Exercise 7.1: Exposing Applications: Expose a Service

Overview

In this lab we will explore various ways to expose an application to other pods and outside the cluster. We will add to the NodePort used in previous labs other service options.

1. We will begin by using the default service type ClusterIP. This is a cluster internal IP, only reachable from within the cluster. Begin by viewing the existing services.

```
student@ckad-1:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        8d
nginx        ClusterIP   10.108.95.67     <none>        443/TCP        8d
registry     ClusterIP   10.105.119.236   <none>        5000/TCP       8d
secondapp    NodePort    10.111.26.8      <none>        80:32000/TCP   7h
```

2. Delete the existing service for secondapp.

```
student@ckad-1:~/app2$ kubectl delete svc secondapp
service "secondapp" deleted
```

3. Create a YAML file for a replacement service, which would be persistent. Use the label to select the secondapp. Expose the same port and protocol of the previous service.

```
student@ckad-1:~/app2$ vim service.yaml
```

service.yaml

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: secondapp
5    labels:
6      run: my-nginx
7  spec:
8    ports:
9    - port: 80
10     protocol: TCP
11   selector:
12     example: second
```

4. Create the service, find the new IP and port. Note there is no high number port as this is internal access only.

```
student@ckad-1:~/app2$ kubectl create -f service.yaml
service/secondapp created

student@ckad-1:~/app2$ kubectl get svc
```

```
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    8d
nginx        ClusterIP   10.108.95.67     <none>        443/TCP    8d
registry     ClusterIP   10.105.119.236   <none>        5000/TCP   8d
secondapp    ClusterIP   10.98.148.52     <none>        80/TCP     14s
```

5. Test access. You should see the default welcome page again.

```
student@ckad-1:~/app2$ curl http://10.98.148.52
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```

6. To expose a port to outside the cluster we will create a NodePort. We had done this in a previous step from the command line. When we create a NodePort it will create a new ClusterIP automatically. Edit the YAML file again. Add type: NodePort. Also add the high-port to match an open port in the firewall as mentioned in the previous chapter. You'll have to delete and re-create as the existing IP is immutable, but not able to be reused. The NodePort will try to create a new ClusterIP instead.

```
student@ckad-1:~/app2$ vim service.yaml
```

service.yaml

```
1 ....
2     protocol: TCP
3     nodePort: 32000          #<-- Add this and following line
4   type: NodePort
5   selector:
6     example: second
```

```
student@ckad-1:~/app2$ kubectl delete svc secondapp ; kubectl create  -f service.yaml
```

```
service "secondapp" deleted
service/secondapp created
```

7. Find the new ClusterIP and ports for the service.

```
student@ckad-1:~/app2$ kubectl get svc
```

```
NAME         TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
kubernetes   ClusterIP    10.96.0.1       <none>        443/TCP         8d
nginx        ClusterIP    10.108.95.67    <none>        443/TCP         8d
registry     ClusterIP    10.105.119.236  <none>        5000/TCP        8d
secondapp    NodePort     10.109.134.221  <none>        80:32000/TCP    4s
```

8. Test the low port number using the new ClusterIP for the secondapp service.

```
student@ckad-1:~/app2$ curl 10.109.134.221
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```

9. Test access from an external node to the host IP and the high container port. Your IP and port will be different. It should work, even with the network policy in place, as the traffic is arriving via a 192.168.0.0 port.

```
user@laptop:~/Desktop$ curl http://35.184.219.5:32000
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```

10. The use of a LoadBalancer makes an asynchronous request to an external provider for a load balancer if one is available. It then creates a NodePort and waits for a response including the external IP. The local NodePort will work even before the load balancer replies. Edit the YAML file and change the type to be LoadBalancer.

```
student@ckad-1:~/app2$ vim service.yaml
```

service.yaml

```
1  ....
2    - port: 80
3      protocol: TCP
4    type: LoadBalancer    #<-- Edit this line
5    selector:
6      example: second
```

```
student@ckad-1:~/app2$ kubectl delete svc secondapp ; kubectl create -f service.yaml
```

```
service "secondapp" deleted
service/secondapp created
```

11. As mentioned the cloud provider is not configured to provide a load balancer; the External-IP will remain in pending state. Some issues have been found using this with VirtualBox.

```
student@ckad-1:~/app2$ kubectl get svc
```

```
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1        <none>        443/TCP        8d
nginx        ClusterIP      10.108.95.67     <none>        443/TCP        8d
registry     ClusterIP      10.105.119.236   <none>        5000/TCP       8d
secondapp    LoadBalancer   10.109.26.21     <pending>     80:32000/TCP   4s
```

12. Test again local and from a remote node. The IP addresses and ports will be different on your node.

```
serewic@laptop:~/Desktop$ curl http://35.184.219.5:32000
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```