Gateway API

Advanced networking configuration in Kubernetes

Francesco Cappa
Politecnico di Torino

Politecnico di Torino

Challenges with Ingress

- Limited protocol support
  - Only HTTP and HTTPS
- Complex routing requirements
  - Basic routing based on hostnames and paths.
- Customization and Extensibility
  - Ingress and ingress controller behaviour could only be "adjusted" through IngressClass
- Standardization
  - Features and capabilities depend on controller implementation
  - Ecosystem adoption

Gateway API & design principles

- Gateway APIs make network services available by using an extensible, role-oriented, protocol-aware configuration mechanism.
  - Gateway API is an add-on containing API kinds that provide dynamic infrastructure provisioning and advanced traffic routing.
- Design principles:
  - **Role-oriented:** Gateway API kinds are modeled after organizational roles that are responsible for managing Kubernetes service networking:
    - **Infrastructure Provider**
    - **Cluster Operator**
    - **Application Developer**
  - **Portable:** Gateway API specifications are defined as custom resources and are supported by many implementations.
  - **Expressive:** Gateway API kinds support functionality for common traffic routing use cases such as header-based matching, traffic weighting, and others that were only possible in Ingress by using custom annotations.
  - **Extensible:** Gateway allows for custom resources to be linked at various layers of the API. This makes granular customization possible at the appropriate places within the API structure.

Resource model

- Gateway API has three stable API kinds:
  - **GatewayClass:** Defines a set of gateways with common configuration and managed by a controller that implements the class.
  - **Gateway:** Defines an instance of traffic handling infrastructure, such as cloud load balancer.
  - **HTTPRoute:** Defines HTTP-specific rules for mapping traffic from a Gateway listener to a representation of backend network endpoints. These endpoints are often represented as a Service.

GatewayClass

- Gateways can be implemented by different controllers, often with different configurations.
- A Gateway must reference a GatewayClass that contains the name of the controller that implements the class.
- See the GatewayClass reference for a full definition of this API kind.

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: example-class
spec:
  controllerName: example.com/gateway-controller
```

Gateway

- A Gateway describes an instance of traffic handling infrastructure.
- It defines a network endpoint that can be used for processing traffic, i.e. filtering, balancing, splitting, etc. for backends such as a Service.
- Example:
  - an instance of traffic handling infrastructure is programmed to listen for HTTP traffic on port 80.
  - Since the `addresses` field is unspecified, an address or hostname is assigned to the Gateway by the implementation's controller.
  - This address is used as a network endpoint for processing traffic of backend network endpoints defined in routes.
- See the Gateway reference for a full definition of this API kind.

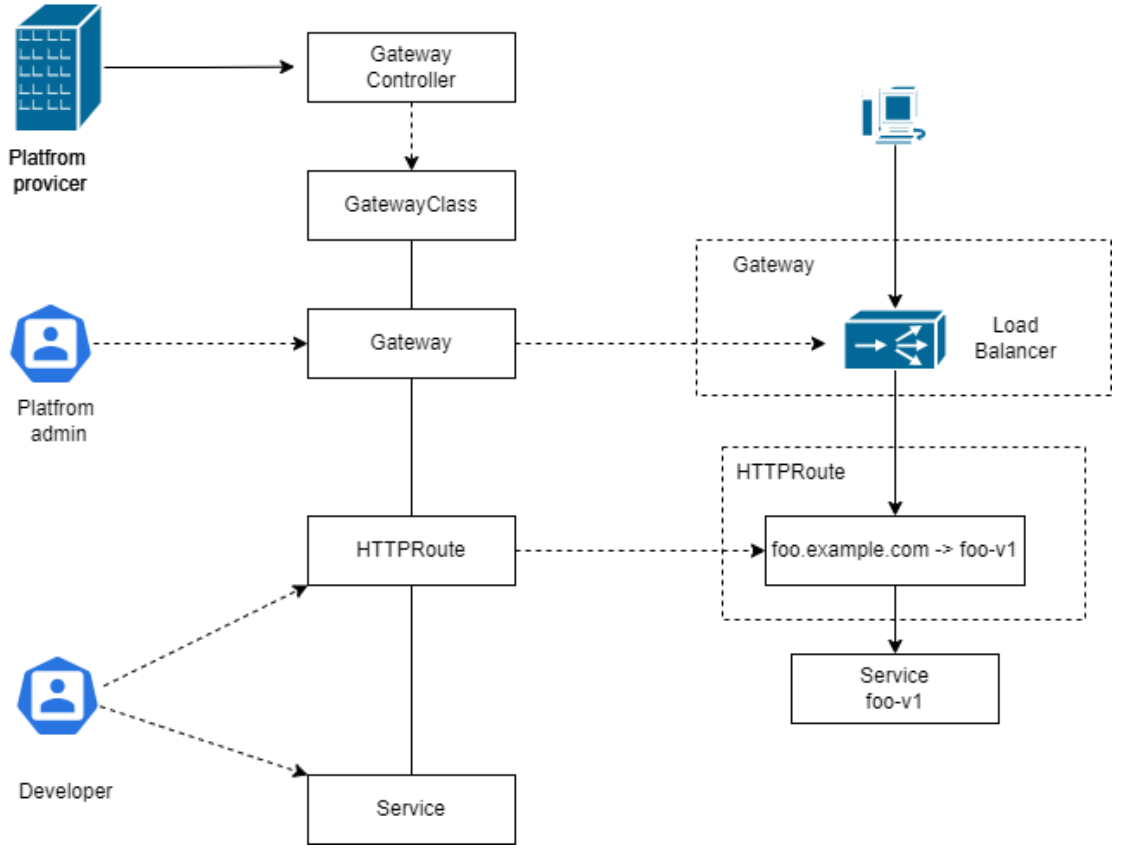
```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: example-gateway
spec:
  gatewayClassName: example-class
  listeners:
  - name: http
    protocol: HTTP
    port: 80
```

HTTPRoute

- The HTTPRoute kind specifies routing behavior of HTTP requests from a Gateway listener to backend network endpoints.
- An HTTPRoute represents configuration that is applied to the underlying Gateway implementation. For example, defining a new HTTPRoute may result in configuring additional traffic routes in a cloud load balancer or in-cluster proxy server.
- See the HTTPRoute reference for a full definition of this API kind.

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-httproute
spec:
  parentRefs:
  - name: example-gateway
  hostnames:
  - "www.example.com"
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /login
    backendRefs:
    - name: example-svc
      port: 8080
```

High-level schema



Example: HTTP header-based matching

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: bar-route
spec:
  parentRefs:
  - name: example-gateway
  hostnames:
  - "bar.example.com"
  rules:
  - matches:
    - headers:
      - type: Exact
        name: env
        value: canary
    backendRefs:
    - name: bar-svc-canary
      port: 8080
  - backendRefs:
    - name: bar-svc
      port: 8080
```

Example: HTTP redirect

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: http-filter-redirect
spec:
  hostnames:
    - redirect.example
  rules:
    - filters:
        - type: RequestRedirect
          requestRedirect:
            scheme: https
            statusCode: 301
```

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: http-filter-redirect
spec:
  hostnames:
    - redirect.example
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /cayenne
      filters:
        - type: RequestRedirect
          requestRedirect:
            path:
              type: ReplaceFullPath
              replaceFullPath: /paprika
            statusCode: 302
```

Example: HTTP rewrite

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: http-filter-rewrite
spec:
  hostnames:
    - rewrite.example
  rules:
    - filters:
        - type: URLRewrite
          urlRewrite:
            hostname: elsewhere.example
      backendRefs:
        - name: example-svc
          weight: 1
          port: 80
```

Example: HTTP request header modification

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: header-http-echo
spec:
  parentRefs:
    - name: acme-gw
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /add-a-request-header
      filters:
        - type: RequestHeaderModifier
          requestHeaderModifier:
            add:
              - name: my-header-name
                value: my-header-value
      backendRefs:
        - name: echo
          port: 8080
```

```yaml
filters:
- type: RequestHeaderModifier
  requestHeaderModifier:
    set:
      - name: my-header-name
        value: my-new-header-value
```

```yaml
filters:
- type: RequestHeaderModifier
  requestHeaderModifier:
    remove: ["x-request-id"]
```

Example: HTTP traffic splitting

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: simple-split
spec:
  rules:
  - backendRefs:
    - name: foo-v1
      port: 8080
      weight: 90
    - name: foo-v2
      port: 8080
      weight: 10
```

Questions