
KubeCon
CloudNativeCon
North America 2019

Evolving the Kubernetes Ingress APIs to GA and Beyond

Bowei Du <bowei@>
Christopher M Luciano <cmluciano@>

Overview

- Current Ingress Usage
- Planned v1 enhancements
- v1 Timeline
- Ingress API Evolution (v2)
- Summary
- Questions

Current State of Ingress

- Provides basic k8s service backend mappings/rules and related TLSconfig
- Majority of Ingress usage provided by off-the-shelf controllers like ingress-nginx, HAProxy, etc.
- Current API is limited in scope to ensure massive portability
  - Side effect of many decorator annotations dangling off of Ingress metadata

v1 API changes

Clean up the object model: `IngressClass`

Tweaks/fix to the specification:

- `backend` to `defaultBackend`
- Path-based prefixes/regex
- Hostname wildcards.

Add flexibility that will be hard to change later:

- Alternate backend types

backend -> defaultBackend

- Rename backend to defaultBackend
  - Clarify use of this field as the explicit default
- Open PR k/k#80179

Path based prefixes & regex

- Current v1beta1 APIs assume that the path is a regex specified with the POSIX IEEE 1003.1 standard
  - Does not match cloud-provider or nginx/haproxy today
  - kubernetes/ingress-nginx#555
- v1 goals
  - Explicitly state path match mode
  - Support existing implementation-specific variance
  - Portable prefix matching with options for future ideas

Path based prefixes & regex


```
type HTTPIngressPath struct {
  ...
 // Path to match against. The interpretation of Path depends on
 // the value of PathType.
 //
 // Defaults to "/" if empty.
 //
 // +Optional
 Path string

 // PathType determines the interpretation of the Path
 // matching. PathType can be one of the following values:
 //
 // Exact  - matches the URL path exactly.
 //
 // Prefix - matches based on a URL path prefix split
 // by '/'. [insert description of semantics described below]
 //
 // ImplementationSpecific - interpretation of the Path
 // matching is up to the IngressClass. Implementations
 // are not required to support ImplementationSpecific matching.
 //
 // +Optional
 PathType string
 ...
}
```

PathType Examples

PathType	Path	Request path	Match?
Exact	`/abc`	`/abc`	Yes
Exact	`/abc`	`/cba`	No
Prefix	`/`	`any path`	Yes
Prefix	`/abc/`	`/abc`	Yes, trailing slash doesn't matter
Prefix	`/aaa/bbb`	`/aaa/bbb/ccc`	Yes, subpath match
Prefix	`/aaa/bbb`	`/aaa/bbbzzz`	No, no prefix match

Hostname wildcards

- Goal
  - ***.foo.com** matches **app.foo.com**
- Validation
  - * must appear as first DNS label
    - Single label
    - Cannot be **Host == “*”**

*.foo.com	bar.foo.com	Matches based on shared suffix
*.foo.com	baz.bar.foo.com	No match, wildcard only covers single label
*.foo.com	foo.com	No match, wildcard only covers single label

IngressClass


```
type IngressSpec struct {
  ...
  // Class is the name of the IngressClass cluster resource. This defines
  // which controller(s) will implement the resource.
  Class string
  ...
}

...

// IngressClass represents the class of the Ingress, referenced by the
// ingress.spec. IngressClass will be a non-namespaced Cluster resource.
type IngressClass struct {
  metav1.TypeMeta
  metav1.ObjectMeta

  // Controller is responsible for handling this class. This should be
  // specified as a domain-prefixed path, e.g. "acme.io/ingress-controller".
  //
  // This allows for different "flavors" that are controlled by the same
  // controller. For example, you may have different Parameters for
  // the same implementing controller.
  Controller string

  // Parameters is a link to a custom resource configuration for
  // the controller. This is optional if the controller does not
  // require extra parameters.
  //
  // +optional
  Parameters *TypeLocalObjectReference
}
```

Current backend types


```
// IngressBackend describes all endpoints for a given service and port.
type IngressBackend struct {
        // Specifies the name of the referenced service.
        ServiceName string `json:"serviceName" protobuf:"bytes,1,opt,name=serviceName"`

        // Specifies the port of the referenced service.
        ServicePort intstr.IntOrString `json:"servicePort" protobuf:"bytes,2,opt,name=servicePort"`
}
```

- Currently Ingress == L7 service sets
  - Only Kubernetes Services are valid backends
- Goal
  - Support alternate backends like storage buckets

Alternate backend types


```
type IngressBackend struct {
  // Only one of the following fields may be specified.

  // Service references a Service as a Backend. This is specially
  // called out as it is required to be supported AND to reduce
  // verbosity.
  // +optional
  Service *ServiceBackend

  // Resource is an ObjectRef to another Kubernetes resource in the namespace
  // of the Ingress object.
  // +optional
  Resource *v1.TypedLocalObjectReference
}

// ServiceBackend references a Kubernetes Service as a Backend.
type ServiceBackend struct {
  // Service is the name of the referenced service. The service must exist in
  // the same namespace as the Ingress object.
  // +optional
  Name string

  // Port of the referenced service. If unspecified and the ServiceName is
  // non-empty, the Service must expose a single port.
  // +optional
  Port ServiceBackendPort
}

// ServiceBackendPort is the service port being referenced.
type ServiceBackendPort struct {
  // Number is the numerical port number (e.g. 80) on the Service.
  Number int
  // Name is the name of the port on the Service.
  Name string
}
```

- Existing Kubernetes Service types MUST remain supported by implementations
- Custom services leverage new Resource field by way of IngressClass

IngressBackend examples


```yaml
kind: Ingress
spec:
  class: magic-lb
  backend:
    service:
      name: magic-service
      port:
        number: 80
```

Kubernetes Service

```yaml
kind: Ingress
spec:
  class: magic-lb
  backend:
    resource:
      apiGroup: magic.blog/backends
      kind: storage-bucket
      name: static-resources
```

Custom Resource

v1 API Timeline

- Kubernetes Release
  - v1.14
    - Ingress API in extensions/v1beta1 copied to networking.k8s.io/v1beta1
    - Mailing list announcement for Ingress extensions/v1beta1 deprecation
  - v1.15
    - Updated all documentation and k/k repo code for networking.k8s.io/v1beta1
    - ingress-nginx v0.25.0
    - ingress-gce v1.7.0
  - v1.18
    - New networking.k8s.io/v1 APIs released
    - Mailing list deprecation warning of Ingress networking.k8s.io/v1beta1 deprecation

v1 API Timeline

- Kubernetes Release
  - v1.19
    - Update ingress-nginx/gce for v1 APIs
    - Remove ability to serve extensions/v1beta1
  - v1.2x ?
    - Remove ability to serve networking.k8s.io/v1beta1

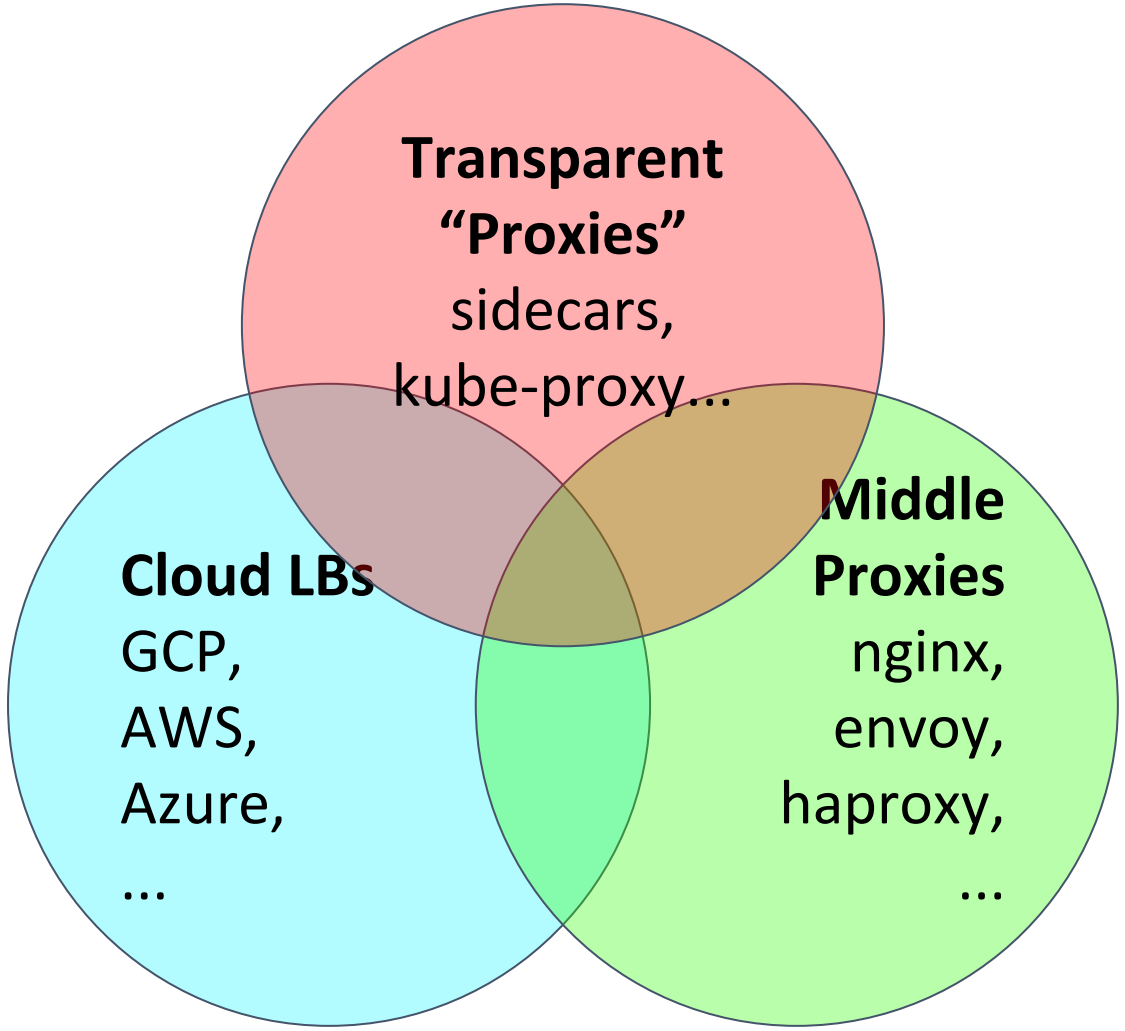
Evolving Landscape

Early resources design were “self-service” oriented:

- Few tenants
- Empowered dev owns whole deployment

Now, we have:

- Multiple team/roles interactions
- Potentially multiple co-existing implementations



Goals for the evolution

Provide a better model **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.

Status

Note: the API is in a proposal stage, so many things can change (Your voice as a user is key part of this process!)

- Describe how goals map to API design
- Give a sense of the overall object model
- Highlight some interesting problems/approaches

Many open questions.

Goals for evolution

Provide a better model **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

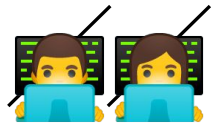
Personas and Roles

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

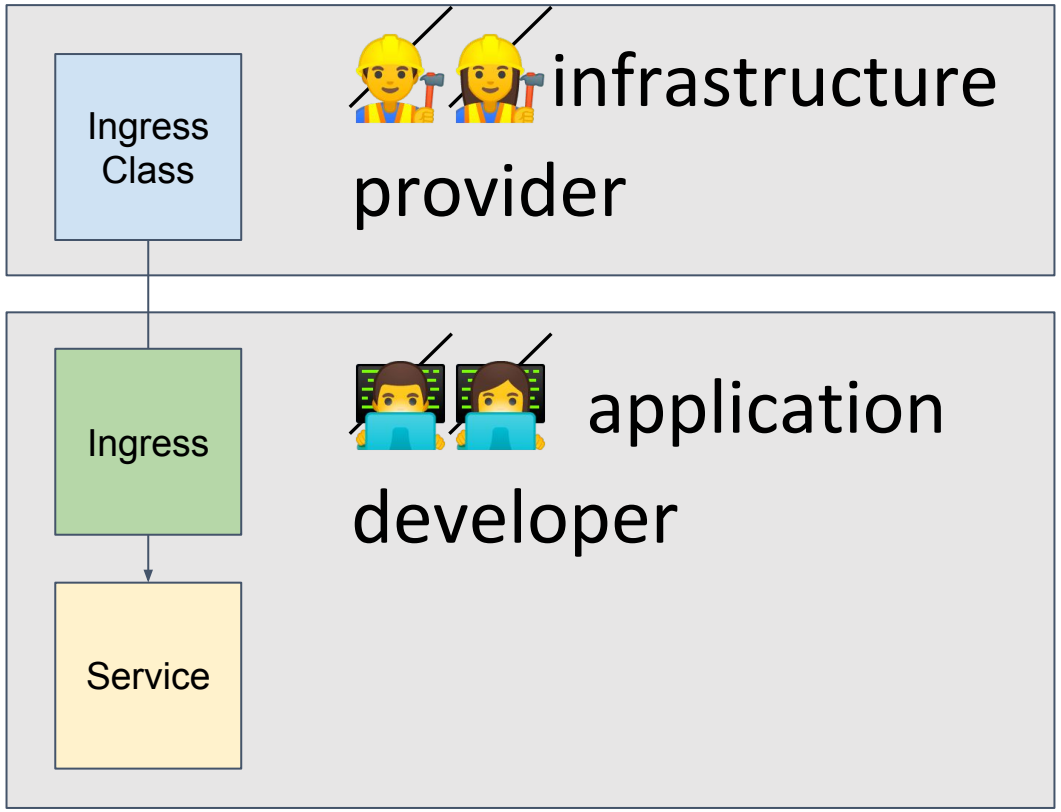
Builds the services and applications and defines traffic routing, services.

Modeling roles: Ingress

Ingress is a self-service model.

IngressClass are created by infrastructure provider

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

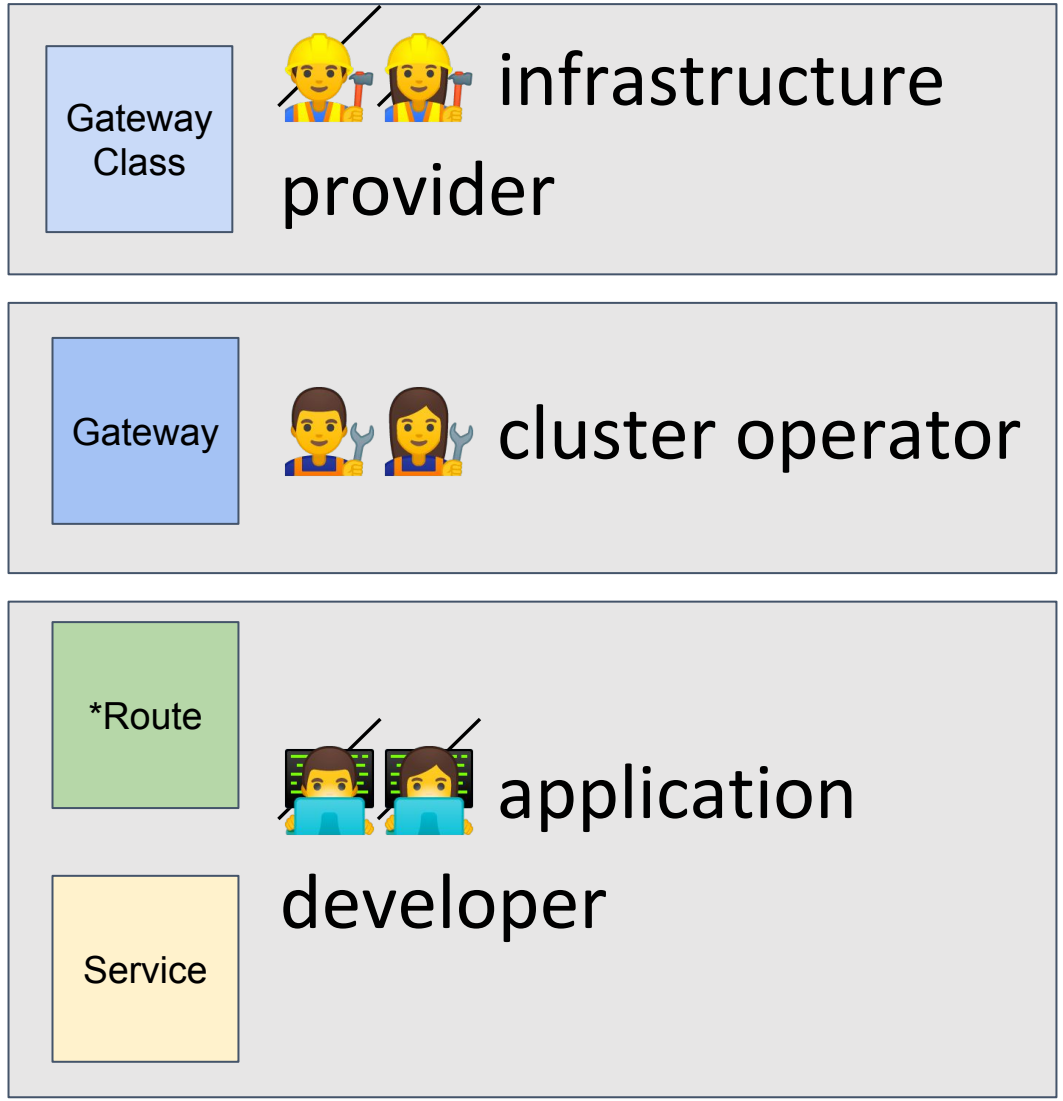


Modeling roles: Evolution

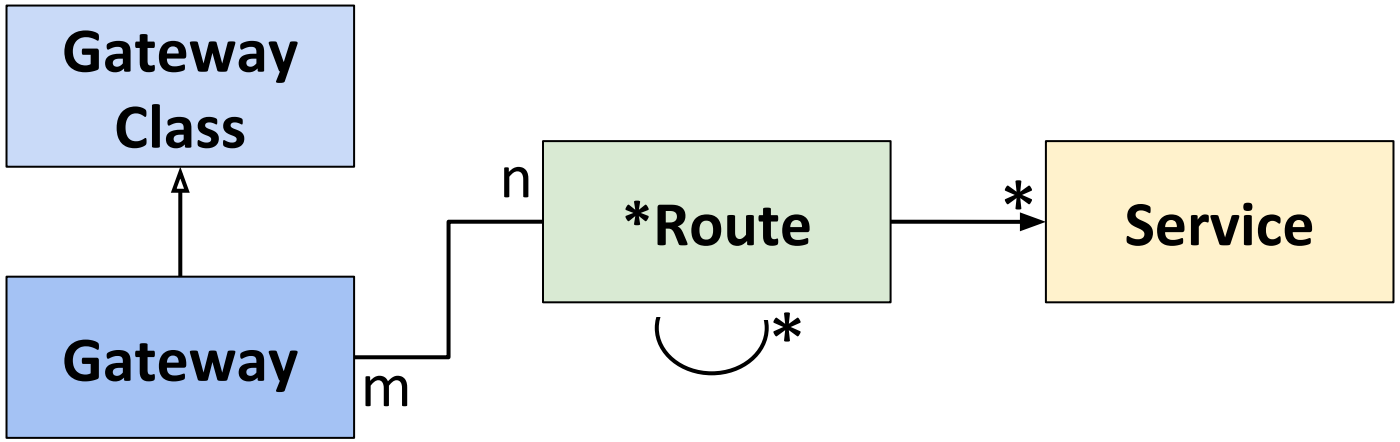
`GatewayClass` is created by the infrastructure provider (kinds of LB available)

`Gateway` is an instantiation of a given LB.

`*Route` (`HTTPRoute`) and `Services` are defined by the developer.



Gateway/Route schema



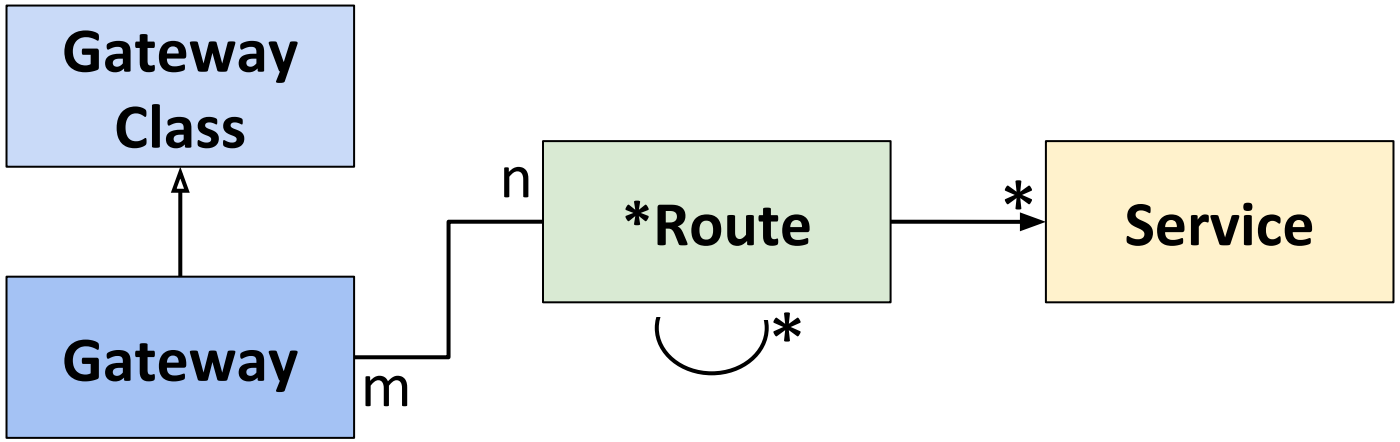

```
kind: GatewayClass
name: internet-lb

provider: acme.io/cloud
parameters:
  apiGroup: acme.io/cloud
  kind: Parameters
  name: ...
```

```
apiGroup: acme.io/cloud
kind: Parameters

public: true
```

```
kind: GatewayClass
name: private-lb

provider: acme.io/cloud
parameters:
  apiGroup: acme.io/cloud
  kind: GatewayParameters
  name: ...
```

```
apiGroup: acme.io/cloud
kind: Parameters

public: false
```

Gateway/Route schema



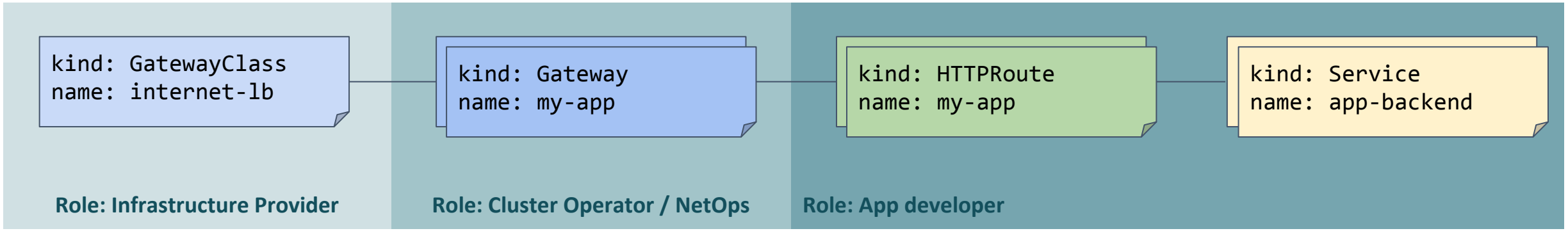

```
kind: GatewayClass
name: internet-lb
...
```

```
kind: Gateway
namespace: net-ops
name: the-gateway

class: internet-lb
listeners:
  -   port: 80
      protocol: http
routes:
  -   kind: HTTPRoute
      name: my-app
```

```
kind: HTTPRoute
name: my-app

rules:
  -   path: /my-app
      ...
gateways:
  -   namespace: net-ops
      name: the-gateway
```

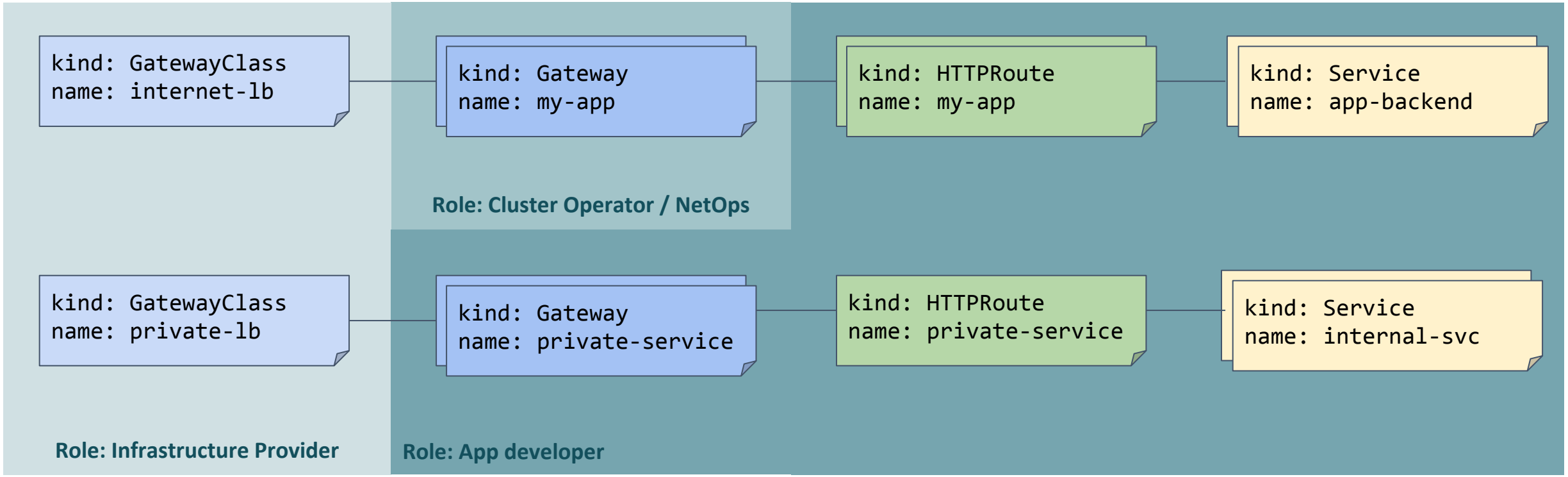
```
kind: Service
name: my-app
```

Roles and resources



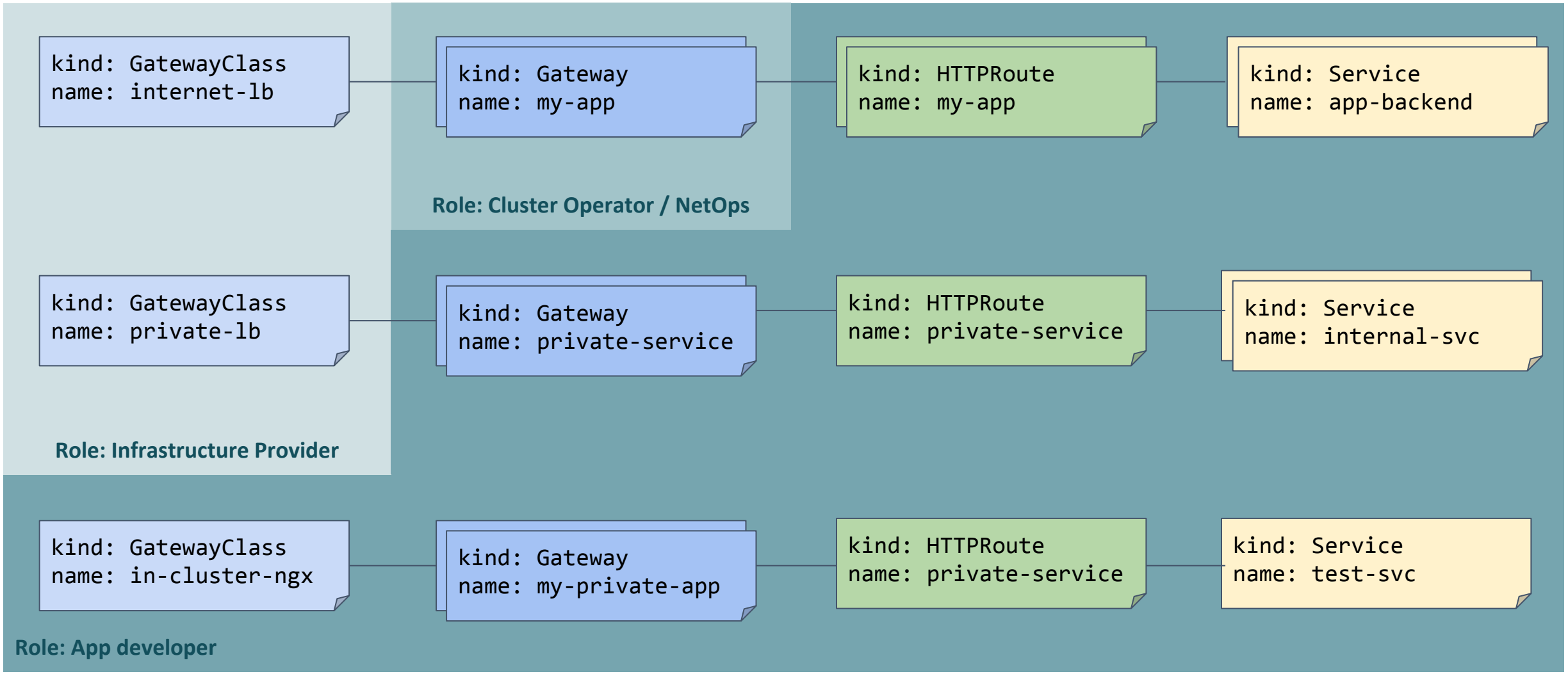
Team role separation

Roles and resources



Self-service Gateway

Roles and resources

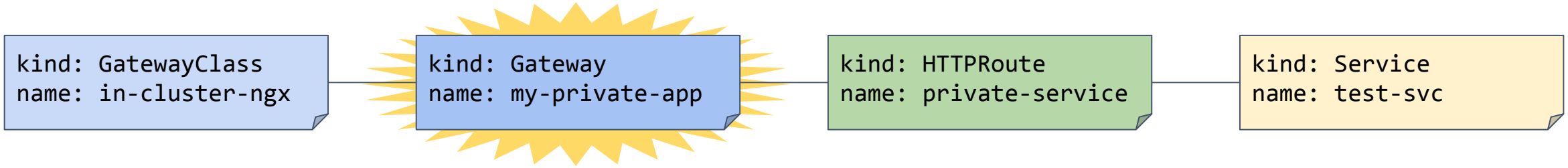


Fully self-service

Design Challenges

- Varied deployment patterns
- Multiple cooperating resources + roles
- Portability
- Extensibility
- Status

Gateway Deployment Models



Controller then

… configures a cloud LB / Gateway

… instantiatiates a proxy instance in cluster / Gateway

… merges Gateways together into a configuration,
reloads configuration.

Designing for RBAC

USER | ROLE | RESOURCE | VERB

USER ROLE

Alice can act as a **Cluster Operator** with

VERB

permission to **Update** the configuration

RESOURCE

of a **Gateway**.

Designing for RBAC

Resource boundaries should be split based on responsibilities.

However, “handshake” required between Gateway/Route:

- Protect who can use a given Gateway (“no Internet for you”)
- Self-service attachment (Gateway ← Route)
- Some users want to control who can export a *Route.

Most natural modeling is **explicit double-sided links**.

Portability

Core

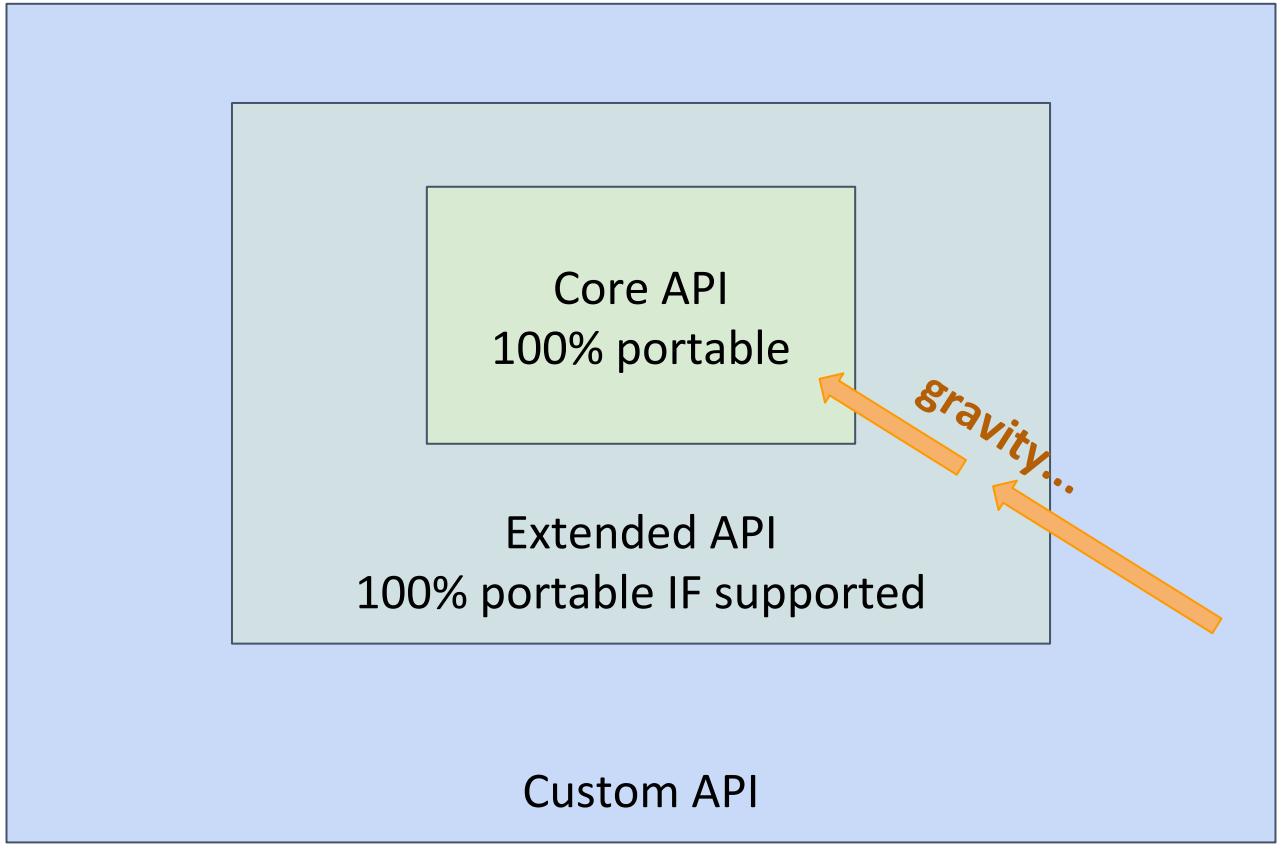
MUST be supported.

Extended

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

Custom

No guarantee for portability, No k8s API schema.

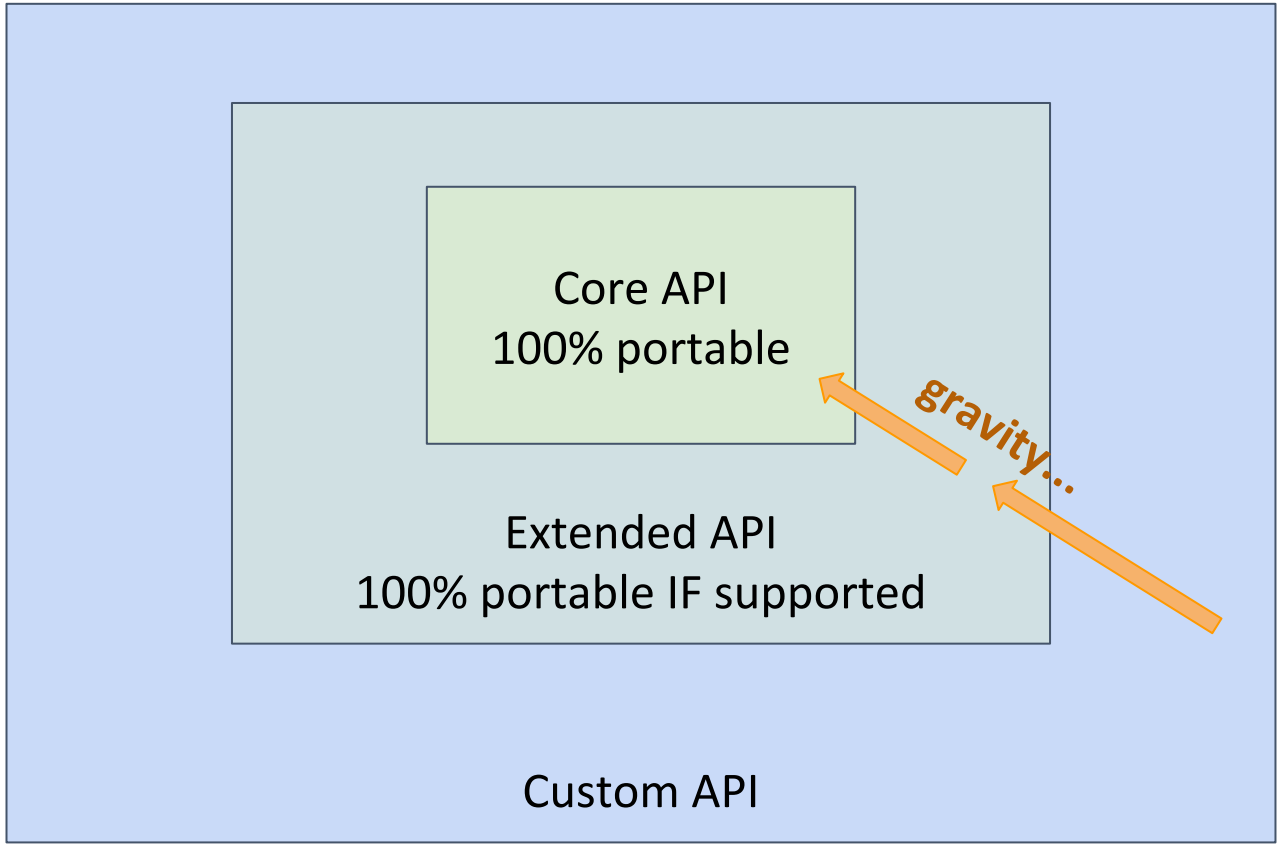


Portability

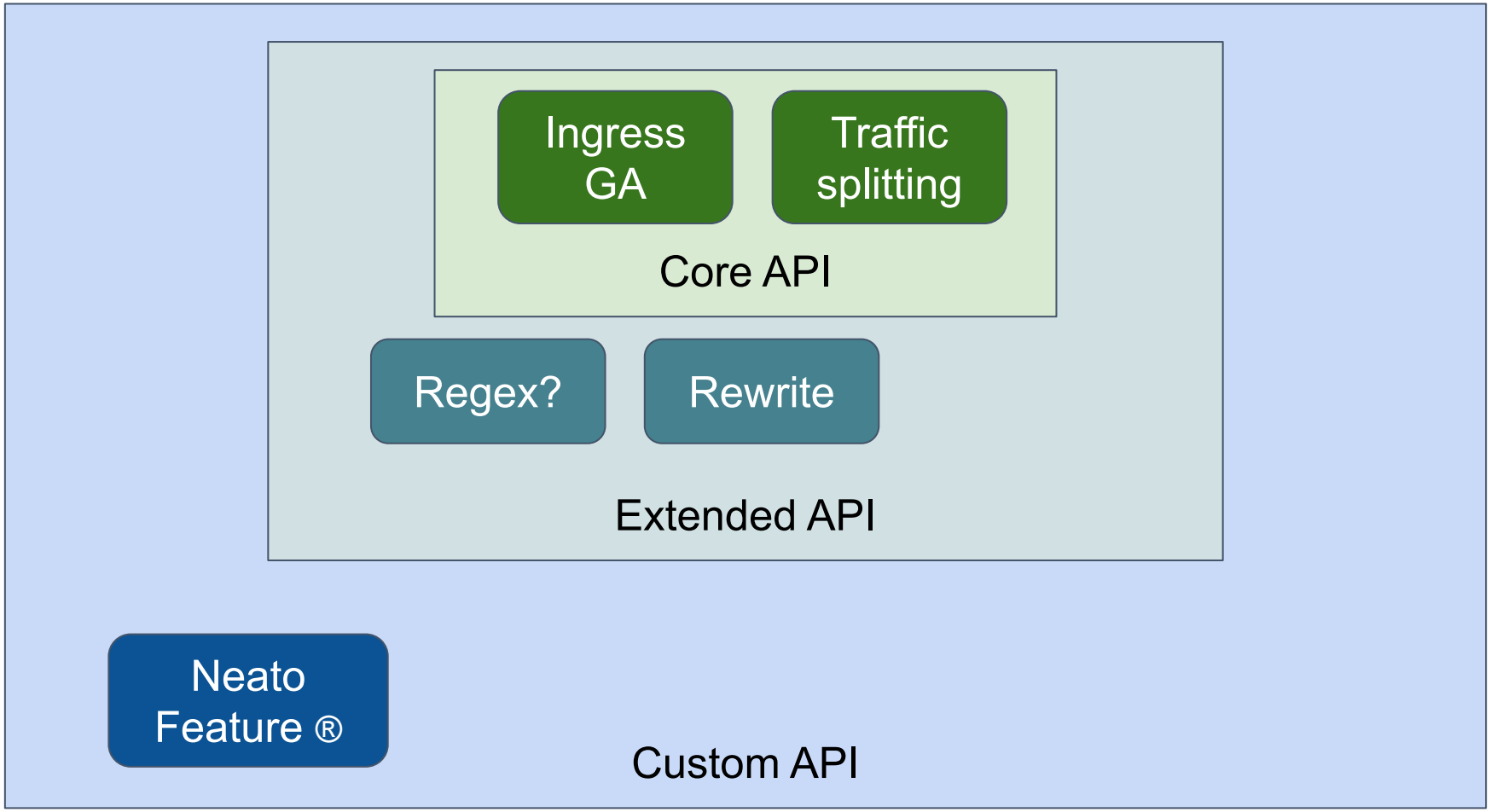
Enforcement by conformance tests.

Extended feature definition **requires** self-contained conformance.

Require all extended features be checkable statically.



Portability



Extensions

Extensibility mechanisms:

annotations
vs map[string]string
vs Custom Resource (polymorphic links)
vs Raw Objects (inline JSON)

Status

Long neglected aspect of LB APIs

More complex due to multiple resource composition/references.

Events : ephemeral (“something changed”)

vs Status (“current state”)

How to get involved

API sketch is here: link

Working group (coming soon, info will go out):

- Bi-weekly meetings
- SIG-NETWORK mailing list (link)
- Slack channel
- github.com/kubernetes-sigs/service-apis

Help wanted:

- Feedback on the proposal **(users AND implementers)**
- Experimental implementations

Q&A

Who are we?

Christopher M Luciano
cmluciano@

Bowei Du
bowei@