Kubernetes have advanced networking capabilities that allow Pods and Services to communicate inside the cluster's network and externally.

In this scenario, you will learn the following types of Kubernetes services.

- Cluster IP
- Target Ports
- NodePort
- External IPs
- Load Balancer

Kubernetes Services are an abstract that defines a policy and approach on how to access a set of Pods. The set of Pods accessed via a Service is based on a Label Selector.

Step 1 - Cluster IP

Cluster IP is the default approach when creating a Kubernetes Service. The service is allocated an internal IP that other components can use to access the pods.

By having a single IP address it enables the service to be load balanced across multiple Pods.

Services are deployed via `kubectl apply -f clusterip.yaml`.

The definition can be viewed at `cat clusterip.yaml`

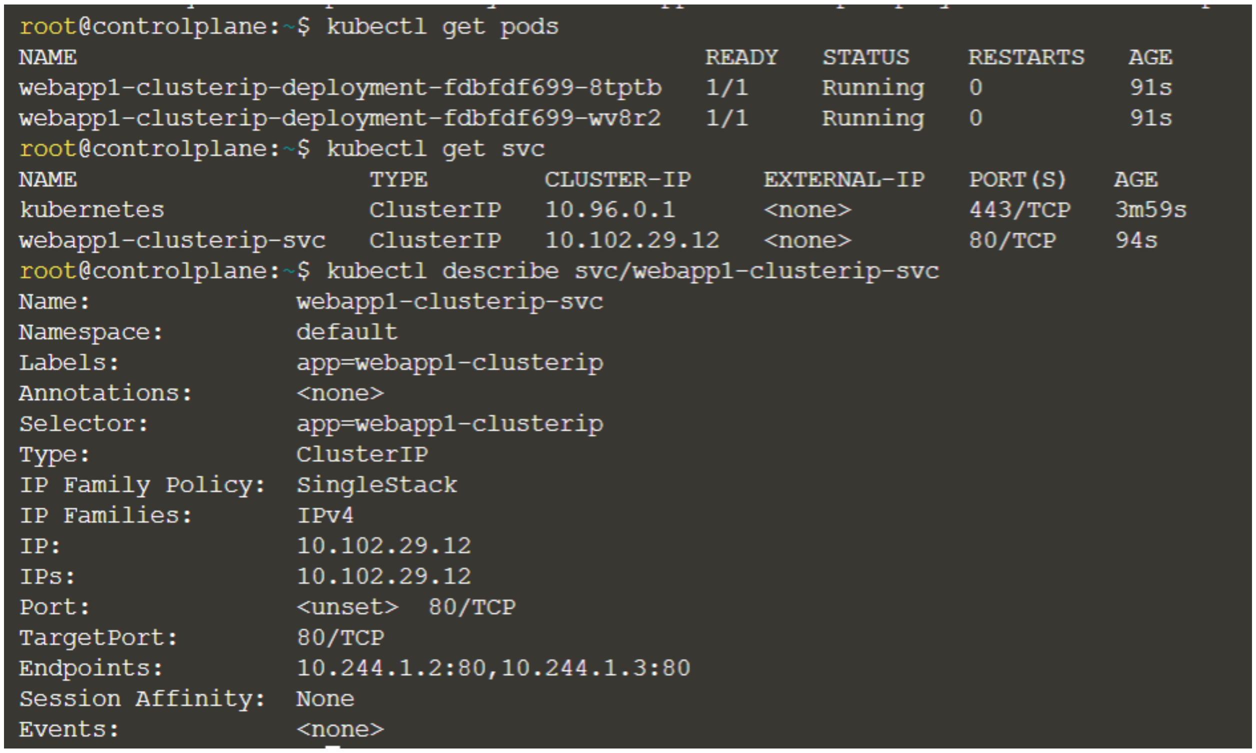
```
oot@controlplane:~$ cat clusterip.yaml
apiVersion: v1
kind: Service
```

```
metadata:
  name: webapp1-clusterip-svc
  labels:
    app: webapp1-clusterip
spec:
  ports:
  - port: 80
  selector:
    app: webapp1-clusterip
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp1-clusterip-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp1-clusterip
  template:
    metadata:
      labels:
        app: webapp1-clusterip
    spec:
      containers:
      - name: webapp1-clusterip-pod
        image: katacoda/docker-http-server:latest
        ports:
        - containerPort: 80
---
```

This will deploy a web app with two replicas to showcase load balancing along with a service. The Pods can be viewed at `kubectl get pods`

It will also deploy a service. `kubectl get svc`

More details on the service configuration and active endpoints (Pods) can be viewed via `kubectl describe svc/webapp1-clusterip-svc`



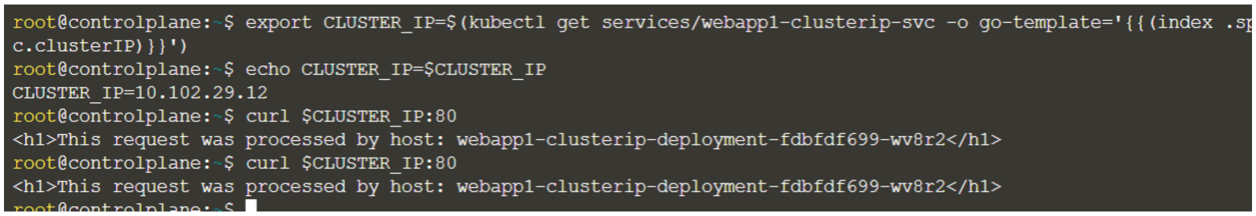
```
root@controlplane:~$ kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
webapp1-clusterip-deployment-fdbfdf699-8tptb  1/1     Running   0          91s
webapp1-clusterip-deployment-fdbfdf699-wv8r2  1/1     Running   0          91s
root@controlplane:~$ kubectl get svc
NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes              ClusterIP   10.96.0.1      <none>        443/TCP   3m59s
webapp1-clusterip-svc   ClusterIP   10.102.29.12   <none>        80/TCP    94s
root@controlplane:~$ kubectl describe svc/webapp1-clusterip-svc
Name:              webapp1-clusterip-svc
Namespace:         default
Labels:            app=webapp1-clusterip
Annotations:       <none>
Selector:          app=webapp1-clusterip
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.102.29.12
IPs:               10.102.29.12
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.244.1.2:80,10.244.1.3:80
Session Affinity:  None
Events:            <none>
```

After deploying, the service can be accessed via the ClusterIP allocated.

```
export CLUSTER_IP=$(kubectl get services/webapp1-clusterip-svc
-o go-template='{{(index .spec.clusterIP)}}') echo
CLUSTER_IP=$CLUSTER_IP curl $CLUSTER_IP:80
```

Multiple requests will showcase how the service load balancers across multiple Pods based on the common label selector.

```
curl $CLUSTER_IP:80
```



```
root@controlplane:~$ export CLUSTER_IP=$(kubectl get services/webapp1-clusterip-svc -o go-template='{{(index .sp
c.clusterIP)}}')
root@controlplane:~$ echo CLUSTER_IP=$CLUSTER_IP
CLUSTER_IP=10.102.29.12
root@controlplane:~$ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-fdbfdf699-wv8r2</h1>
root@controlplane:~$ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-fdbfdf699-wv8r2</h1>
root@controlplane:~$
```

Step 2 - Target Port

Target ports allows us to separate the port the service is available on from the port the application is listening on. TargetPort is the Port which the application is configured to listen on. Port is how the application will be accessed from the outside.

Similar to previously, the service and extra pods are deployed via `kubectl apply -f clusterip-target.yaml`

The following commands will create the service.

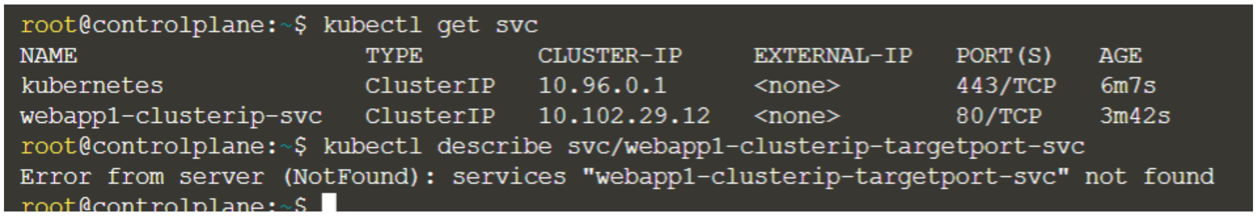
```
cat clusterip-target.yaml
```

```
root@controlplane:~$ cat clusterip-target.yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-clusterip-targetport-svc
  labels:
    app: webapp1-clusterip-targetport
spec:
  ports:
  - port: 8080
    targetPort: 80
  selector:
    app: webapp1-clusterip-targetport
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp1-clusterip-targetport-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp1-clusterip-targetport
  template:
    metadata:
      labels:
```

```
      app: webapp1-clusterip-targetport
    spec:
      containers:
      - name: webapp1-clusterip-targetport-pod
        image: katacoda/docker-http-server:latest
        ports:
        - containerPort: 80
---
```

```
kubectl get svc
```

```
kubectl describe svc/webapp1-clusterip-targetport-svc
```



```
root@controlplane:~$ kubectl get svc
NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes               ClusterIP   10.96.0.1      <none>        443/TCP   6m7s
webapp1-clusterip-svc    ClusterIP   10.102.29.12   <none>        80/TCP    3m42s
root@controlplane:~$ kubectl describe svc/webapp1-clusterip-targetport-svc
Error from server (NotFound): services "webapp1-clusterip-targetport-svc" not found
root@controlplane:~$
```

After the service and pods have deployed, it can be accessed via the cluster IP as before, but this time on the defined port 8080.

```
export CLUSTER_IP=$(kubectl get
services/webapp1-clusterip-targetport-svc -o
go-template='{{(index .spec.clusterIP)}}') echo
CLUSTER_IP=$CLUSTER_IP curl $CLUSTER_IP:8080
```

```
curl $CLUSTER_IP:8080
```

The application itself is still configured to listen on port 80. Kubernetes Service manages the translation between the two.

Step 3 - NodePort

While TargetPort and ClusterIP make it available to inside the cluster, the NodePort exposes the service on each Node’s IP via the defined static port. No matter which Node within the cluster is accessed, the service will be reachable based on the port number defined.

```
kubectl apply -f nodeport.yaml
```

When viewing the service definition, notice the additional type and NodePort property defined `cat nodeport.yaml`

```
root@controlplane:~$ cat nodeport.yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-nodeport-svc
  labels:
    app: webapp1-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30080
  selector:
    app: webapp1-nodeport
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp1-nodeport-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp1-nodeport
  template:
    metadata:
      labels:
        app: webapp1-nodeport
    spec:
      containers:
      - name: webapp1-nodeport-pod
        image: katacoda/docker-http-server:latest
        ports:
        - containerPort: 80
---
```

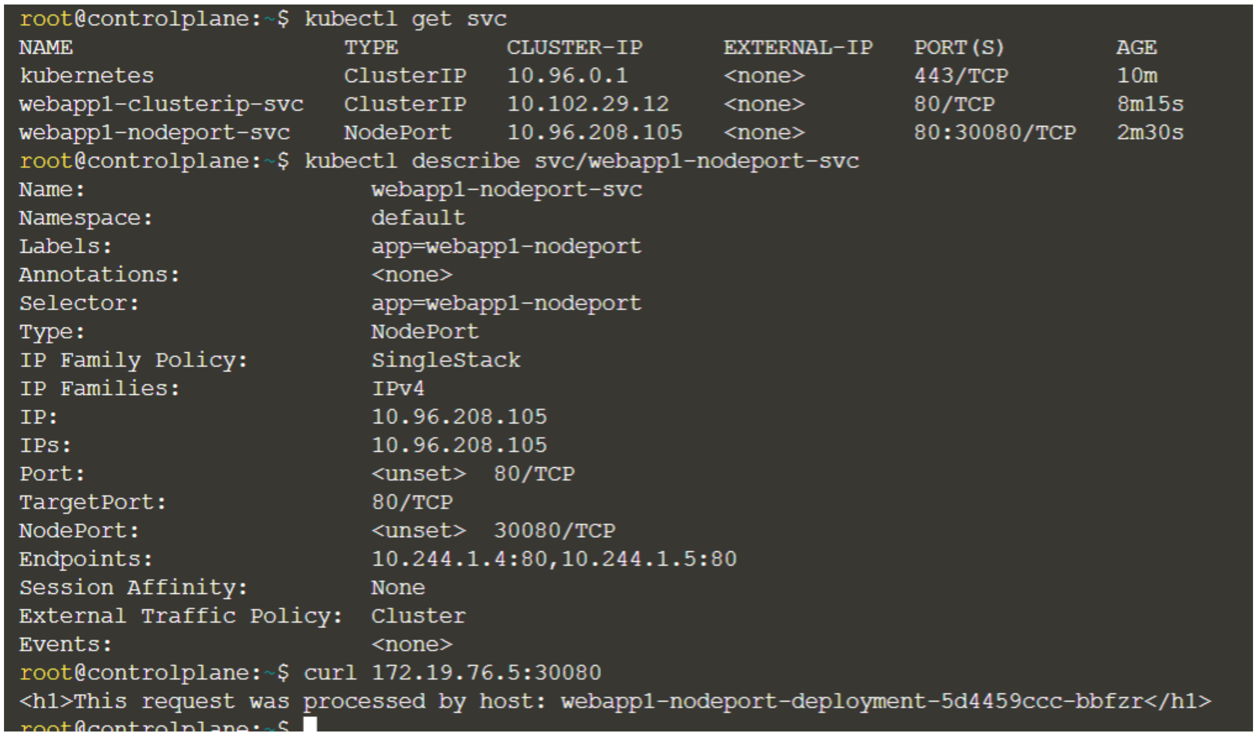
When viewing the service definition, notice the additional type and NodePort property defined `cat nodeport.yaml`

`kubectl get svc`

`kubectl describe svc/webapp1-nodeport-svc`

The service can now be reached via the Node's IP address on the NodePort defined.

`curl 172.19.76.5:30080`



```
root@controlplane:~$ kubectl get svc
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes              ClusterIP   10.96.0.1       <none>        443/TCP        10m
webapp1-clusterip-svc   ClusterIP   10.102.29.12    <none>        80/TCP         8m15s
webapp1-nodeport-svc    NodePort    10.96.208.105   <none>        80:30080/TCP   2m30s
root@controlplane:~$ kubectl describe svc/webapp1-nodeport-svc
Name:                     webapp1-nodeport-svc
Namespace:                default
Labels:                   app=webapp1-nodeport
Annotations:              <none>
Selector:                 app=webapp1-nodeport
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.96.208.105
IPs:                      10.96.208.105
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30080/TCP
Endpoints:                10.244.1.4:80,10.244.1.5:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
root@controlplane:~$ curl 172.19.76.5:30080
<h1>This request was processed by host: webapp1-nodeport-deployment-5d4459ccc-bbfzr</h1>
root@controlplane:~$
```

Step 4 - External IPs

Another approach to making a service available outside of the cluster is via External IP addresses.

Update the definition to the current cluster's IP address with `sed -i 's/HOSTIP/172.19.76.5/g' externalip.yaml`

```
cat externalip.yaml
```

```
root@controlplane:~$ cat externalip.yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-externalip-svc
  labels:
    app: webapp1-externalip
spec:
  ports:
  - port: 80
  externalIPs:
  - 172.19.76.5
  selector:
    app: webapp1-externalip
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp1-externalip-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp1-externalip
  template:
    metadata:
      labels:
        app: webapp1-externalip
    spec:
      containers:
      - name: webapp1-externalip-pod
        image: katacoda/docker-http-server:latest
        ports:
        - containerPort: 80
```

```
kubectl apply -f externalip.yaml
```

```
kubectl get svc
```

```
kubectl describe svc/webapp1-externalip-svc
```

The service is now bound to the IP address and Port 80 of the master node.

```
curl 172.19.76.5
```