SERVICES

About Services:-A grouping of pod end points into a single endpoint is called as services. (pod endpoints are defined as pod ipaddress) it will be used as load balancer. pod can be down at any time and we cannot assure the pod will come back with same ip address and it will be a dynamic assign, so to connect pod consistently, the services will create static endpoints. so, in backend ephemeral pods and endpoints will be combined and give as static endpoint by services, so static point will remain a constant until we remove, however backend whatever pods present that can be deleted or down at any time, that will be taken care by services. With Selectors and labels, pods will be grouping by services. we never bother about backend pods, we will be connecting with services, it will take care of all pods in backend.
There are five types of Services (ClusterIP (default), NodePort, LoadBalancer, ExternalName, Headless).
Ingress is top of services, so in real-time ingress will be used to expose outside.

Actual readme:-

- What is a Service?

The idea of a Service is to group a set of Pod endpoints into a single resource

- Why to use a Service?

In a Kubernetes cluster, each Pod has an internal IP address. But the Pods in a Deployment come and go, and their IP addresses change. So, it doesn't make sense to use Pod IP addresses directly. With a Service, you get a stable IP address that lasts for the life of the Service, even as the IP addresses of the member Pods change.
A Service also provides load balancing. Clients call a single, stable IP address, and their requests are balanced across the Pods that are members of the Service.

- How Pods are Connected to Service?

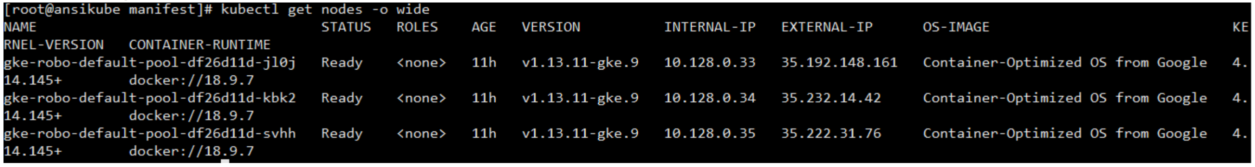
A Service identifies its member Pods with a selector. For a Pod to be a member of the Service, the Pod must have all of the labels specified in the selector. A label is an arbitrary key/value pair that is attached to an object.
Types of Services
There are five types of Services:

1) ClusterIP (default): Internal clients send requests to a stable internal IP address.
2) NodePort: Clients send requests to the IP address of a node on one or more nodePort values that are specified by the Service.
3) LoadBalancer: Clients send requests to the IP address of a network load balancer.
4) ExternalName: Internal clients use the DNS name of a Service as an alias for an external DNS name.
5) Headless: You can use a headless service in situations where you want a Pod grouping, but don't need a stable IP address.

The NodePort type is an extension of the ClusterIP type. So, a Service of type NodePort has a cluster IP address.
The LoadBalancer type is an extension of the NodePort type. So, a Service of type LoadBalancer has a cluster IP address and one or more nodePort values

##ClusterIP (default): Internal clients send request to a stable internal IP address.##
This will be used for internal pod communication only, kubernetes private ip communicate with cluster service, so it will use for internal communication, to communicate internal resources and not to external.

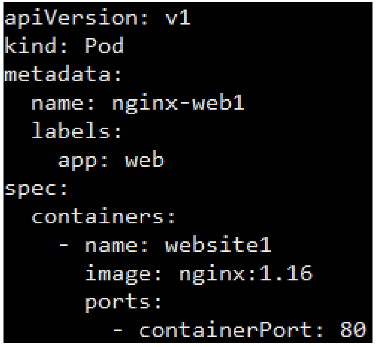
kubectl get nodes -o wide



```
[root@ansikube manifest]# kubectl get nodes -o wide
NAME                                  STATUS   ROLES    AGE   VERSION           INTERNAL-IP   EXTERNAL-IP      OS-IMAGE                             KE
RNEL-VERSION   CONTAINER-RUNTIME
gke-robo-default-pool-df26d11d-jl0j   Ready    <none>   11h   v1.13.11-gke.9    10.128.0.33   35.192.148.161   Container-Optimized OS from Google   4.
14.145+        docker://18.9.7
gke-robo-default-pool-df26d11d-kbk2   Ready    <none>   11h   v1.13.11-gke.9    10.128.0.34   35.232.14.42     Container-Optimized OS from Google   4.
14.145+        docker://18.9.7
gke-robo-default-pool-df26d11d-svhh   Ready    <none>   11h   v1.13.11-gke.9    10.128.0.35   35.222.31.76     Container-Optimized OS from Google   4.
14.145+        docker://18.9.7
```

@Create a pod.

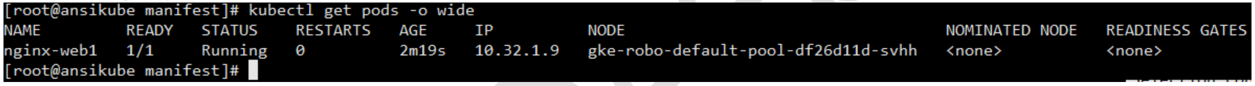
#cat pods.yml



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-web1
  labels:
    app: web
spec:
  containers:
    - name: website1
      image: nginx:1.16
      ports:
        - containerPort: 80
```

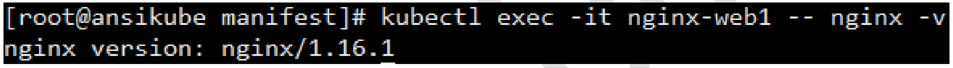
# kubectl apply -f pods.yml

kubectl get pods -o wide



```
[root@ansikube manifest]# kubectl get pods -o wide
NAME         READY   STATUS    RESTARTS   AGE     IP          NODE                                  NOMINATED NODE   READINESS GATES
nginx-web1   1/1     Running   0          2m19s   10.32.1.9   gke-robo-default-pool-df26d11d-svhh   <none>           <none>
[root@ansikube manifest]# 
```

#kubectl exec -it nginx-web1 -- nginx -v



```
[root@ansikube manifest]# kubectl exec -it nginx-web1 -- nginx -v
nginx version: nginx/1.16.1
```

@@To check nginx http access between the containers, we are going to run ubuntu image.

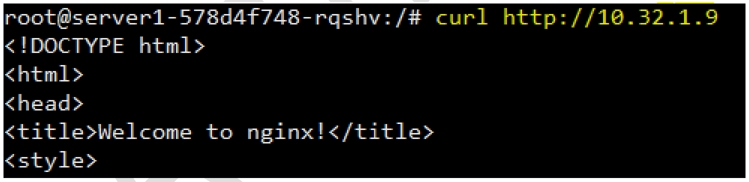
# kubectl run -it server1 --image=ubuntu /bin/bash

:- after login to pod, run below commands.

# apt-get update

# apt-get install curl

curl http://10.32.1.9

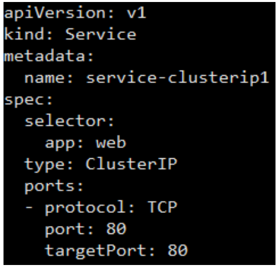


```
root@server1-578d4f748-rqshv:/# curl http://10.32.1.9
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

….

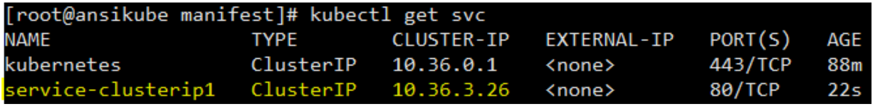
@@ Create cluster IP service.

#cat clusterport.yml



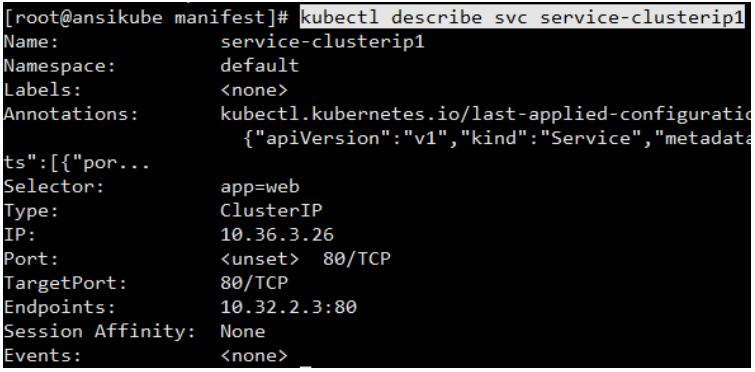
```
apiVersion: v1
kind: Service
metadata:
  name: service-clusterip1
spec:
  selector:
    app: web
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

```
# kubectl apply -f clusterport.yml
# kubectl get svc
```



```
[root@ansikube manifest]# kubectl get svc
NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
kubernetes           ClusterIP   10.36.0.1     <none>        443/TCP   88m
service-clusterip1   ClusterIP   10.36.3.26    <none>        80/TCP    22s
```

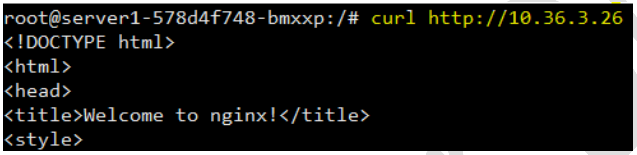
```
# kubectl describe svc service-clusterip1
```



```
[root@ansikube manifest]# kubectl describe svc service-clusterip1
Name:              service-clusterip1
Namespace:         default
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuratic
                     {"apiVersion":"v1","kind":"Service","metadata
ts":[{"por...
Selector:          app=web
Type:              ClusterIP
IP:                10.36.3.26
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.32.2.3:80
Session Affinity:  None
Events:            <none>
```

@Login to pods and check the http connectivity with service ip by using curl command.

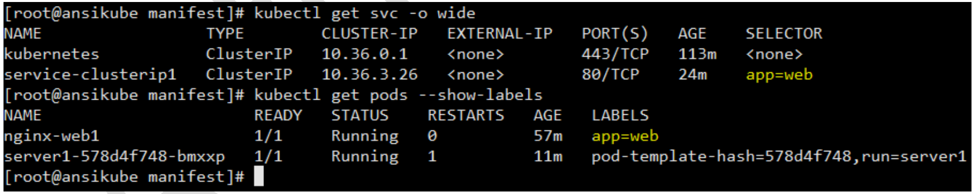
```
# kubectl run -it server1 --image=ubuntu /bin/bash  → run this if pod is not present.
#kubectl get pods
# kubectl exec -it server1-578d4f748-bmxxp /bin/bash
#apt-get update && apt-get install curl -y     → run this command after login into pod.
#curl http://10.36.3.26  → try to access with your cluster ip.
```



```
root@server1-578d4f748-bmxxp:/# curl http://10.36.3.26
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

Note:- cluster ip will be used for internal communication and it will be identified same identity of SELECTOR and pod LABELS.

```
#kubectl get svc -o wide
#kubectl get pods --show-labels
```

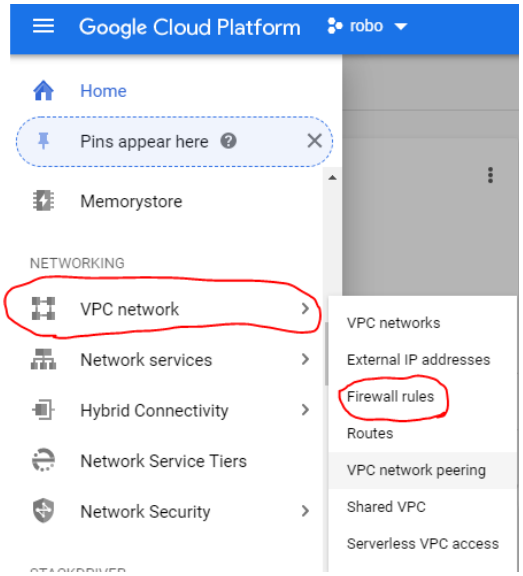


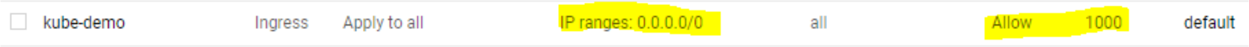
```
[root@ansikube manifest]# kubectl get svc -o wide
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE    SELECTOR
kubernetes           ClusterIP   10.36.0.1    <none>        443/TCP   113m   <none>
service-clusterip1   ClusterIP   10.36.3.26   <none>        80/TCP    24m    app=web
[root@ansikube manifest]# kubectl get pods --show-labels
NAME                       READY   STATUS    RESTARTS   AGE   LABELS
nginx-web1                 1/1     Running   0          57m   app=web
server1-578d4f748-bmxxp    1/1     Running   1          11m   pod-template-hash=578d4f748,run=server1
[root@ansikube manifest]# █
```

##Nodeport: Clients send requests to the IP address of a node on one or more nodePort values that are specified by the Service##

In Kubernetes cluster, on each node, specific port will be opened with that port, we can expose the running application to external client access. on specific node, doesn't matter whether pod is running or not, however the service will run because of kubeproxy. the disadvantage is we access the node via specific port via public ip of VM's and we cannot ask end user to access with port and possibilities are there for hacking via port.

Note:- Make sure to create a firewall rules on GCP (Google Cloud Platform) for node port access (VPC network --> Firewall rules --> create firewall rules)

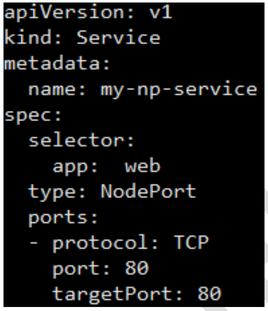




□	kube-demo	Ingress	Apply to all	IP ranges: 0.0.0.0/0	all	Allow	1000	default

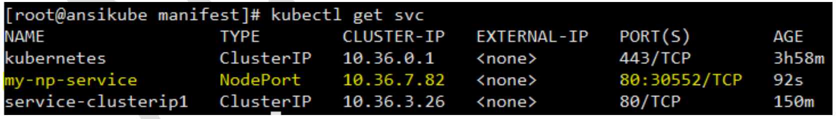
Type1:- Creating Nodeport Service with Dynamic port.

# cat nodeport.yml



```
apiVersion: v1
kind: Service
metadata:
  name: my-np-service
spec:
  selector:
    app:  web
  type: NodePort
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

# kubectl apply -f nodeport.yml



```
[root@ansikube manifest]# kubectl get svc
NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
kubernetes           ClusterIP   10.36.0.1     <none>        443/TCP        3h58m
my-np-service        NodePort    10.36.7.82    <none>        80:30552/TCP   92s
service-clusterip1   ClusterIP   10.36.3.26    <none>        80/TCP         150m
```

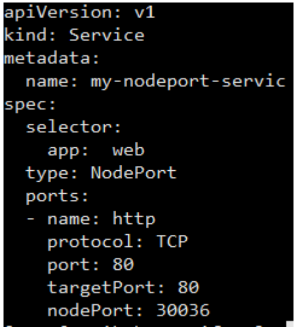
@Try to access the website by using external ip with auto generate port.

# kubectl get pods --show-labels
# kubectl get nodes -o wide

:-Open browser and access with dynamic port http://35.192.148.161:30552/

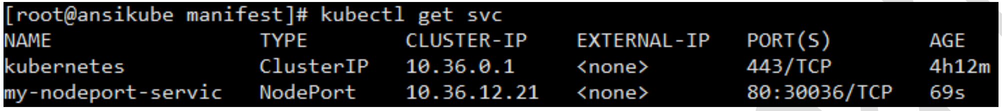
Type2:- Creating Nodeport Service with static port.

# cat nodeport1.yml



```
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-servic
spec:
  selector:
    app:  web
  type: NodePort
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30036
```

# kubectl apply -f nodeport1.yml
kubectl get svc



```
[root@ansikube manifest]# kubectl get svc
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
kubernetes           ClusterIP   10.36.0.1      <none>         443/TCP        4h12m
my-nodeport-servic   NodePort    10.36.12.21    <none>         80:30036/TCP   69s
```

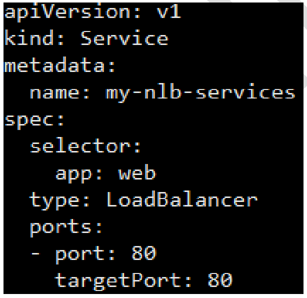
@Try to access the website by using external ip with specific port.

http://35.192.148.161:30036/

##LoadBalancer: Clients send requests to the IP address of a network load balancer##
This will come up with single static ip and listen via 80 port and backend whatever pod running, it will pass the request from the client. so, end user will access the domain without port mention, and this will be used in real-time. the disadvantage is incased four application has exposed in the services means, four load balancers will be created and will be more cost involved, so these kind of setup will be used for cloud providers.

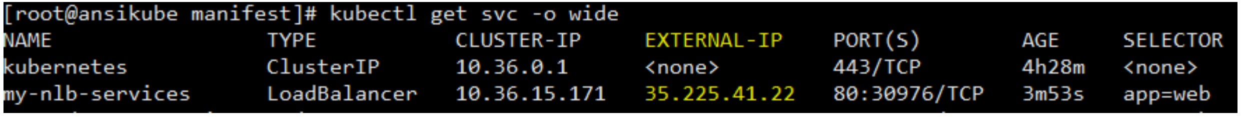
Type1:- In this method, gcloud provider will assign the external ip to load balancer, we can access via that ip.

# cat loadbalancer.yml



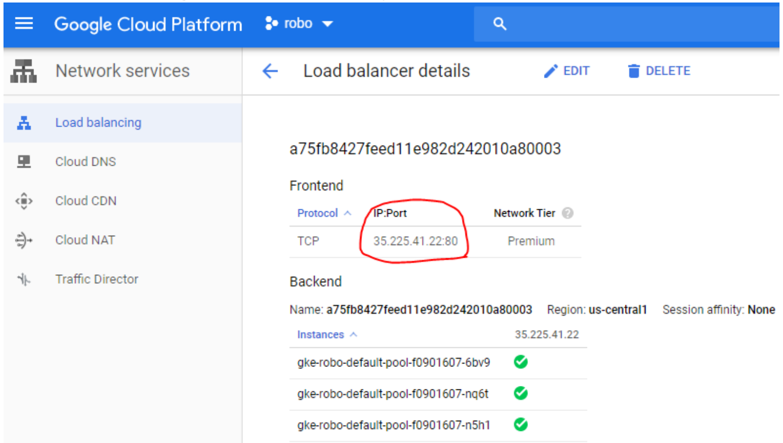
```
apiVersion: v1
kind: Service
metadata:
  name: my-nlb-services
spec:
  selector:
    app: web
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
```

# kubectl apply -f loadbalancer.yml
kubectl get svc -o wide



```
[root@ansikube manifest]# kubectl get svc -o wide
NAME              TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE     SELECTOR
kubernetes        ClusterIP      10.36.0.1      <none>         443/TCP        4h28m   <none>
my-nlb-services   LoadBalancer   10.36.15.171   35.225.41.22   80:30976/TCP   3m53s   app=web
```

@You can see get the external ip info from GCP as well.



Load balancer details

a75fb8427feed11e982d242010a80003

Frontend

Protocol	IP:Port	Network Tier
TCP	35.225.41.22:80	Premium

Backend

Name: a75fb8427feed11e982d242010a80003 Region: us-central1 Session affinity: None

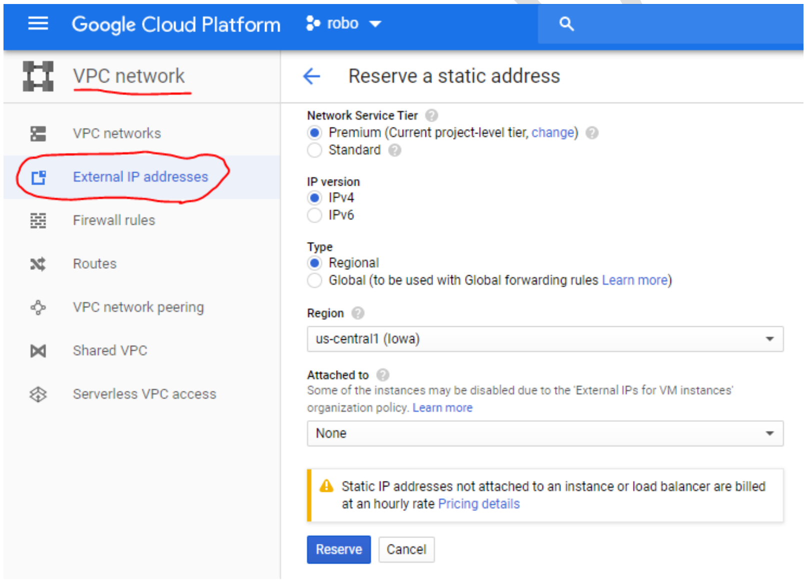
Instances	35.225.41.22
gke-robo-default-pool-f0901607-6bv9	✓
gke-robo-default-pool-f0901607-nq6t	✓
gke-robo-default-pool-f0901607-n5h1	✓

:-Open browser and try to access with loadbalancer ip. http://35.225.41.22/

Note:- Delete once you done.
# kubectl delete -f loadbalancer.yml

Type2:- In this method, we can use static ip which is preferrable for prod, since dynamic ip can be changed by cloud provider and it may disconnect the access.

Note:- Reserve the static ip on GCP cloud by selecting VPC networks --> External IP addresses --> RESERVE STATIC ADDRESS



Reserve a static address

Network Service Tier
● Premium (Current project-level tier, change)
○ Standard

IP version
● IPv4
○ IPv6

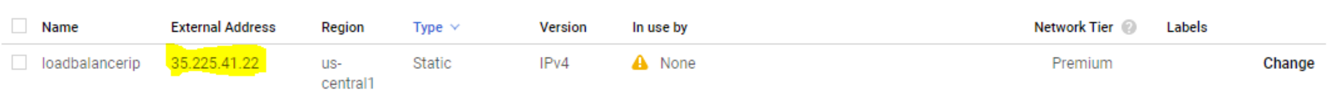
Type
● Regional
○ Global (to be used with Global forwarding rules Learn more)

Region
us-central1 (Iowa)

Attached to
Some of the instances may be disabled due to the 'External IPs for VM instances' organization policy. Learn more
None

Static IP addresses not attached to an instance or load balancer are billed at an hourly rate Pricing details

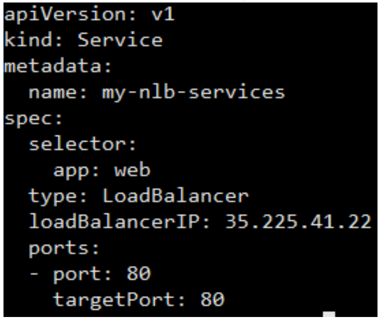
Reserve Cancel



□	Name	External Address	Region	Type	Version	In use by	Network Tier	Labels	
□	loadbalancerip	35.225.41.22	us-central1	Static	IPv4	None	Premium		Change

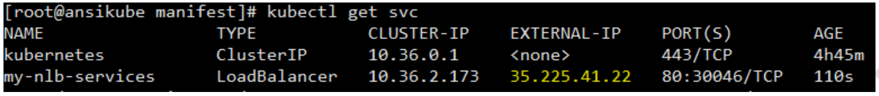
:- with this reserve external ip, we are going to create load balancer on Kubernetes cluster.

#cat loadbalancer1.yml



```
apiVersion: v1
kind: Service
metadata:
  name: my-nlb-services
spec:
  selector:
    app: web
  type: LoadBalancer
  loadBalancerIP: 35.225.41.22
  ports:
  - port: 80
    targetPort: 80
```

# kubectl apply -f loadbalancer1.yml
kubectl get svc

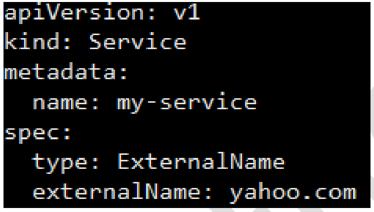


```
[root@ansikube manifest]# kubectl get svc
NAME                TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
kubernetes          ClusterIP      10.36.0.1      <none>         443/TCP        4h45m
my-nlb-services     LoadBalancer   10.36.2.173    35.225.41.22   80:30046/TCP   110s
```

:- Open browser and try to access via LB ip. http://35.225.41.22/

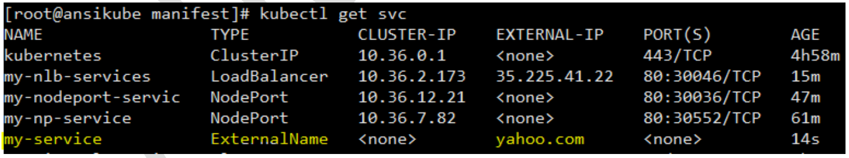
##ExternalName: Internal clients use the DNS name of a Service as an alias for an external DNS name##
for example, our own machine will connect to third-party application is called external service and which will be mapped in this external service and it has an instruct to access . every time no need to do any changes on application level and since external ip has given by apps vendor, so it may change at any time and not required to change on all backend pods. Here we can make changes on external name which we configured and that will manage backend.

# cat externlsvc.yml



```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ExternalName
  externalName: yahoo.com
```

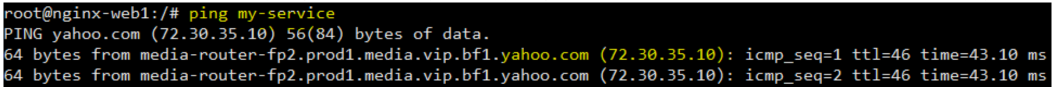
# kubectl apply -f externlsvc.yml
kubectl get svc



```
[root@ansikube manifest]# kubectl get svc
NAME                TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
kubernetes          ClusterIP      10.36.0.1      <none>         443/TCP        4h58m
my-nlb-services     LoadBalancer   10.36.2.173    35.225.41.22   80:30046/TCP   15m
my-nodeport-servic  NodePort       10.36.12.21    <none>         80:30036/TCP   47m
my-np-service       NodePort       10.36.7.82     <none>         80:30552/TCP   61m
my-service          ExternalName   <none>         yahoo.com      <none>         14s
```

# kubectl get pods

@Login to pod and ping with external name.
# kubectl exec -it nginx-web1 /bin/bash
#apt-get update && apt-get install iputils-ping → Run these commands inside the pod.



```
root@nginx-web1:/# ping my-service
PING yahoo.com (72.30.35.10) 56(84) bytes of data.
64 bytes from media-router-fp2.prod1.media.vip.bf1.yahoo.com (72.30.35.10): icmp_seq=1 ttl=46 time=43.10 ms
64 bytes from media-router-fp2.prod1.media.vip.bf1.yahoo.com (72.30.35.10): icmp_seq=2 ttl=46 time=43.10 ms
```