

**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

```
control@control:~/sample/test-repo/KUBERNETES$ minikube start
* minikube v1.35.0 on Ubuntu 24.04 (vbox/amd64)
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.46 ...
* docker "minikube" container is missing, will recreate.
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

- 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.

```
control@control:~/sample/test-repo/KUBERNETES$ ./task2-1.sh
namespace/namespace1 created
namespace/namespace2 created
Certificate request self-signature ok
subject=CN = user1
Certificate request self-signature ok
subject=CN = user2
User "user1" set.
User "user2" set.
Context "user1-context" modified.
Context "user2-context" modified.
role.rbac.authorization.k8s.io/user1-role created
rolebinding.rbac.authorization.k8s.io/user1-binding created
role.rbac.authorization.k8s.io/user2-role created
rolebinding.rbac.authorization.k8s.io/user2-binding created
```

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`

```
control@control:~/sample/test-repo/KUBERNETES$ kubectl apply -f task2-2.yaml
role.rbac.authorization.k8s.io/user1-role configured
rolebinding.rbac.authorization.k8s.io/user1-binding configured
role.rbac.authorization.k8s.io/user2-role configured
rolebinding.rbac.authorization.k8s.io/user2-binding configured
```

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`

```
rolebinding.rbac.authorization.k8s.io/user1-binding unchanged
role.rbac.authorization.k8s.io/user2-role configured
rolebinding.rbac.authorization.k8s.io/user2-binding unchanged
control@control:~/sample/test-repo/KUBERNETES$ kubectl apply -f task2-4.yaml
role.rbac.authorization.k8s.io/user1-role configured
```

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
```

```
control@control:~/sample/test-repo/KUBERNETES$ kubectl apply -f task2-5.yaml
role.rbac.authorization.k8s.io/user2-role configured
```

- 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
```

```
control@control:~/sample/test-repo/KUBERNETES$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
f21a75c1d57a   nginx         "/docker-entrypoint..." 9 seconds ago  Up 8 seconds  0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
385c52b1701e   nginx         "/docker-entrypoint..." 18 minutes ago Up 18 minutes  0.0.0.0:80->80/tcp, ::80->80/tcp
22780f2810ac   gcr.io/k8s-minikube/kicbase:v0.0.46 "/usr/local/bin/entr..." 23 minutes ago Up 23 minutes  127.0.0.1:32771->8443/tcp, 127.0.0.1:32772->32443/tcp
c3fe0a064d6c   jenkins/inbound-agent "/usr/local/bin/jenk..." 53 minutes ago Up 53 minutes
577ba2ad6919   jenkins/inbound-agent "/usr/local/bin/jenk..." 4 hours ago    Up 2 hours
d4aa06acb7c8   jenkins/inbound-agent "/usr/local/bin/jenk..." 4 hours ago    Up 2 hours
control@control:~/sample/test-repo/KUBERNETES$ curl http://localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

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
```

```
control@control:~/sample/test-repo/KUBERNETES$ ./task2-8.sh
service/nginx-service created
service/nginx-service created
networkpolicy.networking.k8s.io/restrict-8080-access created
networkpolicy.networking.k8s.io/restrict-8080-access created
[✓] Services created and network policies applied.
💡 You can now test using the alpine pods in each namespace.
```

- 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
```

```
control@control:~/sample/test-repo/KUBERNETES$ ./task2-9.sh
pod/alpine-pod created
pod/alpine-pod created
⌛ Waiting for pods to be ready...
pod/alpine-pod condition met
pod/alpine-pod condition met
[✓] Alpine pods created and ready.
control@control:~/sample/test-repo/KUBERNETES$ kubectl get pods -n namespace1
NAME          READY   STATUS    RESTARTS   AGE
alpine-pod    1/1     Running   0           64s
control@control:~/sample/test-repo/KUBERNETES$
```

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

```
control@control:~/sample/test-repo/KUBERNETES$ ./task2-10.sh
Creating namespaces...
Warning: resource namespaces/namespace1 is missing the kubect1.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used o
resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
namespace/namespace1 configured
Warning: resource namespaces/namespace2 is missing the kubect1.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used o
resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
namespace/namespace2 configured
Deploying nginx + socat in namespace1 and namespace2...
```

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

```
control@control:~/sample/test-repo/KUBERNETES$ kubectl exec -it alpine-pod -n namespace1 -- /bin/sh
/ # apk add curl
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/community/x86_64/APKINDEX.tar.gz
(1/9) Installing brotli-libs (1.1.0-r2)
(2/9) Installing c-ares (1.34.5-r0)
(3/9) Installing libunistring (1.2-r0)
(4/9) Installing libidn2 (2.3.7-r0)
(5/9) Installing nghttp2-libs (1.64.0-r0)
(6/9) Installing libpsl (0.21.5-r3)
(7/9) Installing zstd-libs (1.5.6-r2)
(8/9) Installing libcurl (8.12.1-r1)
(9/9) Installing curl (8.12.1-r1)
Executing busybox-1.37.0-r12.trigger
```

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

```
control@control:~/sample/test-repo/KUBERNETES$ ./task2-13.sh
===== NAMESPACE CONNECTIVITY TEST =====

🔍 Checking from alpine-pod in namespace1
-----
Own service (port 80): ☒ SUCCESS
Own service (port 8080): ☒ SUCCESS
Other user's service (port 80): ☒ SUCCESS
Other user's service (port 8080 - should be blocked): ☒ UNEXPECTED SUCCESS (Check NetworkPolicy!)
```

## 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).

```
control@control:~/sample/test-repo/KUBERNETES$ ./task2-13.sh
===== NAMESPACE CONNECTIVITY TEST =====

🔍 Checking from alpine-pod in namespace1
-----
Own service (port 80): ☒ SUCCESS
Own service (port 8080): ☒ SUCCESS
Other user's service (port 80): ☒ SUCCESS
Other user's service (port 8080 - should be blocked): ☒ UNEXPECTED SUCCESS (Check NetworkPolicy!)
```