Task 2: Create a K8S cluster (mini cluster like minikube or k3s) on your laptop (no need to setup a separate server for this). Accomplish the following:

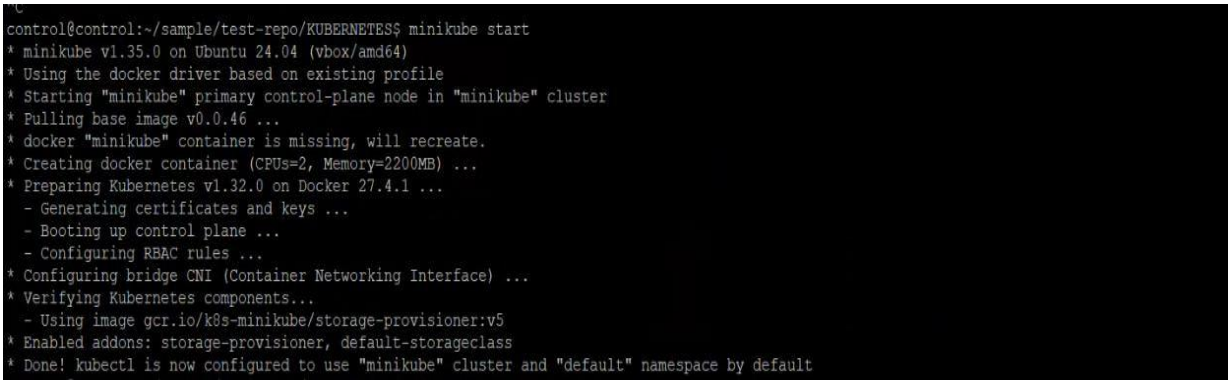
● Create two separate users, two separate name spaces, each user having access to their respective namespaces.

● Let each user create a deployment, where each deployment having two nginx containers (listening on 80 and 8080) in each pod, in their namespaces. Let each user also create an alpine pod in their namespaces.

● From their alpine containers, each user should be able to: ○ Connect (via curl) to their own deployment service via port 80 and port 8080 ○ Connect to other user's deployment service (in the other namespace) via port 80 ○ Restrict access to other user's deployment service via port 8080 (should be stopped via some sort of firewall)

installation for the minikube :

And run the below commands:
minikube start




● Create two separate users, two separate name spaces, each user having access to their respective namespaces.

To create two users in a Minikube cluster, each having access only to their respective namespaces using Role-Based Access Control

**Namespace Creation:**

- Created two separate namespaces using

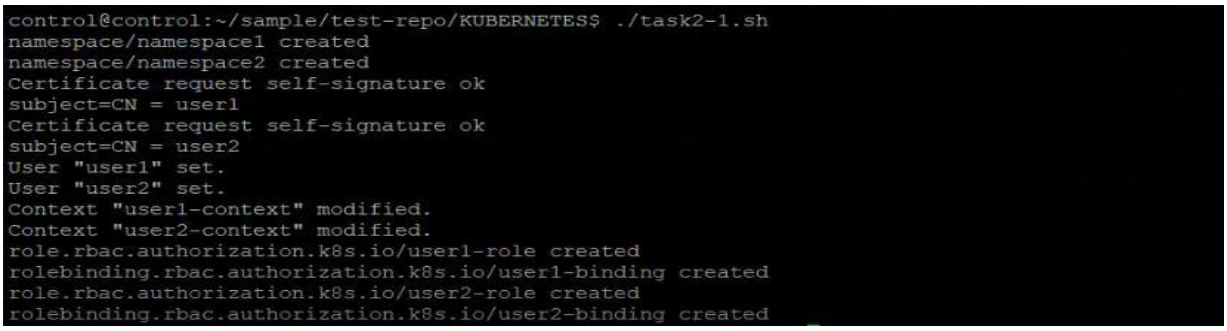
commands:

kubectl create namespace namespace1

kubectl create namespace namespace2

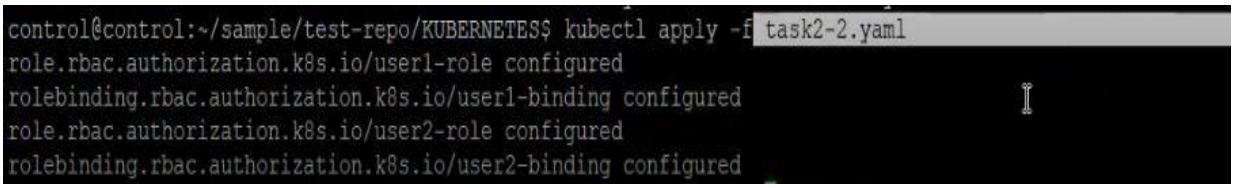
To open the path for the folder.
./task2-1.sh

Attached the screenshot of the terminal showing successful execution of script and context configuration.

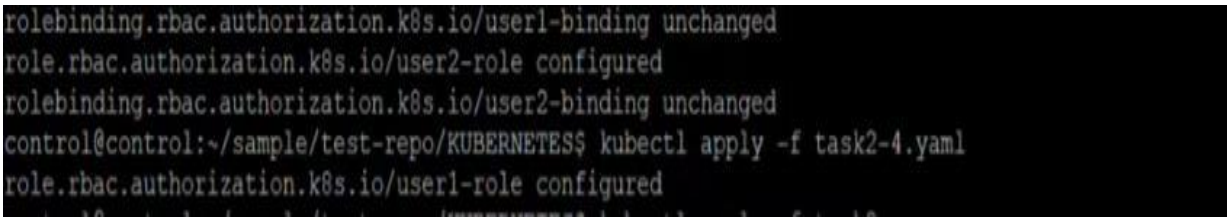



Controls for two users (user1 and user2) within their respective namespaces (namespace1 and namespace2) using Kubernetes Role-Based Access Control following step are done in the task2-2.yaml file.

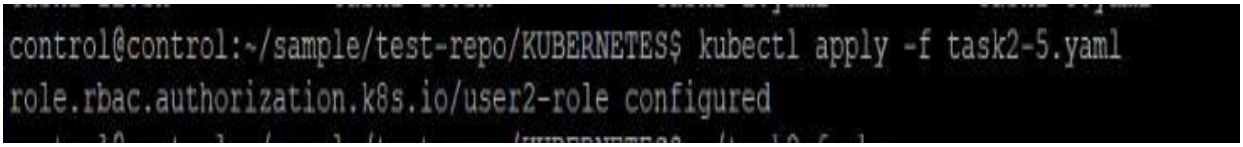
Commands:
kubectl apply -f  task2-2.yaml




To create a Kubernetes Role that limits user1 to read-only operations (list, get, watch) on **pods** and **events** in the namespace1 namespace.
Commands:
kubectl apply -f  task2-4.yaml



To implement a Kubernetes Role that grants user2 limited **read-only access** to essential resources (pods, events) in the namespace2 namespace.
Commands:
kubectl apply -f  task2-5.yaml




● Let each user create a deployment, where each deployment having two nginx containers (listening on 80 and 8080) in each pod, in their namespaces. Let each user also create an alpine pod in their namespaces.

To demonstrate running two instances of the NGINX web server using Docker, each listening on different host ports (80 and 8080) to avoid port conflicts.
Commands:
kubectl apply -f  task2-6.sh

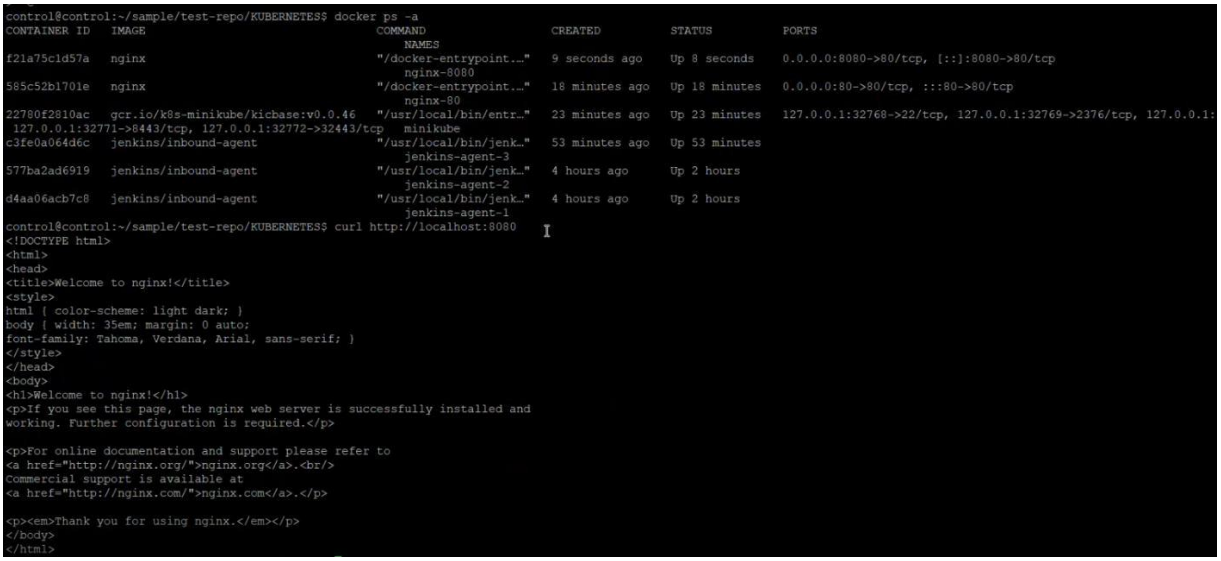
docker run -d --name nginx-80 -p 80:80 nginx
docker run -d --name nginx-8080 -p 8080:80 nginx

**Verification:**

You can test both servers in your browser or using curl:

curl http://localhost:80

curl http://localhost:8080



Commands:

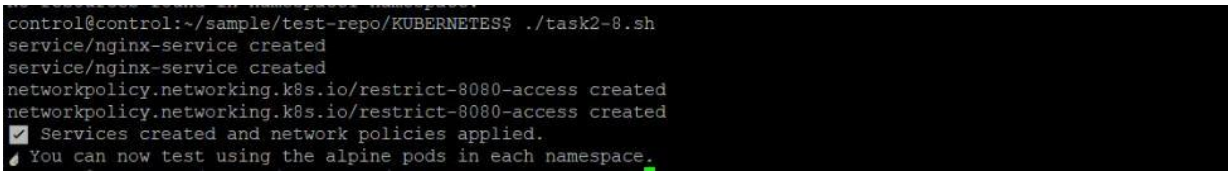
docker exec -it nginx-8080 bash

echo "Hello, NGINX!" > /usr/share/nginx/html/index.html

docker restart nginx-8080

Executed commands inside the nginx-8080 container to customize the default webpage served by NGINX.Modified the index.html to display a custom message: "Hello, NGINX!".Restarted the container to ensure the updated content is loaded properly.Helps verify that port 8080 is correctly exposed and that the NetworkPolicy restrictions are in place by testing access via web browser or curl.
Expose NGINX containers running on ports 80 and 8080 via Services and apply NetworkPolicies to restrict cross-namespace access specifically on port 8080.Now services created the network policies and now test using the alpine pods in each namespace.

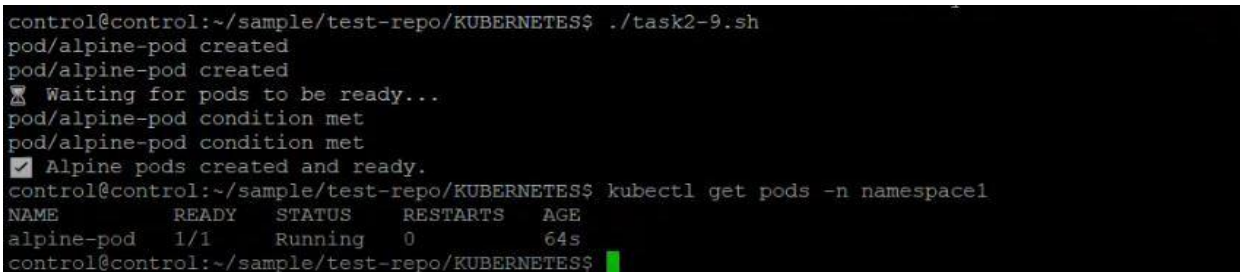
kubectl apply -f  task2-8.sh




● From their alpine containers, each user should be able to: ○ Connect (via curl) to their own deployment service via port 80 and port 8080 ○ Connect to other user's deployment service (in the other namespace) via port 80 ○ Restrict access to other user's deployment service via port 8080 (should be stopped via some sort of firewall)
Deployed test pods (alpine-pod) in both namespace1 and namespace2 for validating network access and service discovery.Each pod runs a simple sleep command to keep it alive for interaction.Used kubectl wait to ensure pods are fully ready before testing begins.

Command:

kubectl apply -f  task2-9.sh
kubectl get pods -n namespace1

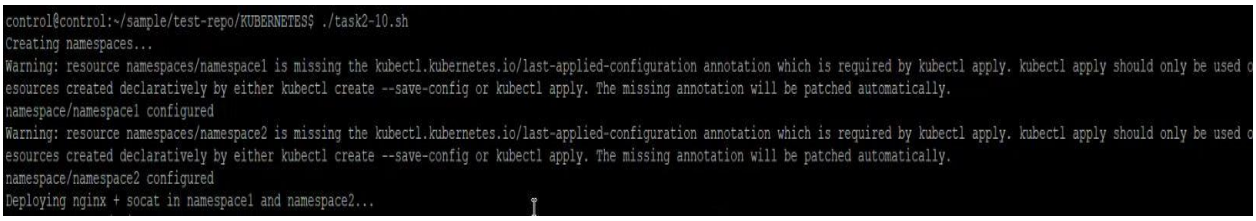



Now the Alpine pods created and ready .

This script automates the deployment of a dual-port NGINX setup (ports 80 and 8080 using socat proxy) across two namespaces, and implements network policies to restrict access to port 8080 across namespaces.

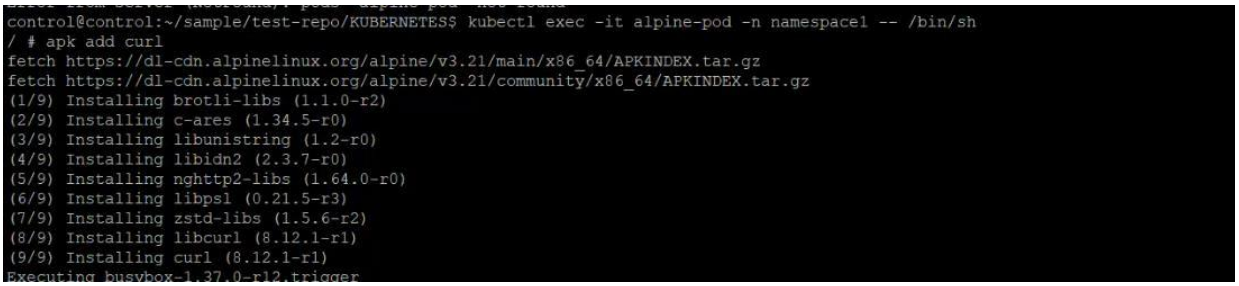
Command:

kubectl apply -f  task2-10.sh




This script checks the existence of a specific NGINX service and deployment in multiple namespaces (namespace1, namespace2). If they are not found, it automatically creates them. This ensures consistency and availability across environments.
Command:

kubectl apply -f  task2-11.sh




This script validates **network connectivity and access control** between two Kubernetes namespaces namespace1. It ensures services are accessible as expected and that network policies are correctly applied—especially to **block cross-namespace access to port 8080**.

Command:

kubectl apply -f  task2-13.sh

# TASK2 OUTPUT:

The validation for the connectivity and access control for the port number 80 and 8080 .when I use the port 80 is success and port 8080 is success .when I gave restrict access to other user's deployment service via port 8080 (should be stopped via some sort of firewall).