Exercise 11.3: Gateway API

1. First step is to Deploy NGINX Gateway Fabric, To install the Gateway API resources, execute the following command

```
student@cp:~$ kubectl kustomize "https://github.com/nginx/nginx-gateway-fabric/\
               config/crd/gateway-api/standard?ref=v1.6.1" | kubectl apply -f -
```

```
customresourcedefinition.apiextensions.k8s.io/gatewayclasses.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/gateways.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/grpcroutes.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/httproutes.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/referencegrants.gateway.networking.k8s.io created
```

2. Deploy the NGINX Gateway Fabric CRDs by executing the following command

```
student@cp:~$ kubectl apply -f https://raw.githubusercontent.com/nginx/\
               nginx-gateway-fabric/v1.6.1/deploy/crds.yaml
```

```
customresourcedefinition.apiextensions.k8s.io/clientsettingspolicies.gateway.nginx.org created
customresourcedefinition.apiextensions.k8s.io/nginxgateways.gateway.nginx.org created
customresourcedefinition.apiextensions.k8s.io/nginxproxies.gateway.nginx.org created
customresourcedefinition.apiextensions.k8s.io/observabilitypolicies.gateway.nginx.org created
customresourcedefinition.apiextensions.k8s.io/snippetsfilters.gateway.nginx.org created
customresourcedefinition.apiextensions.k8s.io/upstreamsettingspolicies.gateway.nginx.org created
```

3. Deploy the NGINX Gateway Fabric by executing the following command

```
student@cp:~$ kubectl apply -f https://raw.githubusercontent.com/nginx/\
              nginx-gateway-fabric/v1.6.1/deploy/default/deploy.yaml
```

```
namespace/nginx-gateway created
serviceaccount/nginx-gateway created
clusterrole.rbac.authorization.k8s.io/nginx-gateway created
clusterrolebinding.rbac.authorization.k8s.io/nginx-gateway created
configmap/nginx-includes-bootstrap created
service/nginx-gateway created
deployment.apps/nginx-gateway created
gatewayclass.gateway.networking.k8s.io/nginx created
nginxgateway.gateway.nginx.org/nginx-gateway-config created
```

4. Verify the NGINX Gateway Fabric is running by executing the following command

```
student@cp:~$ kubectl get all -n nginx-gateway
```

```
NAME                                 READY   STATUS    RESTARTS   AGE
pod/nginx-gateway-7c59cd4cc6-5m84b   2/2     Running   0          92s

NAME                    TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)
↪   AGE
```

```
service/nginx-gateway   LoadBalancer   10.111.70.106   <pending>     80:32500/TCP,443:32730/TCP
↪  92s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-gateway   1/1     1            1           92s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-gateway-7c59cd4cc6   1         1         1       92s
```

5. The NGINX Gateway Fabric service is of the type of LoadBalancer service. The external IP is pending, Kubernetes will randomly allocate two ports on every node of the cluster. To access the NGINX Gateway Fabric, use an IP address of any node of the cluster along with the two allocated ports. For simplicty sake, we can patch the service to be of NodePort.

```
student@cp:~$ kubectl patch service/nginx-gateway -n nginx-gateway -p '{"spec": {"type": "NodePort"}}'
student@cp:~$ kubectl get service/nginx-gateway -n nginx-gateway
```

```
service/nginx-gateway patched
NAME            TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
nginx-gateway   NodePort   10.111.70.106   <none>        80:32500/TCP,443:32730/TCP   11m
```

6. Deploy an application and create a service named books. Then, define two Gateway API resources—a gateway and an HTTPRoute. These components work together to establish a routing rule that captures all HTTP requests for the hostname shop.example.com and directs them to the books service.

```
student@cp:~$ cp /home/student/LFS258/SOLUTIONS/s_11/books.yaml .
```

```
student@cp:~$ vim books.yaml
```

books.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: books
spec:
  replicas: 2
  selector:
    matchLabels:
      app: books
  template:
    metadata:
      labels:
        app: books
    spec:
      containers:
      - name: books
        image: nginx
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: books
spec:
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: books
```

```
student@cp:~$ kubectl create -f books.yaml
```

```
deployment.apps/books created
service/books created
```

7. Verify the books delploytment and service are created and running fine

```
student@cp:~$ kubectl get svc/books deploy/books
```

```
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/books   ClusterIP   10.111.164.248   <none>        80/TCP    2m57s

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/books   2/2     2            2           2m57s
```

8. To route traffic to the books application, we will create a gateway and HTTPRoute. We need a gateway to create an entry point for HTTP traffic coming into the cluster. The shop gateway we are going to create will open an entry point to the cluster on port 80 for HTTP traffic.

```
student@cp:~$ cp /home/student/LFS258/SOLUTIONS/s_11/gateway.yaml .
```

```
student@cp:~$ vim gateway.yaml
```

gateway.yaml

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: shop
spec:
  gatewayClassName: nginx
  listeners:
  - name: http
    port: 80
    protocol: HTTP
```

```
student@cp:~$ kubectl create -f gateway.yaml
```

```
gateway.gateway.networking.k8s.io/shop created
```

9. Verify the shop Gateway API resource has been deployed.

```
student@cp:~$ kubectl get gateway
```

```
NAME   CLASS   ADDRESS   PROGRAMMED   AGE
shop   nginx             True         2m9s
```

10. To route HTTP traffic from the gateway to the books service, we need to create an HTTPRoute named books and attach it to the gateway. This HTTPRoute will have a single routing rule that routes all traffic to the hostname “shop.example.com” from the gateway to the books service. Once NGINX Gateway Fabric processes the shop gateway and books HTTPRoute, it will configure its data plane (NGINX) to route all HTTP requests sent to “shop.example.com” to the pods that the books service targets.

```
student@cp:~$ cp /home/student/LFS258/SOLUTIONS/s_11/httproute.yaml .
```

```
student@cp:~$ vim httproute.yaml
```

httproute.yaml

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: books
spec:
  parentRefs:
  - name: shop
  hostnames:
  - "shop.example.com"
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /
    backendRefs:
    - name: books
      port: 80
```

```
student@cp:~$ kubectl create -f httproute.yaml
```

```
httproute.gateway.networking.k8s.io/books created
```

11. Verify the httproute has been deployed.

```
student@cp:~$ kubectl get httproute
```

```
NAME    HOSTNAMES              AGE
books   ["shop.example.com"]   35s
```

12. The Gateway API and HTTPRoute have been deployed succesfully, test the configuration by sending a request to the Node IP and node port of the NGINX gateway Fabric. First, Lets send a request to the path ’/’. Since the shop HTTPRoute routes all traffic on any path to the books application, the following requests should also be handled by the books pods.

```
student@cp:~$ curl --resolve shop.example.com:32500:10.2.0.30 http://shop.example.com:32500/
```

```
<!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>

...
<output_omitted>
```

13. Requests to hostnames other than “shop.example.com” should not be routed to the books application, since the books HTTPRoute only matches requests with the “shop.example.com” hostname. To verify this, send a request to the hostname “test.example.com”:

```
student@cp:~$ curl --resolve test.example.com:32500:10.2.0.30 http://test.example.com:32500/
```

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```