Exercise 7.2: 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 the moment.

1. 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
```

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
```

2. 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
```

3. Create the Traefik controller. The source web page changes on a regular basis. You can find a recent release by going here https://github.com/containous/traefik/releases, The recent 2.X release has many changes and some "undocumented features" being worked on. Find a copy of the file in the course tarball using the **find** command.

```
student@ckad-1:~/app2$ find ~ -name traefik-ds.yaml
```

4. The output below represents the changes in a **diff** output, from a downloaded version to the edited file in the tarball. One line was added, six lines removed. Also with version 2.0 the dashboard does not appear to work, so we are declaring the use of version `1.7.13`.

```
student@ckad-1:~/app2$ diff download.yaml traefik-ds.yaml
```

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 included file looks like this:

traefik-ds.rule.yaml

```
....
      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
....
```

5. 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
```

6. 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
```

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: /
```

7. Now ingest the rule into the cluster.

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

8. 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 load balancer. 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 10.128.0.7/32 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>


user@laptop:~$ curl -H "Host: www.example.com" http://35.193.3.179
```

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

9. 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 create deployment thirdpage --image=nginx
deployment.apps "thirdpage" created
```

10. Assign a label for the ingress controller to match against. Your pod name is unique, you can use the **Tab** key to complete the name.

```
student@ckad-1:~/app2$ kubectl label pod thirdpage-<tab> example=third
```

11. Expose the new server as a NodePort.

```
student@ckad-1:~/app2$ kubectl expose deployment thirdpage --port=80 --type=NodePort
service/thirdpage exposed
```

12. Now we will customize the installation. Run a bash shell inside the new pod. Your pod name will end differently. Install **vim** or an editor inside the container then edit the `index.html` file of nginx so that the title of the web page will be `Third Page`. Much of the command output is not shown below.

```
student@ckad-1:~/app2$ kubectl exec -it thirdpage-<Tab> -- /bin/bash
```

On Container

```
root@thirdpage-:/# apt-get update

root@thirdpage-:/# apt-get install vim -y

root@thirdpage-:/# vim /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Third Page</title>     #<-- Edit this line
<style>
<output_omitted>


root@thirdpage-:/$ exit
```

Edit the ingress rules to point the thirdpage service.

13. `student@ckad-1:~/app2$ kubectl edit ingress ingress-test`

ingress test

```
1  ....
2    - host: www.example.com
3      http:
4        paths:
5        - backend:
```

```yaml
            serviceName: secondapp
            servicePort: 80
          path: /
    - host: thirdpage.org              #<-- Add this and six following lines
      http:
        paths:
        - backend:
            serviceName: thirdpage
            servicePort: 80
          path: /
  status:
  ....
```

14. Test the second `Host:` setting using **curl** locally as well as from a remote system, be sure the `<title>` shows the non-default page. Use the main IP of either node.

```
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>
```

15. The **Traefik.io** ingress controller also presents a dashboard which allows you to monitor basic traffic. From your local system open a browser and navigate to the public IP of your master node with a like this `<YOURPUBLICIP>:8080/dashboard/`. The trailing slash makes a difference.

    Follow the `HEALTH` and `PROVIDERS` links at the top, as well as the the node IP links and you can view traffic when you reference the pages, from inside or outside the node. Typo the domain names inside the **curl** command and you can also see `404 error traffic`. Explore as time permits.

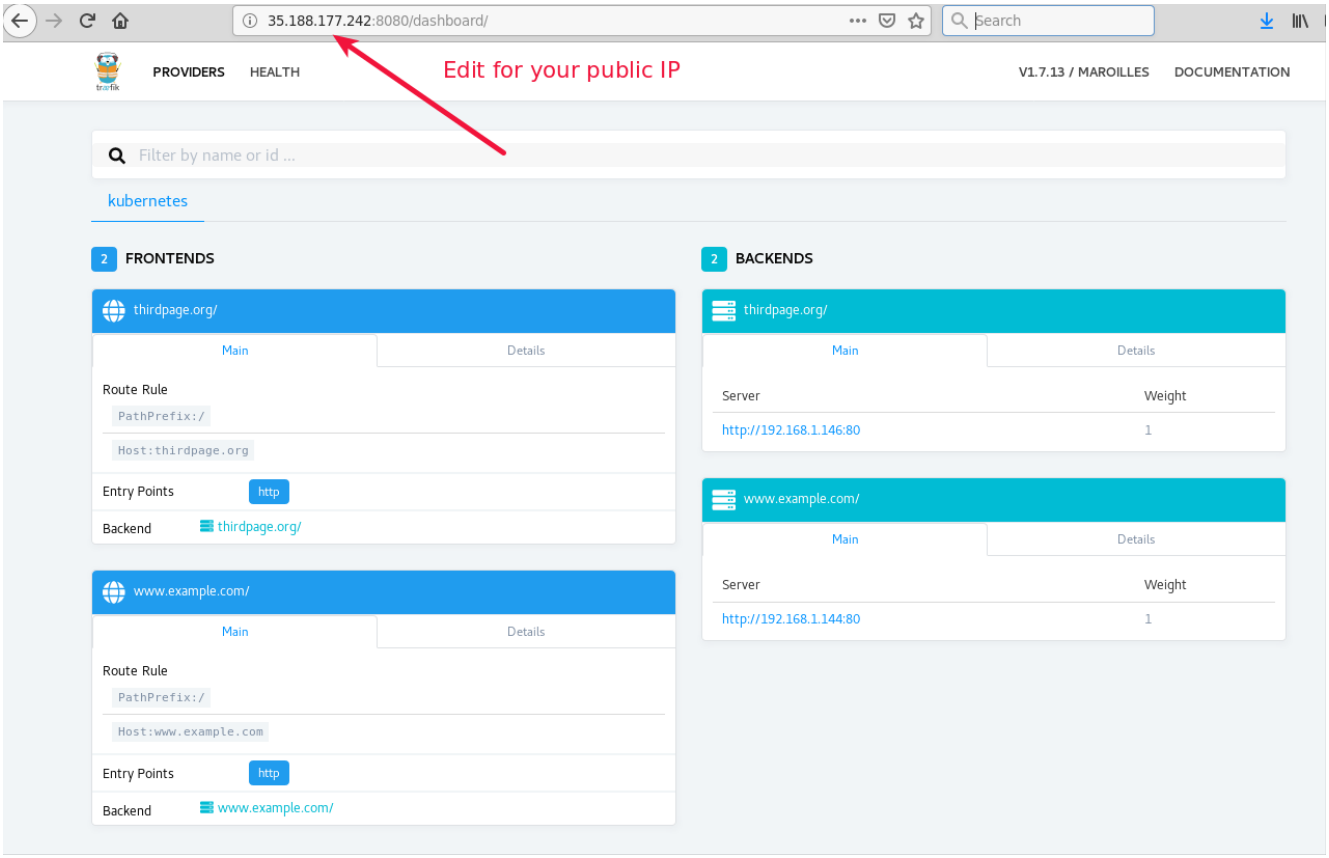

Figure 7.4: **Accessing the API**

```
student@ckad-1:~/app2$ curl -H "Host: thirdpage.org" http://10.128.0.7/

student@ckad-1:~/app2$ curl -H "Host: nopage.net" http://10.128.0.7/
```