# Kubernetes Lab Manual — Expose nginx (ClusterIP & NodePort), Ingress /app, Namespace-restricted NetworkPolicy

**Goal:** you will: - Deploy an nginx app in a dedicated namespace. - Expose it with a **ClusterIP** service and a **NodePort** service. - Deploy an **Ingress** that routes http://lab.example.com/app → the nginx ClusterIP service. - Apply a **NetworkPolicy** that allows ingress **only** from a specific namespace (label-based).

## 1) Prerequisites & dependencies

* **Kubernetes cluster** (minikube, kind, Docker Desktop, k3s, EKS, GKE, AKS — any will work).
* **kubectl** installed and configured to communicate with your cluster.
* **Ingress controller** installed (commonly ingress-nginx). Some clusters (managed cloud) already provide an ingress controller or a cloud load balancer.
* **CNI that supports NetworkPolicy** (e.g., Calico, Cilium). Without a policy-capable CNI, NetworkPolicy resources are ignored.
* Local tools: curl, kubectl, optional jq.

**Verify**

kubectl version --short  
kubectl get nodes  
# Check for an ingress controller (common namespace: ingress-nginx)  
kubectl get pods -n ingress-nginx --no-headers || true

If you don’t have an ingress controller or a policy-capable CNI, see the Notes below (quick options provided).

## 2) Create namespaces

We’ll create three namespaces to demonstrate namespace-restricted access: - lab-ingress — where nginx will run - allowed-ns — the namespace we will allow - blocked-ns — a namespace used to test denied access

kubectl create namespace lab-ingress  
kubectl create namespace allowed-ns  
kubectl create namespace blocked-ns

## 3) Deploy nginx + Services (ClusterIP & NodePort)

Save the following to **nginx-deploy-and-svcs.yaml** (single file containing deployment + both services):

# nginx-deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: nginx-deploy  
 namespace: lab-ingress  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: nginx  
 template:  
 metadata:  
 labels:  
 app: nginx  
 spec:  
 containers:  
 - name: nginx  
 image: nginx:1.25-alpine  
 ports:  
 - containerPort: 80  
  
---  
# ClusterIP service (internal cluster access)  
apiVersion: v1  
kind: Service  
metadata:  
 name: nginx-clusterip  
 namespace: lab-ingress  
spec:  
 type: ClusterIP  
 selector:  
 app: nginx  
 ports:  
 - port: 80  
 targetPort: 80  
  
---  
# NodePort service (external node port)  
apiVersion: v1  
kind: Service  
metadata:  
 name: nginx-nodeport  
 namespace: lab-ingress  
spec:  
 type: NodePort  
 selector:  
 app: nginx  
 ports:  
 - port: 80  
 targetPort: 80  
 nodePort: 30080

Apply it:

kubectl apply -f nginx-deploy-and-svcs.yaml  
kubectl get deploy -n lab-ingress  
kubectl get svc -n lab-ingress  
kubectl get pods -n lab-ingress

**Test NodePort from your workstation** 1. Get a node IP:

NODE\_IP=$(kubectl get nodes -o jsonpath='{.items[0].status.addresses[?(@.type=="InternalIP")].address}')  
echo $NODE\_IP

1. Curl the NodePort:

curl -sS http://$NODE\_IP:30080 | head -n 5  
# you should get the nginx default page HTML

*If running minikube:* NODE\_IP=$(minikube ip) often works. For Docker Desktop / k3d / kind, NodePort may be reachable on localhost or require port-forwarding / special handling — see notes.

## 4) Deploy Ingress for /app

Create an Ingress resource that routes lab.example.com/app to the **ClusterIP** service nginx-clusterip.

Save as **nginx-ingress.yaml**:

apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
 name: nginx-ingress  
 namespace: lab-ingress  
 annotations:  
 kubernetes.io/ingress.class: "nginx" # ensure this matches your ingress controller class  
spec:  
 rules:  
 - host: lab.example.com  
 http:  
 paths:  
 - path: /app  
 pathType: Prefix  
 backend:  
 service:  
 name: nginx-clusterip  
 port:  
 number: 80

Apply it:

kubectl apply -f nginx-ingress.yaml  
kubectl get ingress -n lab-ingress

**Find the Ingress controller IP** - If your ingress controller exposes a Service of type LoadBalancer, get its external IP:

kubectl get svc -n ingress-nginx  
# look for SERVICE with EXTERNAL-IP

* If using **minikube**, you can do minikube ip and point lab.example.com to that IP in /etc/hosts.
* Alternatively, port-forward the ingress controller service to localhost (quick dev trick):

kubectl -n ingress-nginx port-forward svc/ingress-nginx-controller 8080:80 &  
# then test with Host header  
curl -v -H "Host: lab.example.com" http://127.0.0.1:8080/app

**Test Ingress** (example using /etc/hosts local mapping): 1. Find ingress IP (or use minikube ip or port-forward as above). 2. Add to /etc/hosts:

# as root/with sudo — map the ingress IP to host  
# e.g., echo "<INGRESS\_IP> lab.example.com" | sudo tee -a /etc/hosts

1. Curl path:

curl -v http://lab.example.com/app | head -n 5

You should see the nginx default HTML.

## 5) Apply NetworkPolicy: allow traffic only from allowed-ns

**Important:** NetworkPolicy matching by namespace uses labels on the namespace. We’ll label allowed-ns with access=granted and then create a NetworkPolicy in lab-ingress that permits ingress **only** from namespaces with that label.

Label the namespace:

kubectl label namespace allowed-ns access=granted

Create **nginx-namespace-policy.yaml**:

apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
 name: allow-from-allowed-ns  
 namespace: lab-ingress  
spec:  
 podSelector:  
 matchLabels:  
 app: nginx  
 policyTypes:  
 - Ingress  
 ingress:  
 - from:  
 - namespaceSelector:  
 matchLabels:  
 access: granted  
 # you can optionally restrict to certain pods in the allowed namespace using podSelector

Apply it:

kubectl apply -f nginx-namespace-policy.yaml  
kubectl get networkpolicy -n lab-ingress

**What this does:** - Targets pods labeled app=nginx in lab-ingress. - Allows incoming connections *only* from pods running in namespaces that have the label access=granted (i.e., allowed-ns). - All other ingress is denied (because the policy exists and restricts ingress).

**Important note:** this will block **external** NodePort/Ingress traffic too (because those external sources are not in the allowed namespace). This matches the requested requirement “allow traffic only from a specific namespace”. If you **want** to allow external user traffic (e.g., through Ingress) while still restricting cluster-initiated traffic, you’d need more complex policies (for example allow ipBlock ranges or allow traffic from the ingress controller’s namespace/pods). See Troubleshooting notes.

## 6) Test NetworkPolicy behaviour

### 6a) From allowed-ns (should succeed)

Run a temporary pod and curl the ClusterIP service DNS name from inside the cluster:

kubectl run -n allowed-ns --rm -it tester --image=busybox --restart=Never -- /bin/sh -c "sleep 1"  
# in the pod shell (or run with --command):  
# curl the clusterIP service DNS  
nslookup nginx-clusterip.lab-ingress.svc.cluster.local || true  
wget -qO- http://nginx-clusterip.lab-ingress.svc.cluster.local  
# exit the pod shell

You should receive the nginx default page output.

### 6b) From blocked-ns (should be denied)

kubectl run -n blocked-ns --rm -it tester2 --image=busybox --restart=Never -- /bin/sh -c "wget -qO- http://nginx-clusterip.lab-ingress.svc.cluster.local || echo 'DENIED'"

You should see a connection error or DENIED — traffic is denied by the NetworkPolicy.

### 6c) NodePort / Ingress access from outside cluster

Attempt curl http://$NODE\_IP:30080 from your workstation. This will now be **blocked** because the pod only accepts traffic from allowed-ns.

If you *intend* to allow external requests via the Ingress controller while still restricting other cluster namespaces, add an additional from entry in the NetworkPolicy that allows traffic from the ingress controller namespace (label the ingress namespace and reference it). Example snippet to allow ingress-controller namespace ingress-nginx:

- namespaceSelector:  
 matchLabels:  
 app.kubernetes.io/name: ingress-nginx

(Adjust the label to match your controller namespace labels.)

## 7) Quick validation checklist

* lab-ingress namespace created and has nginx pods
* nginx-clusterip (ClusterIP) and nginx-nodeport (NodePort: 30080) exist and return the nginx page when allowed
* Ingress resource for lab.example.com/app exists and routes to nginx-clusterip
* allowed-ns labeled access=granted
* NetworkPolicy allow-from-allowed-ns is present in lab-ingress and blocks traffic from blocked-ns and external hosts (unless explicitly allowed)

## 8) Cleanup

kubectl delete -f nginx-ingress.yaml  
kubectl delete -f nginx-deploy-and-svcs.yaml  
kubectl delete -f nginx-namespace-policy.yaml  
kubectl delete namespace lab-ingress allowed-ns blocked-ns  
# remove host file entry from /etc/hosts if you added it

## 9) Troubleshooting & Notes

* **NetworkPolicy ignored / has no effect**: ensure your cluster CNI supports NetworkPolicy (Calico, Cilium, Canal, etc.). Managed cloud clusters usually enable a policy-capable CNI by default.
* **Ingress not routing**: confirm ingress controller is installed and the ingress annotation/class matches the controller. Check controller logs: kubectl logs -n ingress-nginx deploy/ingress-nginx-controller.
* **NodePort not reachable**: depending on cluster (kind/k3d/docker-desktop), NodePort may not be accessible from your host network. Use minikube ip, minikube tunnel, or port-forwarding for local dev clusters.
* **Allowing external users via Ingress but restricting internal namespaces**: add from entries in the NetworkPolicy to allow the ingress controller’s namespace (or the ingress controller pod selectors), or allow specific ipBlocks for trusted external IPs.

## 10) Instructor tips

* Emphasize how NetworkPolicy is *deny-by-intent*: once a policy selects a pod and defines Ingress, only sources matching its from rules are allowed.
* Show how to inspect packets blocked by policy in CNIs that provide metrics (e.g., Calico Flow Logs).
* For a quick demo cluster: minikube + minikube addons enable ingress + kubectl apply -f calico.yaml (if you need NetworkPolicy support) works well.

**End of lab**