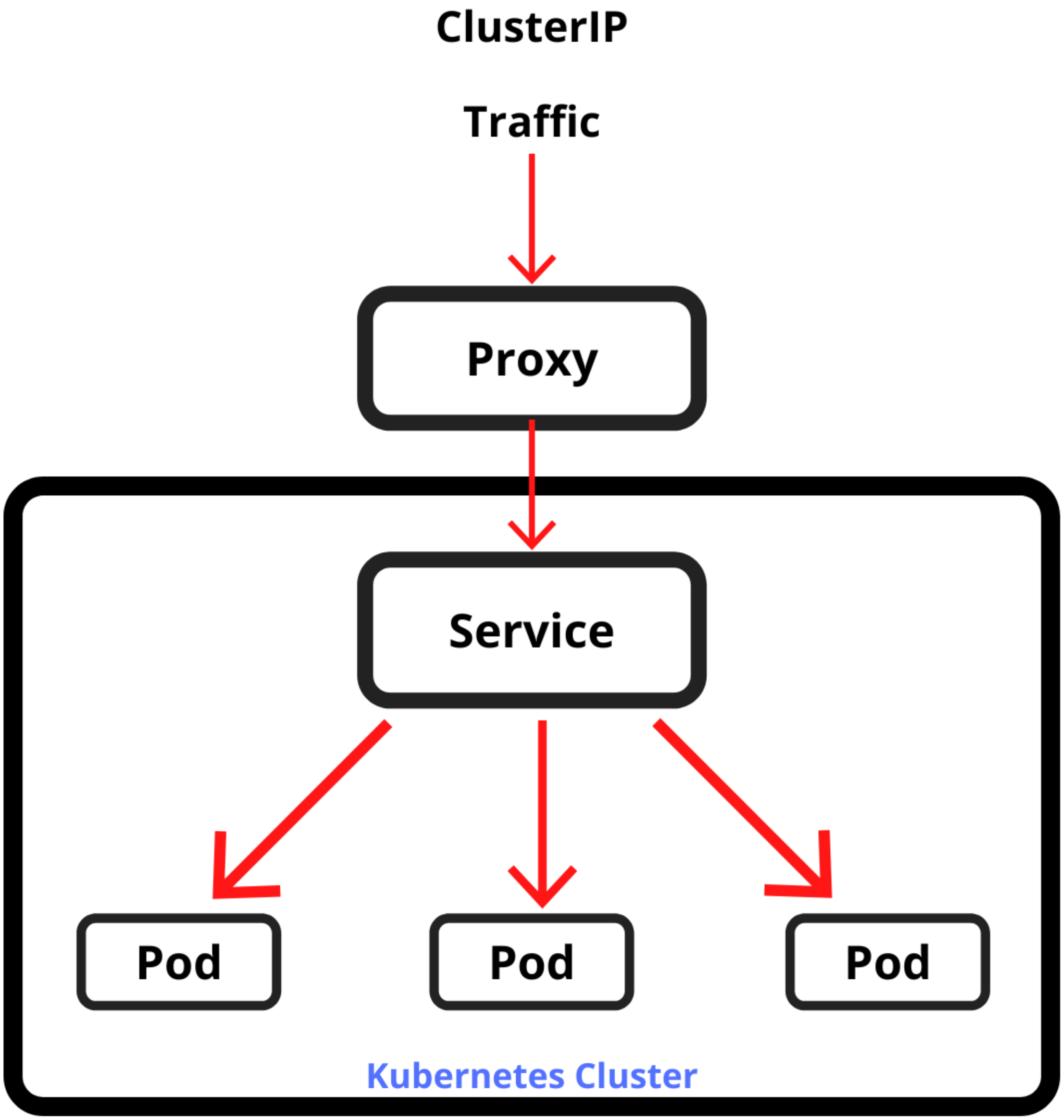
Lab: Exposing Applications using ClusterIP Services

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.



Objective:

- **Create ClusterIP Based Service**
- **Easy way to create Service**
- **Cleanup**

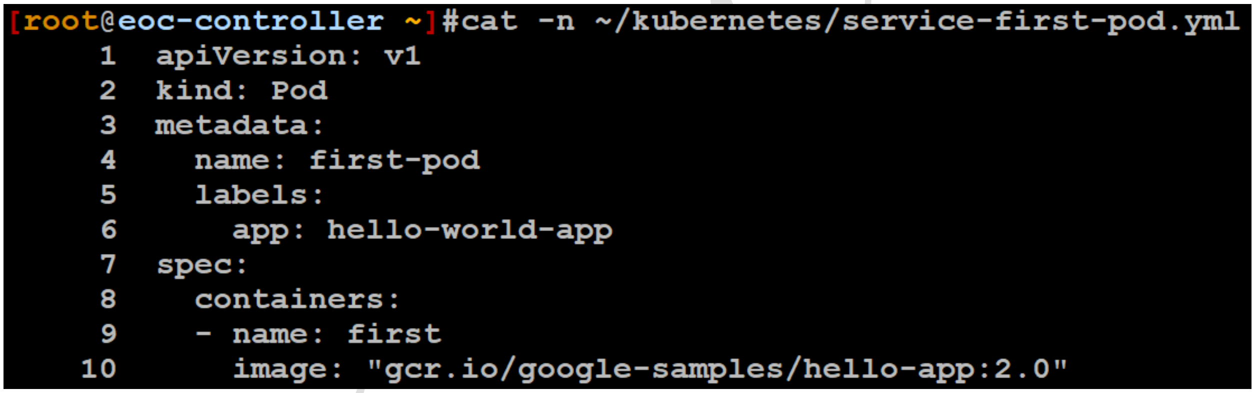
Ensure that you have logged-in as **root** user on **eoc-controller** node.

1 Create ClusterIP Based Service

1.1 Let’s **view** the yaml manifest file of the **first pod**.

```
# cat -n ~/kubernetes/service-first-pod.yml
```

Output:

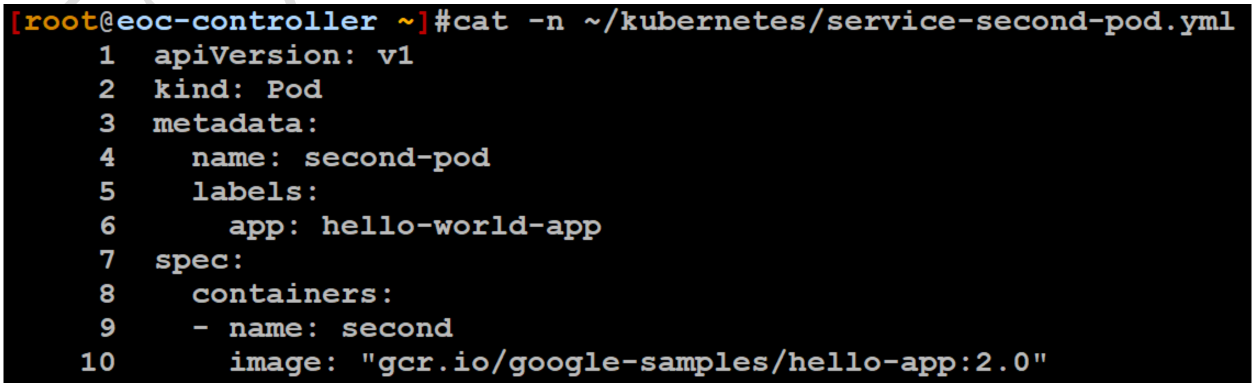


```
[root@eoc-controller ~]#cat -n ~/kubernetes/service-first-pod.yml
     1  apiVersion: v1
     2  kind: Pod
     3  metadata:
     4    name: first-pod
     5    labels:
     6      app: hello-world-app
     7  spec:
     8    containers:
     9    - name: first
    10      image: "gcr.io/google-samples/hello-app:2.0"
```

1.1 Let’s **view** the yaml manifest file of the **second pod.**

```
# cat -n ~/kubernetes/service-second-pod.yml
```

Output:

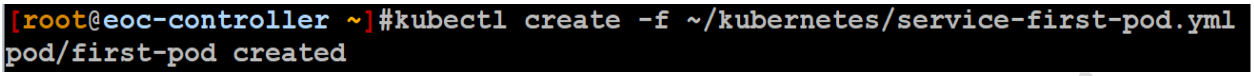


```
[root@eoc-controller ~]#cat -n ~/kubernetes/service-second-pod.yml
     1  apiVersion: v1
     2  kind: Pod
     3  metadata:
     4    name: second-pod
     5    labels:
     6      app: hello-world-app
     7  spec:
     8    containers:
     9    - name: second
    10      image: "gcr.io/google-samples/hello-app:2.0"
```

1.2 Let's create **two pods** named **firstpod** and **secondpod** with **hello-app** image**.** Let us create both the pods, by executing below commands.

```
# kubectl create -f ~/kubernetes/service-first-pod.yml
```

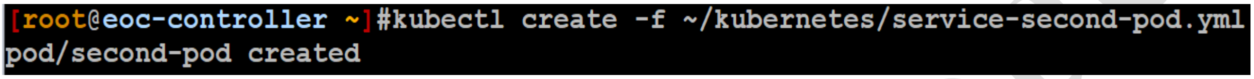
Output:



```
[root@eoc-controller ~]#kubectl create -f ~/kubernetes/service-first-pod.yml
pod/first-pod created
```

```
# kubectl create -f ~/kubernetes/service-second-pod.yml
```

Output:

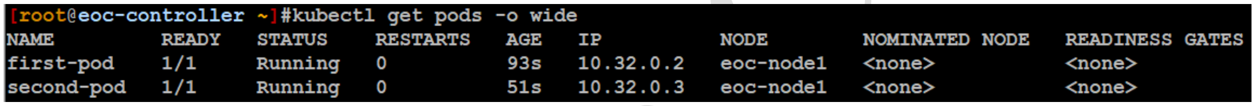


```
[root@eoc-controller ~]#kubectl create -f ~/kubernetes/service-second-pod.yml
pod/second-pod created
```

1.3 Let's **list** running Pods by executing the below command.

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

Output:



```
[root@eoc-controller ~]#kubectl get pods -o wide
NAME         READY   STATUS    RESTARTS   AGE   IP          NODE        NOMINATED NODE   READINESS GATES
first-pod    1/1     Running   0          93s   10.32.0.2   eoc-node1   <none>           <none>
second-pod   1/1     Running   0          51s   10.32.0.3   eoc-node1   <none>           <none>
```

1.4 Let's **capture** pod-ip as variable by executing below command

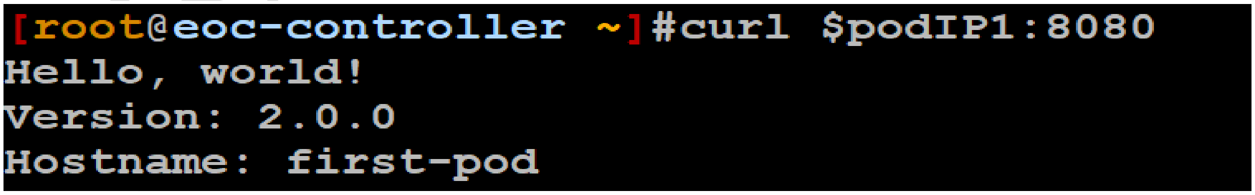
```
# podIP1=$(kubectl get pod first-pod -o
jsonpath='{.status.podIP}')
```

```
# echo $podIP1
```

1.5 Let's **access** the pods using the pod-ip from the above output.

```
# curl $podIP1:8080
```

Output:



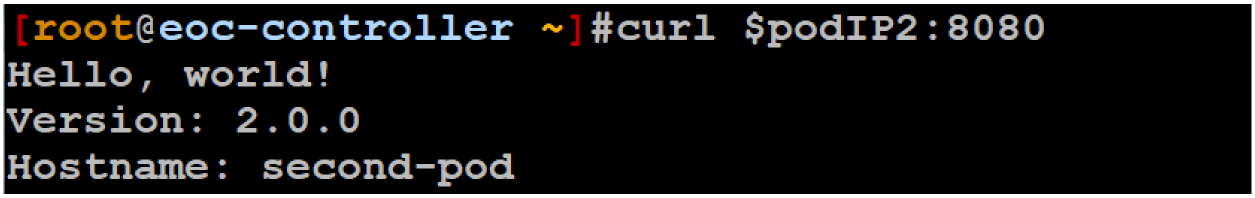
```
[root@eoc-controller ~]#curl $podIP1:8080
Hello, world!
Version: 2.0.0
Hostname: first-pod
```

```
# podIP2=$(kubectl get pod second-pod -o
jsonpath='{.status.podIP}')
```

```
# echo $podIP2
```

```
# curl $podIP2:8080
```

Output:

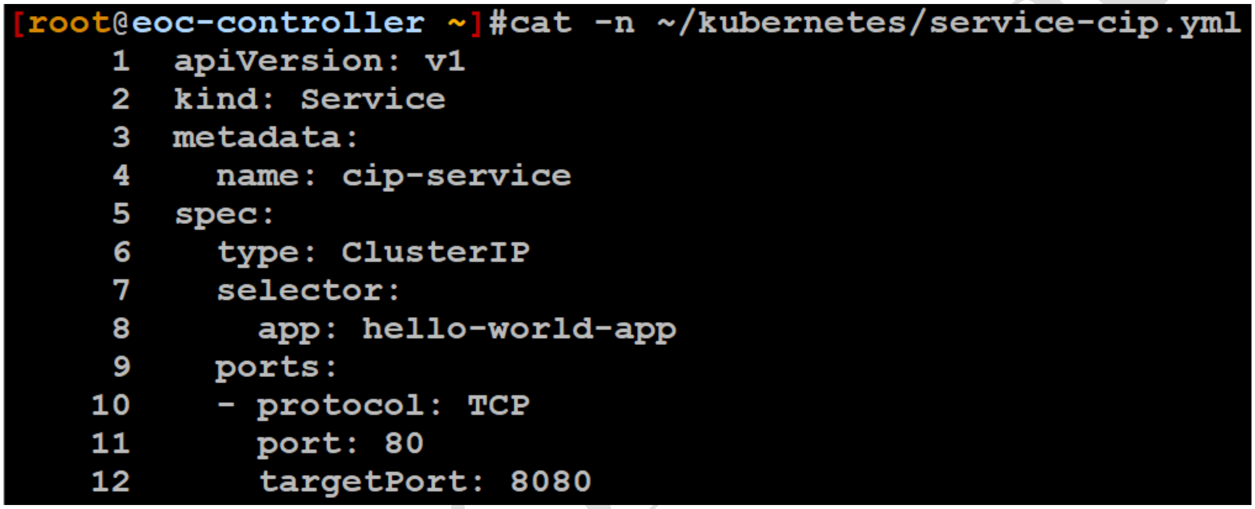


```
[root@eoc-controller ~]#curl $podIP2:8080
Hello, world!
Version: 2.0.0
Hostname: second-pod
```

1.6 Let’s **view** the yaml manifest file for **service** by executing the below command.

```
# cat -n ~/kubernetes/service-cip.yml
```

Output:

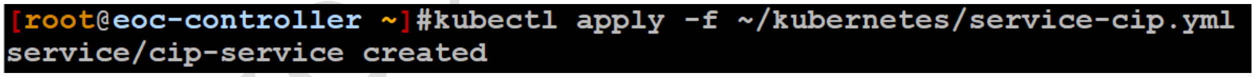


```
[root@eoc-controller ~]#cat -n ~/kubernetes/service-cip.yml
     1  apiVersion: v1
     2  kind: Service
     3  metadata:
     4    name: cip-service
     5  spec:
     6    type: ClusterIP
     7    selector:
     8      app: hello-world-app
     9    ports:
    10    - protocol: TCP
    11      port: 80
    12      targetPort: 8080
```

1.7 Let’s **create** the Service of type **ClusterIP** by using the **service-cip.yml** file.

```
# kubectl apply -f ~/kubernetes/service-cip.yml
```

Output:

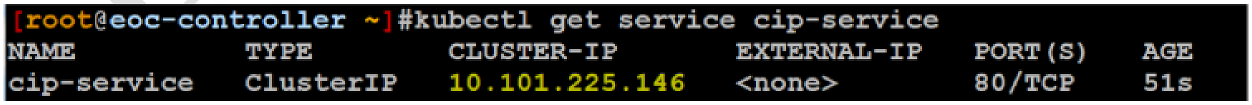


```
[root@eoc-controller ~]#kubectl apply -f ~/kubernetes/service-cip.yml
service/cip-service created
```

1.8 Let’s **list** the service and capture the service ip by executing the below command.

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

Output:



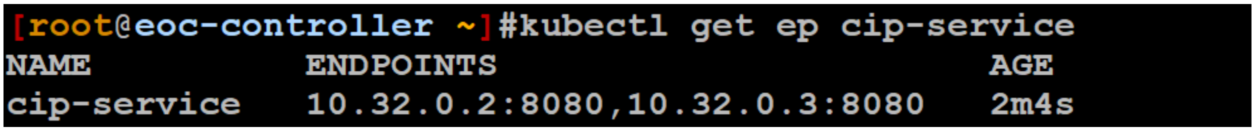
```
[root@eoc-controller ~]#kubectl get service cip-service
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
cip-service   ClusterIP   10.101.225.146   <none>        80/TCP    51s
```

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

1.9 Let’s **list** the endpoints for the above created service

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

Output:



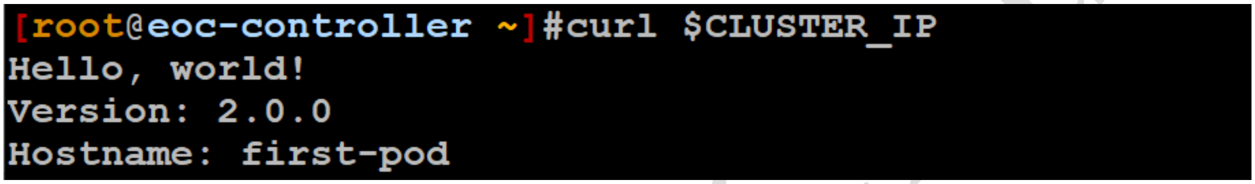
```
[root@eoc-controller ~]#kubectl get ep cip-service
NAME           ENDPOINTS                       AGE
cip-service    10.32.0.2:8080,10.32.0.3:8080   2m4s
```

1.10 Let's **access** by using the **Cluster-ip**.

```
# CLUSTER_IP=$(kubectl get svc cip-service -o
jsonpath='{.spec.clusterIP}')
```

```
# curl $CLUSTER_IP
```

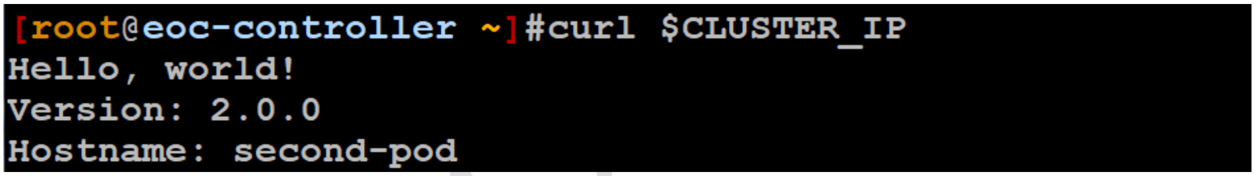
Output:



```
[root@eoc-controller ~]#curl $CLUSTER_IP
Hello, world!
Version: 2.0.0
Hostname: first-pod
```

```
# curl $CLUSTER_IP
```

Output:



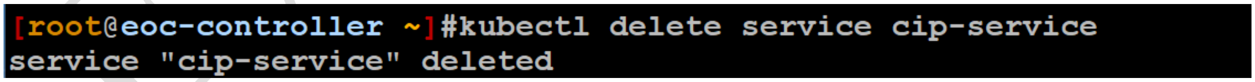
```
[root@eoc-controller ~]#curl $CLUSTER_IP
Hello, world!
Version: 2.0.0
Hostname: second-pod
```

Info: 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.

1.11 Let's **cleanup** the service by execute the below command

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

Output:

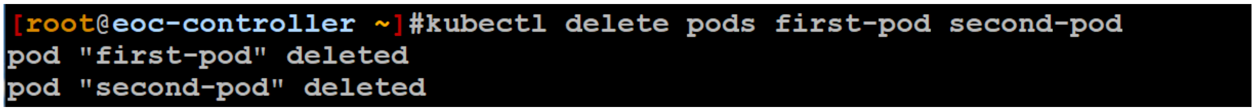


```
[root@eoc-controller ~]#kubectl delete service cip-service
service "cip-service" deleted
```

1.12 Let's **delete** the pods by executing below command.

```
# kubectl delete pods first-pod second-pod
```

Output:



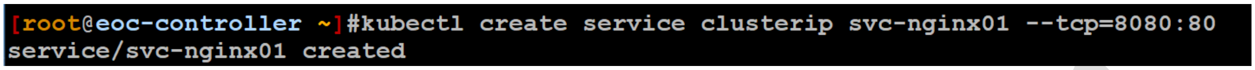
```
[root@eoc-controller ~]#kubectl delete pods first-pod second-pod
pod "first-pod" deleted
pod "second-pod" deleted
```

2 Let's learn easy way to **create** a service.

2.1 Let's **create** a service using imperative method type – **ClusterIP**.

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

Output:

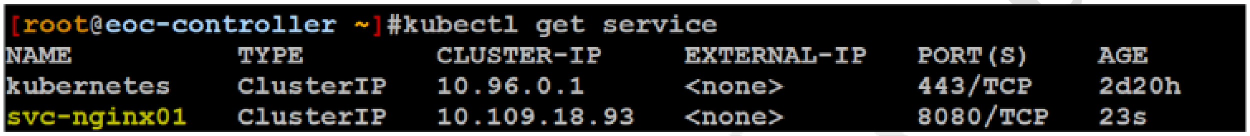


```
[root@eoc-controller ~]#kubectl create service clusterip svc-nginx01 --tcp=8080:80
service/svc-nginx01 created
```

2.2 Let's **list** the service that we just created.

```
# kubectl get service
```

Output:

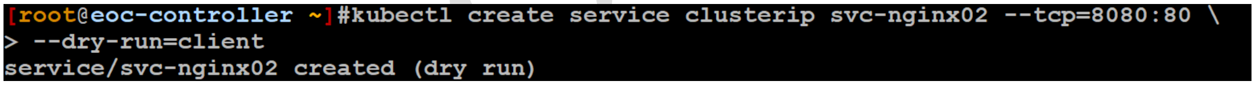


```
[root@eoc-controller ~]#kubectl get service
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP    2d20h
svc-nginx01   ClusterIP   10.109.18.93   <none>        8080/TCP   23s
```

2.3 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:

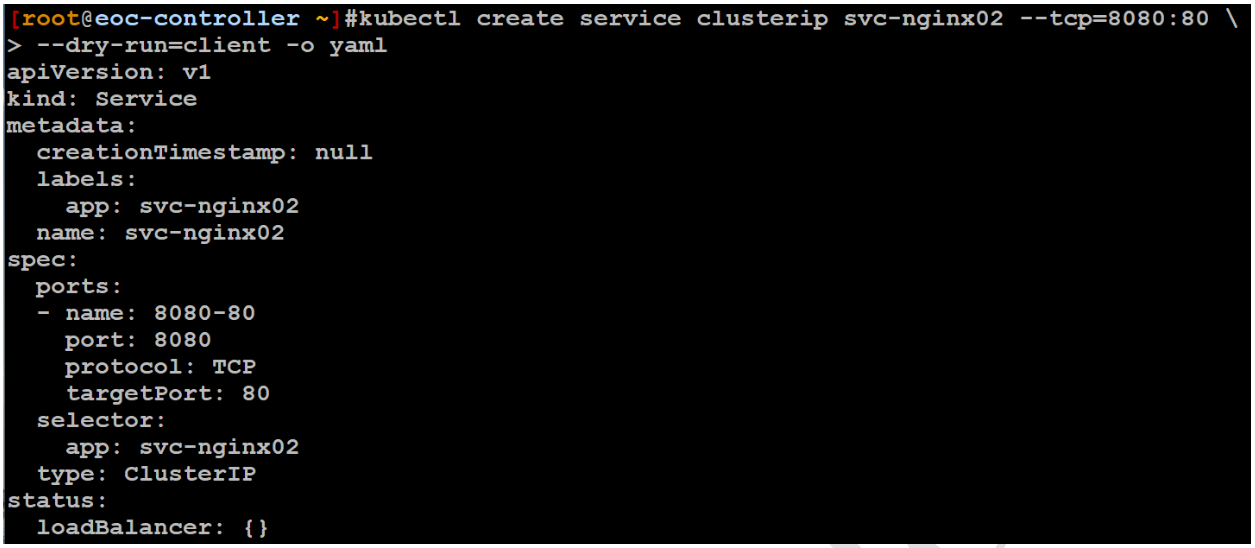


```
[root@eoc-controller ~]#kubectl create service clusterip svc-nginx02 --tcp=8080:80 \
> --dry-run=client
service/svc-nginx02 created (dry run)
```

2.4 Let's **view** the manifest in the **yml** format.

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

Output:



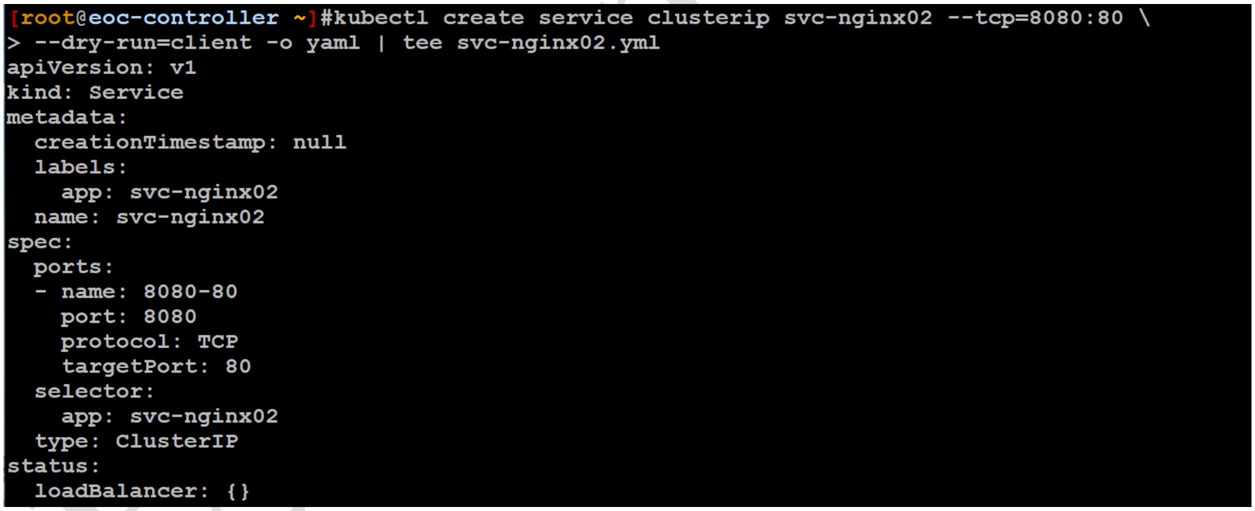
```
[root@eoc-controller ~]#kubectl create service clusterip svc-nginx02 --tcp=8080:80 \
> --dry-run=client -o yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: svc-nginx02
  name: svc-nginx02
spec:
  ports:
  - name: 8080-80
    port: 8080
    protocol: TCP
  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

2.5 Let's **save** the manifest in the yml format to the file **svc-nginx02.yml**.

```
# kubectl create service clusterip svc-nginx02 --tcp=8080:80 \

--dry-run=client -o yaml | tee svc-nginx02.yml
```

Output:

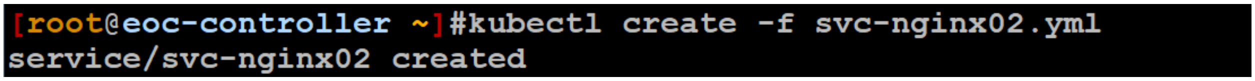


```
[root@eoc-controller ~]#kubectl create service clusterip svc-nginx02 --tcp=8080:80 \
> --dry-run=client -o yaml | tee svc-nginx02.yml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: svc-nginx02
  name: svc-nginx02
spec:
  ports:
  - name: 8080-80
    port: 8080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

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

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

Output:

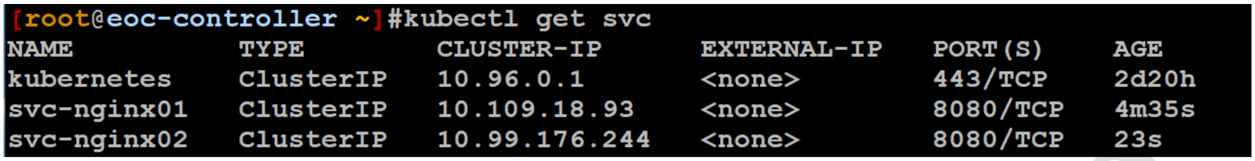


```
[root@eoc-controller ~]#kubectl create -f svc-nginx02.yml
service/svc-nginx02 created
```

2.7 Let's **list** the services by running the below command.

```
# kubectl get svc
```

Output:



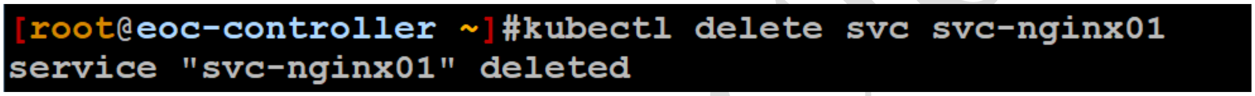
```
[root@eoc-controller ~]#kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP    2d20h
svc-nginx01   ClusterIP   10.109.18.93    <none>        8080/TCP   4m35s
svc-nginx02   ClusterIP   10.99.176.244   <none>        8080/TCP   23s
```

3 Cleanup.

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

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

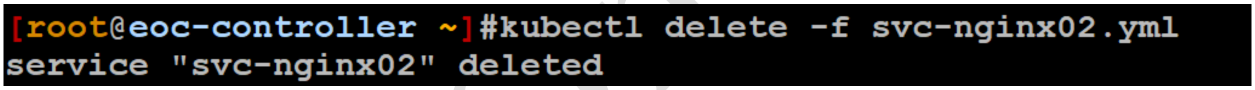
Output:



```
[root@eoc-controller ~]#kubectl delete svc svc-nginx01
service "svc-nginx01" deleted
```

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

Output:



```
[root@eoc-controller ~]#kubectl delete -f svc-nginx02.yml
service "svc-nginx02" deleted
```