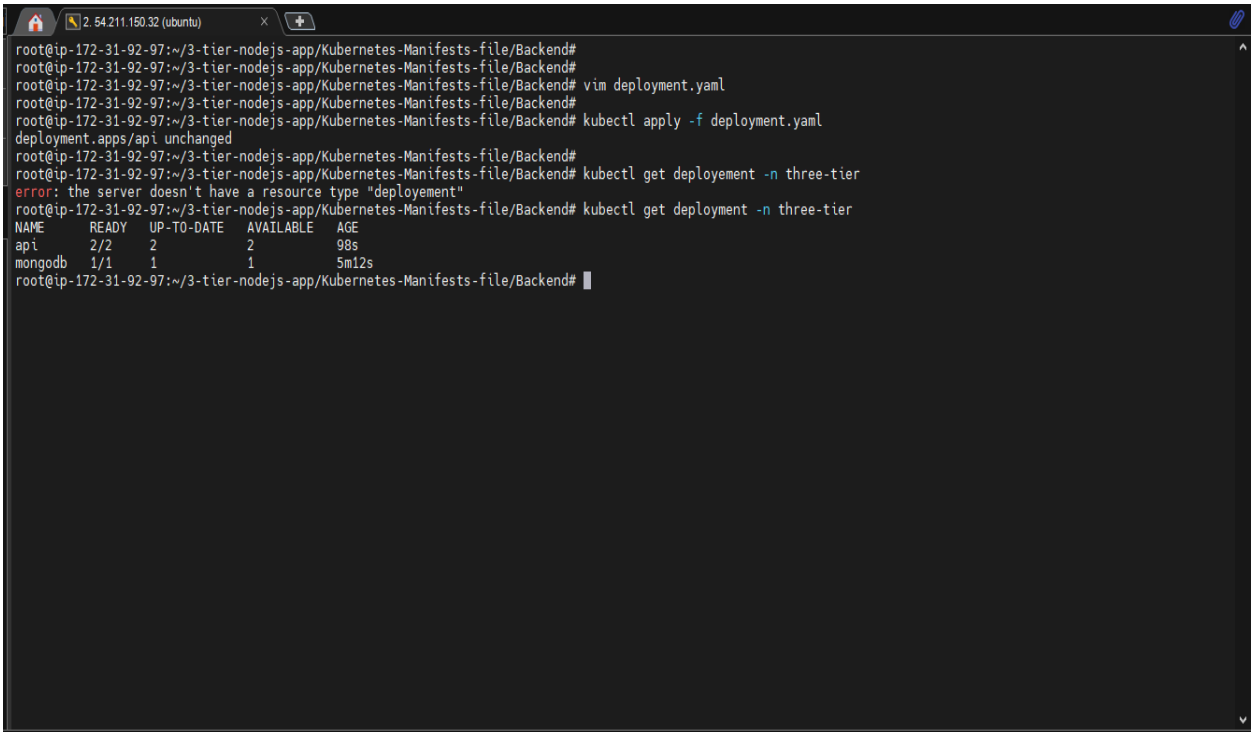


```
2. 54.211.150.32 (ubuntu)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api
  namespace: three-tier
  labels:
    role: api
    env: demo
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 25%
  selector:
    matchLabels:
      role: api
  template:
    metadata:
      labels:
        role: api
    spec:
      containers:
      - name: api
        image: umeshdamale/nodejs:backend
        imagePullPolicy: Always
        env:
          - name: MONGO_CONN_STR
            value: mongodb://mongodb-svc:27017/todo?directConnection=true
          - name: MONGO_USERNAME
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: username
          - name: MONGO_PASSWORD
            valueFrom:
:set nu                                                          1,19          Top
```

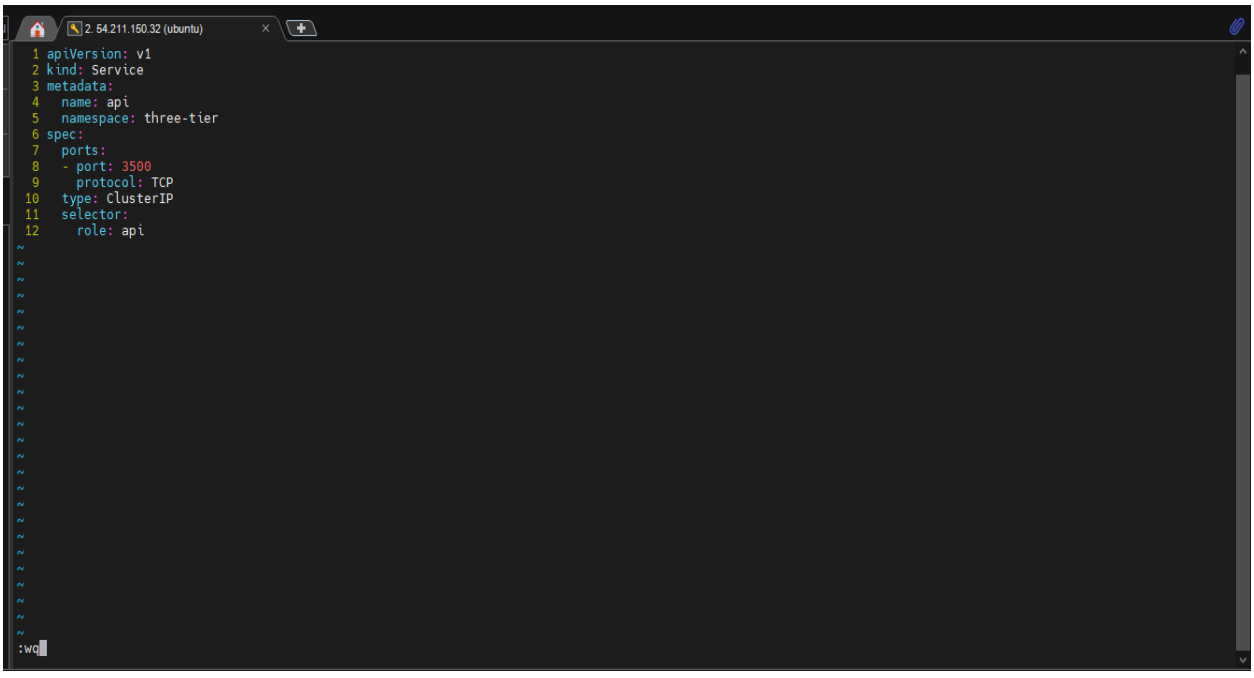


```
2. 54.211.150.32 (ubuntu)
  template:
    metadata:
      labels:
        role: api
    spec:
      containers:
      - name: api
        image: umeshdamale/nodejs:backend
        imagePullPolicy: Always
        env:
          - name: MONGO_CONN_STR
            value: mongodb://mongodb-svc:27017/todo?directConnection=true
          - name: MONGO_USERNAME
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: username
          - name: MONGO_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mongo-sec
                key: password
        ports:
        - containerPort: 3500
        livenessProbe:
          httpGet:
            path: /ok
            port: 3500
          initialDelaySeconds: 2
          periodSeconds: 5
        readinessProbe:
          httpGet:
            path: /ok
            port: 3500
          initialDelaySeconds: 5
          periodSeconds: 5
          successThreshold: 1
:wq
```

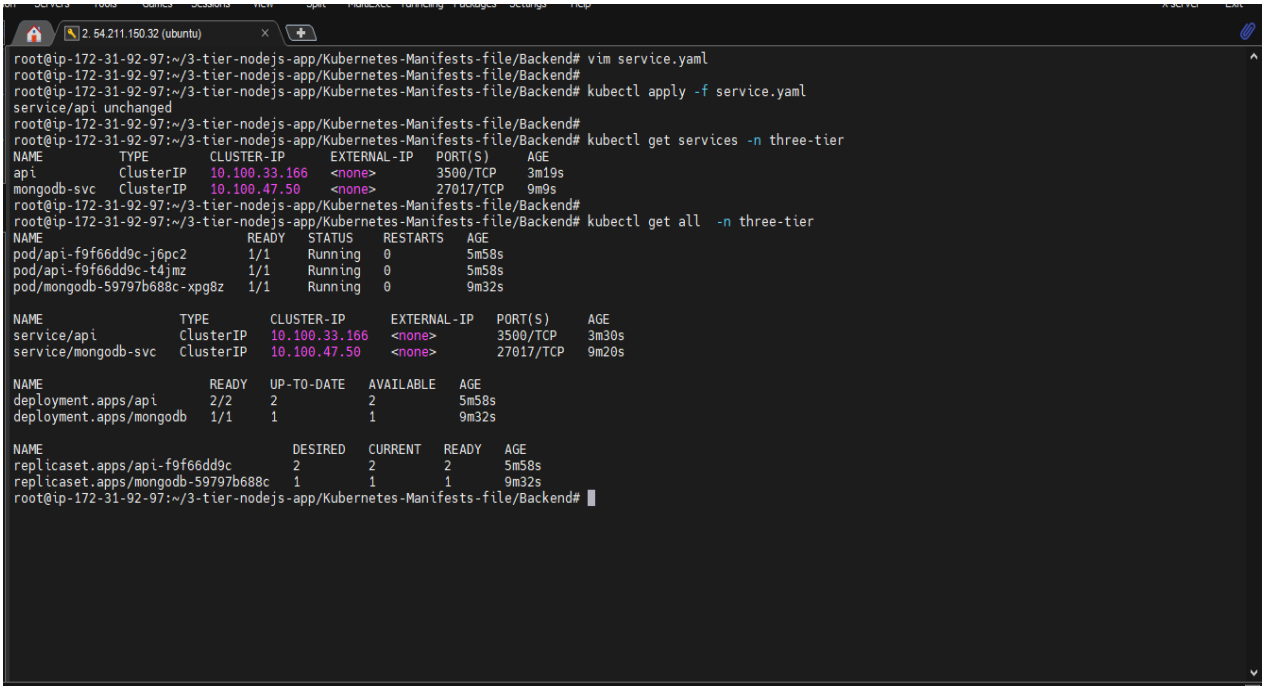
Step 39: Deployment manifest file apply using kubectl apply command after applying deployment will be created.



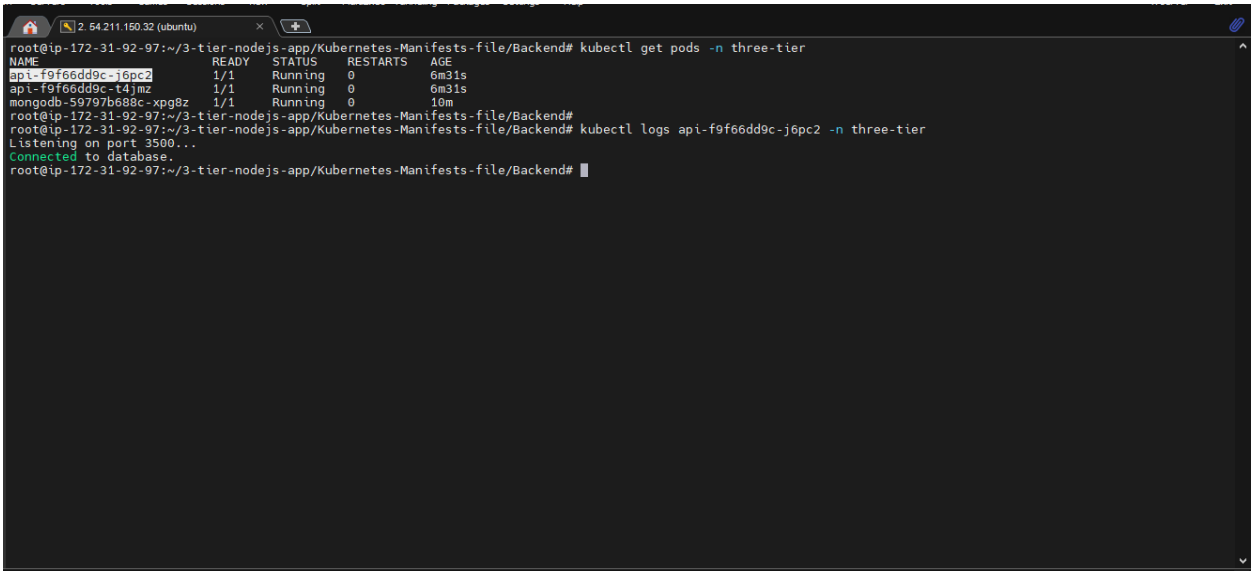
Step 40: next create cluster Ip service for the deployment to enable connectivity to service Within the cluster.



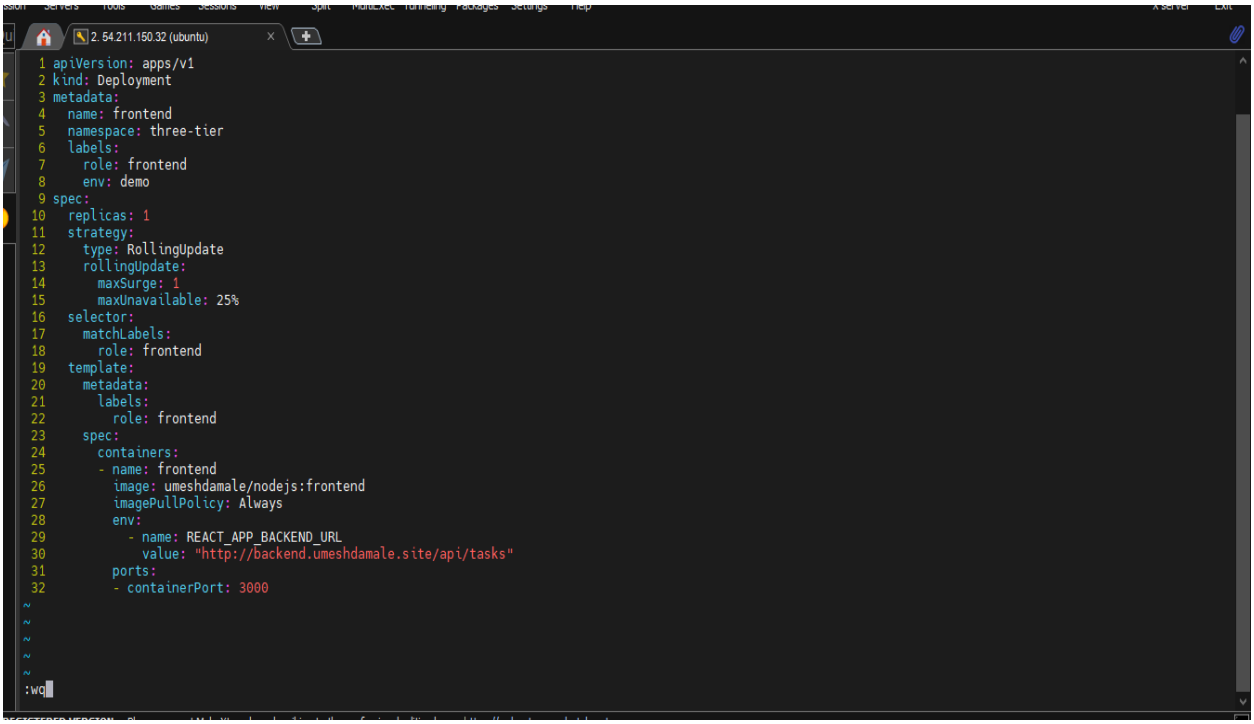
Step 41: Apply service manifest using kubectl apply command after applying service will be created.



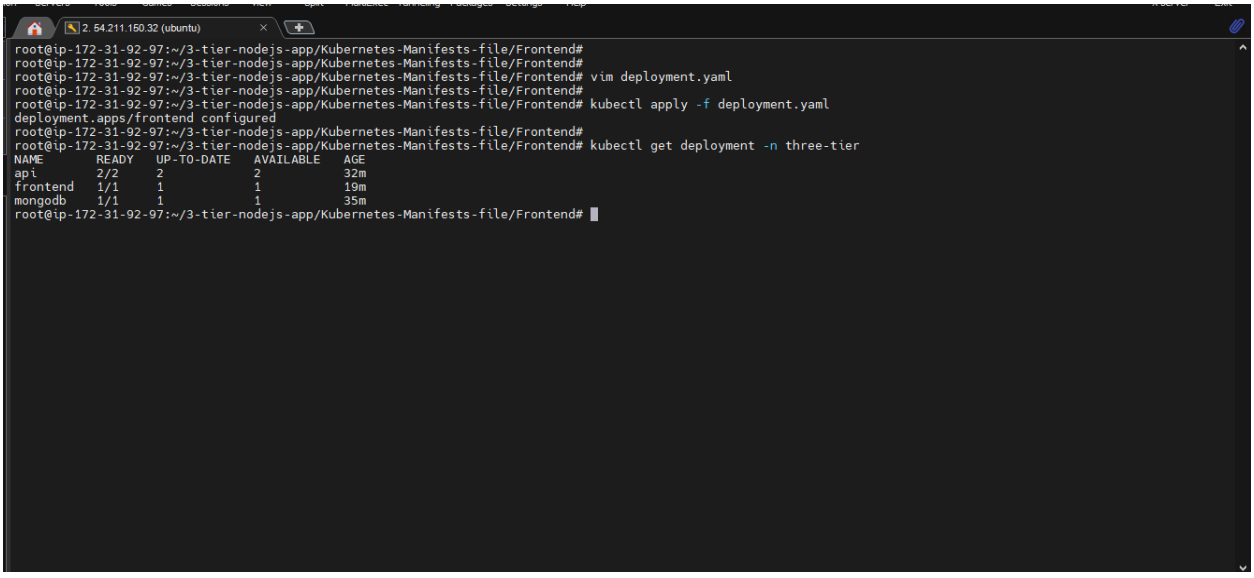
Step 42: Then, check the backend pod logs to confirm that the database has been successfully connected to the backend or not.



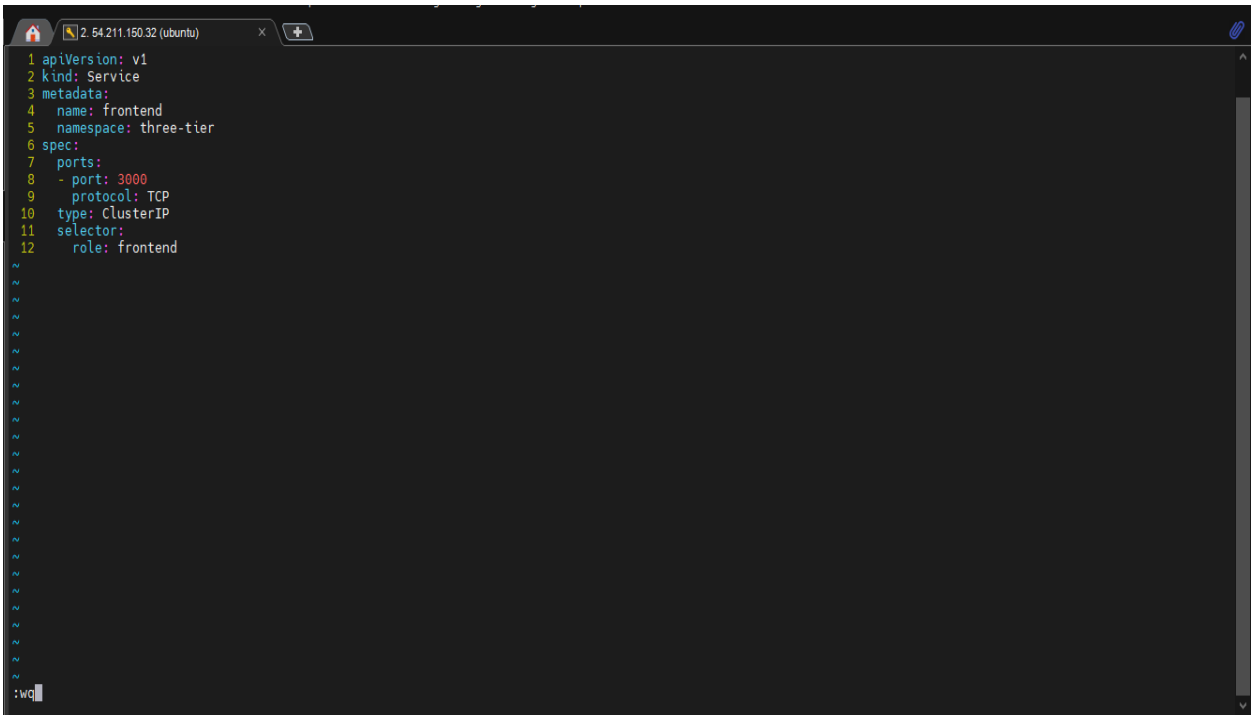
Step 43: next, create frontend deployment manifest file.



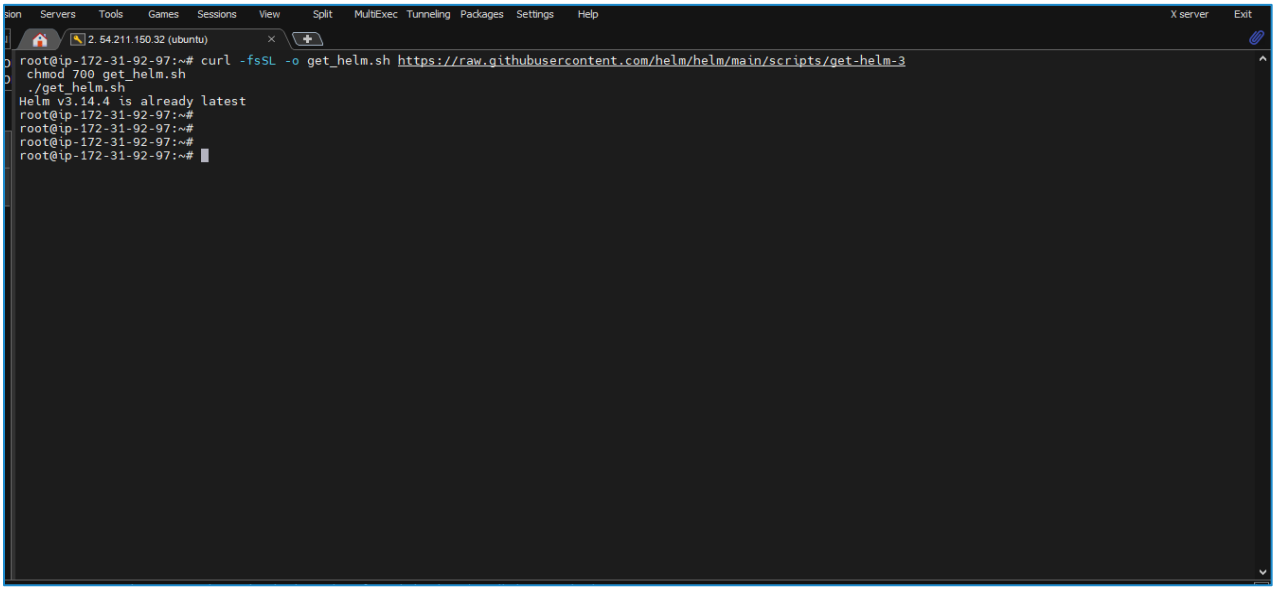
Step 44: Frontend deployment manifest file apply using kubectl apply command and after applying deployment will be create



Step 45: next, create service manifest file for deployment to enable connectivity to the service within the cluster.

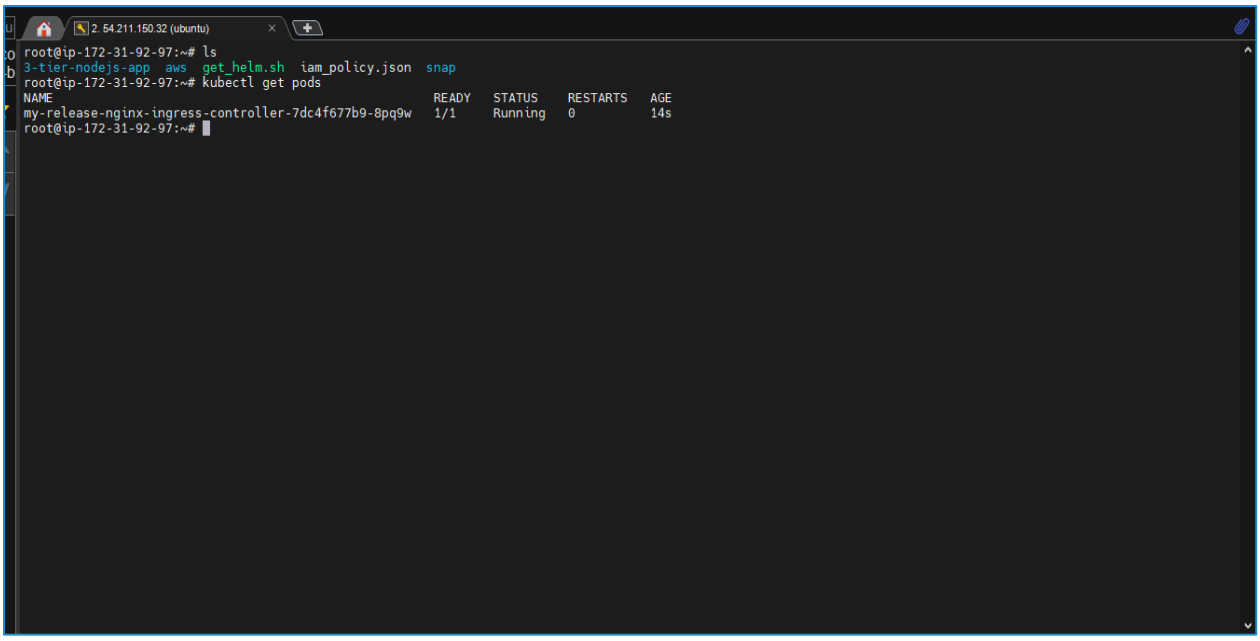


Step 46: Install helm in order to install the ingress controller.

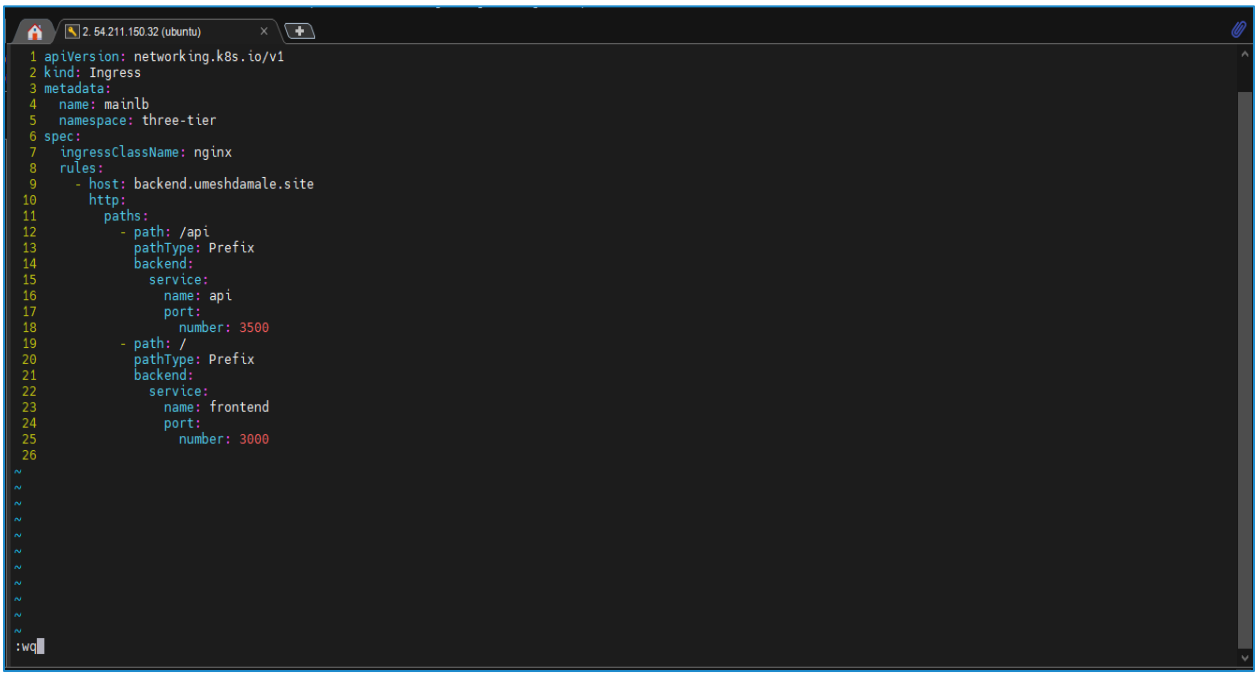


Step 47: Next install nginx ingress controller using helm.

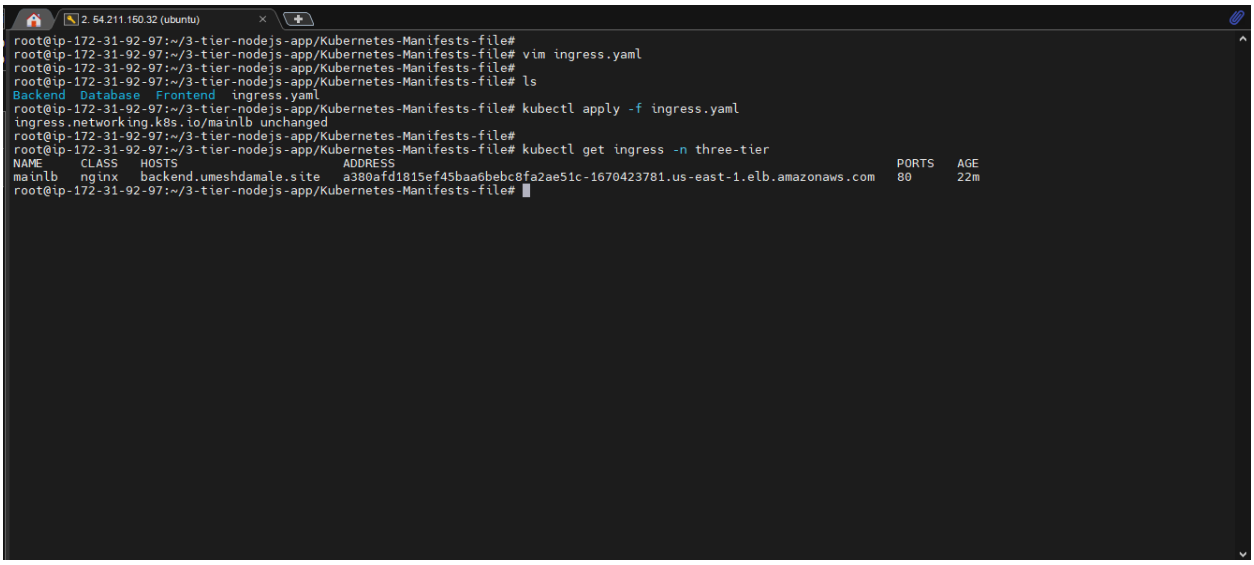
helm install my-release oci://ghcr.io/nginxinc/charts/nginx-ingress --version 1.1.2.



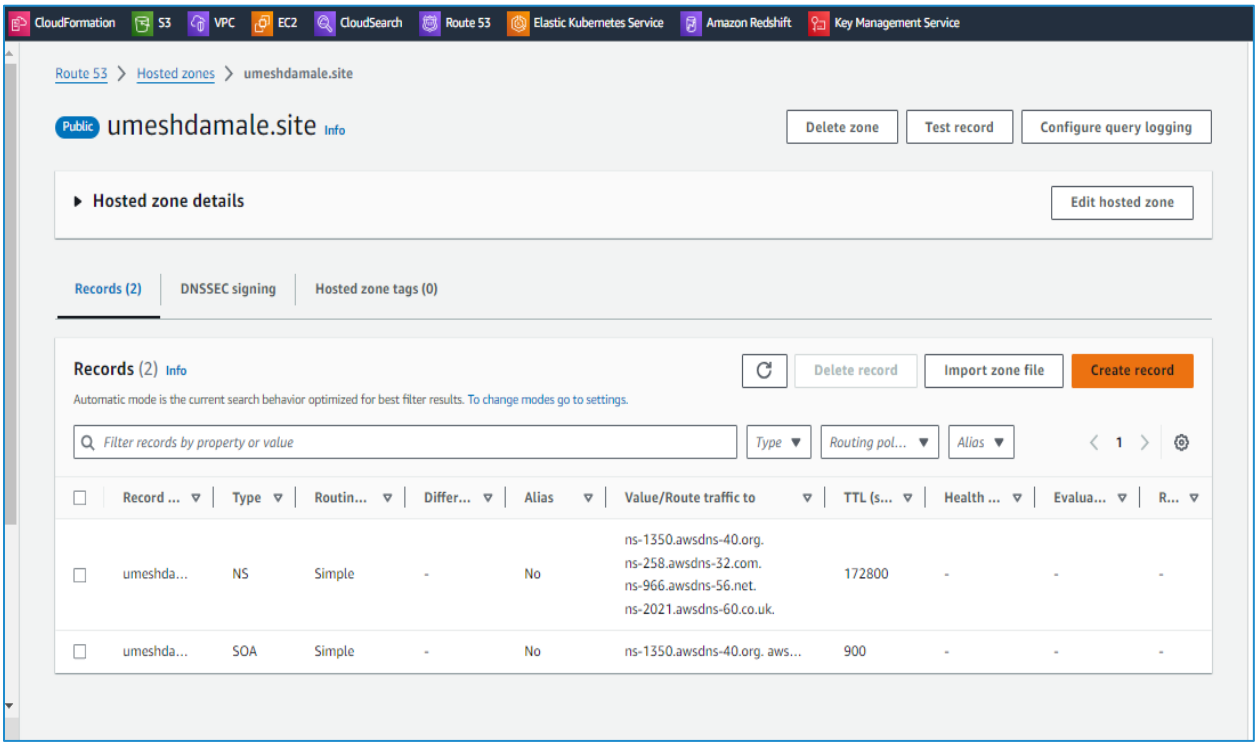
Step 48: Create ingress manifest file in that specifies ingrss class name as nginx and define the domain name in the host field.

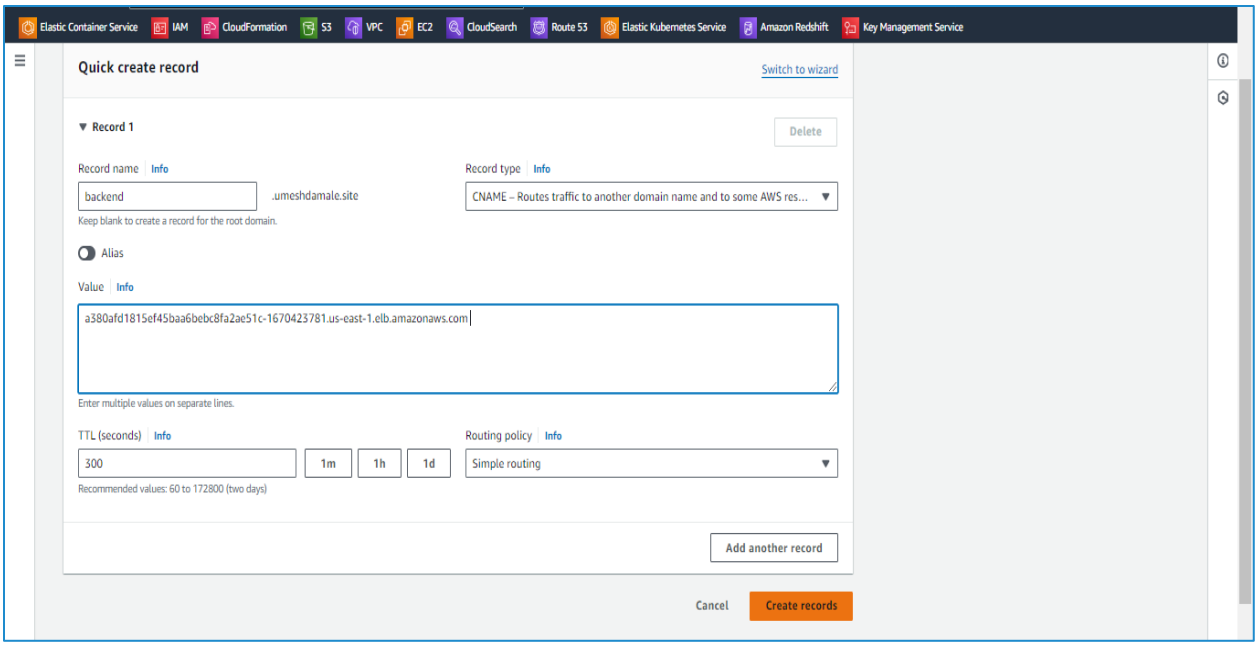


Step 49: Apply ingress manifest using kubectl apply command after applying obtain the address of the ingress.

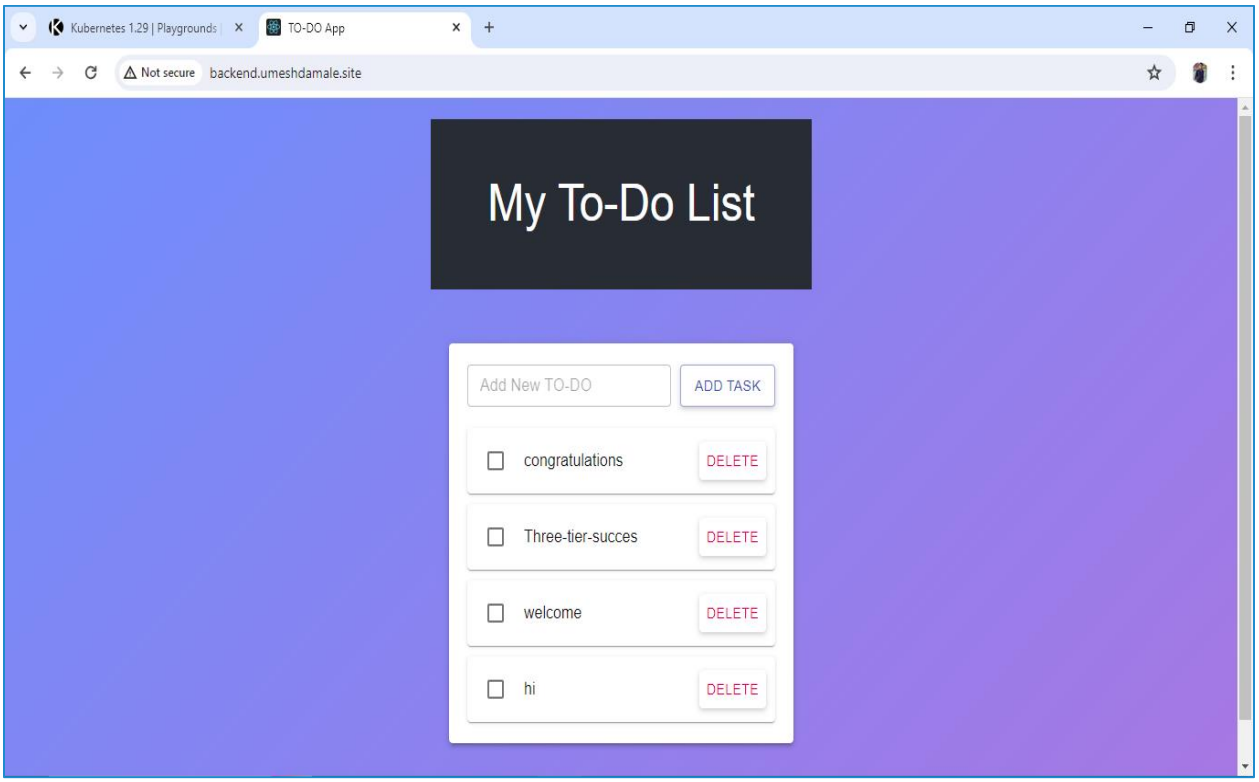


Step 50: Update the Route53 hosted zone by creating a CNAME record and paste the load balancer endpoint into it.

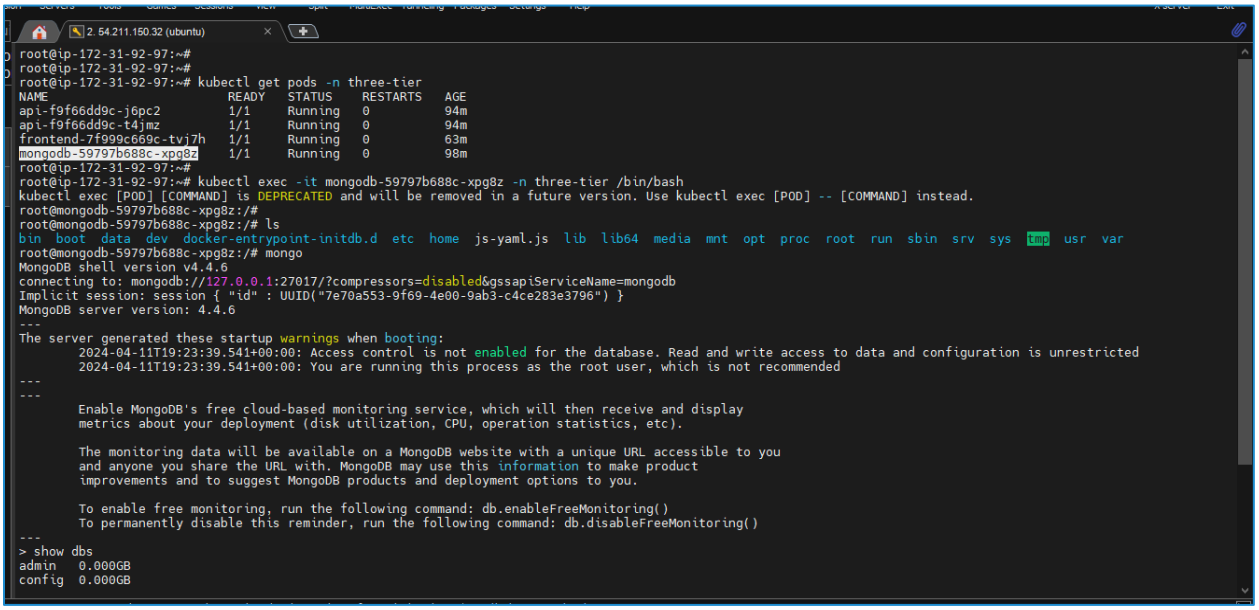




Step 51: Here we have successfully completed 3 tier architecture. We can access the application by visiting the domain name backend.umeshdamale.site in browser.



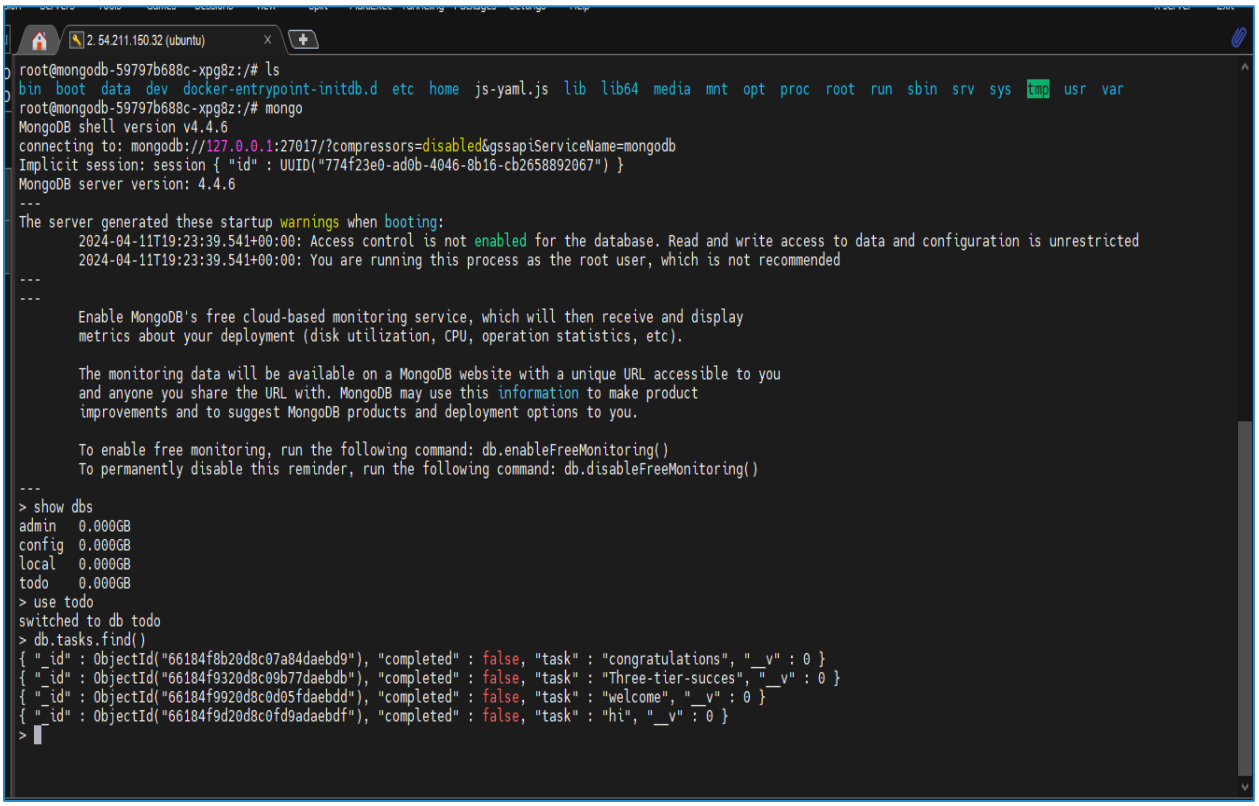
Step 52: Now, let's check if our input data is saved in the MongoDB database.



```
2. 54.211.150.32 (ubuntu)
root@ip-172-31-92-97:~#
root@ip-172-31-92-97:~#
root@ip-172-31-92-97:~# kubectl get pods -n three-tier
NAME                        READY   STATUS    RESTARTS   AGE
api-f9f66dd9c-j6pc2         1/1     Running   0          94m
api-f9f66dd9c-t4jmz         1/1     Running   0          94m
frontend-7f999c669c-tvj7h   1/1     Running   0          63m
mongodb-59797b688c-xpg8z    1/1     Running   0          98m
root@ip-172-31-92-97:~#
root@ip-172-31-92-97:~# kubectl exec -it mongodb-59797b688c-xpg8z -n three-tier /bin/bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@mongodb-59797b688c-xpg8z:/#
root@mongodb-59797b688c-xpg8z:/# ls
bin  boot  data  dev  docker-entrypoint-initdb.d  etc  home  js-yaml.js  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@mongodb-59797b688c-xpg8z:/# mongo
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("7e70a553-9f69-4e00-9ab3-c4ce283e3796") }
MongoDB server version: 4.4.6
---
The server generated these startup warnings when booting:
        2024-04-11T19:23:39.541+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
        2024-04-11T19:23:39.541+00:00: You are running this process as the root user, which is not recommended
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin   0.000GB
config  0.000GB
```



```
2. 54.211.150.32 (ubuntu)
root@mongodb-59797b688c-xpg8z:/# ls
bin  boot  data  dev  docker-entrypoint-initdb.d  etc  home  js-yaml.js  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@mongodb-59797b688c-xpg8z:/# mongo
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("774f23e0-ad0b-4046-8b16-cb2658892067") }
MongoDB server version: 4.4.6
---
The server generated these startup warnings when booting:
        2024-04-11T19:23:39.541+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
        2024-04-11T19:23:39.541+00:00: You are running this process as the root user, which is not recommended
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
todo    0.000GB
> use todo
switched to db todo
> db.tasks.find()
{ "_id" : ObjectId("66184f8b20d8c07a84daebd9"), "completed" : false, "task" : "congratulations", "__v" : 0 }
{ "_id" : ObjectId("66184f9320d8c09b77daebdb"), "completed" : false, "task" : "Three-tier-succes", "__v" : 0 }
{ "_id" : ObjectId("66184f9920d8c0d05fdaebdd"), "completed" : false, "task" : "welcome", "__v" : 0 }
{ "_id" : ObjectId("66184f9d20d8c0fd9adaebdf"), "completed" : false, "task" : "hi", "__v" : 0 }
>
```

- Here our input data has been successfully saved in the MongoDB database.