Kubernetes Gateway selection for Istio in Knative

Shared with Istio Community

Owner(s): *howardjohn* **Working Group:** *Networking*	**Status**: *WIP	In Review |* **Approved** *| Obsolete |* **Delayed** **Created**: *04/01/2021* **Approvers**: *Networking*

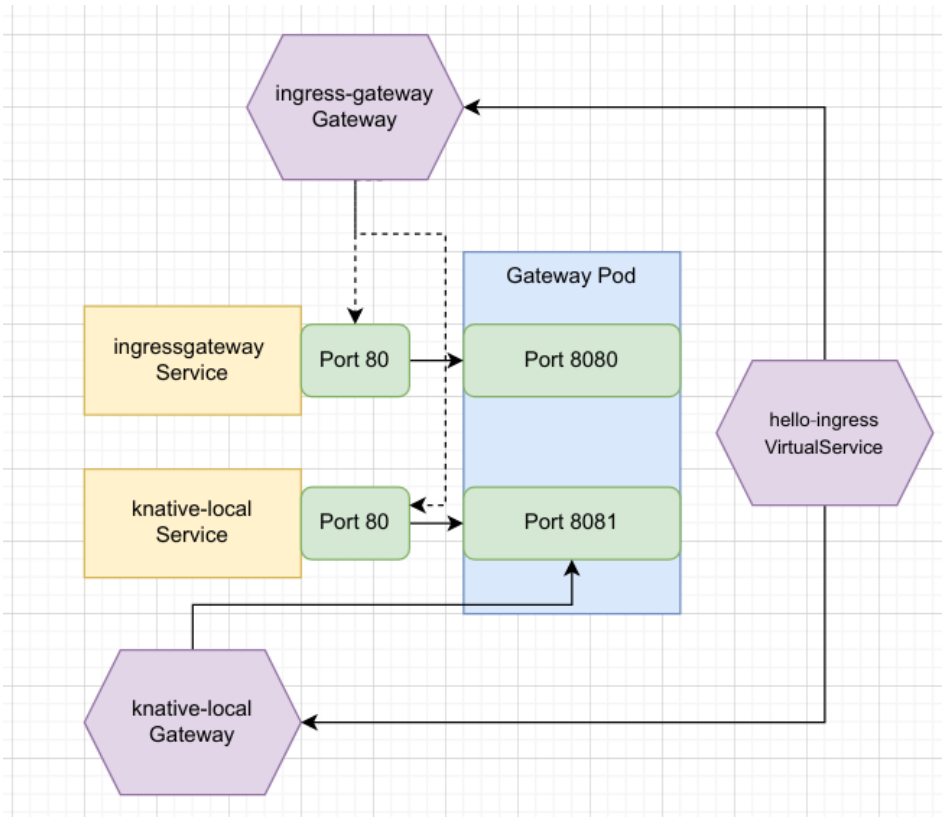
Background

Istio Gateway's associate with workloads by a `selector` field, which selects pod labels. The `ports` setting in the servers may vary depending on associated Services to handle `targetPort`. For example, Istio's default installation sets up port 80 with target 8080. When a user configures a `Gateway`, they should specify the port as 80; internally, Istiod translates this to a listener on port 8080 in Envoy.

However, when there is no associated Service port, the Gateway port will be used directly.

Knative

The following shows an example of a fully Knative setup. This is used as an example of issues with the current API.




```yaml
apiVersion: v1
kind: Service
metadata:
  name: istio-ingressgateway
  namespace: istio-system
spec:
  ports:
  - name: http2
    port: 80
    targetPort: 8080
  selector:
    istio: ingressgateway
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: knative-local-gateway
  namespace: istio-system
spec:
  ports:
  - name: http2
    port: 80
    targetPort: 8081
  selector:
    istio: ingressgateway
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
```

```yaml
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: knative-ingress-gateway
  namespace: knative-serving
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*'
    port:
      name: http
      number: 80
      protocol: HTTP
---
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  labels:
  name: knative-local-gateway
  namespace: knative-serving
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*'
    port:
      name: http
```

```
  name: hello-ingress
spec:
  gateways:
  - knative-serving/knative-ingress-gateway
  - knative-serving/knative-local-gateway
  hosts:
  - hello.default.example.com
  - hello.default
  - hello.default.svc
  - hello.default.svc.cluster.local
  http:
  - match:
    - authority:
        prefix: hello.default
      gateways:
      - knative-serving/knative-local-gateway
    route:
    - destination:
        host: hello-
00001.default.svc.cluster.local
  - match:
    - authority:
        prefix: hello.default.example.com
      gateways:
      - knative-serving/knative-ingress-gateway
    route:
    - destination:
        host: hello-
00001.default.svc.cluster.local
```

```
      number: 8081
      protocol: HTTP
```

This is problematic for a few reasons:

- We may associate port 80 with port 8080 or port 8081. This is a conflict resolved by Service creation order, which is not a good state.
- Gateway port selection impacts other usages. The `port` in Gateway will be used as a match in the vhost. So in the example above, there will be a vhost match on `hello.default.svc.cluster.local:8080`. This is undesirable because the port is likely hit over port 80 (note: we really shouldn't match port at all but we don't have a great way to handle that today, see https://github.com/istio/istio/issues/25350). Additionally, VirtualService has a `match.port` which associates with a Gateway's `port`. This port confusion makes the API inconsistent and hard to reason about.

KGateway API

Currently, the Gateway API has no solid way to actually associate a KGateway with an in-cluster proxy deployment. For our case, we need users to have some way to specify which set of Envoy's should actually be configured by the KGateway

KGateway Status

Another issue to fulfill is the KGateway status. The API expects us to fill in a list of addresses. Because we can have multiple Services associated with a single gateway deployment, this becomes complex to implement and confusing for a user.

This same problem occurs with Ingress today. It is "solved" by explicitly declaring which Service to look up the address for. All other Ingress controllers that I could find do the same. However, this is not suitable for us.

Requirements

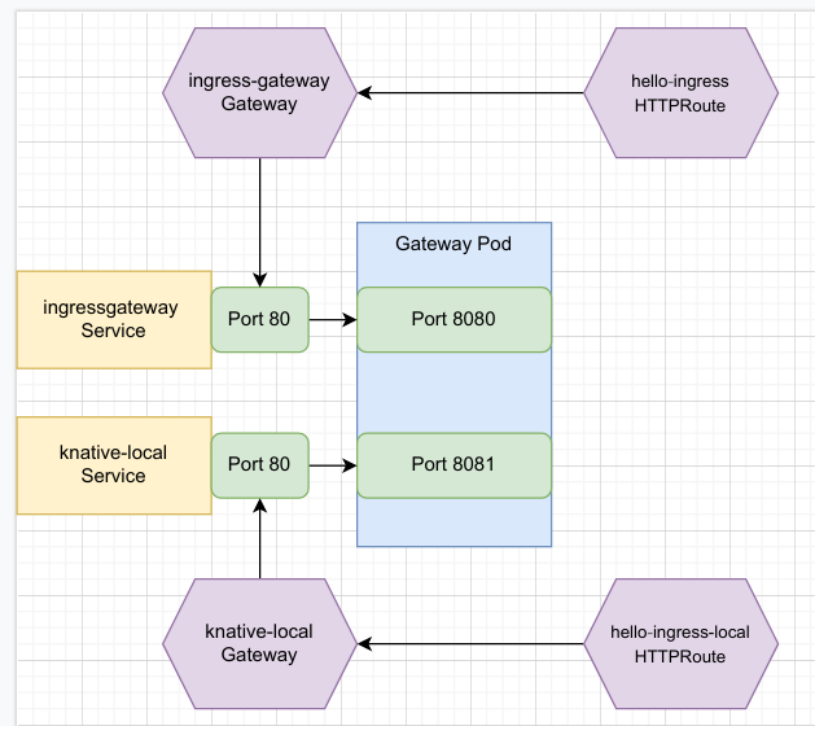
- Users should be able to expose multiple distinct Services over port 80 on the same gateway deployment, if they map those to different targetPorts.
- Ports should be consistent in the API - either referring to the service port or targetPort in all cases.
- Istio must implement the KGateway API, including the status fields
- KGateway must be able to select which envoy will handle the configuration
    - Selection must not use all-namespace selectors, which causes security and usability concerns

Design

This design focuses only on changing the Kubernetes Gateway API/Implementation. No changes to Istio APIs will be made. It attempts to kill 3 birds with one stone: implementing KGateway status, implementing selectors for KGateway, and resolving port conflict issues.

KGateway will add a new field, `gatewayService` or equivalent. If this field cannot be added to the API, it can be used as an annotation. With this, the user will declare what Service will handle the KGateway. The configuration will then apply to all gateway pods selected by that Service.

The above Knative example would look like this:




```yaml
apiVersion: networking.x-k8s.io/v1alpha1
kind: Gateway
metadata:
  name: knative-ingress-gateway
  namespace: istio-system
spec:
  gatewayClassName: istio
  gatewayService: "istio-ingressgateway"
  listeners:
  - port: 80
    protocol: HTTP
    routes:
      kind: HTTPRoute
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: Gateway
metadata:
  name: knative-local-gateway
  namespace: istio-system
spec:
  gatewayClassName: istio
  # Reference the knative-local-gateway Service
explicitly
  gatewayService: "knative-local-gateway"
  listeners:
  - port: 80 # Port refers to Service port of the
knative-local-gateway Service
    protocol: HTTP
    routes:
      kind: HTTPRoute
```

```yaml
apiVersion: networking.x-k8s.io/v1alpha1
kind: HTTPRoute
metadata:
  name: hello-ingress
  namespace: default
spec:
  gateways:
    allow: FromList
    gatewayRefs:
      - name: knative-ingress-gateway
        namespace: istio-system
  hostnames: ["hello.default.example.com"]
  rules:
  - forwardTo:
    - serviceName: hello-
00001.default.svc.cluster.local
      port: 80
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: HTTPRoute
metadata:
  name: hello-ingress-local
  namespace: default
spec:
  gateways:
    allow: FromList
    gatewayRefs:
      - name: knative-local-gateway
        namespace: istio-system
  hostnames: ["hello.default.svc.cluster.local"]
  rules:
  - forwardTo:
    - serviceName: hello-
```

```
00001.default.svc.cluster.local
      port: 80
```

This has a few benefits:

- All parts of the API refer to the Service port
- We are able to unambiguously refer to multiple Services selecting the same pods
- KGateway status is clear - we know exactly which Service we need to look to fill in the address section

Route names

One concern with this design is the route naming. EnvoyFilter API allows matching on route *name* or *port*. In order to support this design, we will need to refactor the route naming to deal with ambiguities of port names. In this process, we will keep compatibility with *port* matches (that is, the port refers to the port in Gateway.servers), but not the name.

Service	Service Port	Target Port	Old Name	New Name	Listener Name
a	80	8080	http.80	http.8080	0.0.0.0_8080
b	80	8080	http.80	http.8080	0.0.0.0_8080
c	81	8080	http.81	http.8080	0.0.0.0_8080
d	81	9090	http.81	http.9090	0.0.0.0_9090

Current behavior:

- A+B = `buildGatewayListeners: found 2 services on port 80`, which picks an arbitrary targetPort (8080 in either case here)
- C+D = `buildGatewayListeners: found 2 services on port 80`, which picks an arbitrary targetPort (could be 8080 or 9090 here)
- A+C = NACK, duplicate listener on 8080 created
- A+D = no conflict, all ports are distinct

Consensus:

- merged target port (a-c) is not safe, can accidentally expose things
    - But we need it for migration between services. we need explicit way to declare these as the "same" service
- agreement on the new name, but if we have a way to indicate the "canonical name" or "merge group", we should use that for the name
- We need to figure out how to declare merge as safe. Look into current behavior to decide what we should do.

One thing to note is that the listener and name route name are now 1:1, which removes potential conflicts between listeners and routes.

Internally, the EnvoyFilter route matching looks at the route name and checks if there is a port match for the API. To support this, we will set up a mapping of targetPort to service ports. For example:

```
{
  8080: [80, 81],
  9090: [81],
}
```

Which will result in EnvoyFilters matching:

- port 80: will match the http.8080 route, which corresponds to service a and b on port 80 **and c on port 81**.
- port 81: will match the http.8080 route, which corresponds to **service a and b on port 80** and c on port 81, as well as the http.9090

This may seem confusing that we have mismatches of ports selected. However, this is fundamentally how the routing works; when a request comes in to Envoy we only know the target port; we do not know which Service it came through (if any), so we cannot do further disambiguation.

Default Service

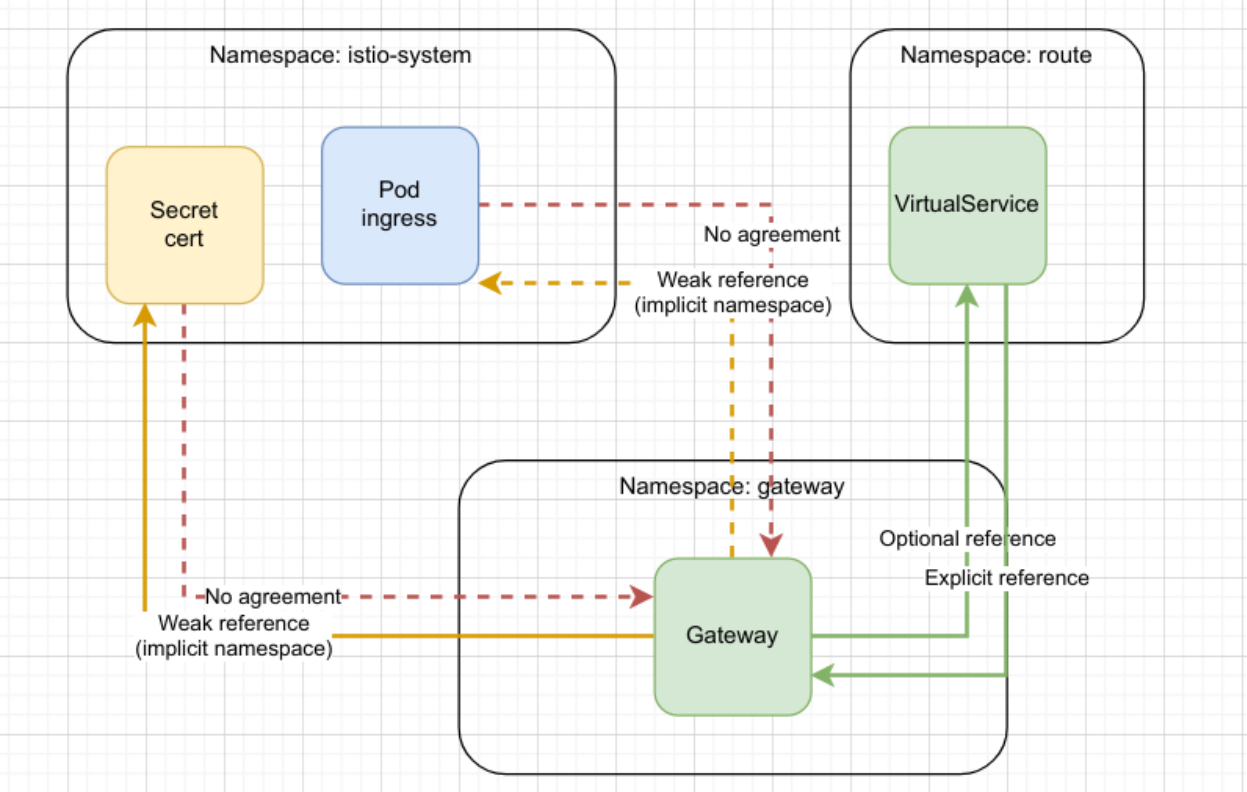
To support minimal Gateway configurations, a single default gateway service may be specified by a label on the Service like `gateway.istio.io/default-gateway: true`. This is similar to Default Ingress Class.

If there is a Gateway without any service specified, and there is a Service in the same namespace with this label, that service will be selected. If there are multiple Services with this label in the same namespace, we will choose the oldest one and emit a warning.

Namespace Isolation

Background

Istio's current API has issues with how references between API objects are managed. This can lead to unexpected (intentional or accidental) interference between namespaces.



Gateway and VirtualService have an acceptable handshake - VirtualService explicitly refers to a Gateway by name and namespace. Gateways *may* choose to limit the VirtualServices that bind to it by using the `namespace/hostname` format, although in practice this is not common.

The references to certificates and gateway deployments are much weaker. Gateway has a cross-namespace label selector, so any Pod can make itself selected, regardless of namespace. When this happens, the Secret reference is also implicitly changed, as it refers to the namespace of the Pod that is selected.

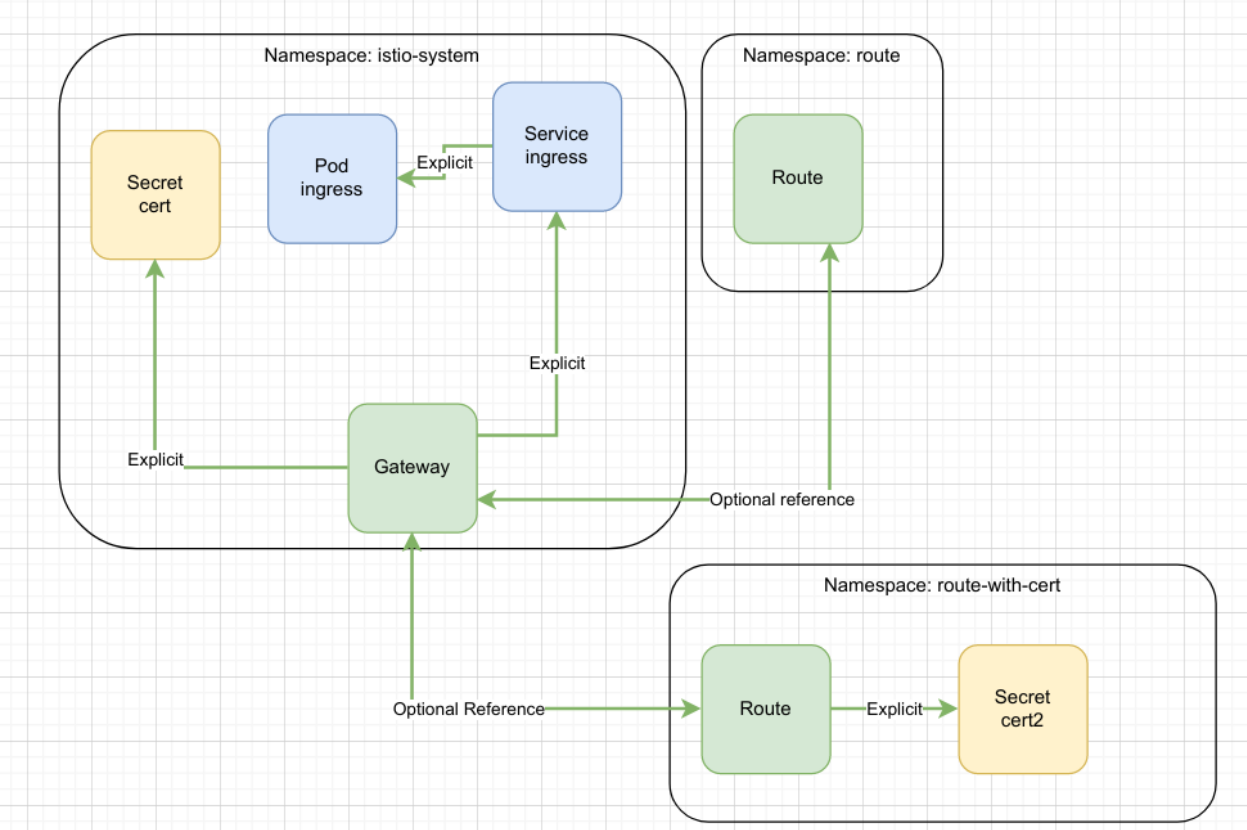
Additionally, a gateway pod cannot restrict which Gateways can bind to it. This can be fixed by the PILOT_SCOPE_GATEWAY_TO_NAMESPACE option, but it's onerous for users as it fully centralizes listener level concerns (such as certificates, etc) to the gateway admin.

Among other concerns, this is preventing us from allowing certificates to be stored in namespaces other than the gateway deployment namespace, which does not adhere to the spec. See Auth for Istiod SDS for more information.

Design

Gateway's will only be able to select Service's in their own namespace.

This changes the permission model to look like this:



Within a namespace, we do not need a "mutual handshake", as namespace is the trust boundary. We have the following one-way references, with the other direction providing implicit trust by the namespace:

- Gateway references Service (by name)
- Service references pod (by selector)
- Gateway references Secret (by name)

The only cross namespace references we have have a mutual handshake. Route and Gateway both have controls to form mutual agreement on selection.

Additionally, routes may override Gateway's certificates. This must be enabled in the Gateway's settings (per the existing spec). The route will explicitly reference a secret in its own namespace, which it is allowed to do since it is within its own namespace.

Because we have end to end trust, we are able to use this configuration to give authorization for Secret access. For example, in the config above, route-with-cert has declared that the `ingress Pod` can access the `cert2` Secret, which tells Istiod to authorize access to it. In the previous mode, because we do not have end to end trust, we cannot ensure there is mutual agreement throughout the entire API chain to provide this authorization.

The main concern with this is breaking use cases that intend to delegate control to application namespaces, rather than having centralized control in the `istio-system` namespace. There are a few use cases:

- Allow app namespace to dynamically control a port. This requires changes to the Service anyways, so istio-system access is already required.
- Allow app namespaces to control arbitrary HTTP ports. This is not a secure default. With the API changes, the gateway admin may choose to allow this with a match on `hostname: *` and `namespaces: All`, or they could just as easily delegate specific hostnames to specific namespaces.
- Allow app namespaces to control TLS settings. In the Istio API, all TLS is in Gateway. However, the Kubernetes API allows route-level TLS overrides if allowed by the Gateway (default is not allowed). This enables the Gateway owner to delegate full control of TLS settings to some namespace. If desired, they could even allow this for `hostname: *` and `namespaces: All`.

As seen above, all existing use cases for allowing Gateways in any namespace is satisfied with the new API. This leads to a stronger isolation of responsibilities. The gateway operator configures the gateway, including what ports are available and which namespaces can control which domains. The application operator controls routing rules for their application, and which gateway's these are exposed on.

Examples

```
apiVersion: networking.x-k8s.io/v1alpha1
kind: Gateway
metadata:
 name: gateway
 namespace: istio-system
spec:
 gatewayClassName: istio
 gatewayService: istio-ingressgateway
 listeners:
 # Set up TLS settings for app1, alow app1 namespace to bind routes
 - hostname: "app1.example.com"
   port: 443
   protocol: HTTPS
   routes:
     kind: HTTPRoute
     namespaces:
       from: Selector
       selector:
         kubernetes.io/metadata.name: app1
   tls:
     mode: Terminate
     certificateRef:
       name: some-other-cert  # lives in istio-system namespace
 # Setup our app2, we allow app2 to configure TLS itself
 - hostname: "app2.example.com"
```

```
    port: 443
    protocol: HTTPS
    routes:
      kind: HTTPRoute
      namespaces:
        from: Selector
        selector:
          kubernetes.io/metadata.name: app1
    tls:
      mode: Terminate
      routeOverride:
        certificate: Allow
  # Setup our dev domain, anyone can bind routes here!
  - hostname: "*.dev.example.com"
    port: 443
    protocol: HTTPS
    routes:
      kind: HTTPRoute
      namespaces:
        from: All
    tls:
      mode: Terminate
      certificateRef:
        name: my-wildcard-cert-tls # lives in istio-system namespace
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: HTTPRoute
metadata:
  name: http
  namespace: app1
spec:
  hostnames: ["app1.example.com"]
  rules:
  - forwardTo:
    - serviceName: httpbin
      port: 80
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: HTTPRoute
metadata:
  name: http
  namespace: app2
spec:
  hostnames: ["app2.example.com"]
  tls:
    certificateRef:
      name: app2-cert # lives in app2 namespace
```

```
rules:
- forwardTo:
  - serviceName: httpbin
    port: 80
```

TODO:

- What if we have two services, both with servicePort:80, targetPort:8080. Do we merge or reject?
- How exactly is this implemented internally?
- Do we support ServiceEntry? We probably should, we can just make gatewayService a FQDN.
- Map out how this works in a non-k8s environment