Lab 7.1 - Exposing Applications

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 and other service options.

Expose A Service

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
```

Delete the existing service for **secondapp**.

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

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
apiVersion: v1
```



```
kind: Service
metadata:
  name: secondapp
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    example: second
```

Create the service, find the new IP and port. Note that 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
```

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>
```

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
apiVersion: v1
```



```
kind: Service
metadata:
  name: secondapp
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    nodePort: 32000
  type: NodePort
  selector:
    example: second

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

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
```

Test the low port number using the **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>
```

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.

```
serewicz@laptop:~/Desktop$ curl http://35.184.219.5:32000
<!DOCTYPE html>
```



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

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
<output_omitted>
  - port: 80
    protocol: TCP
  type: LoadBalancer
  selector:
    example: second

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

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

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
```

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>
```

Ingress Controller

If you have a large number of services to expose outside of the cluster, or to expose a low-number port on the host node, you can deploy an ingress controller. While nginx and GCE have controllers officially supported by Kubernetes.io, the Traefik Ingress Controller is easier to install, at least at the moment.

As we have RBAC configured, we need to make sure the controller will run and be able to work with all necessary ports, endpoints, and resources. Create a YAML file to declare a **clusterrole** and a **clusterrolebinding**:

```
student@ckad-1:~/app2$ vim ingress.rbac.yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
      - secrets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
```



```
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-ingress-controller
subjects:
- kind: ServiceAccount
  name: traefik-ingress-controller
  namespace: kube-system
```

Create the new role and binding:

```
student@ckad-1:~/app2$ kubectl create -f ingress.rbac.yaml
clusterrole.rbac.authorization.k8s.io/traefik-ingress-controller created
clusterrolebinding.rbac.authorization.k8s.io/traefik-ingress-controller
created
```

Create the Traefik controller. We will use a script directly from their website. The shorter, easier URL for the following is https://goo.gl/D2uEEF. We will need to download the script and make some edits before creating the objects.

```
student@ckad-1:~/app2$ wget \
https://raw.githubusercontent.com/containous/traefik/master/examples/k8s/tr
aefik-ds.yaml
```

Edit the downloaded file. The output below represents the changes in a diff type output, from the downloaded to the edited file. One line should be added, six lines should be removed.

```
student@ckad-1:~/app2$ vim traefik-ds.yaml
23a24              ##  Add the following line 24
>       hostNetwork: true
34,39d34         ##  Remove these lines around line 34
<         securityContext:
<           capabilities:
<             drop:
<             - ALL
<             add:
<             - NET_BIND_SERVICE
```


The file should look like this:

```
….
      terminationGracePeriodSeconds: 60
      hostNetwork: True
      containers:
      - image: traefik
        name: traefik-ingress-lb
        ports:
        - name: http
          containerPort: 80
          hostPort: 80
        - name: admin
          containerPort: 8080
          hostPort: 8080
        args:
        - --api
….
```

Create the objects using the edited file:

```
student@ckad-1:~/app2$ kubectl apply -f traefik-ds.yaml
serviceaccount/traefik-ingress-controller created
daemonset.extensions/traefik-ingress-controller created
service/traefik-ingress-service created
```

Now that there is a new controller, we need to pass some rules, so it knows how to handle requests. Note that the host mentioned is www.example.com, which is probably not your node name. We will pass a false header when testing. Also, the service name needs to match the **secondapp** we've been working with:

```
student@ckad-1:~/app2$ vim ingress.rule.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - backend:
```



```
          serviceName: secondapp
          servicePort: 80
        path: /
```

Now, ingest the rule into the cluster:

```
student@ckad-1:~/app2$ kubectl create -f ingress.rule.yaml
ingress.extensions/ingress-test created
```

We should be able to test the internal and external IP addresses, and see the nginx welcome page. The loadbalancer would present the traffic, a **curl** request in this case, to the externally facing interface. Use **ip a** to find the IP address of the interface which would face the loadbalancer. In this example, the interface would be **ens4**, and the IP would be 10.128.0.7.

```
student@ckad-1:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group
default qlen 1000
    link/ether 42:01:0a:80:00:03 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.3/32 brd 10.128.0.3 scope global ens4
       valid_lft forever preferred_lft forever
<output_omitted>

student@ckad-1:~/app2$ curl -H "Host: www.example.com" http://10.128.0.7/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>

serewicz@laptop:~$ curl -H "Host: www.example.com" http://35.193.3.179
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```



```
<output_omitted>
```

At this point, we would keep adding more and more web servers. We'll configure one more, which would then be a process continued as many times as desired.

Begin by deploying another **nginx** server. Give it a label and expose port 80:

```
student@ckad-1:~/app2$ kubectl run thirdpage --image=nginx \
  --port=80 -l example=third
deployment.apps "thirdpage" created
```

Expose the new server as a **NodePort**:

```
student@ckad-1:~/app2$ kubectl expose deployment thirdpage --type=NodePort
service "thirdpage" exposed
```

Now, we will customize the installation. Run a bash shell inside the new pod. Your pod name will end differently. Install **vim** inside the container, then edit the **index.html** file of *nginx* so that the title of the web page will be *Third Page*.

```
student@ckad-1:~/app2$ kubectl exec -it thirdpage-5cf8d67664-zcmfh --
/bin/bash

root@thirdpage-5cf8d67664-zcmfh:/# apt-get update

root@thirdpage-5cf8d67664-zcmfh:#  apt-get install vim -y

root@thirdpage-5cf8d67664-zcmfh:/# vim /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Third Page</title>
<style>
<output_omitted>
```

Edit the ingress rules to point the **thirdpage** service:

```
student@ckad-1:~/app2$ kubectl edit ingress ingress-test
<output_omitted>
  - host: www.example.com
    http:
      paths:
      - backend:
```



```
          serviceName: secondapp
          servicePort: 80
        path: /
  - host: thirdpage.org
    http:
      paths:
      - backend:
          serviceName: thirdpage
          servicePort: 80
        path: /
 status:
<output_omitted>
```

Test the second hostname using **curl** locally, as well as from a remote system:

```
student@ckad-1:~/app2$ curl -H "Host: thirdpage.org" http://10.128.0.7/
<!DOCTYPE html>
<html>
<head>
<title>Third Page</title>
<style>
<output_omitted>
```

