# Lab: Exposing Applications using Services

## Introduction

**Service** is an abstract way to expose an application running on a set of Pods as a network service.

With Kubernetes you don't need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their **own IP addresses** and a **single DNS name** for a set of Pods, and can load-balance across them.

The idea of a Service is to group a **set of Pod endpoints into a single resource**. You can configure various ways to access the grouping. By **default**, you get a stable **cluster IP** address that clients inside the cluster can use to contact Pods in the Service. A client sends a request to the stable IP address, and the request is routed to one of the Pods in the Service.

There are four types of Services:

- ClusterIP (default)
- NodePort
- LoadBalancer
- ExternalName

In this Lab, you will learn below items:

**Objective:**

- Create **ClusterIP** (default) Service
- Create **NodePort** Service
- Create service with **ExternalIP**
- Clean up

## ClusterIP

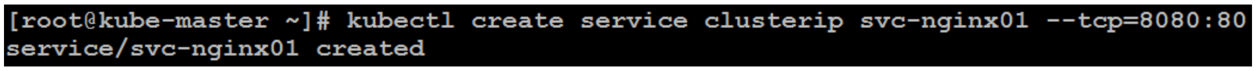
Note: Ensure you have running cluster deployed

1. Ensure that you have logged-in as **root** user with password as **linux** on **kube-master** node.

**1.1** Let us create a service using imperative method, type – **ClusterIP**.

```
# kubectl create service clusterip svc-nginx01 --tcp=8080:80
```

**Output**:

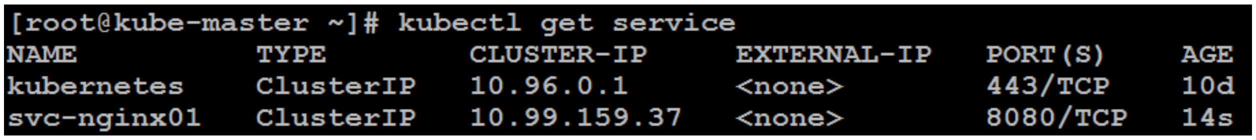


```
[root@kube-master ~]# kubectl create service clusterip svc-nginx01 --tcp=8080:80
service/svc-nginx01 created
```

**1.2** List the service that we just created.

```
# kubectl get service
```

**Output:**

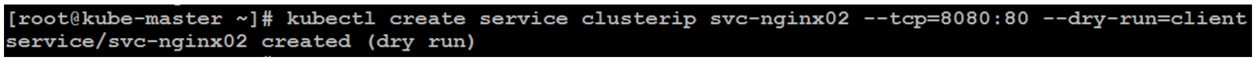


```
[root@kube-master ~]# kubectl get service
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP    10d
svc-nginx01   ClusterIP   10.99.159.37   <none>        8080/TCP   14s
```

**1.3** Let's run an alternate method for creating a service (**--dry-run** to verify the command syntax).

```
# kubectl create service clusterip svc-nginx02 --tcp=8080:80 -
-dry-run=client
```

**Output**:

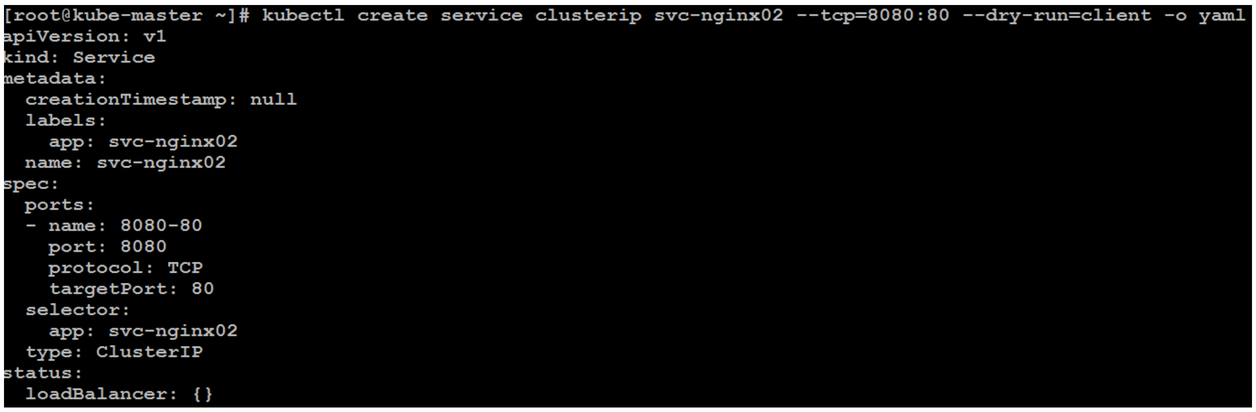


```
[root@kube-master ~]# kubectl create service clusterip svc-nginx02 --tcp=8080:80 --dry-run=client
service/svc-nginx02 created (dry run)
```

**1.4** Let's view the manifest in the **yaml** format.

```
# kubectl create service clusterip svc-nginx02 --tcp=8080:80 -
-dry-run=client -o yaml
```

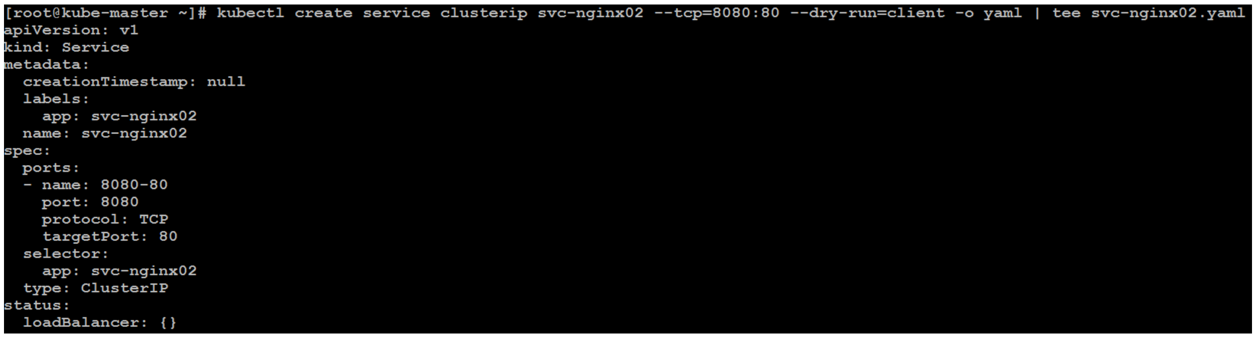
**Output**:


```
[root@kube-master ~]# kubectl create service clusterip svc-nginx02 --tcp=8080:80 --dry-run=client -o yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: svc-nginx02
  name: svc-nginx02
spec:
  ports:
  - name: 8080-80
    port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    app: svc-nginx02
  type: ClusterIP
status:
  loadBalancer: {}
```

**1.5** Let's save the manifest in the yaml format to the file **svc-nginx02.yaml**.

```
# kubectl create service clusterip svc-nginx02 --tcp=8080:80 -
-dry-run=client -o yaml | tee svc-nginx02.yaml
```

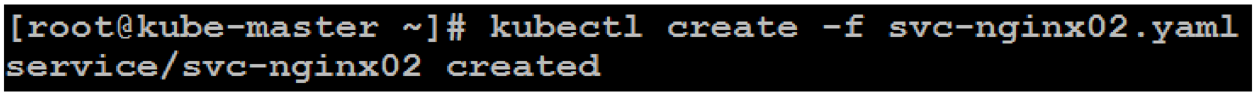
**Output**:


```
[root@kube-master ~]# kubectl create service clusterip svc-nginx02 --tcp=8080:80 --dry-run=client -o yaml | tee svc-nginx02.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: svc-nginx02
  name: svc-nginx02
spec:
  ports:
  - name: 8080-80
    port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    app: svc-nginx02
  type: ClusterIP
status:
  loadBalancer: {}
```

**1.6** Let's create a service using the yaml manifest **svc-nginx02.yaml** that we just created from the above step.

```
# kubectl create -f svc-nginx02.yaml
```

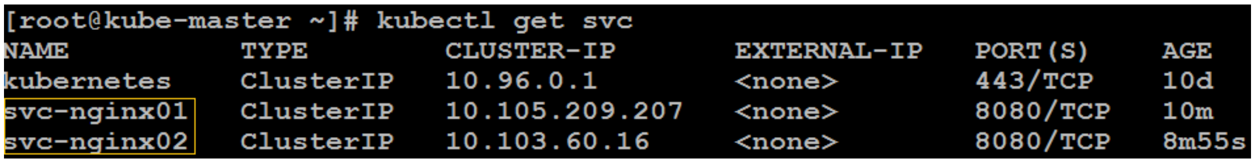
**Output**:


```
[root@kube-master ~]# kubectl create -f svc-nginx02.yaml
service/svc-nginx02 created
```

**1.7** List the services by running the below command.

```
# kubectl get svc
```

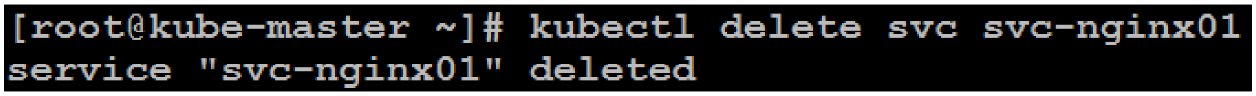
**Output**:


```
[root@kube-master ~]# kubectl get svc
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP    10d
svc-nginx01    ClusterIP   10.105.209.207   <none>        8080/TCP   10m
svc-nginx02    ClusterIP   10.103.60.16     <none>        8080/TCP   8m55s
```

**1.8** Let's delete both the services using different methods.

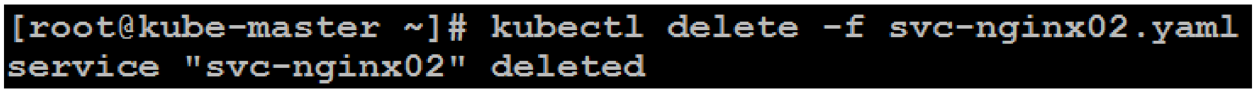
```
# kubectl delete svc svc-nginx01
```

**Output**:


```
[root@kube-master ~]# kubectl delete svc svc-nginx01
service "svc-nginx01" deleted
```

```
# kubectl delete -f svc-nginx02.yaml
```

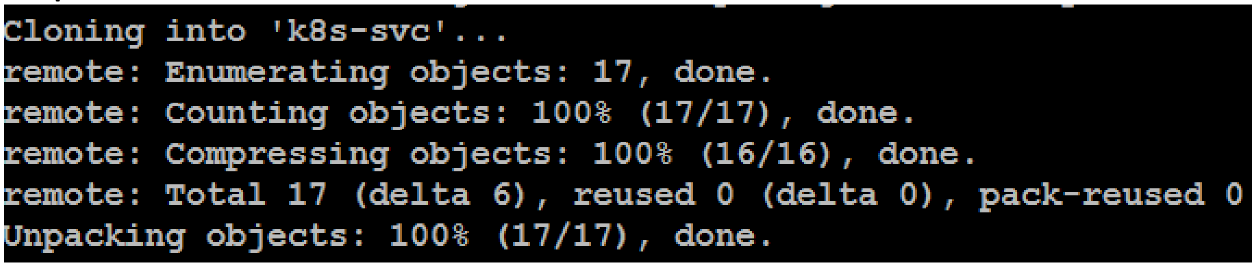
**Output**:


```
[root@kube-master ~]# kubectl delete -f svc-nginx02.yaml
service "svc-nginx02" deleted
```

**2** Let us clone the git repository which contains manifests required for this exercise, by executing the below command.

```
# git clone https://github.com/EyesOnCloud/k8s-svc.git
```

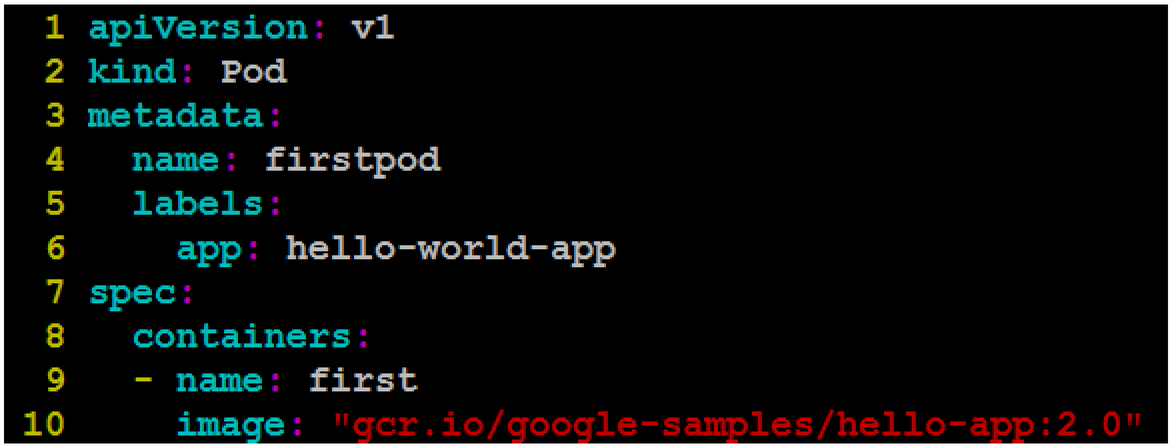
**Output:**


```
Cloning into 'k8s-svc'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 17 (delta 6), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (17/17), done.
```

**2.1** Let us view the manifest files of the pods.

```
# cat -n ~/k8s-svc/first-pod.yaml
```

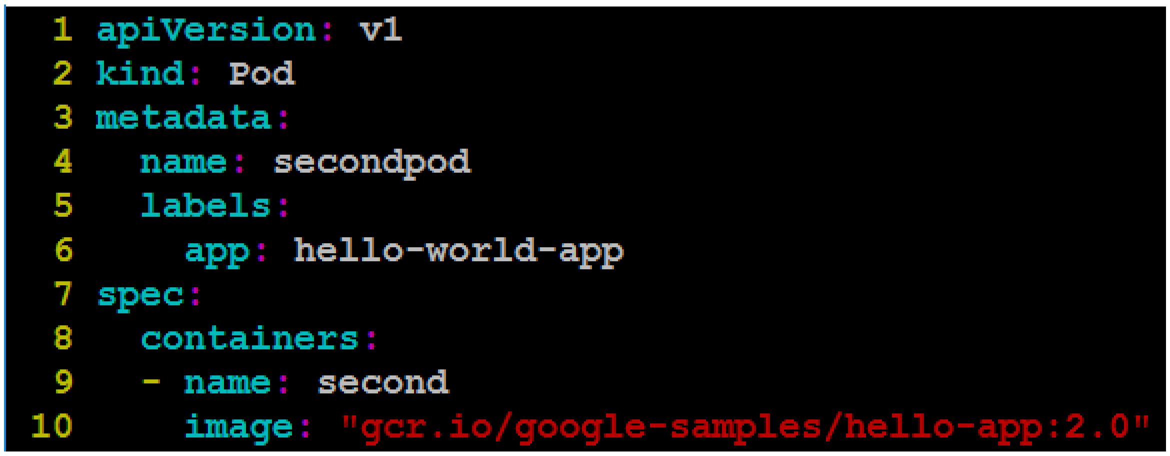
**Output:**



```
apiVersion: v1
kind: Pod
metadata:
  name: firstpod
  labels:
    app: hello-world-app
spec:
  containers:
  - name: first
    image: "gcr.io/google-samples/hello-app:2.0"
```

```
# cat -n ~/k8s-svc/second-pod.yaml
```

**Output**:

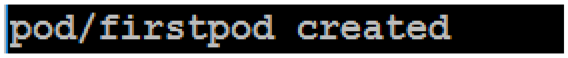


```
apiVersion: v1
kind: Pod
metadata:
  name: secondpod
  labels:
    app: hello-world-app
spec:
  containers:
  - name: second
    image: "gcr.io/google-samples/hello-app:2.0"
```

**2.2** Let us create two pods, named **firstpod** with **version 1** of **hello-app** image and **secondpod** with **version 2** of **hello-app.** Let us create both the pods, by executing below commands**:**

```
# kubectl create -f ~/k8s-svc/first-pod.yaml
```

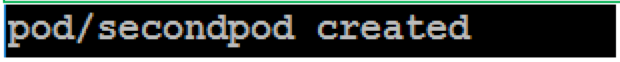
**Output:**



```
pod/firstpod created
```

**Output:**

```
# kubectl create -f ~/k8s-svc/second-pod.yaml
```

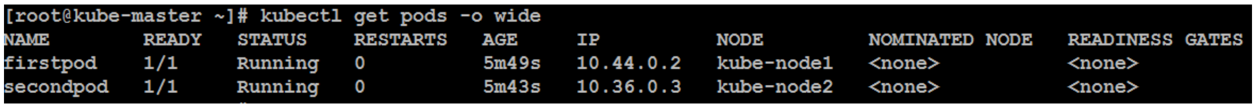


```
pod/secondpod created
```

**2.3** List your running Pods, by executing the below command.

```
# kubectl get pods -o wide
```

**Output:**

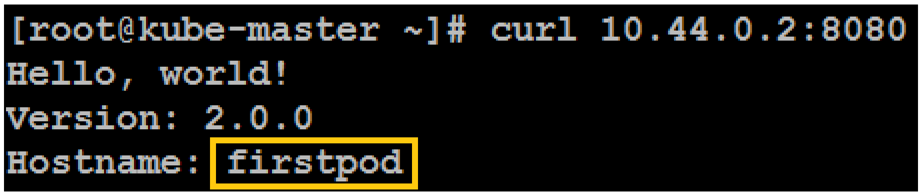


```
[root@kube-master ~]# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE    IP          NODE         NOMINATED NODE   READINESS GATES
firstpod    1/1     Running   0          5m49s  10.44.0.2   kube-node1   <none>           <none>
secondpod   1/1     Running   0          5m43s  10.36.0.3   kube-node2   <none>           <none>
```

**2.4** Let us access the pods using the pod-ip from the above output:

```
# curl 10.44.0.2:8080
```

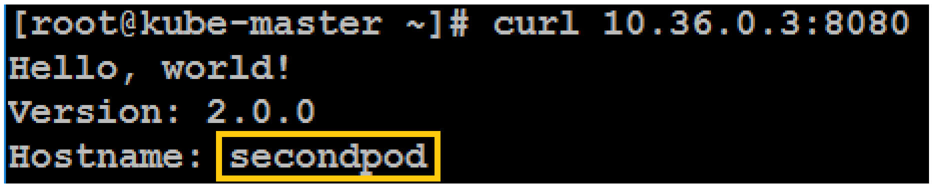
**Output:**



```
[root@kube-master ~]# curl 10.44.0.2:8080
Hello, world!
Version: 2.0.0
Hostname: firstpod
```

```
# curl 10.36.0.3:8080
```

**Output**:

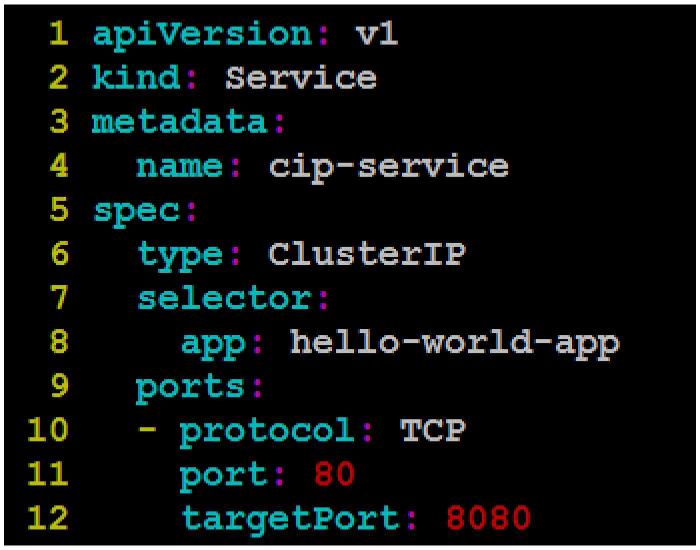


```
[root@kube-master ~]# curl 10.36.0.3:8080
Hello, world!
Version: 2.0.0
Hostname: secondpod
```

**2.5** Let us view the manifest for service, by executing the below command.

```
# cat -n ~/k8s-svc/clusterip-svc.yaml
```

**Output:**

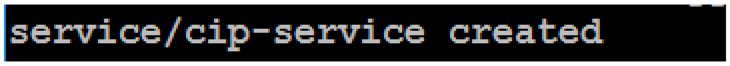


```
apiVersion: v1
kind: Service
metadata:
  name: cip-service
spec:
  type: ClusterIP
  selector:
    app: hello-world-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

**2.6** Let us create the Service of type **ClusterIP** by using the clusterip-svc.yaml file.

```
# kubectl apply -f ~/k8s-svc/clusterip-svc.yaml
```

**Output:**

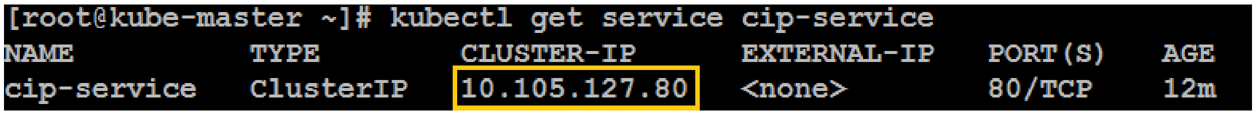


```
service/cip-service created
```

**2.7** Let us list the service and capture the service ip, by executing the below command

```
# kubectl get service cip-service
```

**Output:**



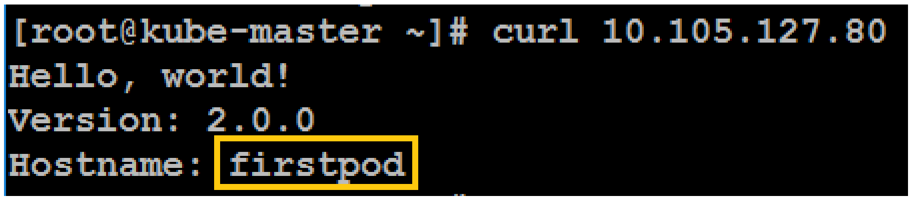
```
[root@kube-master ~]# kubectl get service cip-service
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
cip-service   ClusterIP   10.105.127.80   <none>        80/TCP    12m
```

Note: Make a note of your ClusterIP value for later.

**2.8** Let us access using the Cluster-ip

```
# curl 10.105.127.80
```

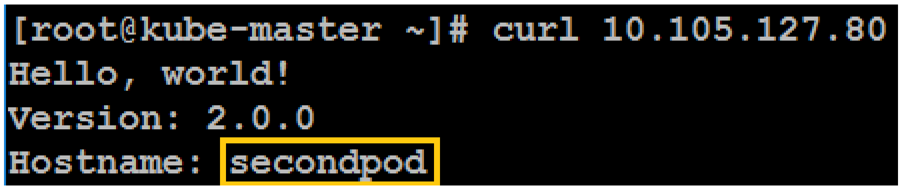
**Output:**



```
[root@kube-master ~]# curl 10.105.127.80
Hello, world!
Version: 2.0.0
Hostname: firstpod
```

```
# curl 10.105.127.80
```

**Output:**



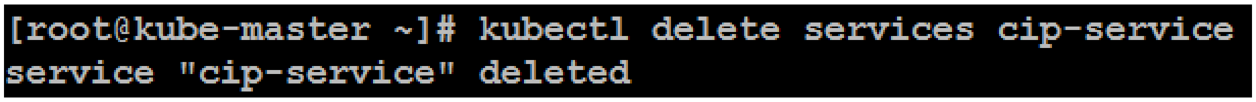
```
[root@kube-master ~]# curl 10.105.127.80
Hello, world!
Version: 2.0.0
Hostname: secondpod
```

Your request is forwarded to one of the member Pods on TCP port **8080**, which is the value of the **targetPort** field. Note that each of the Service's member Pods must have a container listening on port 8080.

**2.9** Let us clean up the service:

```
# kubectl delete services cip-service
```

**Output:**



```
[root@kube-master ~]# kubectl delete services cip-service
service "cip-service" deleted
```

## Nodeport

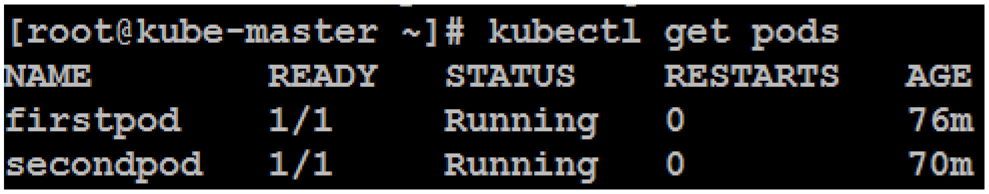
Exposes the Service on each Node's IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service routes, is automatically created. You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

NodePorts are in the **30000-32767** range by default, which means a NodePort is unlikely to match a service's intended port (for example, 8080 may be exposed as 31020).

**3** We will use the pods created in earlier steps; let us verify the status of the pods, by executing the below command

```
# kubectl get pods
```

**Output:**

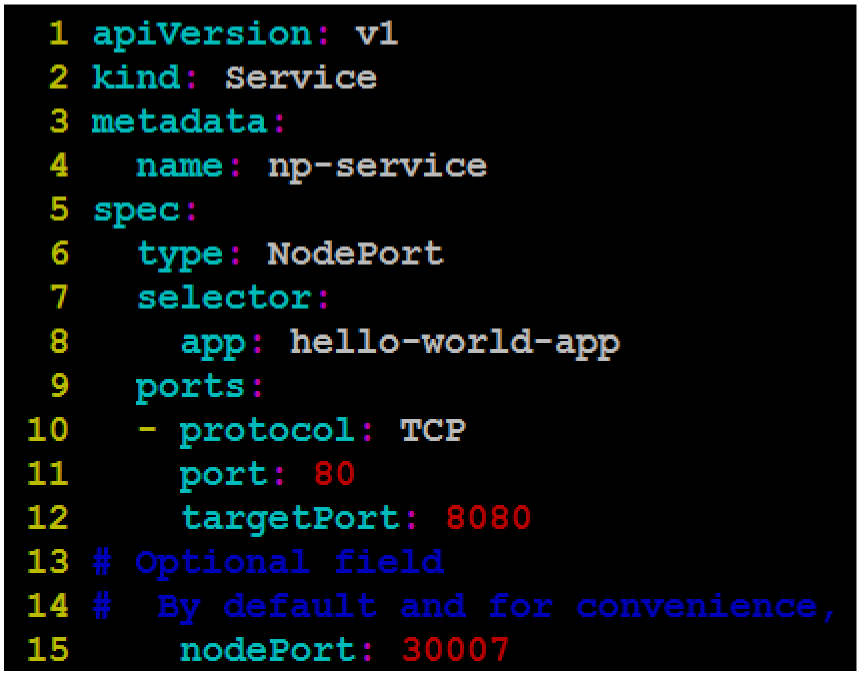


```
[root@kube-master ~]# kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
firstpod     1/1     Running   0          76m
secondpod    1/1     Running   0          70m
```

**3.1** Let us view the manifest.

```
# cat -n ~/k8s-svc/nodeport-svc.yaml
```

**Output:**

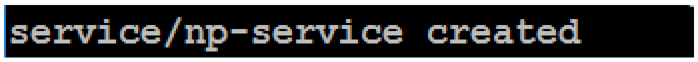


```
apiVersion: v1
kind: Service
metadata:
  name: np-service
spec:
  type: NodePort
  selector:
    app: hello-world-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
# Optional field
#  By default and for convenience,
    nodePort: 30007
```

**3.2** Let us create a Service of type **NodePort,** by executing the below command.

```
# kubectl apply -f ~/k8s-svc/nodeport-svc.yaml
```

**Output:**

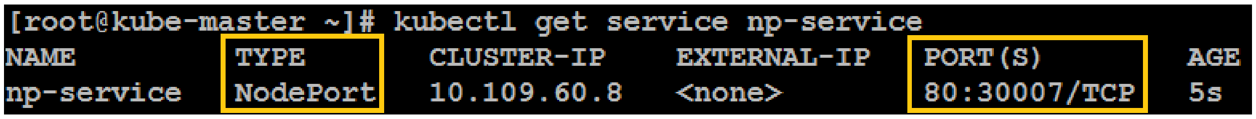


```
service/np-service created
```

**3.3** View the Service:

```
# kubectl get service np-service
```

**Output:**

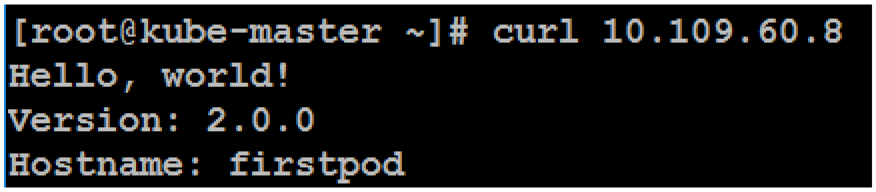


```
[root@kube-master ~]# kubectl get service np-service
NAME         TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
np-service   NodePort   10.109.60.8   <none>        80:30007/TCP   5s
```

**3.4** Let us access the pods via cluster-ip

```
# curl 10.108.156.140
```

**Output:**

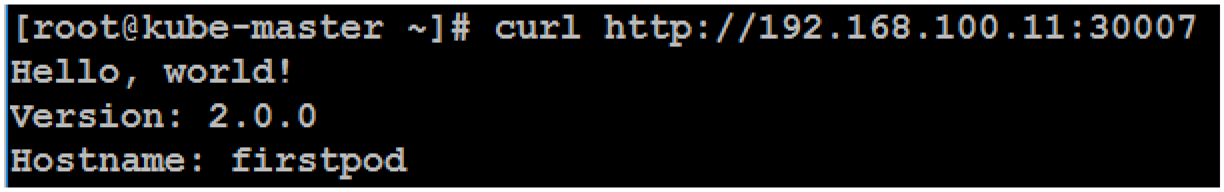


```
[root@kube-master ~]# curl 10.109.60.8
Hello, world!
Version: 2.0.0
Hostname: firstpod
```

**3.5** Let us use the node address and node port to access the application

```
# curl http://192.168.100.11:30007
```

**Output:**

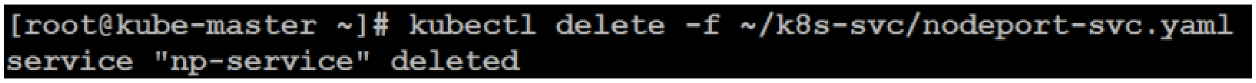


```
[root@kube-master ~]# curl http://192.168.100.11:30007
Hello, world!
Version: 2.0.0
Hostname: firstpod
```

**3.6** Let us clean up the service:

```
# kubectl delete -f ~/k8s-svc/nodeport-svc.yaml
```

**Output:**



```
[root@kube-master ~]# kubectl delete -f ~/k8s-svc/nodeport-svc.yaml
service "np-service" deleted
```

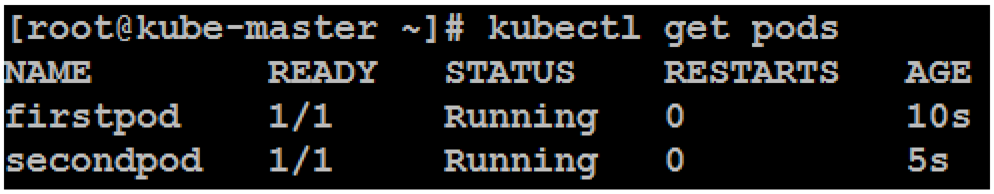
## External IPs

If there are external IPs that route to one or more cluster nodes, Kubernetes Services can be exposed on those externalIPs. Traffic that ingresses into the cluster with the external IP (as destination IP), on the Service port, will be routed to one of the Service endpoints.

**4** We will use the pods created in earlier steps; let us verify the status of the pods, by executing the below command

```
# kubectl get pods
```

**Output:**

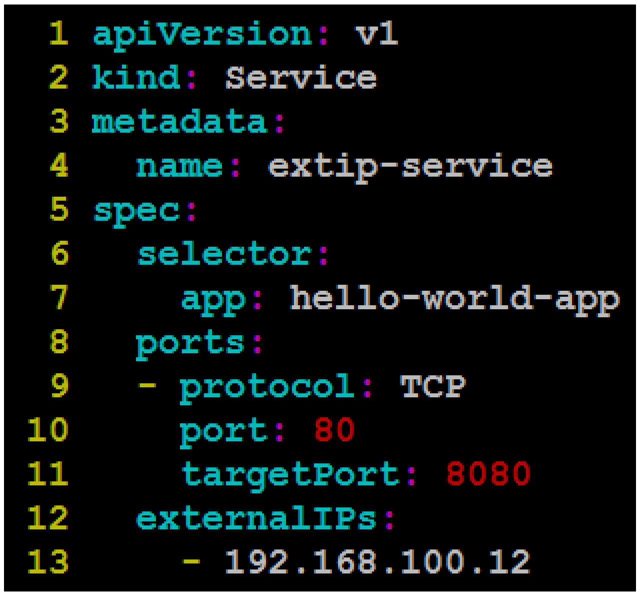


```
[root@kube-master ~]# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
firstpod      1/1     Running   0          10s
secondpod     1/1     Running   0          5s
```

**4.1** Let us view the manifest

```
# cat -n ~/k8s-svc/externalip-svc.yaml
```

**Output:**

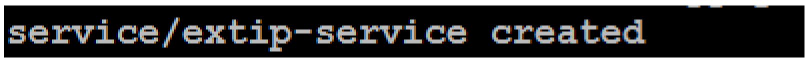


```
apiVersion: v1
kind: Service
metadata:
  name: extip-service
spec:
  selector:
    app: hello-world-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  externalIPs:
    - 192.168.100.12
```

**4.2** Let us create a Service of type **ExternalIPs,** by executing the below command.

```
# kubectl apply -f ~/k8s-svc/externalip-svc.yaml
```

**Output:**

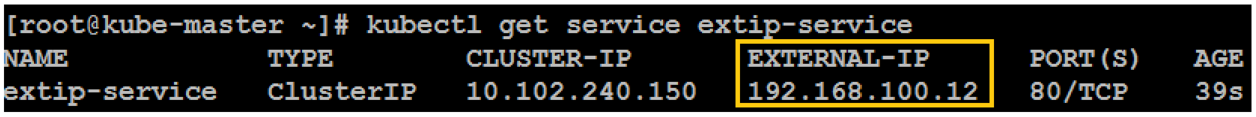


```
service/extip-service created
```

**4.3** Let us view the Service, by executing the below command.

```
# kubectl get service extip-service
```

**Output:**

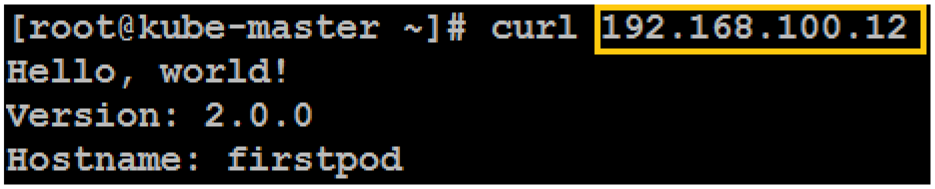


```
[root@kube-master ~]# kubectl get service extip-service
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP      PORT(S)   AGE
extip-service   ClusterIP   10.102.240.150   192.168.100.12   80/TCP    39s
```

**4.4** Let us access the pods via external-ip.

```
# curl 192.168.100.12
```

**Output:**

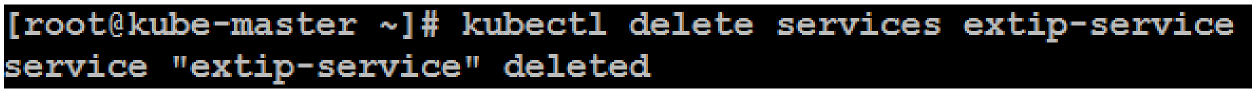


```
[root@kube-master ~]# curl 192.168.100.12
Hello, world!
Version: 2.0.0
Hostname: firstpod
```

**4.5** Let us clean up the service, by executing the below command.

```
# kubectl delete services extip-service
```

**Output:**

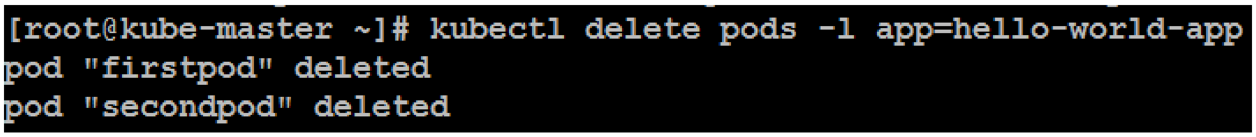


```
[root@kube-master ~]# kubectl delete services extip-service
service "extip-service" deleted
```

**4.6** Let us clean up pods, by executing the below command.

```
# kubectl delete pods -l app=hello-world-app
```

**Output:**



```
[root@kube-master ~]# kubectl delete pods -l app=hello-world-app
pod "firstpod" deleted
pod "secondpod" deleted
```