Create a k8s cluster.

```
[root@ip-172-31-29-237 ~]# cat <<EOF | kind create cluster --name tsk8s --config=-
> kind: Cluster
> apiVersion: kind.x-k8s.io/v1alpha4
> nodes:
> - role: control-plane
>   kubeadmConfigPatches:
>   - |
>     kind: InitConfiguration
>     nodeRegistration:
>       kubeletExtraArgs:
>         node-labels: "ingress-ready=true"
>   extraPortMappings:
>   - containerPort: 80
>     hostPort: 8080
>     protocol: TCP
>   - containerPort: 443
>     hostPort: 8443
>     protocol: TCP
>   - containerPort: 30000
>     hostPort: 30000
>     protocol: TCP
> EOF
Creating cluster "tsk8s" ...
 ✓ Ensuring node image (kindest/node:v1.25.3) 
 ✓ Preparing nodes 
 ✓ Writing configuration 
 ✓ Starting control-plane 
 ✓ Installing CNI 
 ✓ Installing StorageClass 
Set kubectl context to "kind-tsk8s"
You can now use your cluster with:

kubectl cluster-info --context kind-tsk8s

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s.io/#community
```

Illustration:
extraPortMappings: Expose the K8s container (equivalent to the server where K8s is located) port. 8080, 8443, and 30000 are exposed here.
node-labels: Only allow the Ingress controller to run on nodes with the "ingress-ready=true" label.

At this time, we can see that ports 80, 8443, and 30000 have been exposed.

```
[root@ip-172-31-29-237 ~]# docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED              STATUS              PORTS                  NAMES
a30b655fbd13   kindest/node:v1.25.3   "/usr/local/bin/entr…"   About a minute ago   Up About a minute   0.0.0.0:30000->30000/tcp, 0.0.0.0:8080->80/tcp, 0.0.0.0:8443->443/tcp, 127.0.0.1:35759->6443/tcp   tsk8s-control-plane
```

Create a new file my-dep.yaml and add the following content.

root@ip-172-31-29-237:~

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpd-dep
spec:
  replicas: 1  # number of replicas of frontEnd application
  selector:
    matchLabels:
      app: httpd-app
  template:
    metadata:
      labels: # Must match 'Service' and 'Deployment' labels
        app: httpd-app
    spec:
      containers:
      - name: httpd
        image: httpd # docker image of frontend application
        ports:
        - containerPort: 80
~
```

Create Deployment.

```
[root@ip-172-31-29-237 ~]# kubectl apply -f my-dep.yaml
deployment.apps/httpd-dep created
[root@ip-172-31-29-237 ~]#
```

Illustration:
The name of the Deployment is "httpd-dep".
Managed Pods need to have the label "app: httpd-app".
The image specified in the Pod template to run is httpd in the Docker public warehouse.

To view the newly created Pod, -o wide can see more information.

```
[root@ip-172-31-29-237 ~]# kubectl get pods -o wide
NAME                        READY   STATUS    RESTARTS   AGE   IP           NODE                NOMINATED NODE   READINESS GATES
httpd-dep-558bdf4498-6sd5c  1/1     Running   0          13m   10.244.0.5   tsk8s-control-plane <none>           <none>
[root@ip-172-31-29-237 ~]#
```

We can see that the Pod is assigned a K8s internal IP. This IP cannot be accessed from outside K8s, and this IP will change after the Pod is rebuilt.

Create a new file my-svc.yaml and add the following content.

root@ip-172-31-29-237:~

```
kind: Service
apiVersion: v1
metadata:
  name: httpd-svc
spec:
  selector:
      app: httpd-app
  ports:
  - port: 80
```

Create service.

```
[root@ip-172-31-29-237 ~]# kubectl apply -f my-svc.yaml
service/httpd-svc created
[root@ip-172-31-29-237 ~]#
```

View Service information.

```
[root@ip-172-31-29-237 ~]# kubectl get svc/httpd-svc
NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
httpd-svc   ClusterIP   10.96.186.12   <none>        80/TCP    55s
[root@ip-172-31-29-237 ~]#
```

The IP of the Service will not change unless the Service is rebuilt.

View Service details.

```
[root@ip-172-31-29-237 ~]# kubectl describe svc/httpd-svc
Name:              httpd-svc
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=httpd-app
Type:              ClusterIP
IP:                10.96.186.12
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.244.0.5:80
Session Affinity:  None
Events:            <none>
[root@ip-172-31-29-237 ~]#
```

We can see that the IP and port of the Endpoint associated with the Service here are the IP and port of the Pod above. Each time the Pod is rebuilt, the Endpoint here will be refreshed to the new IP.
Currently, this Service only has internal IP and port, so this Service can only be used to expose Pods inside K8s.

Next, we modify the Service configuration so that this Service can be accessed externally using K8s.
Modify my-svc.yaml and add the following content.

root@ip-172-31-29-237:~

```
kind: Service
apiVersion: v1
metadata:
  name: httpd-svc
spec:
  selector:
      app: httpd-app
  type: NodePort #1
  ports:
  - port: 80
    nodePort: 30000 #2
~
```

Illustrate:
#1 Service type defaults to ClusterIP, which means only internal IP. After changing to NodePort, Service will map the internal port of K8s to the server where the cluster is located.
#2 Specify the port mapped by Service to the server as 30000.

Run the kubectl apply command again.

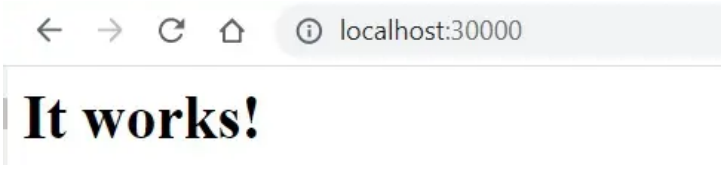
```
[root@ip-172-31-29-237 ~]# kubectl apply -f my-svc.yaml
service/httpd-svc configured
[root@ip-172-31-29-237 ~]#
```

Check the Service information again and we can see that there is an additional 30000 in the port.

```
[root@ip-172-31-29-237 ~]# kubectl get svc/httpd-svc
NAME         TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
httpd-svc    NodePort   10.96.186.12   <none>        80:30000/TCP   13m
[root@ip-172-31-29-237 ~]#
```

The port 30000 is mapped to the server where the K8s cluster is located (that is, the container where K8s runs), and when we created kind K8s, we mapped the container's 30000 port to the local, so now we can use the browser to access the 30000 port locally.

Access the httpd application on port 30000 locally.



It works!