KUBERNETES SERVICES AND NAMESPACES

Kubernetes services

- Service is a method for exposing Pods in your cluster.
- Each Pod gets its own IP address but we need to access from IP of the Node.
- If you want to access pod from inside we use Cluster-IP.
- If the service is of type NodePort or LoadBalancer, it can also be accessed from outside the cluster.
- It enables the pods to be decoupled from the network topology, which makes it easier to manage and scale applications.

Types of services:

- Cluster-IP
- Node Port
- LoadBalancer
- ExternalName

COMPONENTS OF SERVICES

A service is defined using a Kubernetes manifest file that describes its properties and specifications. Some of the key properties of a service include:

- **Selector**: A label selector that defines the set of pods that the service will route traffic to.
- **Port**: The port number on which the service will listen for incoming traffic.
- **TargetPort**: The port number on which the pods are listening for traffic.
- **Type**: The type of the service, such as ClusterIP, NodePort, LoadBalancer, or ExternalName.

TYPES OF SERVICES

- **ClusterIP**: A **ClusterIP** service provides a stable IP address and DNS name for pods within a cluster. This type of service is only accessible within the cluster and is not exposed externally.
- **NodePort**: A **NodePort** service provides a way to expose a service on a static port on each node in the cluster. This type of service is accessible both within the cluster and externally, using the node's IP address and the NodePort.
- **LoadBalancer**: A **LoadBalancer** service provides a way to expose a service externally, using a cloud provider's load balancer. This type of service is typically used when an application needs to handle high traffic loads and requires automatic scaling and load balancing capabilities.
- **ExternalName**: service type allows you to map a Kubernetes service to an external DNS name. Instead of proxying the traffic, it returns a CNAME record that routes traffic directly to the external service.

Step-by-Step: ClusterIP Service

Step 1: Create a Deployment

Create a simple deployment that runs an app (like NGINX).

nginx-deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

Run:

kubectl apply -f nginx-deployment.yaml

```
ubuntu@kiran:~$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
```

Step 2: Create a ClusterIP Service

nginx-service.yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-clusterip-service
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Run:

kubectl apply -f nginx-service.yaml

```
ubuntu@kiran:~$ kubectl apply -f nginx-service.yaml
service/nginx-clusterip-service created
```

Step 3: Verify Deployment and Service

1. **Check pods**

kubectl get po

```
ubuntu@kiran:~$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6c8b449b8f-kc7p6   1/1     Running   0          54s
nginx-deployment-6c8b449b8f-lc9kx   1/1     Running   0          54s
ubuntu@kiran:~$ kubectl get po -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE                  NOMINATED NODE   READINESS GATES
nginx-deployment-6c8b449b8f-kc7p6   1/1     Running   0          66s   100.96.2.5   i-00634086fba6ef0ca   <none>           <none>
nginx-deployment-6c8b449b8f-lc9kx   1/1     Running   0          66s   100.96.1.4   i-037d5d932a411e744   <none>           <none>
```

2. **Check Service**

kubectl get svc

You'll see something like:

```
ubuntu@kiran:~$ kubectl get svc
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes              ClusterIP   100.64.0.1      <none>        443/TCP   35m
nginx-clusterip-service ClusterIP   100.64.186.26   <none>        80/TCP    10s
ubuntu@kiran:~$ kubectl get svc -o wide
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
kubernetes              ClusterIP   100.64.0.1      <none>        443/TCP   35m   <none>
nginx-clusterip-service ClusterIP   100.64.186.26   <none>        80/TCP    19s   app=nginx
```

Step 4: Test Internal Access

Run a temporary busybox pod to test:

kubectl run test-pod --rm -it --image=busybox -- /bin/sh

Then inside the pod:

wget -qO- nginx-clusterip-service

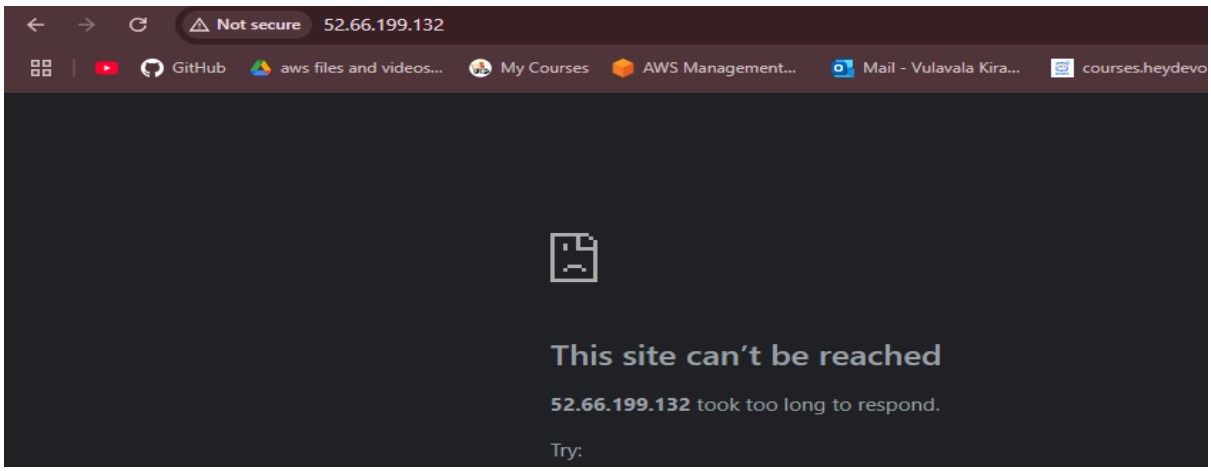
If successful, it will return NGINX HTML content. Exit the pod with `exit`.

```
ubuntu@kiran:~$ kubectl run test-pod --rm -it --image=busybox -- /bin/sh
If you don't see a command prompt, try pressing enter.
/ # wget -qO- nginx-clusterip-service
<!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>
/ #
```

If you test this in external you cannot access the page.



Step-by-Step: NodePort Service

Step 1: Create a Deployment

Create a file nginx-deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

Apply it:

kubectl apply -f nginx-deployment.yaml

```
ubuntu@kiran:~$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
```

Check if pods are running:

kubectl get pods

```
ubuntu@kiran:~$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6c8b449b8f-kc7p6   1/1     Running   0          54s
nginx-deployment-6c8b449b8f-lc9kx   1/1     Running   0          54s
ubuntu@kiran:~$ kubectl get po -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE                  NOMINATED NODE   READINESS GATES
nginx-deployment-6c8b449b8f-kc7p6   1/1     Running   0          66s   100.96.2.5   i-00634086fba6ef0ca   <none>           <none>
nginx-deployment-6c8b449b8f-lc9kx   1/1     Running   0          66s   100.96.1.4   i-037d5d932a411e744   <none>           <none>
```

Step 2: Create a NodePort Service

Create a file nginx-nodeport-service.yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
```

Apply it:

kubectl apply -f nginx-nodeport-service.yaml

```
ubuntu@kiran:~$ kubectl apply -f nginx-nodeport-service.yaml
service/nginx-nodeport-service created
```

Step 3: Get Node External IP (EC2 Public IP)

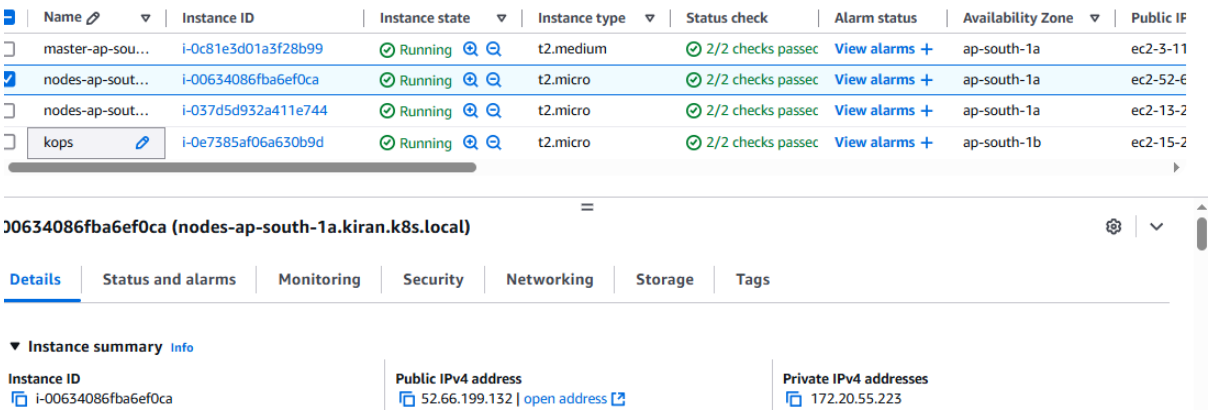
Run:

kubectl get nodes -o wide

```
ubuntu@kiran:~$ kubectl get nodes -o wide
NAME                  STATUS   ROLES           AGE   VERSION   INTERNAL-IP     EXTERNAL-IP      OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
i-00634086fba6ef0ca   Ready    node            66m   v1.24.17  172.20.55.223   52.66.199.132    Ubuntu 20.04.6 LTS   5.15.0-1077-aws  containerd://1.6.6
i-037d5d932a411e744   Ready    node            66m   v1.24.17  172.20.46.153   13.233.190.183   Ubuntu 20.04.6 LTS   5.15.0-1077-aws  containerd://1.6.6
i-0c81e3d01a3f28b99   Ready    control-plane   68m   v1.24.17  172.20.57.39    3.111.217.160    Ubuntu 20.04.6 LTS   5.15.0-1077-aws  containerd://1.6.6
ubuntu@kiran:~$
```

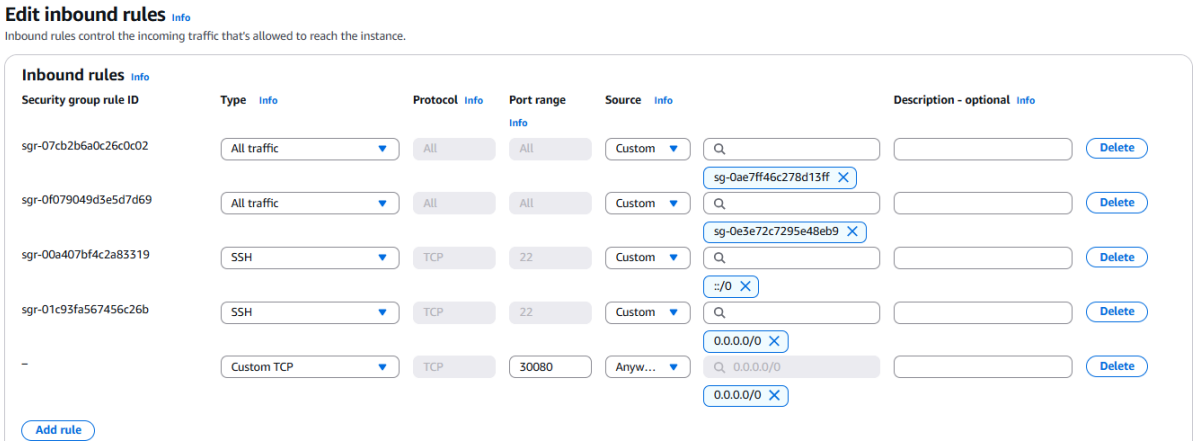
Check the **EXTERNAL-IP**. If it's `<none>`, then:

- Go to **AWS EC2 Console**
- Find your worker nodes (name usually includes "nodes")
- Copy **Public IPv4 address** of any node



Step 4: Allow Port in Security Group

1. Go to AWS EC2 → **Security Groups**
2. Find the group attached to the worker node
3. Edit **Inbound rules**:
    - Add:
        - **Type**: Custom TCP
        - **Port range**: 30080
        - **Source**: 0.0.0.0/0 (for public testing)

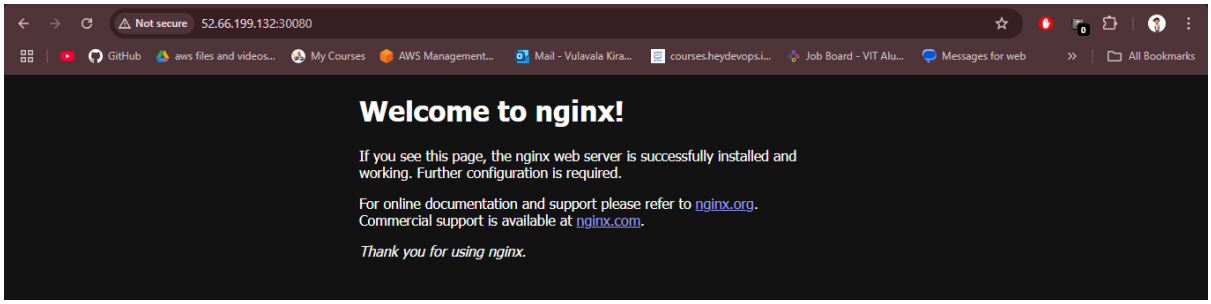


Step 5: Access the App

Open in browser:

http://<EC2-Public-IP>:30080

You should see the **Nginx Welcome Page**!



Step-by-Step: LoadBalancer Service

Step 1: Create Nginx Deployment

Create a file called nginx-deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

Apply it:

kubectl apply -f nginx-deployment.yaml

```
ubuntu@kiran:~$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
```

Verify:

kubectl get pods

```
ubuntu@kiran:~$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6c8b449b8f-kc7p6   1/1     Running   0          54s
nginx-deployment-6c8b449b8f-lc9kx   1/1     Running   0          54s
ubuntu@kiran:~$ kubectl get po -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE                  NOMINATED NODE   READINESS GATES
nginx-deployment-6c8b449b8f-kc7p6   1/1     Running   0          66s   100.96.2.5   i-00634086fba6ef0ca   <none>           <none>
nginx-deployment-6c8b449b8f-lc9kx   1/1     Running   0          66s   100.96.1.4   i-037d5d932a411e744   <none>           <none>
```

Step 2: Create LoadBalancer Service

Create nginx-loadbalancer-service.yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-loadbalancer
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
```

Apply the service:

kubectl apply -f nginx-loadbalancer-service.yaml

```
ubuntu@kiran:~$ vi nginx-loadbalancer-service.yaml
ubuntu@kiran:~$ kubectl apply -f nginx-loadbalancer-service.yaml
service/nginx-loadbalancer created
ubuntu@kiran:~$
```

Check the service:

kubectl get svc nginx-loadbalancer

You'll see an **EXTERNAL-IP** being provisioned by AWS ELB (Elastic Load Balancer). It may take a couple of minutes.

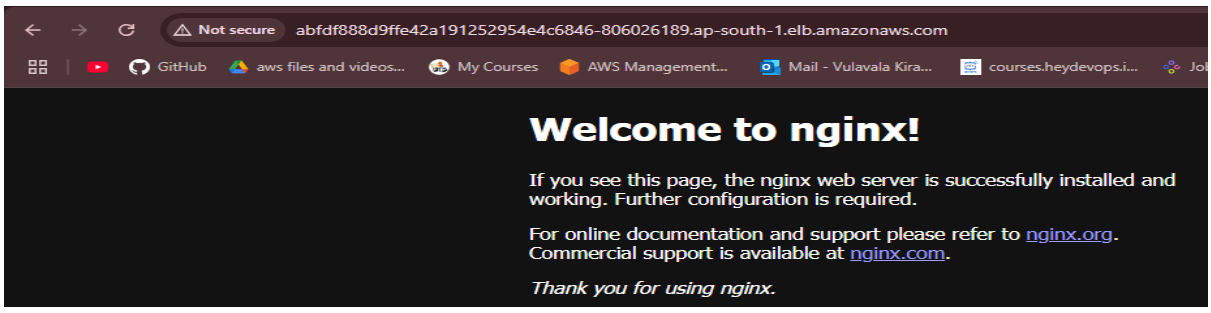
```
ubuntu@kiran:~$ kubectl get svc nginx-loadbalancer
NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)        AGE
nginx-loadbalancer   LoadBalancer   100.68.215.222   abfdf888d9ffe42a191252954e4c6846-806026189.ap-south-1.elb.amazonaws.com   80:30648/TCP   38s
ubuntu@kiran:~$
```

Step 3: Access the App

Once the EXTERNAL-IP is available, open it in your browser:

http://<EXTERNAL-IP>

You should see the **Nginx Welcome Page**.



Step 4: Cleanup (Optional)

kubectl delete -f nginx-deployment.yaml
kubectl delete -f nginx-loadbalancer-service.yaml

```
ubuntu@kiran:~$ kubectl delete -f nginx-deployment.yaml
kubectl delete -f nginx-loadbalancer-service.yaml
deployment.apps "nginx-deployment" deleted
service "nginx-loadbalancer" deleted
```

This LoadBalancer service uses an **AWS ELB**, which means:

- Traffic hits the ELB
- ELB forwards it to your worker nodes
- It reaches the nginx pods through kube-proxy.

Feature	ClusterIP	NodePort	LoadBalancer
Access Scope	Internal only	External via Node IP and port	External via cloud provider's IP
Default Type	✅ Yes	❌ No	❌ No
External Access	❌ No	✅ Yes (manual via Node IP)	✅ Yes (automatic via LB)
Port Range	Random within cluster	30000–32767	Automatically assigned by cloud provider
Cloud Support	Not needed	Not needed	✅ Required (AWS, Azure, GCP, etc.)
Use Case	Internal communication	Testing or dev environments	Production & internet-facing apps
Scalability	High within cluster	Limited	Scalable and balanced
Example URL	`my-service:80`	`http://<public-ip>:30080`	`http://<public-ip>`
Firewall Rules	Not applicable	Required for open ports	Handled by cloud provider

KUBERNETES NAMESPACE:

- **Namespaces** are used to group Kubernetes components like Pods, Services, and Deployments.
- Useful for separating environments such as **Development, Staging, QA, and Production**, or different **teams/projects**.
- In real-time, **frontend pods** may be mapped to one namespace, while **backend pods** go to another.
- A Namespace represents a **cluster within a cluster**.
- You can have multiple namespaces in a Kubernetes cluster; they are **logically isolated** from each other.
- Namespaces **provide logical separation** between resources used by different users, teams, projects, or customers.

Namespace Communication and Isolation

- Within the same namespace, **Pod-to-Pod communication** is direct.
- Namespaces are **logically separated**, not fully isolated—communication is possible with proper addressing.
- One service in a namespace can talk to another in a different namespace.
- **Resource names must be unique within a namespace**.
- When a namespace is **deleted**, all its resources are also deleted.

Default Namespaces in Kubernetes:

Name	Purpose
default	Where resources are created if no namespace is specified.
kube-public	For public resources available to all users.
kube-system	For resources created by Kubernetes system components.
kube-node-lease	Manages node lease objects to improve heartbeat performance at scale.

When to Use Namespaces

Use namespaces when:

- You want to separate environments (dev/test/prod).
- Multiple teams share a Kubernetes cluster.
- You need to apply different resource limits or policies per project/team.

If your cluster is small or used by a single team/project, you might not need namespaces at all beyond the default.

List all namespaces

kubectl get namespaces

```
ubuntu@kiran:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   113m
kube-node-lease   Active   113m
kube-public       Active   113m
kube-system       Active   113m
```

2. Create a new namespace

kubectl create namespace dev

```
ubuntu@kiran:~$ kubectl create namespace dev
namespace/dev created
```

3. Create a pod in a specific namespace

kubectl run nginx-pod --image=nginx --namespace=dev

```
ubuntu@kiran:~$ kubectl run nginx-pod --image=nginx --namespace=dev
pod/nginx-pod created
```

4. View all pods in a namespace

kubectl get pods -n dev

```
ubuntu@kiran:~$ kubectl get pods -n dev
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          9s
```

5. Set your current context to a namespace

kubectl config set-context --current --namespace=dev

```
ubuntu@kiran:~$ kubectl config set-context --current --namespace=dev
Context "kiran.k8s.local" modified.
```

6. Create a deployment in a namespace

kubectl create deployment myapp --image=nginx -n dev

```
ubuntu@kiran:~$ kubectl create deployment myapp --image=nginx -n dev
deployment.apps/myapp created
```

7. Delete a namespace

kubectl delete namespace dev

Caution: This will delete all resources in that namespace.

```
ubuntu@kiran:~$ kubectl delete namespace dev
namespace "dev" deleted
```

YAML Example

Create a namespace using a manifest:

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev-team
```

Apply it:

kubectl apply -f namespace.yaml

Summary

Namespaces are essential for multi-tenant environments in Kubernetes. They simplify resource management, enable isolation, and improve security. Mastering namespaces helps in building well-organized, enterprise-grade clusters.