Exploring (Gateway API)

As Kubernetes continues to evolve, so do the networking challenges that come with managing complex, distributed microservices at scale. Traditional tools like the **Ingress API** have long been the backbone for exposing services to external traffic, but as environments grow more sophisticated, developers and operators are looking for greater flexibility, extensibility, and fine-grained control over network traffic.

The Evolution of Kubernetes Networking

Earlier, the **Ingress resource** was introduced as a way to manage external access to services in a cluster. While Ingress served its purpose, it had limitations:

1. Limited support for complex routing scenarios
2. Lack of support for non-HTTP protocols
3. Difficulty in extending functionality across different implementations

Recognizing these challenges, the Kubernetes community began work on the Gateway API in 2019. The Gateway API was designed to address the limitations of Ingress while providing a more flexible, extensible, and powerful way to manage traffic routing in Kubernetes.

Key features of the Gateway API include:

1. **Improved Resource Model**: The API introduces new custom resources like GatewayClass, Gateway, and Route (HTTPRoute, TCPRoute, etc.), providing a more granular and expressive way to define routing rules.
2. **Protocol Agnostic**: Unlike Ingress, which was primarily designed for HTTP, the Gateway API supports multiple protocols, including TCP, UDP, and TLS.
3. **Enhanced Security**: Built-in support for TLS configuration and more granular access control.
4. **Cross-Namespace Support**: Ability to route traffic to services in different namespaces, enabling more flexible architectures.
5. **Extensibility**: The API is designed to be easily extended with custom resources and policies.
6. **Role-Oriented**: Clear separation of concerns between cluster operators, application developers, and security teams.

Core Components of the Gateway API

Here are the main components of the Gateway API and their roles:

1. GatewayClass

- **What It Is**: Like how a **StorageClass** defines different types of storage, a **GatewayClass** defines how networking infrastructure (like load balancers, proxies, etc.) is provisioned and managed.

- **Role**: It represents the overall configuration of a "type" of Gateway and is used by an administrator to define specific implementation details for that Gateway. It abstracts the specific network configuration like whether the load balancer is cloud-based, on-premises, or a service mesh.
- **Practical Use**: When creating a Gateway, we must specify the GatewayClass as we want the Gateway to use.

2. Gateway

- **What It Is**: The **Gateway** resource describes the specific details of the networking infrastructure being used, such as the ports it listens on and the protocol it supports.
- **Role**: It serves as the logical entry point for traffic into the cluster. Gateways can listen for HTTP(S), TCP, UDP, and other protocols. It binds to a specific GatewayClass, which dictates the underlying infrastructure (e.g., load balancer, reverse proxy).
- **Practical Use**: The Gateway can specify one or more listeners (for different protocols like HTTP or HTTPS) and act as a load balancer, receiving external traffic and directing it based on rules defined in HTTPRoute, TCPRoute, etc.

3. Route Resources

Routes are the rules that define how traffic is routed to backend services once it enters through a Gateway. They allow for complex traffic routing and splitting logic.

My Setup Components

1. **Deployment & Service (app.yaml)**

- Deploys an HTTP echo server inside a namespace demo.
- Exposes it via a Kubernetes Service.

2. **Gateway Class (gateway-api.yaml)**

- GatewayClass: Defines which controller to use. In my case: **projectcontour.io/gateway-controller.**
- Gateway: Specifies port 80, listens for HTTP traffic, and is bound to the GatewayClass.
- HTTPRoute: Maps HTTP paths to my web service.

How It Works (Workflow)

Here's the step-by-step flow:

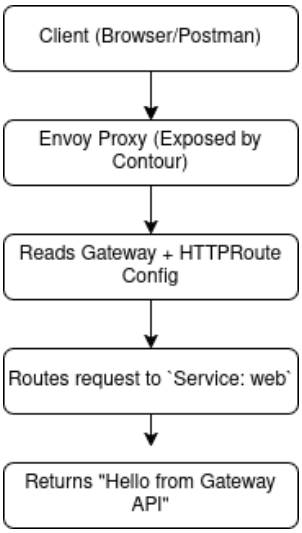
1. **GatewayClass**
    a. create a GatewayClass and specify the controller.
    b. Here, the controller is projectcontour.io/gateway-controller.

2. **Gateway**
    a. A Gateway object is created in the namespace (demo) that refers to the GatewayClass.
    b. It listens for incoming HTTP requests on port 80.
3. **HTTPRoute**
    a. A HTTPRoute object tells the Gateway how to route traffic.
    b. It forwards traffic to the Kubernetes Service named web.
4. **Contour**
    a. Contour is the actual implementation (controller) that watches for GatewayClass, Gateway, and HTTPRoute resources.
    b. It sets up an Envoy proxy that handles routing based on your Gateway API configs.

Contour is preferred because:

- It supports the Gateway API natively.
- Comes with an easy-to-use **Envoy proxy** backend.
- No need for separate CRDs/configs like NGINX requires advanced features.

WORK-FLOW-DAIGRAM



app.yaml

```yaml
# app.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: demo

---
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: demo
spec:
  selector:
    app: web
  ports:
    - port: 80
      targetPort: 5678
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  namespace: demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: hashicorp/http-echo
          args:
            - "-text=Hello from Gateway API"
          ports:
            - containerPort: 5678
```

gateway-api.yaml

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: example-gatewayclass
spec:
  controllerName: projectcontour.io/gateway-controll



---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: example-gateway
  namespace: demo
spec:
  gatewayClassName: example-gatewayclass
  listeners:
    - name: http
      port: 80
      protocol: HTTP
      allowedRoutes:
        namespaces:
          from: Same


---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: web-route
  namespace: demo
spec:
  parentRefs:
  - name: example-gateway
  rules:
  - backendRefs:
    - name: web
      port: 80

```

Result

✅ 1. Check if Minikube is Running

```
xs-514ansana@xs-host603-037:~$ sudo minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

xs-514ansana@xs-host603-037:~$
```

✅ 2. Check All Pods in All Namespaces

```
xs-514ansana@xs-host603-037:~$ sudo kubectl get pods -A
[sudo] password for xs-514ansana:
NAMESPACE        NAME                                        READY   STATUS      RESTARTS      AGE
demo             contour-example-gateway-6bc6948887-7ns4d    1/1     Running     2 (95m ago)   24h
demo             contour-example-gateway-6bc6948887-frwsx    1/1     Running     2 (95m ago)   24h
demo             debug                                       1/1     Running     0             14h
demo             debug-container                             1/1     Running     0             14h
demo             envoy-example-gateway-xj87g                 2/2     Running     2 (96m ago)   24h
demo             web-dcf45dc88-jf8mn                         1/1     Running     1 (96m ago)   24h
kube-system      coredns-668d6bf9bc-4b6b2                    1/1     Running     1 (96m ago)   25h
kube-system      coredns-668d6bf9bc-gmvbf                    1/1     Running     1 (96m ago)   25h
kube-system      etcd-minikube                               1/1     Running     1 (96m ago)   25h
kube-system      kube-apiserver-minikube                     1/1     Running     1 (96m ago)   25h
kube-system      kube-controller-manager-minikube            1/1     Running     1 (96m ago)   25h
kube-system      kube-proxy-ckzp6                            1/1     Running     1 (96m ago)   25h
kube-system      kube-scheduler-minikube                     1/1     Running     1 (96m ago)   25h
kube-system      storage-provisioner                         1/1     Running     3 (95m ago)   25h
projectcontour   contour-5579599456-5btn5                    1/1     Running     2 (95m ago)   24h
projectcontour   contour-5579599456-mscsw                    1/1     Running     2 (95m ago)   24h
projectcontour   contour-certgen-v1-30-3-j7mdj               0/1     Completed   0             24h
projectcontour   contour-gateway-provisioner-6d58f9ff8d-d2tlc 1/1     Running     1 (96m ago)   24h
projectcontour   envoy-b5qbt                                 2/2     Running     2 (96m ago)   24h
xs-514ansana@xs-host603-037:~$
```

✅ 3. Check Services and Deployments

```
xs-514ansana@xs-host603-037:~$ sudo kubectl get svc -A
sudo kubectl get deploy -A
NAMESPACE        NAME                      TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
default          kubernetes                ClusterIP      10.96.0.1        <none>        443/TCP                      25h
demo             contour-example-gateway   ClusterIP      10.96.41.67      <none>        8001/TCP                     24h
demo             envoy-example-gateway     LoadBalancer   10.103.147.7     127.0.0.1     80:31044/TCP                 24h
demo             web                       ClusterIP      10.108.162.236   <none>        80/TCP                       24h
kube-system      kube-dns                  ClusterIP      10.96.0.10       <none>        53/UDP,53/TCP,9153/TCP       25h
projectcontour   contour                   ClusterIP      10.97.132.204    <none>        8001/TCP                     24h
projectcontour   envoy                     LoadBalancer   10.105.246.117   127.0.0.1     80:31723/TCP,443:32680/TCP   24h
NAMESPACE        NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
demo             contour-example-gateway       2/2     2            2           24h
demo             web                           1/1     1            1           24h
kube-system      coredns                       2/2     2            2           25h
projectcontour   contour                       2/2     2            2           24h
projectcontour   contour-gateway-provisioner   1/1     1            1           24h
xs-514ansana@xs-host603-037:~$
```

✅ 4. Check Gateway API Resources

```
xs-514ansana@xs-host603-037:~$ sudo kubectl get gatewayclass
NAME                   CONTROLLER                                   ACCEPTED   AGE
example-gatewayclass   projectcontour.io/gateway-controller   True       24h
xs-514ansana@xs-host603-037:~$ sudo kubectl get gateway -n demo
NAME              CLASS                  ADDRESS     PROGRAMMED   AGE
example-gateway   example-gatewayclass   127.0.0.1   True         24h
xs-514ansana@xs-host603-037:~$ sudo kubectl get httproute -n demo
NAME        HOSTNAMES   AGE
web-route               24h
xs-514ansana@xs-host603-037:~$
```

✅ 5. Find the envoy service:

```
xs-514ansana@xs-host603-037:~$ sudo kubectl get svc -n projectcontour
NAME      TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
contour   ClusterIP      10.97.132.204    <none>        8001/TCP                     25h
envoy     LoadBalancer   10.105.246.117   127.0.0.1     80:31723/TCP,443:32680/TCP   25h
xs-514ansana@xs-host603-037:~$
```

✅ 5. Port forwarding:

```
xs-514ansana@xs-host603-037:~$ sudo kubectl port-forward -n demo service/web 8080:80
Forwarding from 127.0.0.1:8080 -> 5678
Forwarding from [::1]:8080 -> 5678
Handling connection for 8080
```