

NAME: DEEPA T

BATCH: B11

DATE: 29.10.2025

TASK: K8s

1) Question: Create a deployment with the latest nginx image and two replicas.

- Expose its port 80 through a service of type NodePort.
- Show all elements, including the endpoints.
- Get the nginx index page through the NodePort.

Ans: We can create deployment in two ways:

1. CLI

```
Kubectl create deploy deploynginx - -image=nginx:latest - -replicas=2
```

2. Template file

Create a Deployment with the latest NGINX image and two replicas using YAML file:

```
deepa@ubuntu:~/devops/K8s/2910/Q1$ kubectl apply -f nginxdeployment.yaml
deployment.apps/nginxdeployment created
deepa@ubuntu:~/devops/K8s/2910/Q1$ cat nginxdeployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginxdeployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

Service YAML:

```
deepa@ubuntu:~/devops/K8s/2910/Q1$ kubectl apply -f nginxservice.yaml
service/nginxservice created
deepa@ubuntu:~/devops/K8s/2910/Q1$ cat nginxservice.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginxservice
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 31333

deepa@ubuntu:~/devops/K8s/2910/Q1$
```

Verify the Deployment and Pods:

```
deepa@ubuntu:~/devops/K8s/2910/Q1$ kubectl get pods -o wide
NAME                               READY   STATUS             RESTARTS
AGE      IP           NODE          NOMINATED-NODE  READINESS GATES
demo-keyvault-pod                 0/1    ContainerCreating 0
5d20h   <none>       minikube     <none>        <none>
nginxdeployment-6f9664446b-f9bnd  1/1    Running           0
8m24s   10.244.0.11  minikube     <none>        <none>
nginxdeployment-6f9664446b-x6jjx  1/1    Running           0
8m24s   10.244.0.10  minikube     <none>        <none>
deepa@ubuntu:~/devops/K8s/2910/Q1$
```

Check the Endpoints of the Service:

```
deepa@ubuntu:~/devops/K8s/2910/Q1$ kubectl get endpoints
Warning: v1 Endpoints is deprecated in v1.33+; use discovery.k8s.io/v1 EndpointSlice
NAME          ENDPOINTS          AGE
kubernetes    192.168.49.2:8443  5d21h
nginxservice  10.244.0.10:80,10.244.0.11:80  8m27s
deepa@ubuntu:~/devops/K8s/2910/Q1$
```

View All Kubernetes Resources: Get the NodePort Number

```
deepa@ubuntu:~/devops/K8s/2910/Q1$ kubectl get all
NAME                               READY   STATUS             RESTARTS   AGE
pod/demo-keyvault-pod            0/1    ContainerCreating 0          5d20h
pod/nginxdeployment-6f9664446b-f9bnd  1/1    Running           0          10m
pod/nginxdeployment-6f9664446b-x6jjx  1/1    Running           0          10m
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/kubernetes  ClusterIP  10.96.0.1      <none>        443/TCP     5d21h
service/nginxservice  NodePort   10.106.0.226  <none>        80:31333/TCP  4m23s
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginxdeployment  2/2      2           2           10m
NAME          DESIRED  CURRENT    READY   AGE
replicaset.apps/nginxdeployment-6f9664446b  2        2        2        10m
deepa@ubuntu:~/devops/K8s/2910/Q1$
```

Access the NGINX index page

```
deepa@ubuntu:~/devops/K8s/2910/Q1$ minikube service nginxservice --url
http://192.168.49.2:31333
deepa@ubuntu:~/devops/K8s/2910/Q1$
```

```
deepa@ubuntu:~/devops/K8s/2910/Q1$ curl http://192.168.49.2:31333
<!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>
deepa@ubuntu:~/devops/K8s/2910/Q1$
```

Node IP, open this in browser: default NGINX welcome page (HTML)



2) Question:

- Create a pod and mount a volume with hostPath directory.
- Check that the contents of the directory are accessible through the pod.

Ans:

Before the pod is created, hostpath directory must be created or exist.

Minikube node runs Kubernetes **inside a virtual machine** (e.g. using Docker, KVM, or Hyper-V). HostPath volume, the path, refers to a directory on the **Kubernetes node's filesystem, not the local machine's filesystem**. **Kubernetes pod runs inside Minikube's VM, not directly on Ubuntu host.**

Minikube ssh, we can access the Minikube VM — the **actual Kubernetes node**.

```
deepa@ubuntu:~/devops/K8s$ cd 2910
deepa@ubuntu:~/devops/K8s/2910$ ls
Q1
deepa@ubuntu:~/devops/K8s/2910$ mkdir Q2
deepa@ubuntu:~/devops/K8s/2910$ cd Q2
deepa@ubuntu:~/devops/K8s/2910/Q2$ ls
deepa@ubuntu:~/devops/K8s/2910/Q2$ minikube ssh
docker@minikube:~$ sudo mkdir -p /MinikubeVM/hostpath
docker@minikube:~$ sudo echo "Have a nice day" | sudo tee /MinikubeVM/hostpath/fileABC.txt
Have a nice day
docker@minikube:~$ ls
docker@minikube:~$ ls
docker@minikube:~$ pwd
/home/docker
docker@minikube:~$ ls /MinikubeVM/hostpath
fileABC.txt
docker@minikube:~$ cat /MinikubeVM/hostpath
cat: /MinikubeVM/hostpath: Is a directory
docker@minikube:~$ cat /MinikubeVM/hostpath/fileABC.t
xt
Have a nice day
docker@minikube:~$ exit
logout
```

Creating a Pod using YAML file:

```
deepa@ubuntu:~/devops/K8s/2910/Q2$ cat pod1.yaml
apiVersion: v1
kind: pod
metadata:
  name: pod1
spec:
  containers:
  - name: container1
    image: busybox
    command: ["/bin/sh", "-c", "sleep 3600"]
    volumeMounts:
    - name: host-volume
      mountPath: /mnt/hostdata
  volumes:
  - name: host-volume
    hostPath:
      path: /MinikubeVM/hostpath
      type: Directory
```

```
deepa@ubuntu:~/devops/K8s/2910/Q2$ █
```

Pod name: pod1

Container name: container1

image : uses the lightweight busybox image (handy for testing).

```
command: ["/bin/sh", "-c", "sleep 3600"]
```

This tells the container to run sh -c "sleep 3600", which makes the container do nothing but stay alive for 1 hour (3600 seconds).

It's just a placeholder so the container doesn't exit immediately.

BusyBox and many other small images will exit right away unless you tell them to "stay running."

Storage creatd : host-volume, source

/MinikubeVM/hostpath : host path

/mnt/hostdata : mounting path in the pod's container

So anything written or saved in the hostpath will appear in the container path , and vice versa.

```
deepa@ubuntu:~/devops/K8s/2910/Q2$ vi pod1.yaml
deepa@ubuntu:~/devops/K8s/2910/Q2$ kubectl apply -f pod1.yaml
pod/pod1 created
deepa@ubuntu:~/devops/K8s/2910/Q2$ █
```

Now, access the pod1:

```
deepa@ubuntu:~/devops/K8s/2910/Q2$ kubectl exec -it pod1 -- sh
/ # ls
bin dev etc home lib lib64 mnt proc root sys tmp usr var
/ # ls /mnt/hostdata
fileABC.txt
/ # cat /mnt/hostdata
cat: read error: Is a directory
/ # cat /mnt/hostdata/fileABC.txt
Have_a nice day
```

Contents of the directory on host path are accessible, copied to the pod.

3) Question: Create a persistent volume from hostPath and a persistent volume claim corresponding to that PV. Create a pod that uses the PVC and check that the volume is mounted in the pod.

- Create a file from the pod in the volume then delete it and create a new pod with the same volume and show the created file by the first pod.

Ans: Create podpv.yaml, podpvc.yaml, and pod1.yaml

```
deepa@ubuntu:~/devops/K8s/2910/Q3$ ls
pod1.yaml podpvc.yaml podpv.yaml
deepa@ubuntu:~/devops/K8s/2910/Q3$ cat podpv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-demo
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: /host/volume
deepa@ubuntu:~/devops/K8s/2910/Q3$ kubectl apply -f podpv.yaml
persistentvolume/pv-demo unchanged
deepa@ubuntu:~/devops/K8s/2910/Q3$ kubectl apply -f podpvc.yaml
persistentvolumeclaim/pvc-demo unchanged
deepa@ubuntu:~/devops/K8s/2910/Q3$ kubectl apply -f pod1.yaml
pod/pvc-pod1 configured
deepa@ubuntu:~/devops/K8s/2910/Q3$ cat podpvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-demo
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: manual
  resources:
    requests:
      storage: 500Mi
```

```
deepa@ubuntu:~/devops/K8s/2910/Q3$ kubectl apply -f pod1.yaml
pod/pvc-pod1 created
deepa@ubuntu:~/devops/K8s/2910/Q3$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginxdeployment-6f9664446b-f9bnd   1/1     Running   2 (41h ago)   3d2h
nginxdeployment-6f9664446b-x6jjx   1/1     Running   2 (41h ago)   3d2h
pvc-pod1      1/1     Running   0          11s
```

```
deepa@ubuntu:~/devops/K8s/2910/Q3$ cat pod1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pvc-pod1
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["/bin/sh", "-c", "sleep infinity"]
    volumeMounts:
    - name: mypvc
      mountPath: /mnt/pvdata
  volumes:
  - name: mypvc
    persistentVolumeClaim:
      claimName: pvc-demo  # ← FIXED HERE
```

Verify:

```
deepa@ubuntu:~/devops/K8s/2910/Q3$ kubectl describe pod pvc-pod1
Name:           pvc-pod1
Namespace:      default
Priority:       0
Service Account: default
Node:           minikube/192.168.49.2
Start Time:     Sat, 01 Nov 2025 05:30:51 -0700
Labels:          <none>
Annotations:    <none>
Status:         Running
IP:             10.244.0.22
IPs:
  IP: 10.244.0.22
Containers:
  busybox:
    Container ID: docker://f148a48e428f6e9e9218a3a12ae38b3fea98cdb3db70ad6da030bdc13369d48b
    Image:          busybox
    Image ID:      docker-pullable://busybox@sha256:e3652a00a2fabd16ce889f0aa32c38eec347b997e73bd09e69c962ec7f8732ee
    Port:          <none>
    Host Port:    <none>
    Command:
      /bin/sh
      -c
      sleep infinity
    State:        Running
    Started:     Sat, 01 Nov 2025 05:30:56 -0700
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
      /mnt/pvdata from mypvc (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-6kzjt (ro)
Conditions:
  Type        Status
  PodReadyToStartContainers  True
  Initialized  True
  Ready        True
  ContainersReady  True
  PodScheduled  True
Volumes:
  mypvc:
    Type:     PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: pvc-demo
    ReadOnly:  false
```

```

kube-api-access-6kzjt:
  Type:          Projected (a volume that contains injected data from multiple sources)
  TokenExpirationSeconds: 3607
  ConfigMapName:   kube-root-ca.crt
  Optional:       false
  DownwardAPI:    true
  QoS Class:     BestEffort
  Node-Selectors: <none>
  Tolerations:   node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                  node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

Events:
  Type  Reason  Age   From           Message
  ----  -----  --   --            --
  Normal Scheduled 11s  default-scheduler  Successfully assigned default/pvc-pod1 to minikube
  Normal Pulling  10s  kubelet        Pulling image "busybox"
  Normal Pulled   6s   kubelet        Successfully pulled image "busybox" in 3.715s (3.715s including waiting). Image size: 4429382 bytes.
  Normal Created  6s   kubelet        Created container: busybox
  Normal Started  5s   kubelet        Started container busybox
deepa@ubuntu:~/devops/K8s/2910/03$ 

```

Volume /mnt/pvdata mounted inside the pod.

Test the persistent storage:

Create a file inside the pod

```

deepa@ubuntu:~/devops/K8s/2910/03$ kubectl exec -it pvc-pod1 -- sh
/ # echo "Hello from pvc-pod1" > /mnt/pvdata/testfile.txt
/ # echo "Hello from pvc-pod1" > /mnt/pvdata/testfile.txt
/ # ls -l /mnt/pvdata
total 4
-rw-r--r--  1 root      root           20 Nov  1 12:37 testfile.txt
/ # cat /mnt/pvdata/testfile.txt
Hello from pvc-pod1
/ # 

```

Confirms the **PersistentVolume (PV)** is correctly mounted and writable inside the pod.

Now, to **demonstrate the persistence** (that the data survives when the pod is deleted):

Delete the pod1.

Create pod2

```

deepa@ubuntu:~/devops/K8s/2910/03$ cat pod2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pvc-pod2
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["/bin/sh", "-c", "sleep infinity"]
    volumeMounts:
    - name: mypvc
      mountPath: /mnt/pvdata
  volumes:
  - name: mypvc
    persistentVolumeClaim:
      claimName: pvc-demo  # Same PVC as the previous pod
deepa@ubuntu:~/devops/K8s/2910/03$ kubectl apply -f pod2.yaml
pod/pvc-pod2 created
deepa@ubuntu:~/devops/K8s/2910/03$ 

```

Verify the file from the new pod2:

```
deepa@ubuntu:~/devops/K8s/2910/Q3$ kubectl exec -it pvc-pod2 -- sh
/ # ls /mnt/pvdata
testfile.txt
/ # at /mnt/pvdata/testfile.txt
sh: at: not found
/ # cat /mnt/pvdata/testfile.txt
Hello from pvc-pod1
/ #
```

This proves your **PersistentVolumeClaim** retains data even when the original pod is deleted.

4) Question:

Create a new deployment called **nginx-deploy**, with image **nginx:1.16** and 1 replica. Record the version. Next upgrade the deployment to version 1.17 using rolling update. Make sure that the version upgrade is recorded in the resource annotation.

Create a file called **nginx-deploy.yaml** :

```
deepa@ubuntu:~/devops/kind/311025/01$ kubectl get nodes
NAME           STATUS   ROLES      AGE    VERSION
dev-cluster-control-plane   Ready    control-plane   3h53m   v1.34.0
dev-cluster-worker     Ready    <none>       3h53m   v1.34.0
dev-cluster-worker2    Ready    <none>       3h52m   v1.34.0
deepa@ubuntu:~/devops/kind/311025/01$ kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deploy created
deepa@ubuntu:~/devops/kind/311025/01$ cat nginx-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  labels:
    app: nginx
  annotations:
    kubernetes.io/change-cause: "Initial deployment with nginx:1.16"
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.16
          ports:
            - containerPort: 80
deepa@ubuntu:~/devops/kind/311025/01$
```

Verify deployment:

```
deepa@ubuntu:~/devops/kind/311025/01$ kubectl get deploy nginx-deploy
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deploy   1/1     1           1           3m13s
```

Annotation, version:

```
deepa@ubuntu:~/devops/kind/311025/01$ kubectl describe deploy nginx-deploy
Name:           nginx-deploy
Namespace:      default
CreationTimestamp: Fri, 31 Oct 2025 03:34:45 -0700
Labels:          app=nginx
Annotations:    deployment.kubernetes.io/revision: 1
                 kubernetes.io/change-cause: Initial deployment with nginx:1.16
Selector:        app=nginx
Replicas:        1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:          app=nginx
  Containers:
    nginx:
      Image:          nginx:1.16 ←
      Port:          80/TCP
      Host Port:     0/TCP
      Environment:   <none>
      Mounts:         <none>
      Volumes:        <none>
      Node-Selectors: <none>
      Tolerations:   <none>
  Conditions:
    Type        Status  Reason
    ----        ----  -----
    Available   True    MinimumReplicasAvailable
    Progressing True    NewReplicaSetAvailable
  OldReplicaSets: <none>
  NewReplicaSet:  nginx-deploy-bd6fc657 (1/1 replicas created)
Events:
  Type      Reason           Age   From               Message
  ----      ----           --   --                --
  Normal   ScalingReplicaSet 3m44s  deployment-controller  Scaled up replica set nginx-deploy-bd6fc657 from 0 to 1
```

Upgrade the deployment to version 1.17 using rolling update:

```
deepa@ubuntu:~/devops/kind/311025/Q1$ cat nginx-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  labels:
    app: nginx
  annotations:
    kubernetes.io/change-cause: "Initial deployment with nginx:1.17"
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.17
          ports:
            - containerPort: 80
deepa@ubuntu:~/devops/kind/311025/Q1$
```

Apply manifest yaml file:

```
deepa@ubuntu:~/devops/kind/311025/Q1$ kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deploy configured
deepa@ubuntu:~/devops/kind/311025/Q1$
```

Rollout : upgrading to 1.17

```
deepa@ubuntu:~/devops/kind/311025/Q1$ kubectl rollout status deployment/nginx-deploy
Waiting for deployment "nginx-deploy" rollout to finish: 1 old replicas are pending termination ...
Waiting for deployment "nginx-deploy" rollout to finish: 1 old replicas are pending termination ...
deployment "nginx-deploy" successfully rolled out
deepa@ubuntu:~/devops/kind/311025/Q1$
```

Rolling Update Strategy:

```
deepa@ubuntu:~/devops/kind/311025/Q1$ kubectl describe deploy nginx-deploy
Name:           nginx-deploy
Namespace:      default
CreationTimestamp:   Fri, 31 Oct 2025 03:34:45 -0700
Labels:         app=nginx
Annotations:    deployment.kubernetes.io/revision: 2
                kubernetes.io/change-cause: Initial deployment with nginx:1.17
Selector:       app=nginx
Replicas:      1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:  RollingUpdate  ←
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
    nginx:
      image:  nginx:1.17
      Port:   80/TCP
      Host Port:  0/TCP
      Environment: <none>
      Mounts:
      Volumes:
      Node Selectors:
      Tolerations:
  Conditions:
    Type        Status  Reason
    Available   True    MinImageReplicasAvailable
    Progressing True    NewReplicaSetsAvailable
    OldReplicaSets: nginx-deploy-bd6fc657 (0/0 replicas created)
    NewReplicaSet:  nginx-deploy-65bc4bdf84 (1/1 replicas created)
  Events:
    Type     Reason   Age   From             Message
    Normal   ScalingReplicaSet  14m   deployment-controller  Scaled up replica set nginx-deploy-bd6fc657 from 0 to 1
    Normal   ScalingReplicaSet  3m50s  deployment-controller  Scaled up replica set nginx-deploy-65bc4bdf84 from 0 to 1
    Normal   ScalingReplicaSet  50s    deployment-controller  Scaled down replica set nginx-deploy-bd6fc657 from 1 to 0
```

5) Question:

Taint the worker node to be Unschedulable. Once done, create a pod called dev-redis, image redis:alpine to ensure workloads are not scheduled to this worker node. Finally, create a new pod called prod-redis and image redis:alpine with toleration to be scheduled on node01.

Ans:

Identify worker node

Kubectl get nodes

```
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl get nodes
NAME           STATUS   ROLES      AGE    VERSION
dev-cluster-control-plane Ready    control-plane 7h36m  v1.34.0
dev-cluster-worker     Ready    <none>     7h36m  v1.34.0
dev-cluster-worker2    Ready    <none>     7h36m  v1.34.0
deepa@ubuntu:~/devops/kind/311025/Q2$
```

Select the node ‘dev-cluster-worker’ to taint:Taint the worker node:

```
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl taint nodes dev-cluster-worker key1=value1:NoSchedule
node/dev-cluster-worker tainted
deepa@ubuntu:~/devops/kind/311025/Q2$
```

This makes the node “unschedulable” for normal pods.

```
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl describe node dev-cluster-worker
Name:           dev-cluster-worker
Roles:          <none>
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                kubernetes.io/arch=amd64
                kubernetes.io/hostname=dev-cluster-worker
                kubernetes.io/os=linux
Annotations:   node.alpha.kubernetes.io/ttl: 0
                volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp: Thu, 30 Oct 2025 23:41:23 -0700
Taints:         key1=value1:NoSchedule
Conditions:
  Type        Status  LastHeartbeatTime           LastTransitionTime        Reason           Message
  MemoryPressure False   Fri, 31 Oct 2025 07:29:51 -0700 Thu, 30 Oct 2025 23:41:23 -0700 KubeletHasSufficientMemory  kubelet
  has sufficient memory available
  DiskPressure  False   Fri, 31 Oct 2025 07:29:51 -0700 Thu, 30 Oct 2025 23:41:23 -0700 KubeletHasNoDiskPressure  kubelet
  has no disk pressure
  PIDPressure  False   Fri, 31 Oct 2025 07:29:51 -0700 Thu, 30 Oct 2025 23:41:23 -0700 KubeletHasSufficientPID  kubelet
  has sufficient PID available
  Ready       True    Fri, 31 Oct 2025 07:29:51 -0700 Thu, 30 Oct 2025 23:43:29 -0700 KubeletReady            kubelet
  is posting ready status
Addresses:
  InternalIP: 172.21.0.3
  Hostname:   dev-cluster-worker
Capacity:
  cpu:        8
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl describe node dev-cluster-worker | grep -i taints
Taints:         key1=value1:NoSchedule
deepa@ubuntu:~/devops/kind/311025/Q2$
```

Create a pod called dev-redis, image redis:alpine to ensure workloads are not scheduled to this worker node:

Create dev-redis.yaml

Create a pod *without* toleration

```
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl apply -f dev-redis.yaml
pod/dev-redis created
```

```
deepa@ubuntu:~/devops/kind/311025/Q2$ cat dev-redis.yaml
apiVersion: v1
kind: Pod
metadata:
  name: dev-redis
spec:
  containers:
    - name: redis
      image: redis:alpine
deepa@ubuntu:~/devops/kind/311025/Q2$
```

```
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE      IP           NODE      NOMINATED NODE   READINESS GATES
dev-redis   1/1     Running   0          15m     10.244.2.6  dev-cluster-worker2  <none>    <none>
hello-world-7f6f4c8b6-5pphb 1/1     Running   0          7h26m   10.244.2.3  dev-cluster-worker2  <none>    <none>
hello-world-7f6f4c8b6-6phbj 1/1     Running   0          7h26m   10.244.1.6  dev-cluster-worker  <none>    <none>
nginx-deploy-65bc4bdf84-hppgg 1/1     Running   0          4h24m   10.244.2.5  dev-cluster-worker2  <none>    <none>
deepa@ubuntu:~/devops/kind/311025/Q2$
```

The taint on dev-cluster-worker (key1=value1:NoSchedule) is working properly.

Scheduled on the untainted node — because it has no toleration, so it avoided dev-cluster-worker (which is tainted).

Kubernetes’ scheduler is functioning correctly.

Now schedule a pod that *can* tolerate the taint on the first worker node.

Create a file called prod-redis.yaml:

```

deepa@ubuntu:~/devops/kind/311025/Q2$ vi prod-redis.yaml
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl apply -f prod-redis.yaml
pod/prod-redis created
deepa@ubuntu:~/devops/kind/311025/Q2$ cat prod-redis.yaml
apiVersion: v1
kind: Pod
metadata:
  name: prod-redis
spec:
  tolerations:
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoSchedule"
  containers:
    - name: redis
      image: redis:alpine
deepa@ubuntu:~/devops/kind/311025/Q2$ █

```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
dev-redis	1/1	Running	0	66m	10.244.2.6	dev-cluster-worker2	<none>	<none>
nginx-deploy-65bc4bdf84-hppgg	1/1	Running	0	5h15m	10.244.2.5	dev-cluster-worker2	<none>	<none>
prod-redis	1/1	Running	0	12s	10.244.2.8	dev-cluster-worker2	<none>	<none>

dev-redis → running on dev-cluster-worker2

prod-redis → running on dev-cluster-worker2

Tolerations don't force a pod onto a tainted node.

They only allow it to run there *if the scheduler chooses that node*.

Kubernetes will still pick any node that satisfies the pod's requirements — and since both nodes can run the pod, the scheduler picked worker2 (no taint).

A toleration lets a pod ignore a taint, but doesn't target that node.

How to make prod-redis run *specifically* on dev-cluster-worker

We can use node affinity or a nodeSelector.

```

deepa@ubuntu:~/devops/kind/311025/Q2$ vi prod-redis.yaml
deepa@ubuntu:~/devops/kind/311025/Q2$ cat prod-redis.yaml
apiVersion: v1
kind: Pod
metadata:
  name: prod-redis
spec:
  tolerations:
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoSchedule"           added nodeSelector
  nodeSelector:
    kubernetes.io/hostname: dev-cluster-worker
  containers:
    - name: redis
      image: redis:alpine

```

Apply manifest yaml : delete the earlier pod.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	R
EADINESS GATES								
dev-redis	1/1	Running	0	76m	10.244.2.6	dev-cluster-worker2	<none>	<none>
nginx-deploy-65bc4bdf84-hppgg	1/1	Running	0	5h25m	10.244.2.5	dev-cluster-worker2	<none>	<none>
prod-redis	0/1	ContainerCreating	0	20s	<none>	dev-cluster-worker	<none>	<none>

Verify:

```
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE      NOMINATED NODE   READINESS GATES
dev-redis      1/1     Running   0          78m    10.244.2.6   dev-cluster-worker2   <none>        <none>
nginx-deploy-65bc4bdf84-hppgg  1/1     Running   0          5h27m  10.244.2.5   dev-cluster-worker2   <none>        <none>
prod-redis     1/1     Running   0          2m33s  10.244.1.7   dev-cluster-worker   <none>        <none>
deepa@ubuntu:~/devops/kind/311025/Q2$
```

Other option: Node affinity:

```
deepa@ubuntu:~/devops/kind/311025/Q2$ vi prod-redis.yaml
deepa@ubuntu:~/devops/kind/311025/Q2$ cat prod-redis.yaml
apiVersion: v1
kind: Pod
metadata:
  name: prod-redis
spec:
  tolerations:
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoSchedule"
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                  - dev-cluster-worker
  containers:
    - name: redis
      image: redis:alpine

deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl apply -f prod-redis.yaml
pod/prod-redis created
deepa@ubuntu:~/devops/kind/311025/Q2$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE      NOMINATED NODE   READINESS GATES
dev-redis      1/1     Running   0          93m    10.244.2.6   dev-cluster-worker2   <none>        <none>
nginx-deploy-65bc4bdf84-hppgg  1/1     Running   0          5h42m  10.244.2.5   dev-cluster-worker2   <none>        <none>
prod-redis     1/1     Running   0          27s    10.244.1.8   dev-cluster-worker   <none>        <none>
deepa@ubuntu:~/devops/kind/311025/Q2$
```

NodeSelector is the simplest way to tell Kubernetes:. It is used when we already know the exact node name or label.

Not flexible (can only match exact key-value).

Small clusters (like kind, minikube, dev/test) suitable.

NodeAffinity is the advanced version of nodeSelector.

Can define *required* or *preferred* rules.

Used in production for smarter scheduling., Production-grade, dynamic clusters.

Slightly more complex to read/write.

6) Question:

- Create a deployment with the latest nginx image and two replicas. add a new deployment using the image bitnami/apache with two replicas.
- Expose its port 8080 through a service and query it.
- Deploy nginx ingress controller
- Create an ingress service that redirects /nginx to the nginx service and /apache to the apache service.

Ans:

Create NGINX Deployment and Service

```

deepa@ubuntu:~/devops/kind/311025/Q3$ vi nginx-deploy.yaml
deepa@ubuntu:~/devops/kind/311025/Q3$ cat nginx-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deploy created
deepa@ubuntu:~/devops/kind/311025/Q3$ █

```

```

deepa@ubuntu:~/devops/kind/311025/Q3$ vi nginx-service.yaml
deepa@ubuntu:~/devops/kind/311025/Q3$ cat nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - port: 8080
      targetPort: 80
      protocol: TCP
  type: ClusterIP
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl apply -f nginx-service.yaml
service/nginx-service created

```

Create Apache (Bitnami) Deployment and Service

```

deepa@ubuntu:~/devops/kind/311025/Q3$ vi apache-deploy.yaml
deepa@ubuntu:~/devops/kind/311025/Q3$ cat apache-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
        - name: apache
          image: bitnami/apache:latest
          ports:
            - containerPort: 8080
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl apply -f apache-deploy.yaml
deployment.apps/apache-deploy created

```

```

deepa@ubuntu:~/devops/kind/311025/Q3$ vi apache-service.yaml
deepa@ubuntu:~/devops/kind/311025/Q3$ cat apache-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: apache-service
spec:
  selector:
    app: apache
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: ClusterIP
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl apply -f apache-service.yaml
service/apache-service created

```

Deploy NGINX Ingress Controller

Since you're using kind, there's a prebuilt setup command.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
```

```
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
deepa@ubuntu:~/devops/kind/311025/Q3$ 
deepa@ubuntu:~/devops/kind/311025/Q3$ █
```

```
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl get pods -n ingress-nginx
NAME                               READY   STATUS      RESTARTS   AGE
ingress-nginx-controller-6c948dc88d-58ggf   0/1    ContainerCreating   0          5m11s
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl get pods -n ingress-nginx
NAME                               READY   STATUS      RESTARTS   AGE
ingress-nginx-controller-6c948dc88d-58ggf   0/1    ContainerCreating   0          7m9s
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl get pods -n ingress-nginx
NAME                               READY   STATUS      RESTARTS   AGE
ingress-nginx-controller-6c948dc88d-58ggf   1/1    Running    0          9m43s
deepa@ubuntu:~/devops/kind/311025/Q3$ █
```

Create Ingress Resource and Apply:

```
deepa@ubuntu:~/devops/kind/311025/Q3$ vi web-ingress.yaml
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl apply -f web-ingress.yaml
ingress.networking.k8s.io/web-ingress created
deepa@ubuntu:~/devops/kind/311025/Q3$ cat web-ingress.yaml
apiVersion: networking.k8s.io/v1
Kind: Ingress
metadata:
  name: web-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
          - path: /nginx
            pathType: Prefix
            backend:
              service:
                name: nginx-service
                port:
                  number: 8080
          - path: /apache
            pathType: Prefix
            backend:
              service:
                name: apache-service
                port:
                  number: 8080
deepa@ubuntu:~/devops/kind/311025/Q3$ █
```

Verify:

```
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl get ingress
NAME      CLASS  HOSTS  ADDRESS      PORTS  AGE
web-ingress  nginx *  localhost  80      2m11s
deepa@ubuntu:~/devops/kind/311025/Q3$
```

Test Access via Port Forwarding:

```
deepa@ubuntu:~/devops/kind/311025/Q3$ 
deepa@ubuntu:~/devops/kind/311025/Q3$ kubectl port-forward -n ingress-nginx service/ingress-nginx-controller 8080:80
Forwarding from 127.0.0.1:8080 → 80
Forwarding from [::1]:8080 → 80
Handling connection for 8080
Handling connection for 8080
█
```

In another terminal test:

```
deepa@ubuntu:~/devops/kind/311025/Q3$ curl http://localhost:8080/nginx
<!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>
```



```
deepa@ubuntu:~/devops/kind/311025/Q3$ curl http://localhost:8080/apache
<html>
<head><title>503 Service Temporarily Unavailable</title></head>
<body>
<center><h1>503 Service Temporarily Unavailable</h1></center>
<hr><center>nginx</center>
</body>
</html>
deepa@ubuntu:~/devops/kind/311025/Q3$
```



7) Question:

Create a new ClusterRole named deployment-clusterrole, which only allows to create the following resource types:

Deployment

Stateful Set

DaemonSet

Create a new ServiceAccount named cicd-token in the existing namespace app-team1.

Bind the new ClusterRole deployment-clusterrole to the new ServiceAccount cicd-token, limited to the namespace app-team1.

Ans:

RBAC (Role-Based Access Control)

Create a ClusterRole

```
deepa@ubuntu:~/devops/kind/311025/04$ vi deployment-clusterrole.yaml
deepa@ubuntu:~/devops/kind/311025/04$ kubectl apply -f deployment-clusterrole.yaml
clusterrole.rbac.authorization.k8s.io/deployment-clusterrole created
deepa@ubuntu:~/devops/kind/311025/04$ cat deployment-clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: deployment-clusterrole
rules:
- apiGroups: ["apps"]
  resources: ["deployments", "statefulsets", "daemonsets"]
  verbs: ["create"]
```

Create a ServiceAccount:

create the ServiceAccount in the existing namespace app-team1.

```
deepa@ubuntu:~/devops/kind/311025/04$ vi app-team1-ns.yaml
deepa@ubuntu:~/devops/kind/311025/04$ kubectl apply -f app-team1-ns.yaml
namespace/app-team1 created
deepa@ubuntu:~/devops/kind/311025/04$ cat app-team1-ns.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: app-team1
deepa@ubuntu:~/devops/kind/311025/04$
```

```
deepa@ubuntu:~/devops/kind/311025/04$ kubectl apply -f cicd-token-sa.yaml
serviceaccount/cicd-token created
deepa@ubuntu:~/devops/kind/311025/04$ cat cicd-token-sa.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cicd-token
  namespace: app-team1
deepa@ubuntu:~/devops/kind/311025/04$
```

```
deepa@ubuntu:~/devops/kind/311025/04$ kubectl get namespaces
NAME          STATUS   AGE
app-team1     Active   56s
default       Active   12h
ingress-nginx Active   98m
kube-node-lease Active   12h
kube-public   Active   12h
kube-system   Active   12h
local-path-storage Active   12h
```

```
deepa@ubuntu:~/devops/kind/311025/04$ vi cicd-rolebinding.yaml
deepa@ubuntu:~/devops/kind/311025/04$ cat cicd-rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
Kind: RoleBinding
metadata:
  name: cicd-deployment-binding
  namespace: app-team1
subjects:
- kind: ServiceAccount
  name: cicd-token
  namespace: app-team1
roleRef:
  kind: ClusterRole
  name: deployment-clusterrole
  apiGroup: rbac.authorization.k8s.io
deepa@ubuntu:~/devops/kind/311025/04$ kubectl apply -f cicd-rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/cicd-deployment-binding created
deepa@ubuntu:~/devops/kind/311025/04$
```

Verify:

```
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl get clusterrole deployment-clusterrole
NAME          CREATED AT
deployment-clusterrole  2025-10-31T19:19:27Z
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl get sa -n app-team1
NAME      SECRETS   AGE
cicd-token  0        42m
default     0        43m
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl get rolebinding -n app-team1
NAME          ROLE          AGE
cicd-deployment-binding  ClusterRole/deployment-clusterrole  43s
deepa@ubuntu:~/devops/kind/311025/Q4$
```

After v1.24:

- Kubernetes no longer stores those tokens as Secrets by default.
- Tokens are short-lived and automatically provided to Pods at runtime via the projected service account token mechanism.

Check if ServiceAccount has permission to create Deployments in the app-team1 namespace:

```
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl auth can-i create deployments \
>   --as=system:serviceaccount:app-team1:cicd-token \
>   -n app-team1
yes
deepa@ubuntu:~/devops/kind/311025/Q4$
```

```
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl auth can-i delete pods \
>   --as=system:serviceaccount:app-team1:cicd-token \
>   -n app-team1
no
deepa@ubuntu:~/devops/kind/311025/Q4$
```

Use the ServiceAccount in a Pod

```
deepa@ubuntu:~/devops/kind/311025/Q4$ vi rbac-test.yaml
deepa@ubuntu:~/devops/kind/311025/Q4$ cat rbac-test.yaml
apiVersion: v1
kind: Pod
metadata:
  name: rbac-test
  namespace: app-team1
spec:
  serviceAccountName: cicd-token
  containers:
  - name: kubectl
    image: bitnami/kubectl:latest
    command: ["sleep", "3600"]
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl apply -f rbac-test.yaml
pod/rbac-test created
deepa@ubuntu:~/devops/kind/311025/Q4$
```

Test permissions: inside container

```
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl get pods -n app-team1
NAME      READY  STATUS    RESTARTS   AGE
rbac-test  1/1    Running   0          5m37s
deepa@ubuntu:~/devops/kind/311025/Q4$ kubectl exec -it rbac-test -n app-team1 -- bash
I have no name! [ / ]$
kubectl auth can-i create deployments -n app-team1
yes
I have no name! [ / ]$ kubectl auth can-i delete pods -n app-team1
no
I have no name! [ / ]$
```

8) Question:

Create a new NetworkPolicy named allow-port-from-namespace in the existing namespace fubar.

Ensure that the new NetworkPolicy allows Pods in namespace internal to connect to port 9000 of Pods in namespace fubar.

Further ensure that the new NetworkPolicy:

does not allow access to Pods, which don't listen on port 9000

does not allow access from Pods, which are not in namespace internal

Ans.

Create two namespaces with a label :

- fubar — where the protected Pods live, label=fubar
- internal — where allowed Pods live, label=internal

kubectl create ns fubar

kubectl create ns internal

Or we can use yaml file:

Create namespace fubar with a label

Verify:

kubectl get ns

fubar-namespace.yaml

```
deepa@ubuntu:~/devops/kind/311025/Q5$ vi fubar-namespace.yaml
deepa@ubuntu:~/devops/kind/311025/Q5$ cat fubar-namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: fubar
  labels:
    purpose: fubar
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f fubar-namespace.yaml

namespace/fubar created
deepa@ubuntu:~/devops/kind/311025/Q5$
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get ns fubar --show-labels
NAME      STATUS   AGE     LABELS
fubar     Active   11s    kubernetes.io/metadata.name=fubar,purpose=fubar
deepa@ubuntu:~/devops/kind/311025/Q5$
```

internal-namespace.yaml

```
deepa@ubuntu:~/devops/kind/311025/Q5$ vi internal-namespace.yaml
deepa@ubuntu:~/devops/kind/311025/Q5$ cat internal-namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: internal
  labels:
    purpose: internal
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f internal-namespace.yaml

namespace/internal created
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get ns internal --show-labels
NAME      STATUS   AGE     LABELS
internal  Active   2m10s  kubernetes.io/metadata.name=internal,purpose=internal
deepa@ubuntu:~/devops/kind/311025/Q5$
```

Create the NetworkPolicy YAML:

```
deepa@ubuntu:~/devops/kind/311025/Q5$ vi allow-port-from-namespace.yaml
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f allow-port-from-namespace.yaml
networkpolicy.networking.k8s.io/allow-port-from-namespace created
deepa@ubuntu:~/devops/kind/311025/Q5$ cat allow-port-from-namespace.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port-from-namespace
  namespace: fubar
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              purpose: internal
      ports:
        - protocol: TCP
          port: 9000
deepa@ubuntu:~/devops/kind/311025/Q5$
```

```
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get networkpolicy -n fubar
NAME                POD-SELECTOR   AGE
allow-port-from-namespace <none>     88s
deepa@ubuntu:~/devops/kind/311025/Q5$
```

Policy definition:

```
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl describe networkpolicy allow-port-from-namespace -n fubar
Name:      allow-port-from-namespace
Namespace: fubar
Created on: 2025-11-01 00:50:35 -0700 PDT
Labels:    <none>
Annotations: <none>
Spec:
  PodSelector: <none> (Allowing the specific traffic to all pods in this namespace)
    Allowing ingress traffic:
      To Port: 9000/TCP
      From:
        NamespaceSelector: purpose=internal
      Not affecting egress traffic
      Policy Types: Ingress
deepa@ubuntu:~/devops/kind/311025/Q5$
```

Testing the policy:

Create a test pod in fubar that listens on port 9000:

```
deepa@ubuntu:~/devops/kind/311025/Q5$ vi fubar-server.yaml
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f fubar-server.yaml
pod/fubar-server created
deepa@ubuntu:~/devops/kind/311025/Q5$ cat fubar-server.yaml
apiVersion: v1
kind: Pod
metadata:
  name: fubar-server
  namespace: fubar
  labels:
    app: fubar-server
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 9000
deepa@ubuntu:~/devops/kind/311025/Q5$ 
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get pod -n fubar
NAME      READY   STATUS    RESTARTS   AGE
fubar-server   1/1     Running   0          40s
deepa@ubuntu:~/devops/kind/311025/Q5$ 
```

Create a pod in internal namespace (should be able to connect):

```
deepa@ubuntu:~/devops/kind/311025/Q5$ vi internal-pod1.yaml
deepa@ubuntu:~/devops/kind/311025/Q5$ cat internal-pod1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: internal-pod1
  namespace: internal
  labels:
    app: internal-pod1
spec:
  containers:
  - name: client
    image: busybox:latest
    command: ["sleep", "3600"] # keep the pod alive for testing
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f internal-pod1.yaml
pod/internal-pod1 created
deepa@ubuntu:~/devops/kind/311025/Q5$ 
```

```
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get pods -n internal
NAME      READY   STATUS    RESTARTS   AGE
internal-pod1   1/1     Running   0          26s
deepa@ubuntu:~/devops/kind/311025/Q5$ 
```

Test connectivity to the server pod in fubar

```
deepa@ubuntu:~/devops/kind/311025/Q5$ 
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl exec -it internal-pod1 -n internal -- sh
/ # wget -qO- http://fubar-server.fubar.svc.cluster.local:9000
wget: bad address 'fubar-server.fubar.svc.cluster.local:9000'
/ # wget -qO- http://fubar-server.fubar.svc.cluster.local:9000
wget: bad address 'fubar-server.fubar.svc.cluster.local:9000'
/ # 
```

Error:

```
wget: bad address 'fubar-server.fubar.svc.cluster.local:9000'  
means DNS lookup failed — Kubernetes can't resolve the name fubar-server.fubar.svc.cluster.local.  
wget: bad address 'fubar-server.fubar.svc.cluster.local:9000'  
/ # exit  
command terminated with exit code 1  
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get svc -n fubar  
No resources found in fubar namespace.  
deepa@ubuntu:~/devops/kind/311025/Q5$ █
```

Service doesn't created.

```
deepa@ubuntu:~/devops/kind/311025/Q5$ vi fubar-server-svc.yaml  
deepa@ubuntu:~/devops/kind/311025/Q5$ cat fubar-server-svc.yaml  
apiVersion: v1  
kind: Service  
metadata:  
  name: fubar-server  
  namespace: fubar  
spec:  
  selector:  
    app: fubar-server  
  ports:  
    - protocol: TCP  
      port: 9000  
      targetPort: 9000  
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f fubar-server-svc.yaml  
service/fubar-server created  
deepa@ubuntu:~/devops/kind/311025/Q5$ █
```

```
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get svc -n fubar  
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE  
fubar-server   ClusterIP  10.96.193.238  <none>        9000/TCP   25s  
deepa@ubuntu:~/devops/kind/311025/Q5$ █
```

Confirm the target Pod is running:

```
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get pods -n fubar -o wide  
NAME        READY   STATUS    RESTARTS   AGE   IP           NODE      NOMINATED-NODE  READINESS GATES  
fubar-server  1/1     Running   0          19m   10.244.2.8   dev-cluster-worker2  <none>        <none>  
deepa@ubuntu:~/devops/kind/311025/Q5$ █
```

DNS is working

```
/ # wget -qO- http://fubar-server.fubar.svc.cluster.local:9000  
wget: can't connect to remote host (10.96.193.238): Connection refused  
/ # nslookup fubar-server.fubar.svc.cluster.local  
Server: 10.96.0.10  
Address: 10.96.0.10:53  
/ # █
```

Connection is refused:

The Service exists and resolves to an IP, but the target Pod isn't actually listening on **port 9000**.

```
/ # wget -qO- http://fubar-server.fubar.svc.cluster.local:9000  
wget: can't connect to remote host (10.96.193.238): Connection refused  
/ # █
```

Pod **exposes containerPort 9000**,

but...

the **nginx process inside the container still listens on port 80**, not 9000 —

```

deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get pods -n fubar -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP           NODE      NOMINATED NODE   READINESS GATES
fubar-server 1/1     Running   0          26m    10.244.2.8   dev-cluster-worker2   <none>        <none>
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl describe pod fubar-server -n fubar
Name:           fubar-server
Namespace:      fubar
Priority:       0
Service Account: default
Node:          dev-cluster-worker2/172.21.0.3
Start Time:    Sat, 01 Nov 2025 00:55:39 -0700
Labels:         app=fubar-server
Annotations:   <none>
Status:        Running
IP:            10.244.2.8
IPs:
  IP:  10.244.2.8
Containers:
  nginx:
    Container ID:  containerd://2c08a305521719832c76682b59874b6880a062be807521be75445db9ac88f9a1
    Image:         nginx:latest
    Image ID:     docker.io/library/nginx@sha256:f547e3d0d5d02f7009737b284abc87d808e4252b42dceea361811e9fc606287f
    Port:          9000/TCP ←
    Host Port:    0/TCP
    State:        Running
      Started:   Sat, 01 Nov 2025 00:55:44 -0700
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-8974m (ro)
Conditions:
  Type  Status
  PodReadyToStartContainers  True
  Initialized  True

```

Make the Service route **port 9000 → container port 80**, where nginx is really serving content.

Update the **fubar-server-svc.yaml** to this:

```

deepa@ubuntu:~/devops/kind/311025/Q5$ vi fubar-server-svc.yaml
deepa@ubuntu:~/devops/kind/311025/Q5$ cat fubar-server-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: fubar-server
  namespace: fubar
spec:
  selector:
    app: fubar-server
  ports:
    - protocol: TCP
      port: 9000      # port clients (internal pods) connect to
      targetPort: 80    # real nginx port inside container

deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f fubar-server-svc.yaml
service/fubar-server configured
deepa@ubuntu:~/devops/kind/311025/Q5$ █

```

Create a test pod that already has curl

Let's make a slightly bigger test pod using the official curlimages/curl image (it has curl preinstalled).

Create YAML file — **internal-curl.yaml**:

```
deepa@ubuntu:~/devops/kind/311025/Q5$ vi internal-curl.yaml
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl apply -f internal-curl.yaml
pod/internal-curl created
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl get pods -n internal
NAME           READY   STATUS    RESTARTS   AGE
internal-curl  0/1     Pending   0          0s
internal-pod1  1/1     Running   2 (23m ago) 3h8m
deepa@ubuntu:~/devops/kind/311025/Q5$ cat internal-curl.yaml
apiVersion: v1
kind: Pod
metadata:
  name: internal-curl
  namespace: internal
  labels:
    app: internal-curl
spec:
  containers:
  - name: curl
    image: curlimages/curl:latest
    command: ["sleep", "3600"]
deepa@ubuntu:~/devops/kind/311025/Q5$ █
```

```
deepa@ubuntu:~/devops/kind/311025/Q5$ kubectl exec -it internal-curl -n internal -- sh
~ $ curl -v http://fubar-server.fubar.svc.cluster.local:9000
* Host fubar-server.fubar.svc.cluster.local:9000 was resolved.
* IPv6: (none)
* IPv4: 10.96.193.238
*   Trying 10.96.193.238...
* connect to 10.96.193.238 port 9000 from 10.244.223.193 port 38754 failed: Operation timed out
* Failed to connect to fubar-server.fubar.svc.cluster.local port 9000 after 129815 ms: Could not connect to server
* closing connection #0
curl: (28) Failed to connect to fubar-server.fubar.svc.cluster.local port 9000 after 129815 ms: Could not connect to server
~ $ █
```

DNS works

Service (ClusterIP 10.96.193.238) exists

Connection to that ClusterIP times out

That means **the network plugin isn't allowing traffic** between pods according to the NetworkPolicy rules — in this case because Kind's built-in network plugin (kindnet) **does not implement NetworkPolicy enforcement**.

So even though YAML and logic are correct, the traffic control part of the policy is being ignored.