Request Filtering Between Gateways and Namespaced Routes

Relevant Issues

- https://github.com/kubernetes-sigs/gateway-api/issues/634
- https://github.com/kubernetes-sigs/gateway-api/issues/411

A method for Gateway administrators to use routing rules to filter which traffic can be forwarded to downstream routes.

As a Gateway administrator managing an HTTP and/or TLS (SNI) load balancer I would like to delegate particular path prefixes, headers, as well as subdomains to namespaced HTTPRoutes/TLSRoutes to prevent namespace administrators from writing rules that conflict with the rules from other namespaces.

Path Prefix

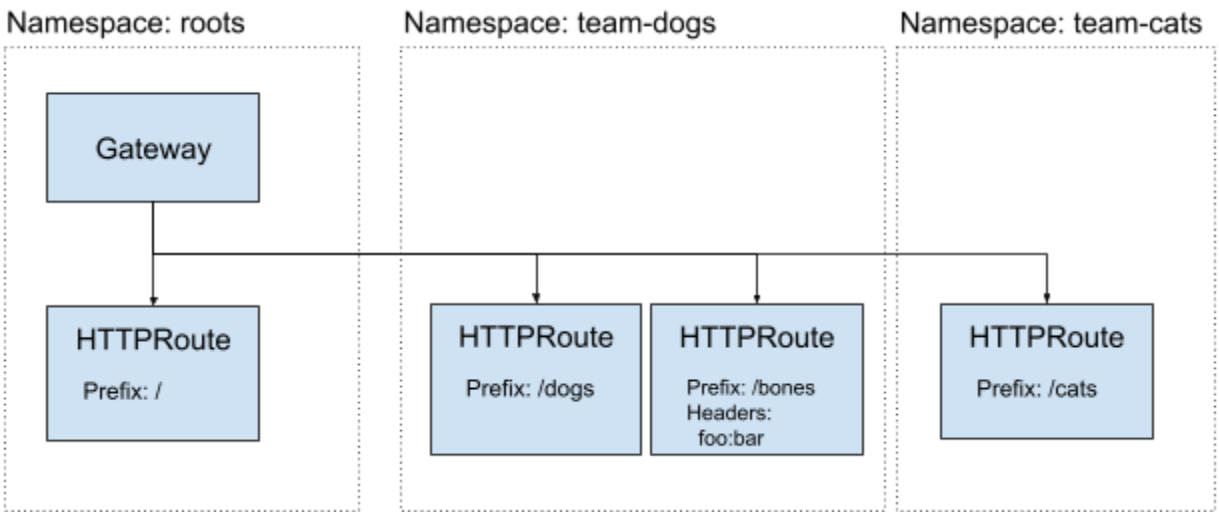
I should be able to authorize the path prefix `/cats` to the cats namespace and the path prefix `/dogs` to the dogs namespace. Then the administrators for each of those namespaces can safely write any rule they want, but with the expectation that the path rules are prefixed with /cats or /dogs, /cats/food and /dogs/food would never collide because the Gateway administrator had already pre-filtered the traffic.

Headers

I should be able to delegate any number of headers to match against the cats namespace and a different set of headers to the dogs namespace. Then the administrators for each of those namespaces can safely write any rule they want, but with the expectation that the path rules are prefixed with their respective headers.

DNS Domain/Subdomain

I manage `example.com` and authorize `team1.example.com` to a namespace. That team then had full authority over the `team1.example.com` fqdn and could further authorize other domains to another namespace, `app-a.team1.example.com` for example.



Existing Implementations

Contour HTTPProxy

Contour has implemented a CRD named “HTTPProxy” which implements a top-down approach to solving this problem. (IngressRoute)

In HTTPProxy, there are effectively two types of objects:

- “Root" HTTPProxy: Specifies an FQDN
- "Child" HTTPProxy: Can’t specify a FQDN

An `Include` defines a set of path prefixes and/or headers which will be applied to the **child** route when building out configuration. At processing time, all the “includes” that are configured for a resource are applied, then the resulting config is passed off.

The following example shows a “root” HTTPProxy that routes traffic for `foo.bar.com`. Requests to `foo.bar.com/blog` will be handled by the `childroute` HTTPProxy in the namespace `child`. Requests to `foo.bar.com/blog/info` will be handled by the `childroute` HTTPProxy in the namespace `child`. Any other request other than `/blog` or `/blog/info` will be handled by the root HTTPProxy in the `roots` namespace.

Any other HTTPProxy who attempts to configure `foo.bar.com/blog` will be rejected.

```
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: root
  namespace: roots
spec:
  virtualhost:
    fqdn: root.bar.com
    tls:
      secretName: mysecret
  includes:
  - name: childroute
    namespace: child
    conditions:
    - prefix: /blog
  routes:
  - services:
    - name: rootservice
      port: 80
    conditions:
    - prefix: /
---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: childroute
  namespace: child
spec:
  routes:
  - services:
    - name: service1
      port: 80
    conditions:
    - prefix: / # ← path prefix `/blog` (IngressRoute define /blog)
  - services:
    - name: service2
      port: 80
    conditions:
    - prefix: /info  # ← path prefix is `/blog/info`
      - services:
        - name: service
          port: 80
```

Referencing (including) a "root" HTTPProxy inside another root is an error, the include will be marked as invalid.

Issues

- Requires admins to manage the “roots”
- Keyed off of route name which must be known to the root setting up the inclusion

Existing Conversations:

- https://github.com/projectcontour/contour/issues/2206
- https://github.com/projectcontour/contour/issues/2645
- https://github.com/projectcontour/contour/issues/2189

Solutions

A Route can forward to another Route (ref)

Change the spec to allow `ForwardTo` to name another Route resource.

Benefits

- Easy to add to the spec, no new objects or complex rules required

Issues

- Routing Cycles: Difficult to detect manually
- Verify each route is compatible with each other: Type/Protocol
- Some Route fields may not make sense when included (for example, what do hostname fields in included HTTPRoutes do?

Create a new “Delegated/Included/*name*” Route type

This allows for a new type to be explicitly referred to when configuring such that it’s clear that a delegation model is being implemented.

Benefits

- Can simplify structure to only include fields relevant to Delegated routes
- Makes the different models very clear

Issues

- Duplication of code
- Adds unneeded user confusion as to why each type is being used
- Will require a DelegatedXRoute for each Route that supports delegation, probably. So, DelegatedHTTPRoute, DelegatedTLSRoute, etc.

Add limited redirection capability (ref)

Add a validation requirement that `*Route.ForwardTo` may only select another `*Route` provided that the destination route is terminal (i.e. doesn't forward to another route).

Benefits

- Keeps the current API, no new objects required.

Issues

- Still requires a lot of checking of included/delegate Route fields (Doesn't answer "what do included hostname or TLS fields do?"

HTTP Listeners have prefix