Services

a Service is an abstraction which defines a logical set of Pods and a policy by which to access them. Services are using a selector field to target the Pods. Once the selector field of a service matches with labels assigned to the pods, these pods become the endpoints of the service.

Types of services:

1. NodePort
2. ClusterIP
3. LoadBalancer

Objectives:

1. Creating a deployment and exposing it to a NodePort service
2. Creating a deployment and exposing it to a ClusterIP service
3. Creating a deployment and exposing it to a LoadBalancer service

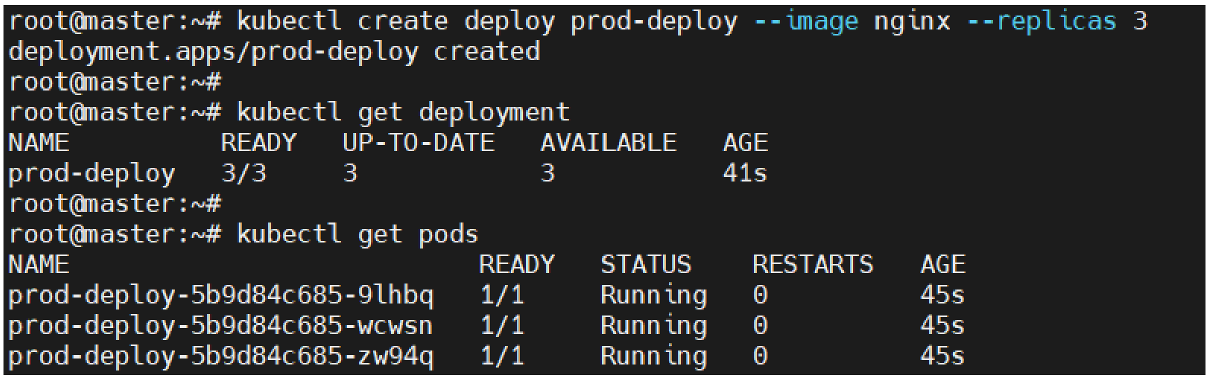
1. Creating a deployment and exposing it to a NodePort service:

It exposes the Service on each Node's IP at a static port. In other word, the traffic from outside world will use NodePort service to connect with the endpoints. Kubernetes allocates 30000-32767 ports to NodePort service

Step1: Create a deployment

We are going to create a deployment named **prod-deploy** with nginx image. Use the below command to create this deployment.

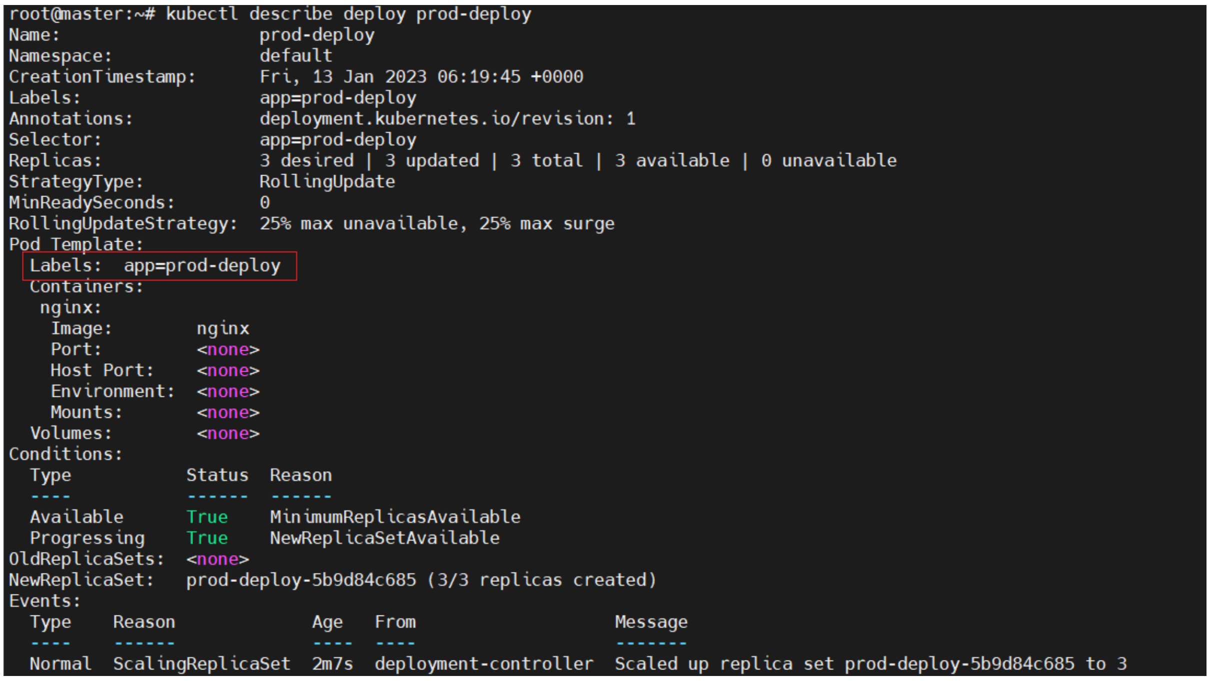
```
kubectl create deploy prod-deploy --image nginx --replicas 3
```



```
root@master:~# kubectl create deploy prod-deploy --image nginx --replicas 3
deployment.apps/prod-deploy created
root@master:~#
root@master:~# kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
prod-deploy   3/3     3            3           41s
root@master:~#
root@master:~# kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
prod-deploy-5b9d84c685-9lhbq   1/1     Running   0          45s
prod-deploy-5b9d84c685-wcwsn   1/1     Running   0          45s
prod-deploy-5b9d84c685-zw94q   1/1     Running   0          45s
```

From above output, we can see that our deployment with 3 pods has been created. Now we will describe the deployment to check the labels which will be used in our service under selector field.

```
kubectl describe deploy prod-deploy
```



```
root@master:~# kubectl describe deploy prod-deploy
Name:                   prod-deploy
Namespace:              default
CreationTimestamp:      Fri, 13 Jan 2023 06:19:45 +0000
Labels:                 app=prod-deploy
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=prod-deploy
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=prod-deploy
  Containers:
   nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   prod-deploy-5b9d84c685 (3/3 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  2m7s  deployment-controller  Scaled up replica set prod-deploy-5b9d84c685 to 3
```

From above output, we can see that the pods have the label **app=prod-deploy**

<u>**Step2: Create a NodePort service**</u>

We can simply use an imperative command or we can create a Yaml file to create a NodePort service.

```
kubectl expose deploy prod-deploy --name prod-service --type NodePort --port 80
--target-port=80
```

Above command will automatically expose our deployment and Kubernetes will assign the NodePort automatically from range 30000-32767.

Or Create the Yaml file

```
kubectl expose deploy prod-deploy --name prod-service --type NodePort --port 80
--target-port=80 --dry-run=client -o yaml > prod-service.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: prod-deploy
  name: prod-service
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30010
  selector:
    app: prod-deploy
  type: NodePort
status:
  loadBalancer: {}
```

We have our Yaml file ready and let’s understand the important filed inside this Yaml file.

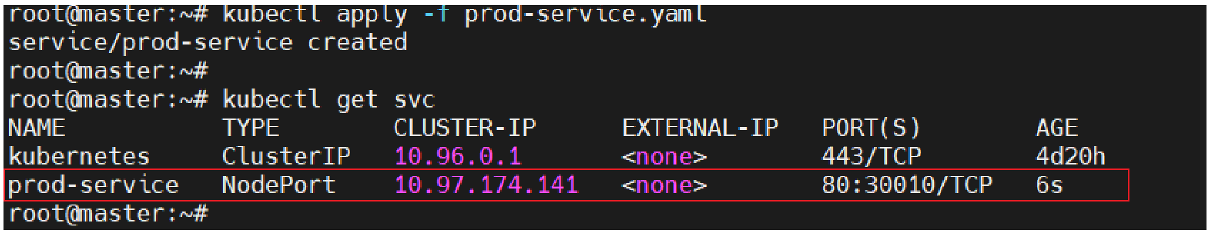
Port: the port of the service
targetPort: port of the pod this service is targeting to.
NodePort: Port on the node which is exposed outside. We are explicitly using 300010, if we do not mention then Kubernetes will automatically assign the port.
Selector: It is used to target the Pods which are having the same label. So the pods having app=prod-deploy label they will become the endpoint of this service.

Now let’s apply this Yaml file and we will check the status further

```
kubectl apply -f prod-service.yaml
```

Check the services available in our cluster in current namespace.

```
kubectl get svc
```

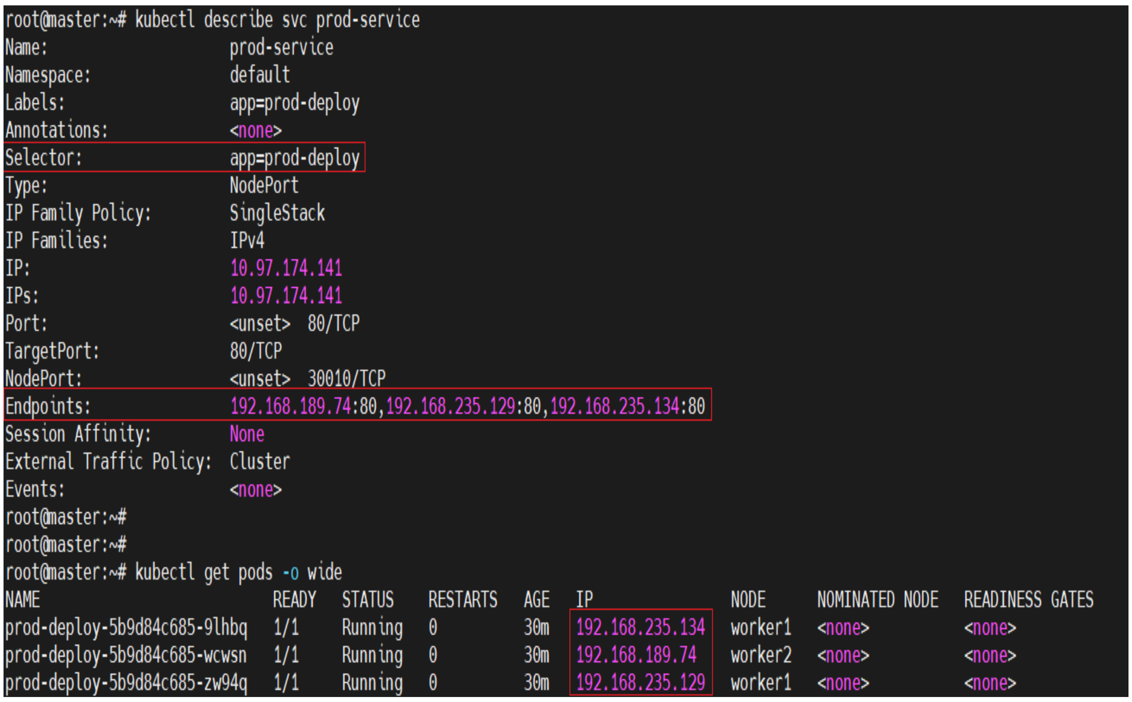


```
root@master:~# kubectl apply -f prod-service.yaml
service/prod-service created
root@master:~#
root@master:~# kubectl get svc
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP        4d20h
prod-service   NodePort    10.97.174.141   <none>        80:30010/TCP   6s
root@master:~#
```

Our NodePort prod-service has been created and with IP address 10.97.174.141 which is provided by Kubernetes only. We can also see that 30010 nodePort has been assigned to this service.

Let's describe this service.

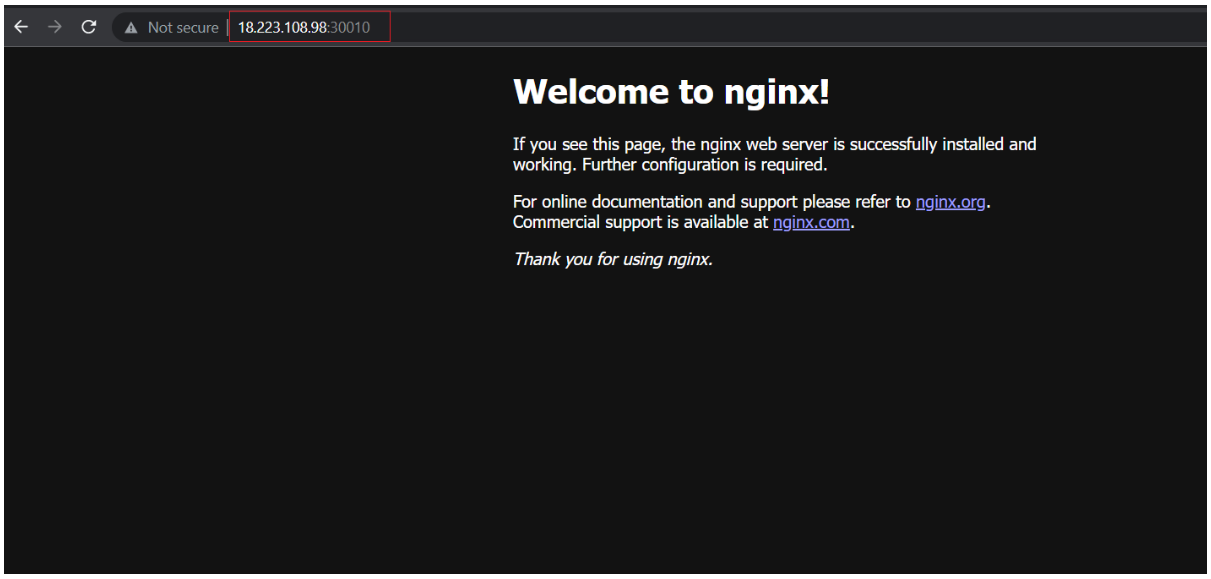
```
kubectl describe svc prod-service
```



```
root@master:~# kubectl describe svc prod-service
Name:                     prod-service
Namespace:                default
Labels:                   app=prod-deploy
Annotations:              <none>
Selector:                 app=prod-deploy
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.97.174.141
IPs:                      10.97.174.141
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30010/TCP
Endpoints:                192.168.189.74:80,192.168.235.129:80,192.168.235.134:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
root@master:~#
root@master:~#
root@master:~# kubectl get pods -o wide
NAME                           READY   STATUS    RESTARTS   AGE   IP                NODE      NOMINATED NODE   READINESS GATES
prod-deploy-5b9d84c685-9lhbq   1/1     Running   0          30m   192.168.235.134   worker1   <none>           <none>
prod-deploy-5b9d84c685-wcwsn   1/1     Running   0          30m   192.168.189.74    worker2   <none>           <none>
prod-deploy-5b9d84c685-zw94q   1/1     Running   0          30m   192.168.235.129   worker1   <none>           <none>
```

Above we can see that under the endpoints, we are having the same pods which are having **prod-deploy** label.

Now this has been exposed to the outside world. We can access it using the below in the browser.
node-ip-address:nodeport



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

So, we are able to access our application using NodePort service.

Now delete the service using below command.

```
kubectl delete svc prod-service
```

2. Creating a deployment and exposing it to a ClusterIP service

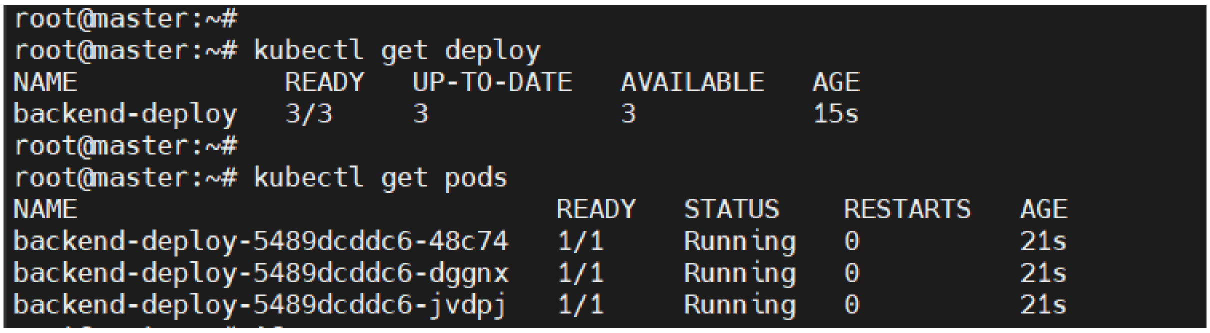
It exposes the Service on a cluster-internal IP. It is used access from within the cluster. This is the default that is used if you don't explicitly specify a type for a Service. It uses the same selector concept to access the

Step1: Create a deployment

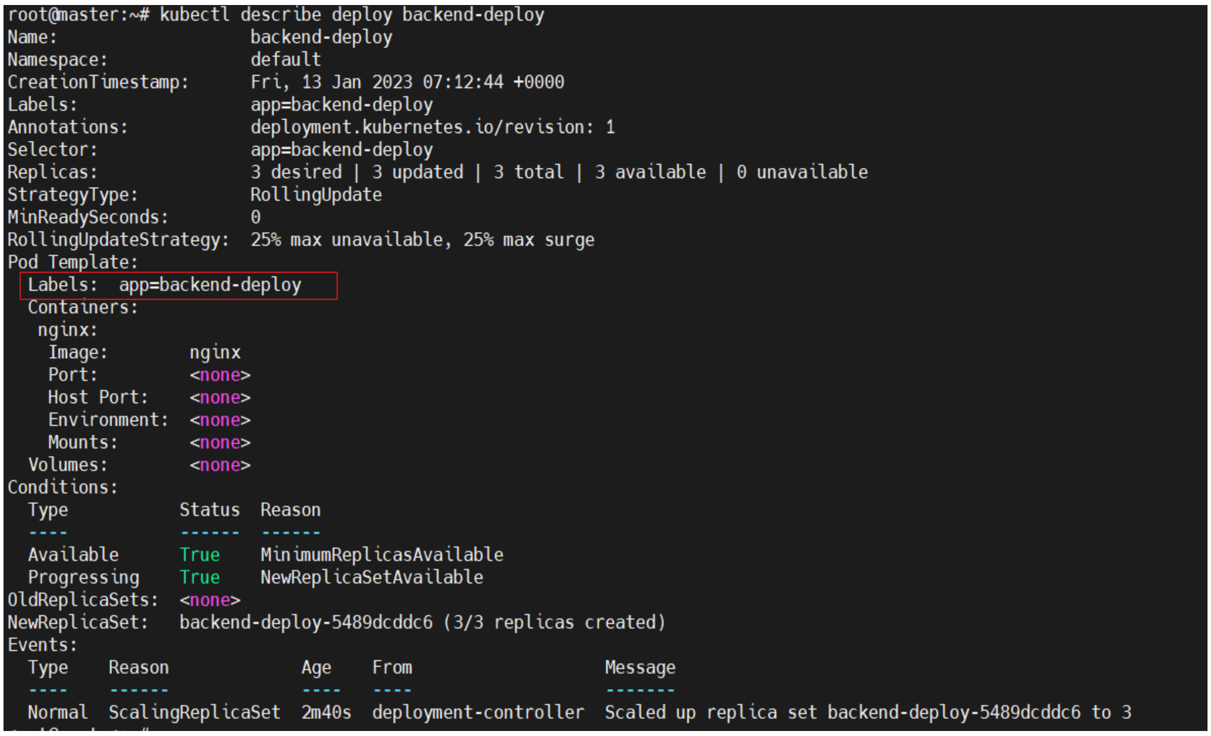
Create a deployment using below command.

```
kubectl create deploy backend-deploy --image nginx --replicas=3
```

It will create a deployment **backend-deploy** having 3 pods with **nginx** image. Let’s describe the deployment and we will check use the same labels for our **clusterIP** service.



```
root@master:~#
root@master:~# kubectl get deploy
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
backend-deploy   3/3     3            3           15s
root@master:~#
root@master:~# kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
backend-deploy-5489dcddc6-48c74   1/1     Running   0          21s
backend-deploy-5489dcddc6-dggnx   1/1     Running   0          21s
backend-deploy-5489dcddc6-jvdpj   1/1     Running   0          21s
```



```
root@master:~# kubectl describe deploy backend-deploy
Name:                   backend-deploy
Namespace:              default
CreationTimestamp:      Fri, 13 Jan 2023 07:12:44 +0000
Labels:                 app=backend-deploy
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=backend-deploy
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=backend-deploy
  Containers:
   nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   backend-deploy-5489dcddc6 (3/3 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  2m40s  deployment-controller  Scaled up replica set backend-deploy-5489dcddc6 to 3
```

Above we can see that pods are having **app=backend-deploy** label which we will be used under **selector** field of our **ClusterIP** service.

Step2: Create a ClusterIP service

We can use the imperative command as well as the Yaml file to create the ClusterIP service.

Use the below command to simply expose backend-deploy.

```
kubectl expose deploy backend-deploy --name backend-service --port 80
```

Or we can create a Yaml template using below command.

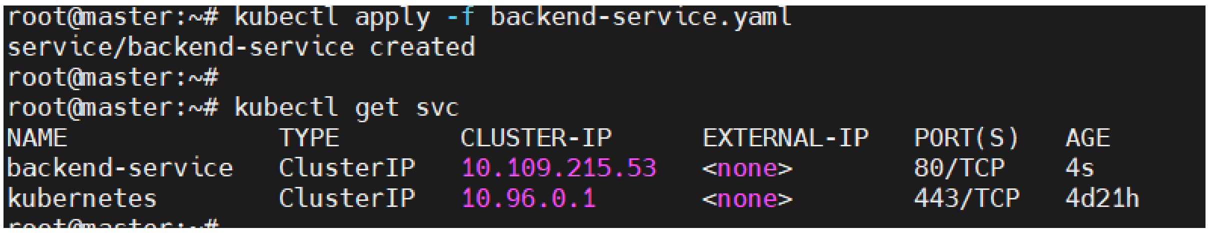
```
kubectl expose deploy backend-deploy --name backend-service --port 80 --dry-run=client -o yaml > backend-service.yaml
```

The above command will create a template. We can see below that this service is having the selector field **app: backend-deploy** which will target the pod having the same label.
As we have discussed in NodePort, port is for the service itself and targetPort is the pod's port.

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: backend-deploy
  name: backend-service
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: backend-deploy
status:
   loadBalancer: {}
```

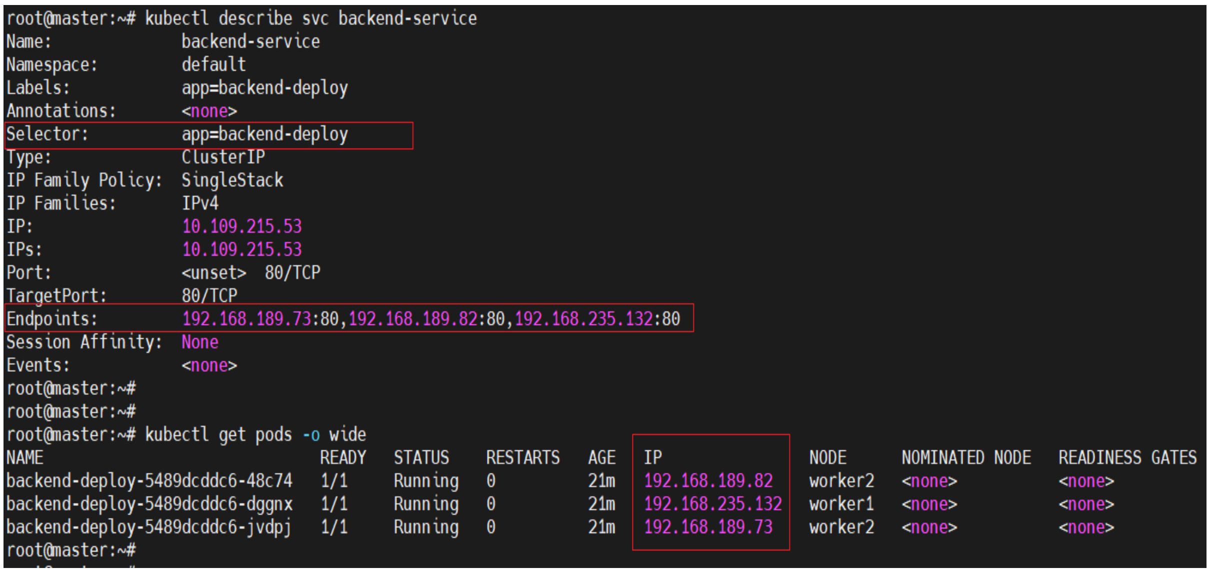
Let's apply this file and describe it to get more details.

```
kubectl apply -f backend-service.yaml
```



```
root@master:~# kubectl apply -f backend-service.yaml
service/backend-service created
root@master:~#
root@master:~# kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
backend-service   ClusterIP   10.109.215.53   <none>        80/TCP    4s
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP   4d21h
```

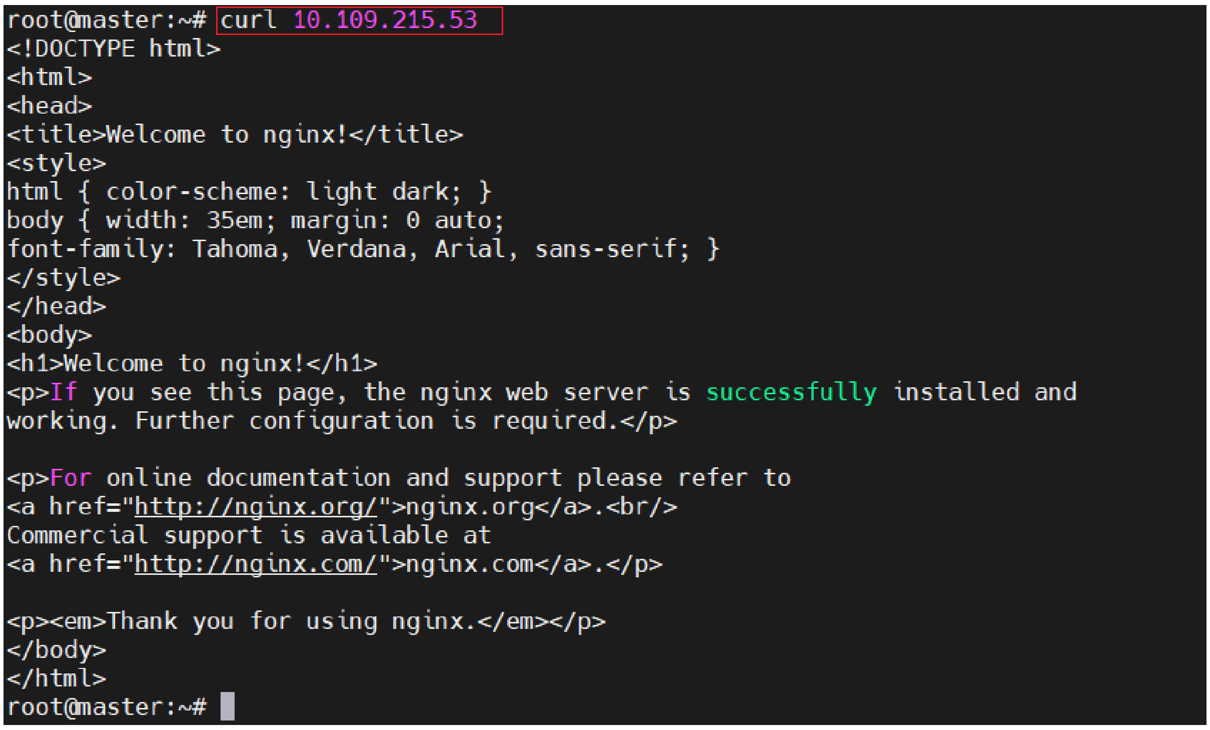
Above we can see that our clusterIP service has been created with **10.109.215.53** IP address which has been assigned by Kubernetes only.


```
root@master:~# kubectl describe svc backend-service
Name:              backend-service
Namespace:         default
Labels:            app=backend-deploy
Annotations:       <none>
Selector:          app=backend-deploy
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.109.215.53
IPs:               10.109.215.53
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.189.73:80,192.168.189.82:80,192.168.235.132:80
Session Affinity:  None
Events:            <none>
root@master:~#
root@master:~#
root@master:~# kubectl get pods -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP                NODE      NOMINATED NODE   READINESS GATES
backend-deploy-5489dcddc6-48c74   1/1     Running   0          21m   192.168.189.82    worker2   <none>           <none>
backend-deploy-5489dcddc6-dggnx   1/1     Running   0          21m   192.168.235.132   worker1   <none>           <none>
backend-deploy-5489dcddc6-jvdpj   1/1     Running   0          21m   192.168.189.73    worker2   <none>           <none>
root@master:~#
```

So our ClusterIP service has been created and it is targeting the pods which are having the label **app=backend-deploy**. Under the endpoints field we can see the targeted pods.

We can access the pods using the service IP address within the cluster.

```
Curl 10.109.215.53
```


```
root@master:~# curl 10.109.215.53
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@master:~#
```

Now delete the service using below command.

```
kubectl delete svc backend-service
```

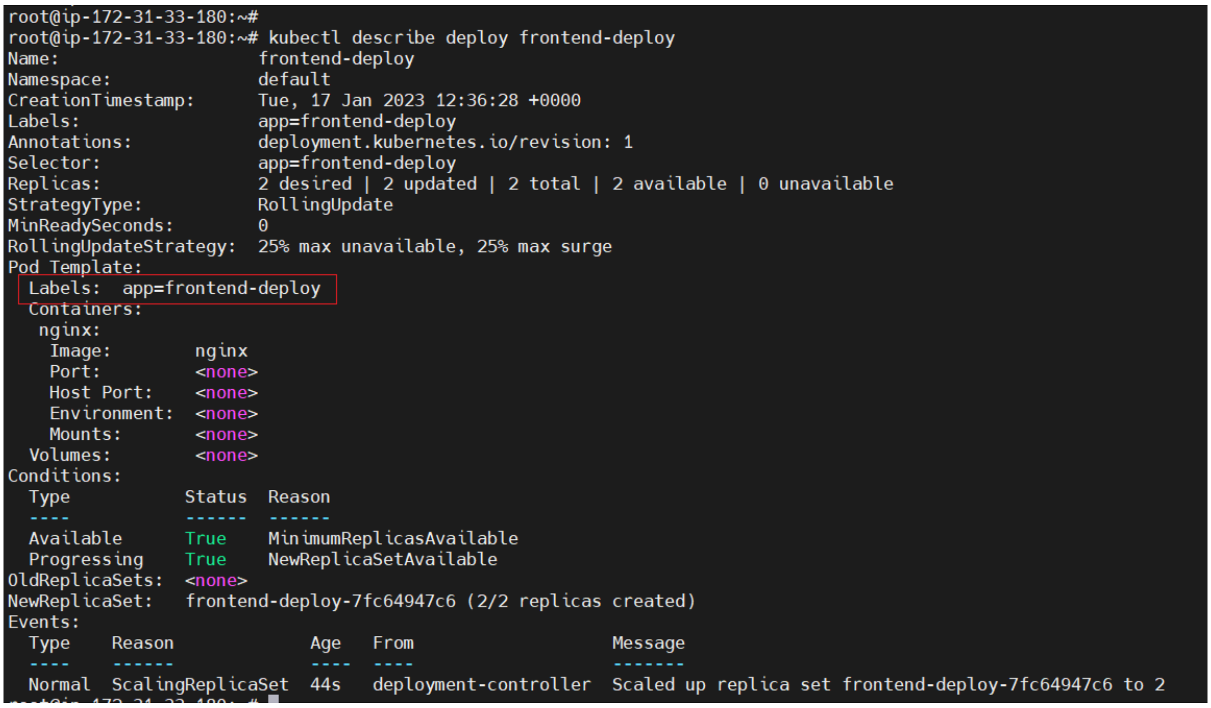
3. Creating a deployment and exposing it to a LoadBalancer service

It exposes the Service externally using a cloud provider's load balancer. It is same as NodePort but instead of using the node's IP address, here we will get a single DNS or IP address which cloud provider create for us. We use this service if we are going for PAAS services lie EKS or GKE etc.

Step1: Create a deployment

Use the below command to create a deployment.

```
kubectl create deploy frontend-deploy --image nginx --replicas=2
```



```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl describe deploy frontend-deploy
Name:                   frontend-deploy
Namespace:              default
CreationTimestamp:      Tue, 17 Jan 2023 12:36:28 +0000
Labels:                 app=frontend-deploy
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=frontend-deploy
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=frontend-deploy
  Containers:
   nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   frontend-deploy-7fc64947c6 (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  44s   deployment-controller  Scaled up replica set frontend-deploy-7fc64947c6 to 2
```

The above output shows us the pods will be using the label **app=frontend-deploy**.

Step2: Create a LoadBalancer service

We will be using the below command or a Yaml file to create a LoadBalancer service. Cloud provider will automatically create a physical Load balancer for us.

```
kubectl expose deploy frontend-deploy --name frontend-service --type
LoadBalancer --port 80
```

Or create a template from the above command.

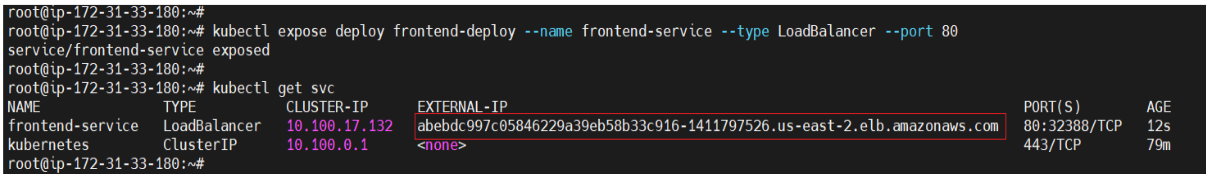
```
kubectl expose deploy frontend-deploy --name frontend-service --type
LoadBalancer --port 80 --dry-run=client -o yaml > frontend-service.yaml
```

```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: frontend-deploy
  name: frontend-service
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: frontend-deploy
  type: LoadBalancer
status:
   loadBalancer: {}
```

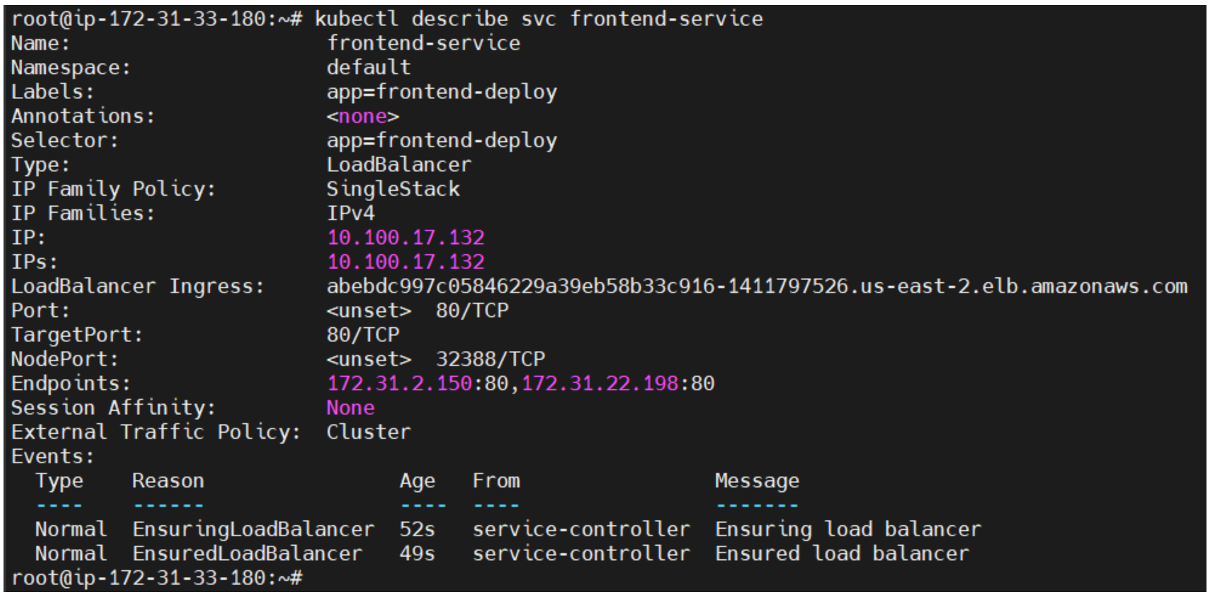
Use the below command to apply the definition file.

```
kubectl apply -f frontend-service.yaml
```

We can see below that a Load balancer service has been created and AWS create a physical load balancer.



```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl expose deploy frontend-deploy --name frontend-service --type LoadBalancer --port 80
service/frontend-service exposed
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl get svc
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP                                                               PORT(S)        AGE
frontend-service   LoadBalancer   10.100.17.132   abebdc997c05846229a39eb58b33c916-1411797526.us-east-2.elb.amazonaws.com   80:32388/TCP   12s
kubernetes         ClusterIP      10.100.0.1      <none>                                                                    443/TCP        79m
root@ip-172-31-33-180:~#
```



```
root@ip-172-31-33-180:~# kubectl describe svc frontend-service
Name:                     frontend-service
Namespace:                default
Labels:                   app=frontend-deploy
Annotations:              <none>
Selector:                 app=frontend-deploy
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.100.17.132
IPs:                      10.100.17.132
LoadBalancer Ingress:     abebdc997c05846229a39eb58b33c916-1411797526.us-east-2.elb.amazonaws.com
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32388/TCP
Endpoints:                172.31.2.150:80,172.31.22.198:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason                Age   From                Message
  ----    ------                ----  ----                -------
  Normal  EnsuringLoadBalancer  52s   service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer   49s   service-controller  Ensured load balancer
root@ip-172-31-33-180:~#
```