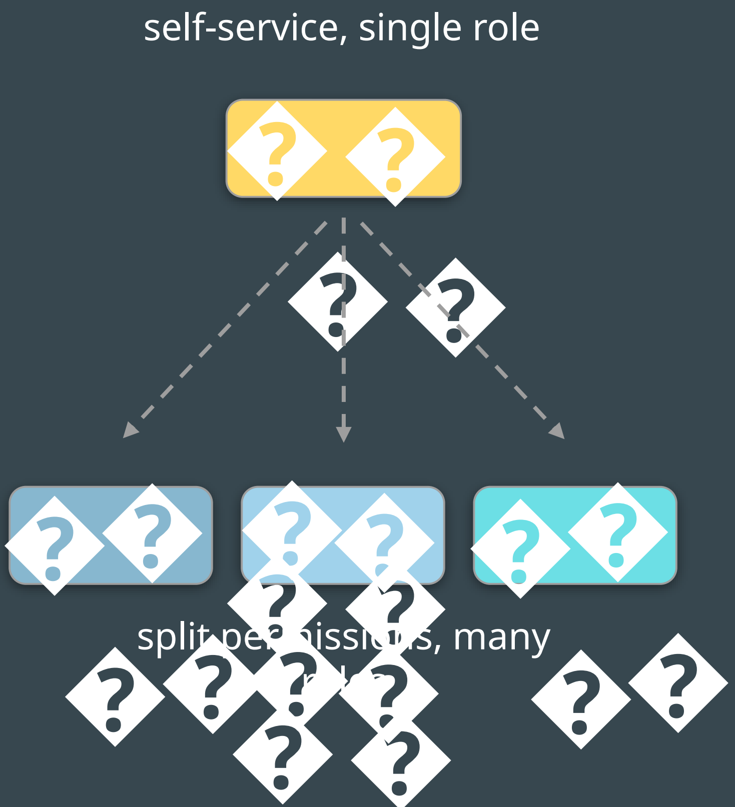
The evolution of Ingress through the Gateway API …

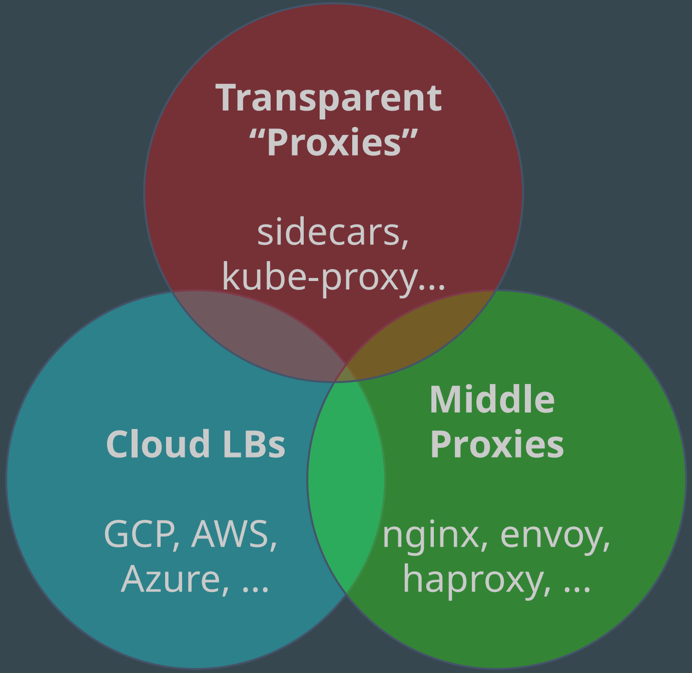
Bowei Du <bowei@google.com>
Kaslin Fields <kaslin@google.com>

Kaslin Fields <
kaslin@google.com>

Bowei Du <bowei@google.com>

Evolving Landscape





Goals

Better model the **personas and roles** involved with services and load-balancing.
Support **modern load-balancing features** while maintaining **portability** (or maybe “**predictability**”)
Have **standard mechanisms for extension** for API growth / implementation / vendor-specific behaviors.

Goals (and how to get there)

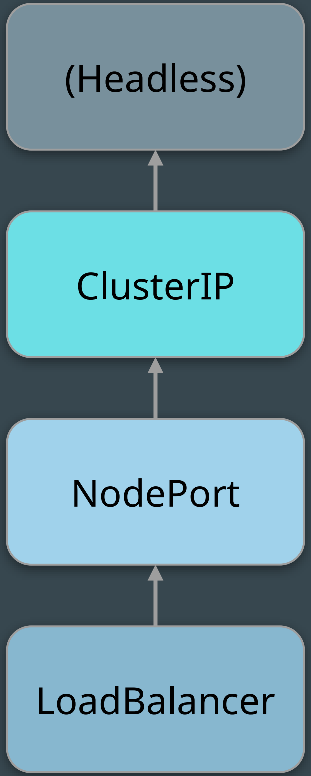
Better model the **personas and roles** involved with services and load-balancing.	→	Resource model + RBAC
Support **modern load-balancing features** while maintaining **portability** (or maybe “**predictability**”)	→	Levels of support, specification and conformance
Have **standard mechanisms for extension** for API growth / implementation / vendor-specific behaviors.	→	Resource model, polymorphism-like

Modeling: Services

A Service resource describes many things:

- **Method of exposure** (ClusterIP, NodePort, LoadBalancer)
- **Grouping of Backends** (i.e. selector)
- **Attributes** (ExternalTrafficPolicy, SessionAffinity, ...)

Evolving and extending the resource becomes harder and harder due to interactions between fields...



Service hierarchy

Personas and Roles

	Infrastructure Provider	Provides the infrastructure for cluster creation, e.g. cloud provider, internal PaaS team.
	Cluster Operator / NetOps / SRE	Manages the cluster overall once its created. Responsible for overall policies, e.g. which services expose to Internet.
	Application Developer	Builds the services and applications and defines traffic routing, services.

Designing for RBAC

USE
R

ROL
E

can act as a

VER
B

with permission to the

RESOUR
CE

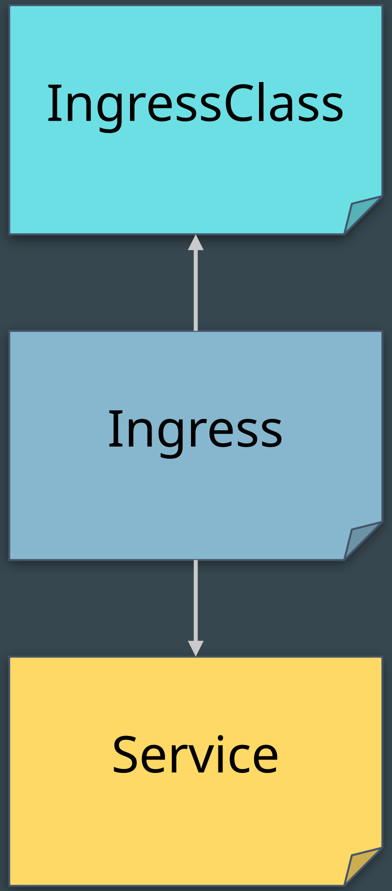
configuration of a

Modeling roles: Ingress

Ingress is a self-service model.

IngressClass is created by the infrastructure provider

Application developer manages Ingress + Service; Ingress limited to simple L7 descriptions.



Modeling roles: Gateway

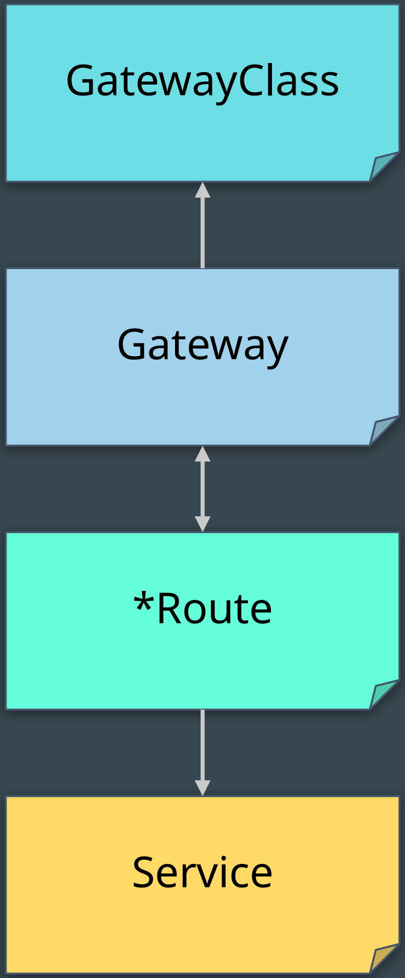
Idea: Decouple along role, concept axes:

Roles:

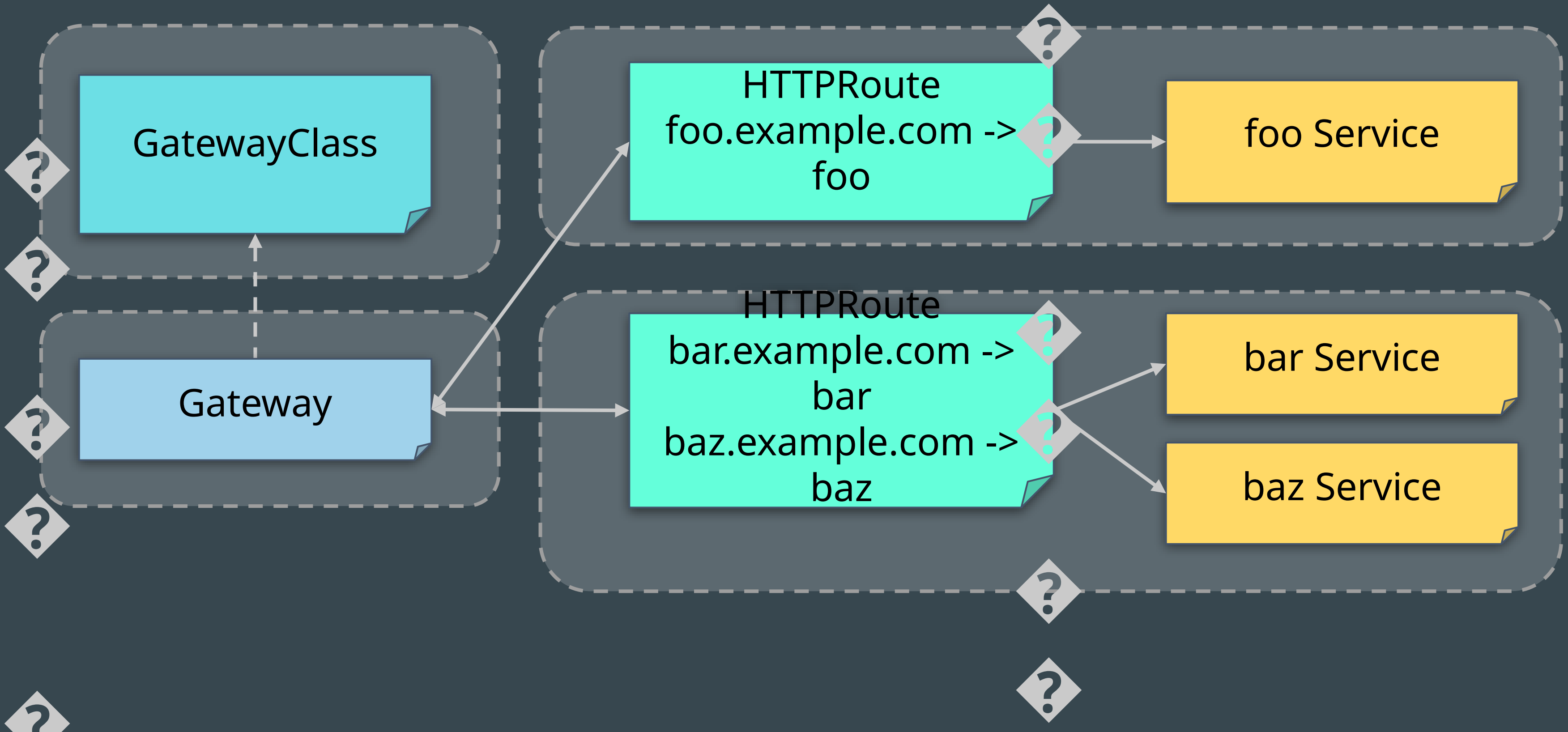
- Infrastructure Provider (GatewayClass)
- Cluster Operator / NetOps (Gateway)
- Application Developer (Routes and Services)

Concepts:

- Exposure and access (Gateway)
- Routing, protocol specific attributes (Routes)
- Grouping, selection (Service)



API schema


GatewayClass
Gateway
HTTPRoute
foo.example.com ->
foo
foo Service
HTTPRoute
bar.example.com ->
bar
baz.example.com ->
baz
bar Service
baz Service

Portability vs predictability

Core

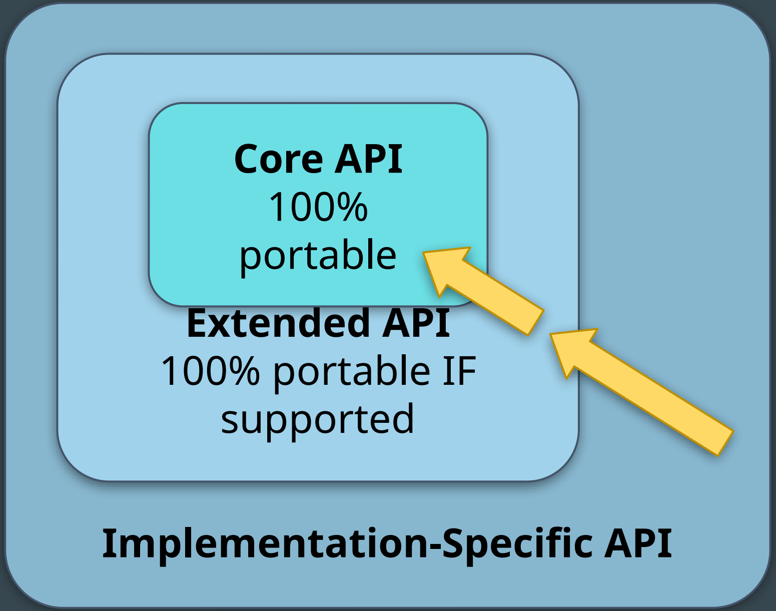
- MUST be supported.

Extended

- Feature by feature.
- MAYBE supported, but MUST be portable.
- Part of API schema.

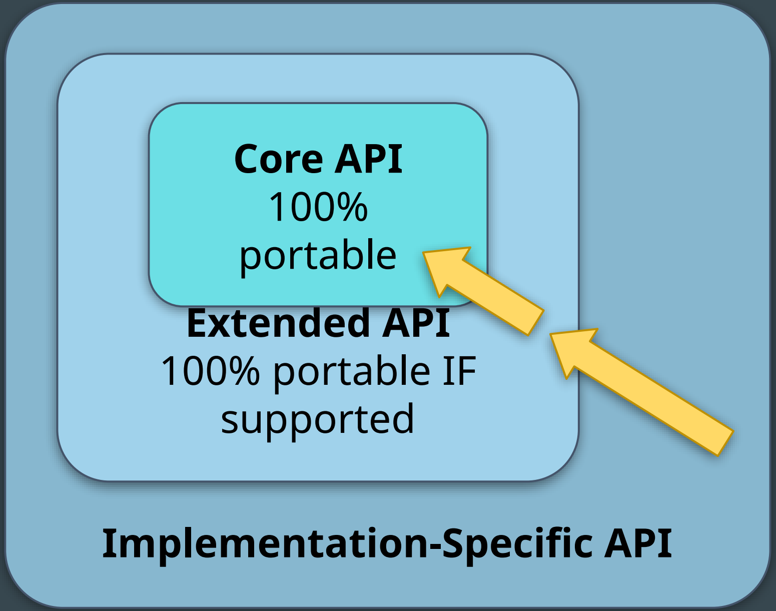
Implementation-specific

- No guarantee for portability, No k8s API schema.



Portability vs predictability

- Enforcement by conformance tests
- Extended feature definition **requires** self-contained conformance
- Require all extended features be checkable statically

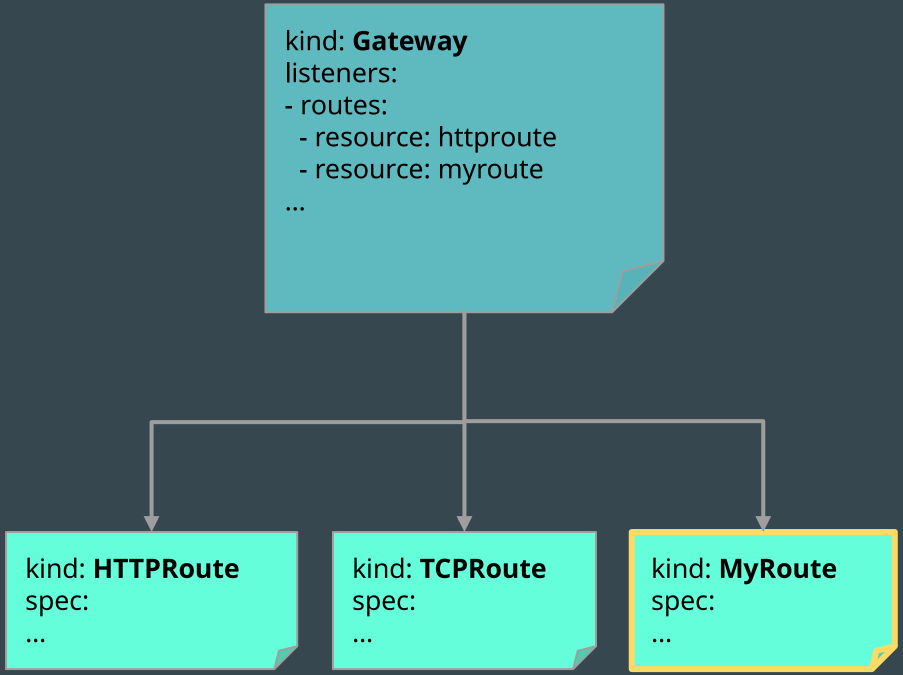


Extensibility

Polymorphic object references:

- Gateways can refer to different kinds of Routes: {HTTP, TCP, UDP, SNI} Route
- Route to backends also polymorphic (not just to Services)

Use of Custom Resources (CRs) for extension.



```
kind: Gateway
listeners:
- routes:
  - resource: httproute
  - resource: myroute
...
```

```
kind: HTTPRoute
spec:
...
```

```
kind: TCPRoute
spec:
...
```

```
kind: MyRoute
spec:
...
```

A taste of the API

User story

Alice the IaaS provider

Bob the SRE

Carol the application developer

User story: Alice the IaaS

Alice the IaaS provider offers two flavors of load-balancers:

- external: Public access to the Internet
- internal: Internal to the VPC

```
kind: GatewayClass
meta:
  name: external
spec:
  controller: alice.io/gw-ctrl
...
```

```
kind: GatewayClass
meta:
  name: internal
spec:
  controller: alice.io/gw-ctrl
...
```

User story: Bob the SRE

Bob wants:

- Only certain namespaces can deploy external LBs (namespaces with label internet:external)
- Anyone can deploy an internal LB
- Anyone can deploy a in-cluster proxy for testing (Bob installs an acme.io/proxy)

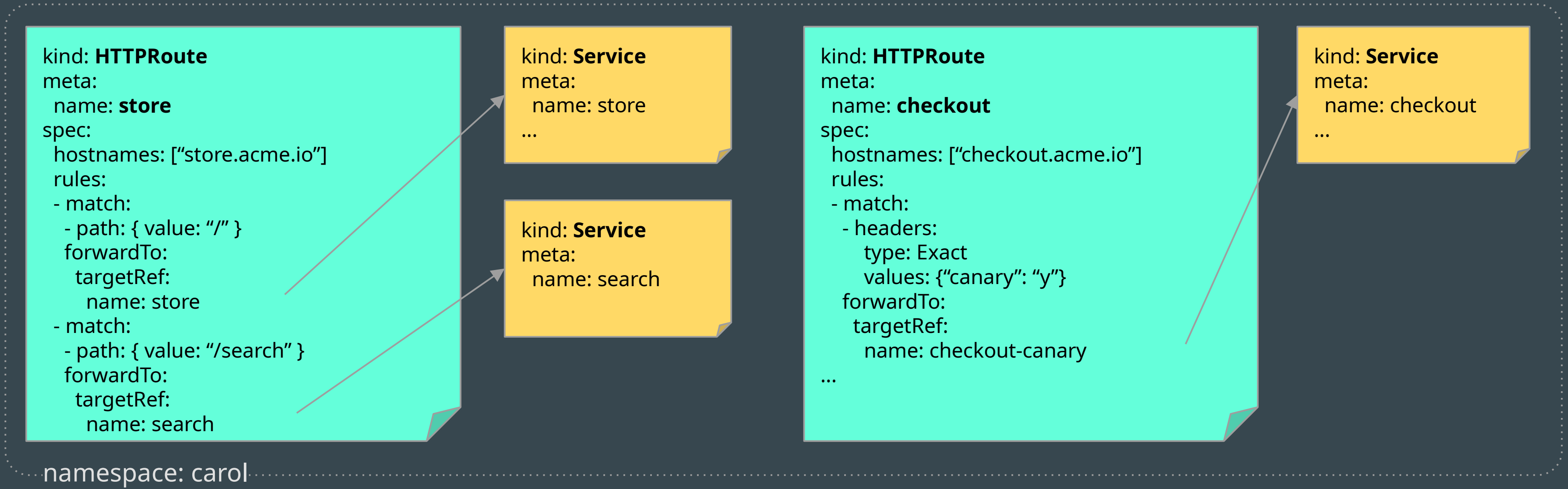
```
kind: GatewayClass
meta:
  name: external
spec:
  controller: alice.io/gw-ctrl
  allowedGatewayNamespaces:
    matchLabels:
      internet: external
```

```
kind: GatewayClass
meta:
  name: internal
spec:
  controller: alice.io/gw-ctrl
  # Default allows all ns
```

```
kind: GatewayClass
meta:
  name: test
spec:
  controller: acme.io/proxy
  # Default allows all ns
```

User story: Carol the dev

Creates Routes and Services to describe her applications: “store”, “checkout”



```
kind: HTTPRoute
meta:
  name: store
spec:
  hostnames: [“store.acme.io”]
  rules:
  - match:
    - path: { value: “/” }
    forwardTo:
      targetRef:
        name: store
  - match:
    - path: { value: “/search” }
    forwardTo:
      targetRef:
        name: search
```

```
kind: Service
meta:
  name: store
…
```

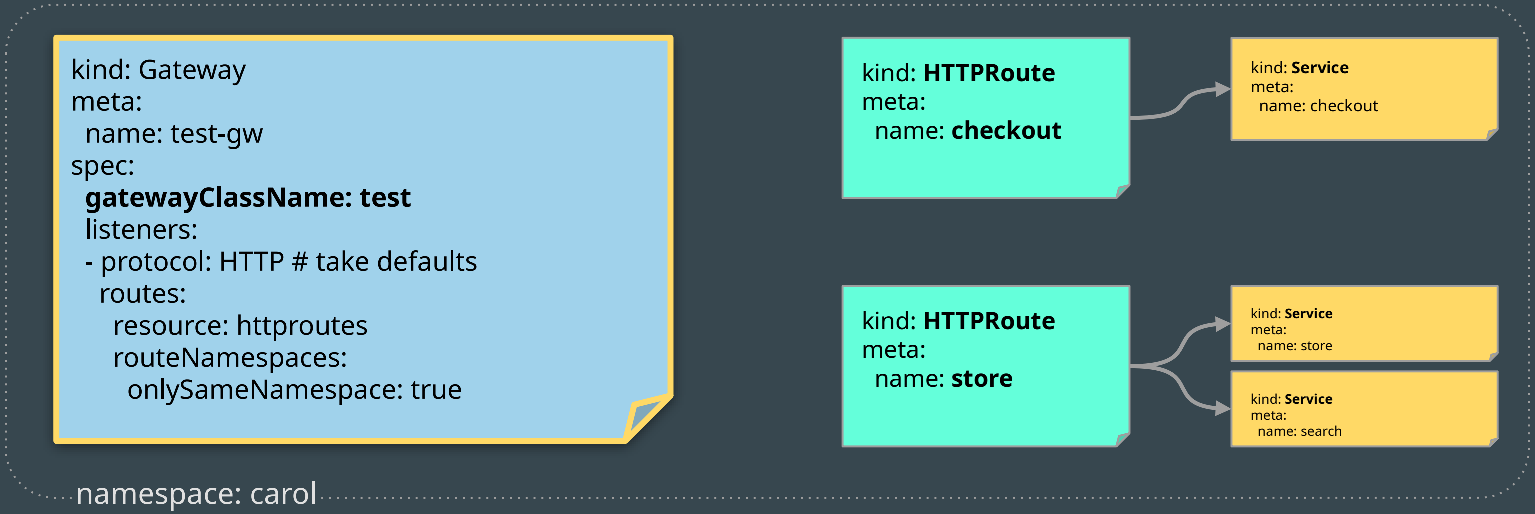
```
kind: Service
meta:
  name: search
```

```
kind: HTTPRoute
meta:
  name: checkout
spec:
  hostnames: [“checkout.acme.io”]
  rules:
  - match:
    - headers:
        type: Exact
        values: {“canary”: “y”}
    forwardTo:
      targetRef:
        name: checkout-canary
…
```

```
kind: Service
meta:
  name: checkout
…
```

User story: Carol the dev

To test out her applications, she creates a Gateway of class “test” in her namespace:



```
kind: Gateway
meta:
  name: test-gw
spec:
  gatewayClassName: test
  listeners:
  - protocol: HTTP # take defaults
    routes:
      resource: httproutes
      routeNamespaces:
        onlySameNamespace: true
```

```
kind: HTTPRoute
meta:
  name: checkout
```

```
kind: Service
meta:
  name: checkout
```

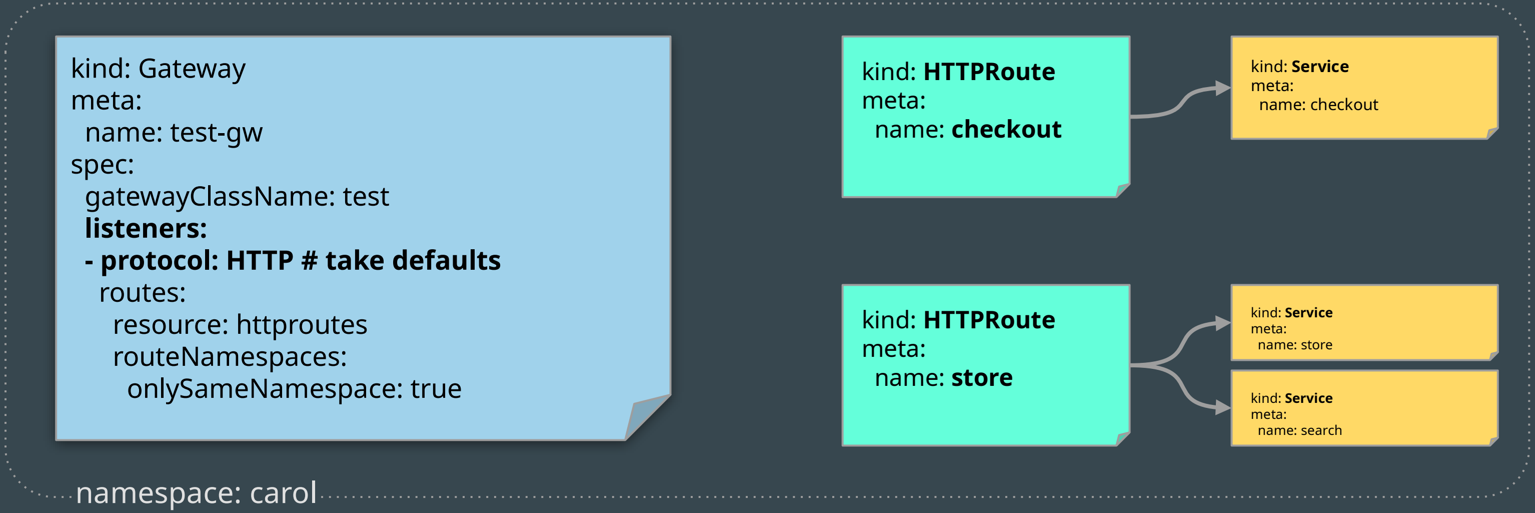
```
kind: HTTPRoute
meta:
  name: store
```

```
kind: Service
meta:
  name: store
```

```
kind: Service
meta:
  name: search
```

User story: Carol the dev

A Gateway is a “request” for an LB. It may be underspecified and the controller for the class will fill in the blanks:



```
kind: Gateway
meta:
  name: test-gw
spec:
  gatewayClassName: test
  listeners:
  - protocol: HTTP # take defaults
    routes:
      resource: httproutes
      routeNamespaces:
        onlySameNamespace: true
```

```
kind: HTTPRoute
meta:
  name: checkout
```

```
kind: Service
meta:
  name: checkout
```

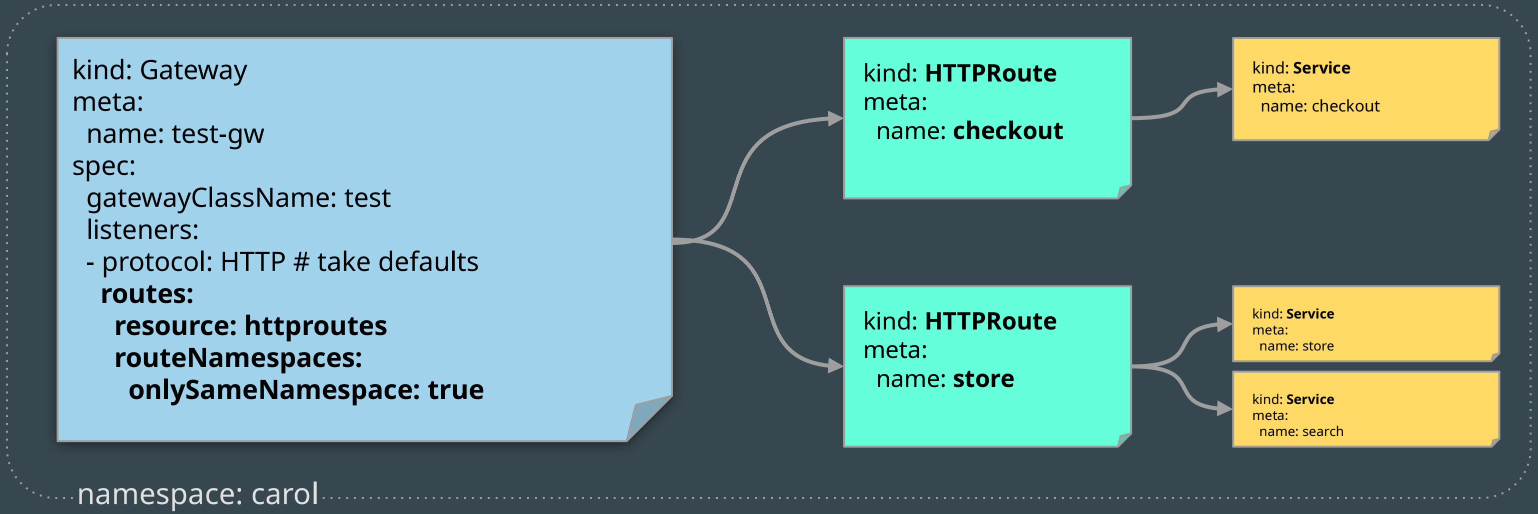
```
kind: HTTPRoute
meta:
  name: store
```

```
kind: Service
meta:
  name: store
```

```
kind: Service
meta:
  name: search
```

User story: Carol the dev

A Gateway serves Routes. Carol uses the simplest configuration which is to pick up routes in her own namespace:



```
kind: Gateway
meta:
  name: test-gw
spec:
  gatewayClassName: test
  listeners:
  - protocol: HTTP # take defaults
    routes:
      resource: httproutes
      routeNamespaces:
        onlySameNamespace: true
```

```
kind: HTTPRoute
meta:
  name: checkout
```

```
kind: Service
meta:
  name: checkout
```

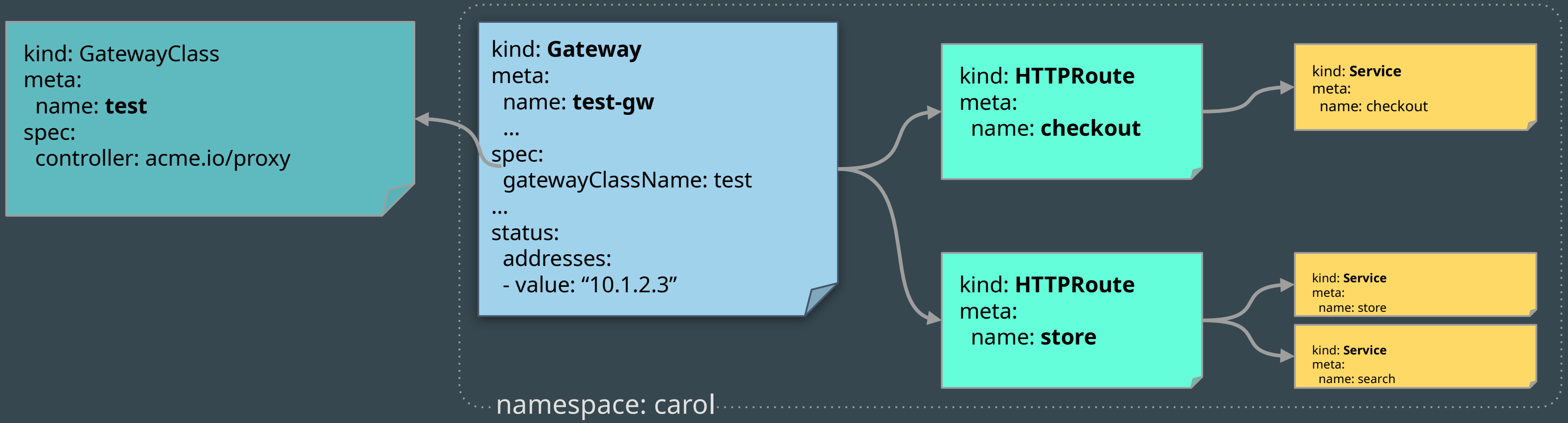
```
kind: HTTPRoute
meta:
  name: store
```

```
kind: Service
meta:
  name: store
```

```
kind: Service
meta:
  name: search
```

User story: Carol the dev

Creating the Gateway wires up the app(s) and the acme.io/proxy starts serving traffic:



```
kind: GatewayClass
meta:
  name: test
spec:
  controller: acme.io/proxy
```

```
kind: Gateway
meta:
  name: test-gw
  …
spec:
  gatewayClassName: test
…
status:
  addresses:
  - value: “10.1.2.3”
```

```
kind: HTTPRoute
meta:
  name: checkout
```

```
kind: Service
meta:
  name: checkout
```

```
kind: HTTPRoute
meta:
  name: store
```

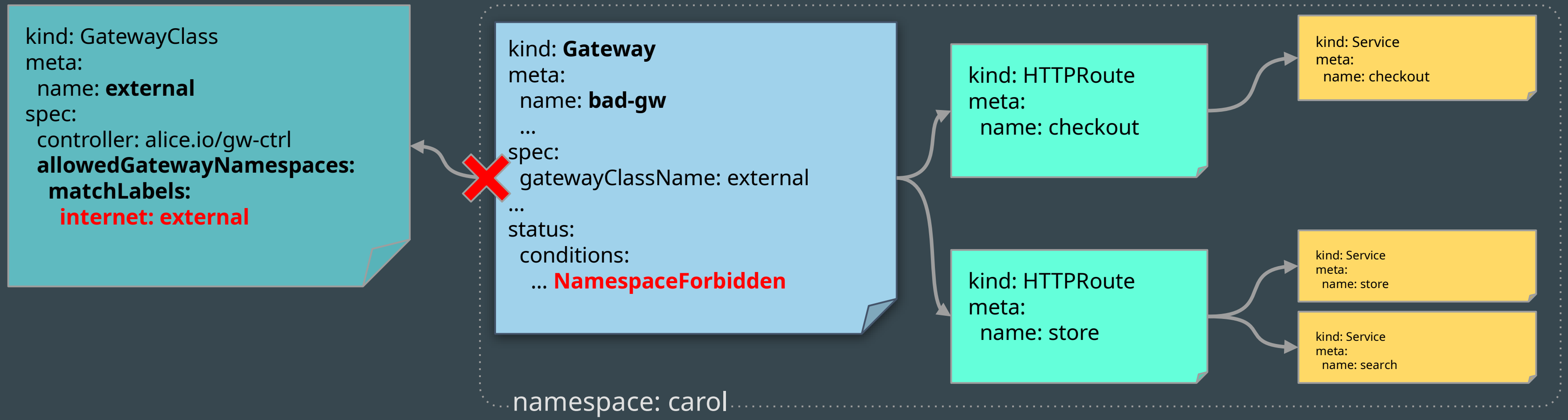
```
kind: Service
meta:
  name: store
```

```
kind: Service
meta:
  name: search
```

```
carol@acme$ curl -H store.acme.io http://10.1.2.3
Hello, world! ….
```

User story: Carol the dev

Carol is cannot use the external class and serve production traffic - she isn’t allowed:



```
kind: GatewayClass
meta:
  name: external
spec:
  controller: alice.io/gw-ctrl
  allowedGatewayNamespaces:
    matchLabels:
      internet: external
```

```
kind: Gateway
meta:
  name: bad-gw
  ...
spec:
  gatewayClassName: external
...
status:
  conditions:
    ... NamespaceForbidden
```

```
kind: HTTPRoute
meta:
  name: checkout
```

```
kind: HTTPRoute
meta:
  name: store
```

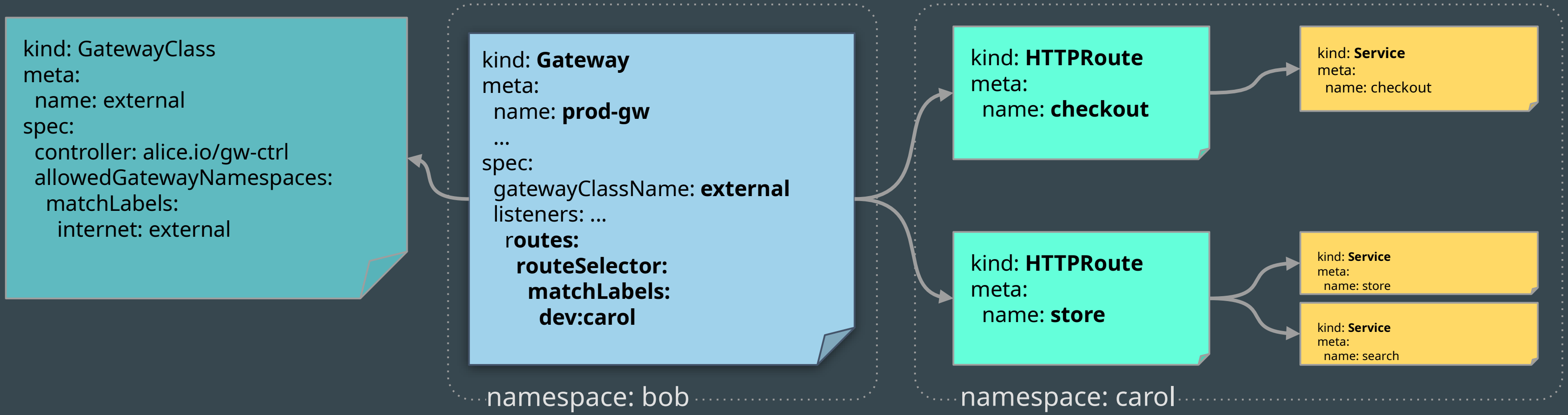
```
kind: Service
meta:
  name: checkout
```

```
kind: Service
meta:
  name: store
```

```
kind: Service
meta:
  name: search
```

User story: Bob the SRE

Bob, who manages external Gateways for the cluster, can reference Carol's Routes and put them into production:



```
kind: GatewayClass
meta:
  name: external
spec:
  controller: alice.io/gw-ctrl
  allowedGatewayNamespaces:
    matchLabels:
      internet: external
```

```
kind: Gateway
meta:
  name: prod-gw
  ...
spec:
  gatewayClassName: external
  listeners: ...
    routes:
      routeSelector:
        matchLabels:
          dev:carol
```

```
kind: HTTPRoute
meta:
  name: checkout
```

```
kind: HTTPRoute
meta:
  name: store
```

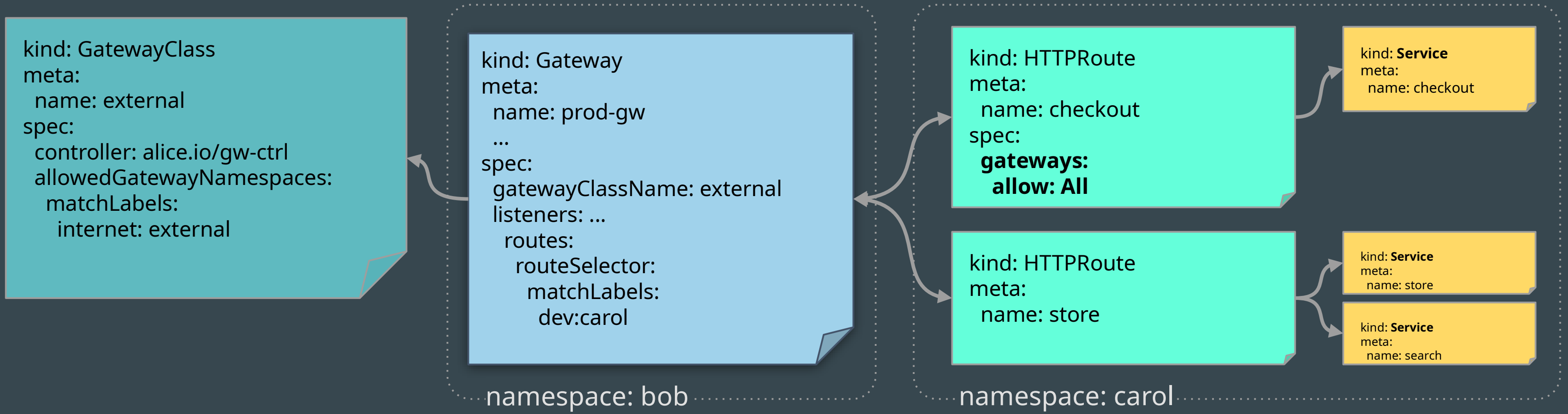
```
kind: Service
meta:
  name: checkout
```

```
kind: Service
meta:
  name: store
```

```
kind: Service
meta:
  name: search
```

User story: Carol the dev

Carol also needs to allow her Route to the prod Gateway - otherwise anyone with a Gateway can start serving traffic (default is to allow in the same namespace):



```
kind: GatewayClass
meta:
  name: external
spec:
  controller: alice.io/gw-ctrl
  allowedGatewayNamespaces:
    matchLabels:
      internet: external
```

```
kind: Gateway
meta:
  name: prod-gw
  ...
spec:
  gatewayClassName: external
  listeners: ...
    routes:
      routeSelector:
        matchLabels:
          dev:carol
```

```
kind: HTTPRoute
meta:
  name: checkout
spec:
  gateways:
    allow: All
```

```
kind: Service
meta:
  name: checkout
```

```
kind: HTTPRoute
meta:
  name: store
```

```
kind: Service
meta:
  name: store
```

```
kind: Service
meta:
  name: search
```

```
carol@acme$ curl -H http://store.acme.io
Hello, world! ...
```

User story: recap

GatewayClass: Supports multiple classes of load-balancing, reflecting capabilities of the IaaS or deployed infrastructure

Gateway: defines how your app is **exposed** (e.g. VIP:port, what kind of proxy used)

Route: defines the **routing** of your app for a given **protocol**

Service: defines your Backends (**grouping**)

User story: recap

Supports multiple classes of load-balancing, reflecting capabilities of the IaaS or deployed infrastructure.

Access to GatewayClasses is controlled.

Gateways can reference and aggregate cross namespace Routes. Access is controlled using handshake between Gateway and Route

Conflicts

Merging Routes into Gateways can result in conflicts (same host, same path)...

Conflicts principles

Do no harm

- Don’t break things that are working
- Drop as little traffic as possible

Be consistent

- Provide a consistent behavior when conflicts occur (A then B ≡ B then A)
- Prefer more specific matches to less specific ones
- Decide using stable properties: creation timestamp, canonical order <namespace, name>

Be clear

- Make it clear which configuration has been chosen
- Communicate conflicts via object status

Extension points

GatewayClass parameters for overall LB configuration.

Gateway.Listeners have an ExtensionRef to customize Listener properties

Routes

- Custom filters via ExtensionRef
- Backends can be more than Services

Feedback needed!

What in the Alpha?

Basic applications, data types:

- GatewayClass, Gateway
- {HTTP, TCP, UDP, SNI} Route
- TLS (HTTPS, TLS)

Implementations:

- Merging style (multiple Gateways hosted on in-cluster proxy)
- Provisioning/Cloud (Gateways mapped to externally managed resources)

What remains to be done:

- Feedback (users, users, users)!
- Conformance tests
- Delegation within Routes

→ Gain confidence towards Beta

Demo:

https://bit.ly/2RY9Xzr

Thanks for coming!

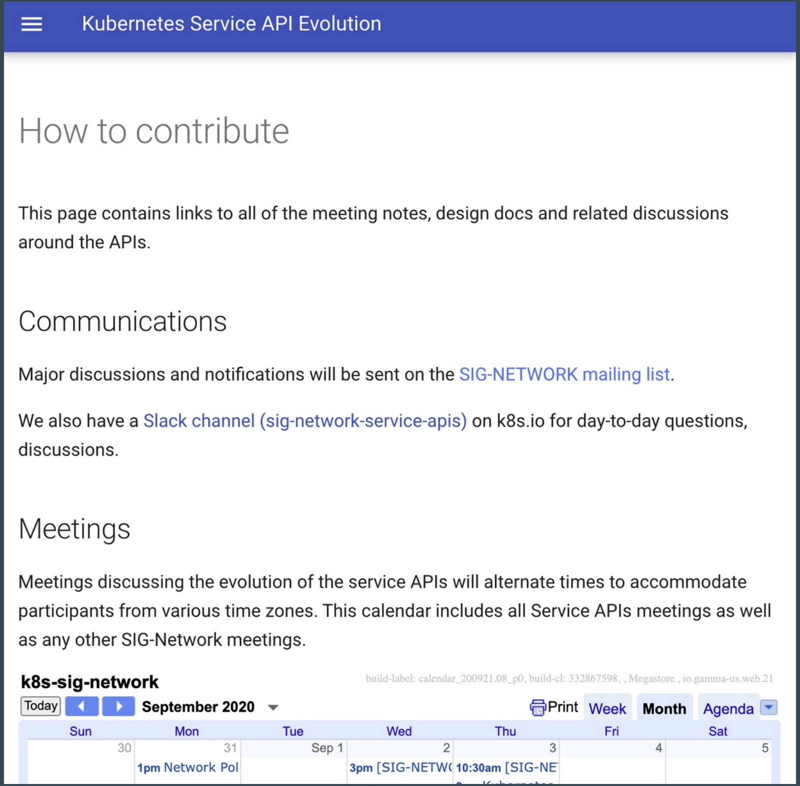
Kubernetes SIG-NETWORK subproject

Project homepage
github/kubernetes-sigs/service-apis

How to contribute
kubernetes-sigs.github.io/service-apis/community/

Meetings
Wednesday, Thursdays

Alternating times AM/PM Pacific, check calendar, meeting code: “77777”



Kubernetes Service API Evolution

How to contribute

This page contains links to all of the meeting notes, design docs and related discussions around the APIs.

Communications

Major discussions and notifications will be sent on the SIG-NETWORK mailing list.

We also have a Slack channel (sig-network-service-apis) on k8s.io for day-to-day questions, discussions.

Meetings

Meetings discussing the evolution of the service APIs will alternate times to accommodate participants from various time zones. This calendar includes all Service APIs meetings as well as any other SIG-Network meetings.

k8s-sig-network

build-label: calendar_200921.08_p0, build-cl: 332867598, , Megastore , io.gamma-us.web.21

Today September 2020

Print Week Month Agenda

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	31	Sep 1	2	3	4	5
	1pm Network Pol		3pm [SIG-NETW(	10:30am [SIG-NE		

Fin

References

- https://www.youtube.com/watch?v=Ne9UJL6irXY (very old)
- Kubecon San Diego 2019 - Evolving the Kubernetes Ingress APIs to GA and Beyond [PUBLIC]
- Virtual KubeCon EU 2020: SIG-Network Intro and DeepDive
- https://github.com/kubernetes-sigs/service-apis