Going All-in on Gateway API

Replacing Ingresses and Enhancing a Service Mesh

Who am I?

- 20+ years working in IT
- Largely focused on systems administration and devops
- Administered & used GKE in production for ~4 years
- Have also done most other things too
- I’m all about using the right tool for the job at hand

Mastodon: @genebean@fosstodon.org

GitHub: https://github.com/genebean

LinkedIn: https://www.linkedin.com/in/geneliverman/

Origin of this talk

- Heard about Gateway API at SCaLE 20x (or maybe 21x)
- Started a greenfield project at work
- Project results will hang around a long time…
  - let's be as up-to-date as possible
  - let's not carry known pain points forward into this project
- Wow… documentation is hard to find on *the new stuff*

But things didn’t go as planned…

Before we talk about Gateway API…

Why not just use the Ingress resource?

“The main limitation of Ingress is that it only works at Layer 7, specifically optimizing for HTTP and HTTPS traffic. Other L7 protocols (like gRPC) and non-L7 protocols (like TCP and UDP) must be handled using custom controller extensions rather than native Ingress capabilities.”

konghq.com/blog/engineering/gateway-api-vs-ingress

Why not just use the Ingress resource?

- gRPC support was needed
- TCP & UDP traffic
- Progressive rollouts a la Argo Rollouts
- Service mesh integration for observability

Gateway API

Why Gateway API?

"Overall, Gateways define a whole new way of declaring and managing traffic targeting Kubernetes services that avoids the limitations teams experience using only Ingress resources. The Gateway API creates a standardized model for enabling features like L4 support, advanced HTTP routing, and built-in traffic management in a portable fashion across all compliant gateway controllers. This will prevent vendor lock-in and give developers expanded declarative management without having to touch low-level controller configurations."

konghq.com/blog/engineering/gateway-api-vs-ingress

Why Gateway API?

“It’s important to note that Ingress is now frozen, and all new features are being added to the Gateway API going forward.”

konghq.com/blog/engineering/gateway-api-vs-ingress

	Ingress	Gateway API
Protocol Support	HTTP/HTTPS only	L4 & L7 support
Traffic Management	Limited, vendor extensions required	Built-in advanced support
Portability	Vendor specific definitions	Standardized across implementations
Resource Objects	Ingress resource only	GatewayClass, Gateway, HTTPRoute, etc.
Routing Rules	Host/path-based only	Header-based also supported
Extending Capabilities	Custom annotations needed	Built-in advanced functionality

The Plan

Components

- Traefik for north/south traffic instead of Ingress resources
- Linkerd for east/west traffic traversing the service mesh
- Argo Rollouts for progressive delivery / safer deployments

North/South traffic

Traffic from outside a cluster to inside a cluster (and vice versa).

gateway-api.sigs.k8s.io/concepts/glossary#northsouth-traffic

Put another way, traffic from users of the things running inside the cluster and traffic from your cluster to external systems and services

East/West traffic

Traffic from workload to workload within a cluster.

gateway-api.sigs.k8s.io/concepts/glossary/#eastwest-traffic

Put another way, traffic between program A and program B or between pods within the same application stack

Safer deployment because of Argo Rollouts?

- Deploy (roll out) applications to a subset of users, say 1% of traffic
- Roll out new versions a little at a time
- Halt, and even revert, rollout if metrics go sideways… automatically!

argoproj.github.io/rollouts

Why Traefik?

Simple: it seemed to be ubiquitous in all conversations, podcasts, presentations, and blog posts about Kubernetes and/or Docker

Also, fully supports Gateway API

Why Linkerd?

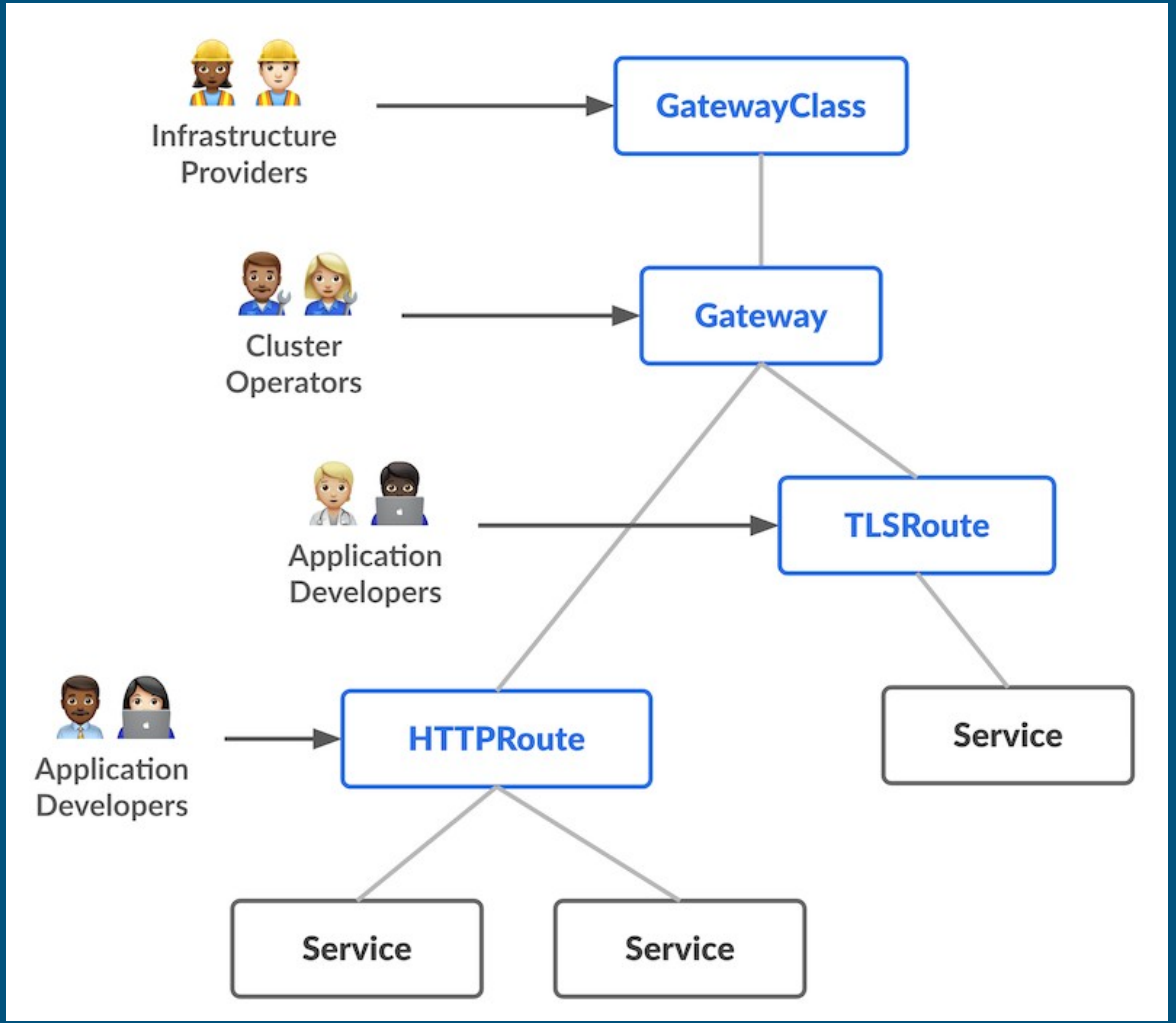
- Super lightweight
- Purpose built to just be a service mesh
- Really fast
  - livewyer.io/blog/service-meshes-decoded-istio-vs-linkerd-vs-cilium/

Introducing Gateway API

Resource Model

The overall resource model focuses on 3 separate personas and corresponding resources that they are expected to manage

https://gateway-api.sigs.k8s.io/concepts/roles-and-personas/



Sample GatewayClass

```
apiVersion: gateway.networking.k8s.io/v1

kind: GatewayClass

metadata:

  name: traefik-v3

spec:

  controllerName: traefik.io/gateway-controller
```

“GatewayClasses formalize types of load balancing implementations. These classes make it easy and explicit for users to understand what kind of capabilities are available via the Kubernetes resource model.”

gateway-api.sigs.k8s.io

Sample Gateway

```
apiVersion: gateway.networking.k8s.io/v1

kind: Gateway

metadata:

  name: argocd-server-gateway

  annotations:

      cert-manager.io/cluster-issuer: "le[…]"

spec:

  gatewayClassName: traefik-v3

  listeners:

      [ … see next slide … ]
```

“A Gateway describes how traffic can be translated to Services within the cluster. That is, it defines a request for a way to translate traffic from somewhere that does not know about Kubernetes to somewhere that does.”

“It defines a request for a specific load balancer config that implements the GatewayClass’ configuration and behaviour contract. The resource may be created by an operator directly, or may be created by a controller handling a GatewayClass.”

gateway-api.sigs.k8s.io/concepts/api-overview/#gateway

Sample Gateway listeners block

```
listeners:
    - name: http
      protocol: HTTP
      port: 80
      allowedRoutes:
        namespaces:
            from: Same
```

```
- name: https
  protocol: HTTPS
  port: 443
  hostname: {{ .Values.hostname }}
  allowedRoutes:
    namespaces:
        from: Same
  tls:
    mode: Terminate
    certificateRefs:
        - name: argocd-server-tls
```

Routes

“Gateway API supports typed Route resources and also different types of backends. This allows the API to be flexible in supporting various protocols (like HTTP and gRPC) and various backend targets (like Kubernetes Services, storage buckets, or functions).”

gateway-api.sigs.k8s.io/#gateway-api-concepts

Sample HTTPRoute: HTTP → HTTPS redirect

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: argocd-server-http
spec:
  hostnames:
    - {{ .Values.hostname }}
  parentRefs:
    - name: argocd-server-gateway
      sectionName: http
      kind: Gateway
  rules:
    - filters:
      - type: RequestRedirect
        requestRedirect:
          scheme: https
```

Sample HTTPRoute: HTTPS

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: argocd-server-https
spec:
  hostnames:
    - {{ .Values.hostname }}
  parentRefs:
    - name: argocd-server-gateway
      sectionName: https
      kind: Gateway
  rules:
    - backendRefs:
        - name: argocd-server
          port: 80
      matches:
        - path:
            type: PathPrefix
            value: /
```

Service Mesh Variation

```
spec:
  parentRefs:
    - name: some-http-service
      kind: Service
      group: core
      port:
{{ .Values.collector.service.apiPort }}
  rules:
    - backendRefs:
        - name: some-http-service
          kind: Service
               group: core
               port:
{{ .Values.collector.service.apiPort }}
               weight: 100
```

```
spec:
  parentRefs:
    - name: some-grpc-service
      kind: Service
      group: core
      port:
{{ .Values.collector.service.grpcPort }}
  rules:
    - backendRefs:
        - name: some-grpc-service
          kind: Service
          group: core
               port:
{{ .Values.collector.service.grpcPort }}
               weight: 100
```

Challenges along the way

Not everything is rosy

- Gateway API is not in tree with k8s so you have to manage CRDs
  - Easy enough with Argo CD
- Linkerd doesn't support GRPCRoute v1 because of cloud providers
  - #13032 [Gateway API] Support for GRPCRoute v1 (stable channel)
- Getting Traefik to make CRDs optional was challenging:
  - #1209 Don't bundle Gateway API CRDs
  - #1223 feat(Chart): 📦 add optional separated chart for CRDs
- Gateway API v1.1.1 - fixed GRPCRoute with help from Flynn from Linkerd
  - *"It's issue 3411. Its original PR was PR 3412, which has been merged. PR 3419 is next in line (the changelog for the 1.1.1 release); I think that's the last PR before release."*
  - https://github.com/kubernetes-sigs/gateway-api/releases/tag/v1.1.1

Even with challenges, still worth it.

Why is it worth it?

- Easier to reason about
- More flexible
- Provides better visibility

Questions?