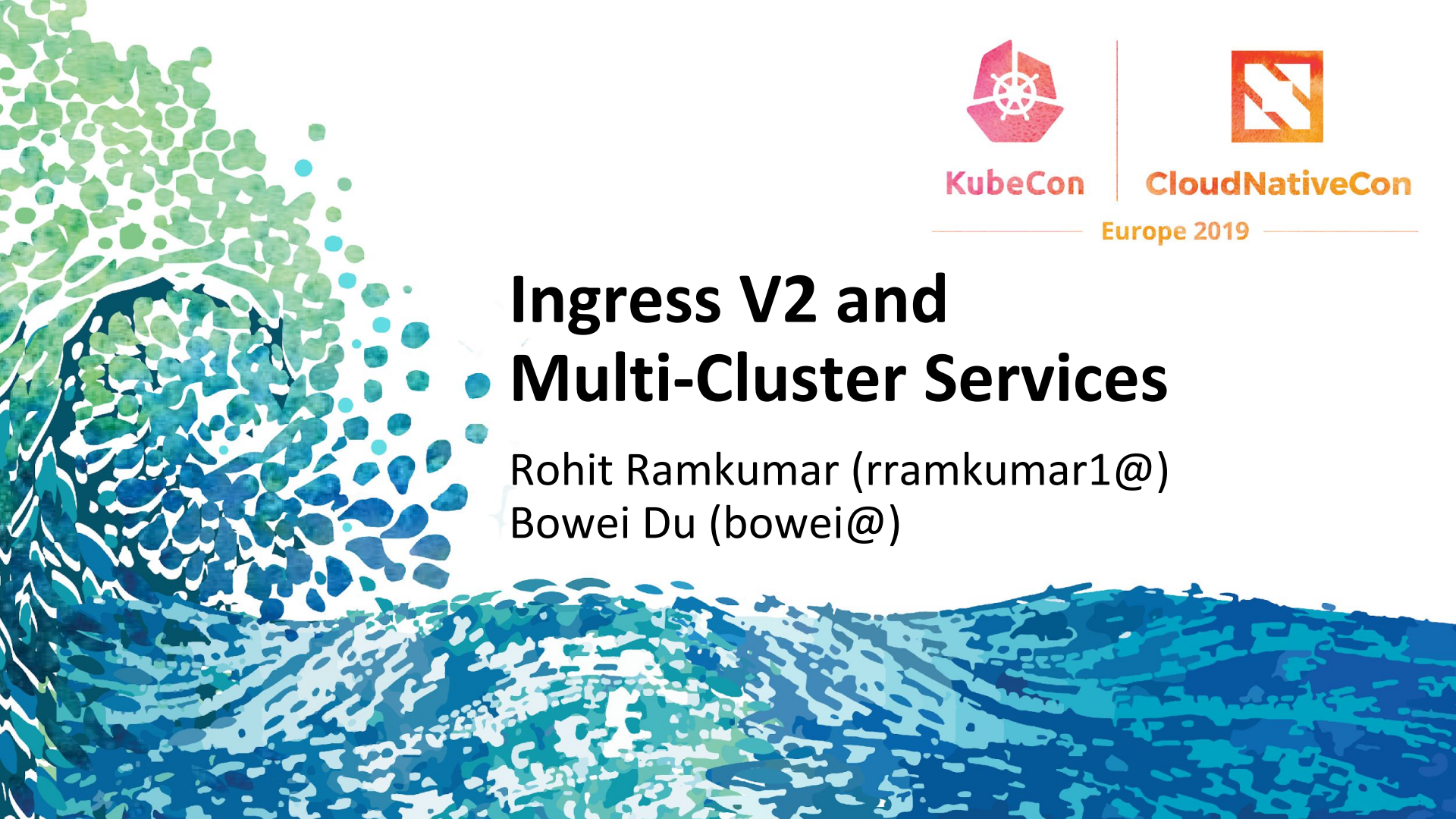
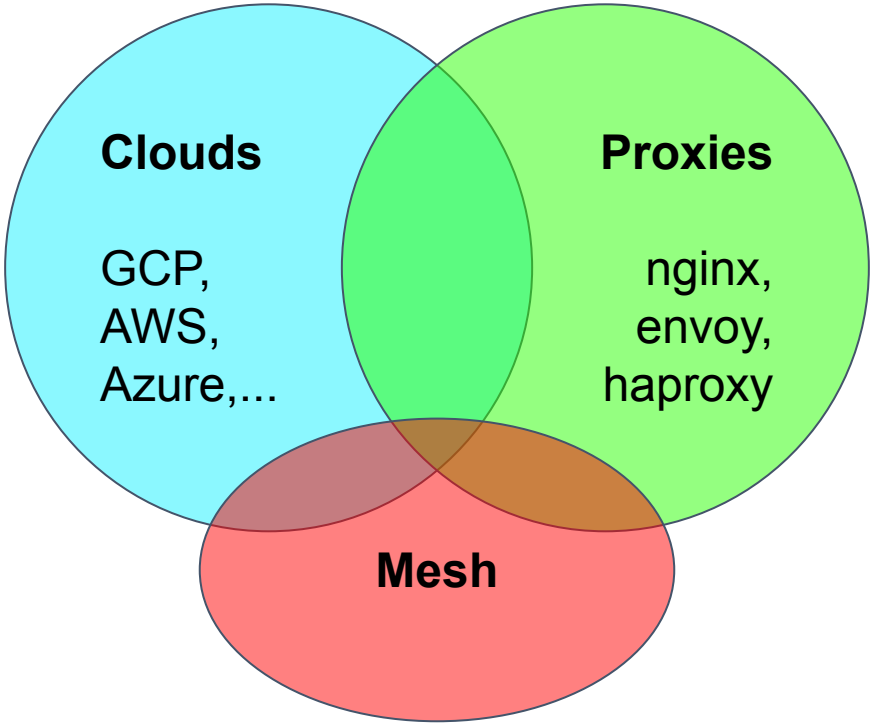


Ingress V2 and Multi-Cluster Services

Rohit Ramkumar (rramkumar1@)
Bowei Du (bowei@)

Next “Ingress” API



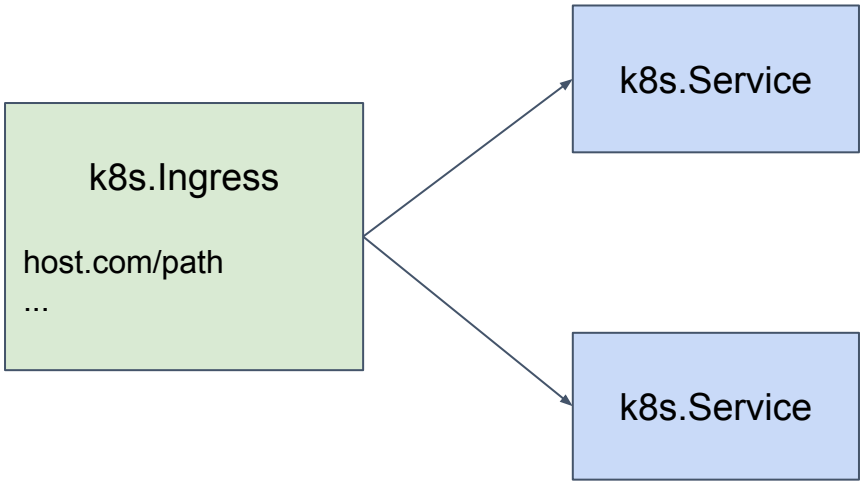
Next “Ingress” API

Challenges: portability, expressiveness, features

Lots of APIs now in this space.

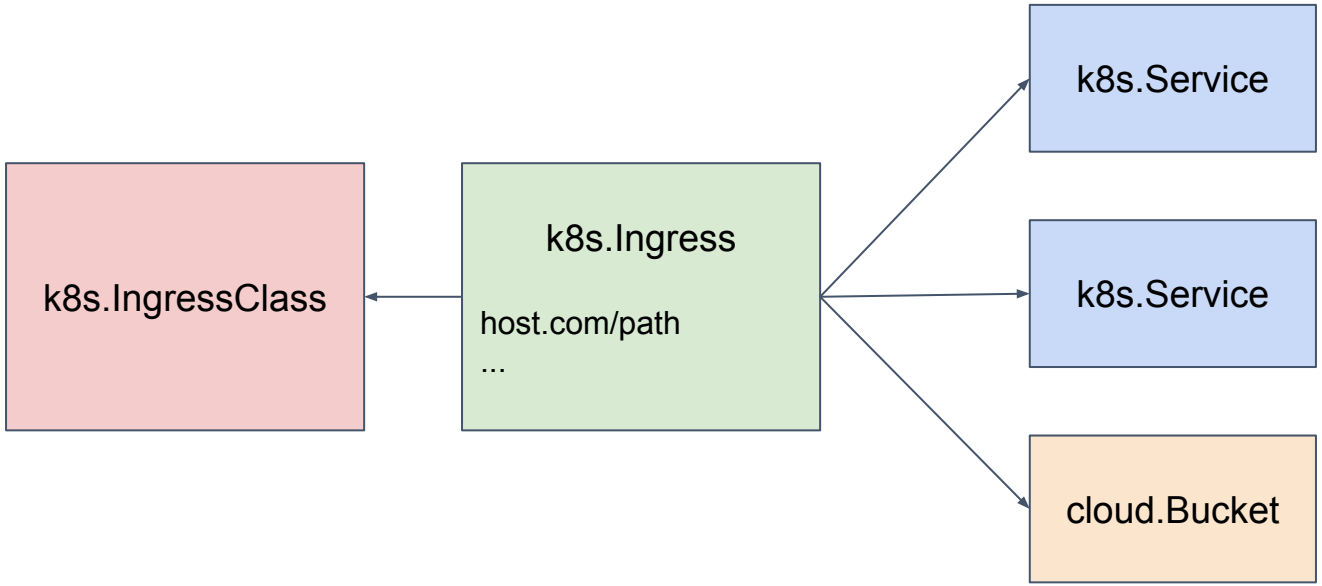
What can we can learn from them?

API models: Ingress



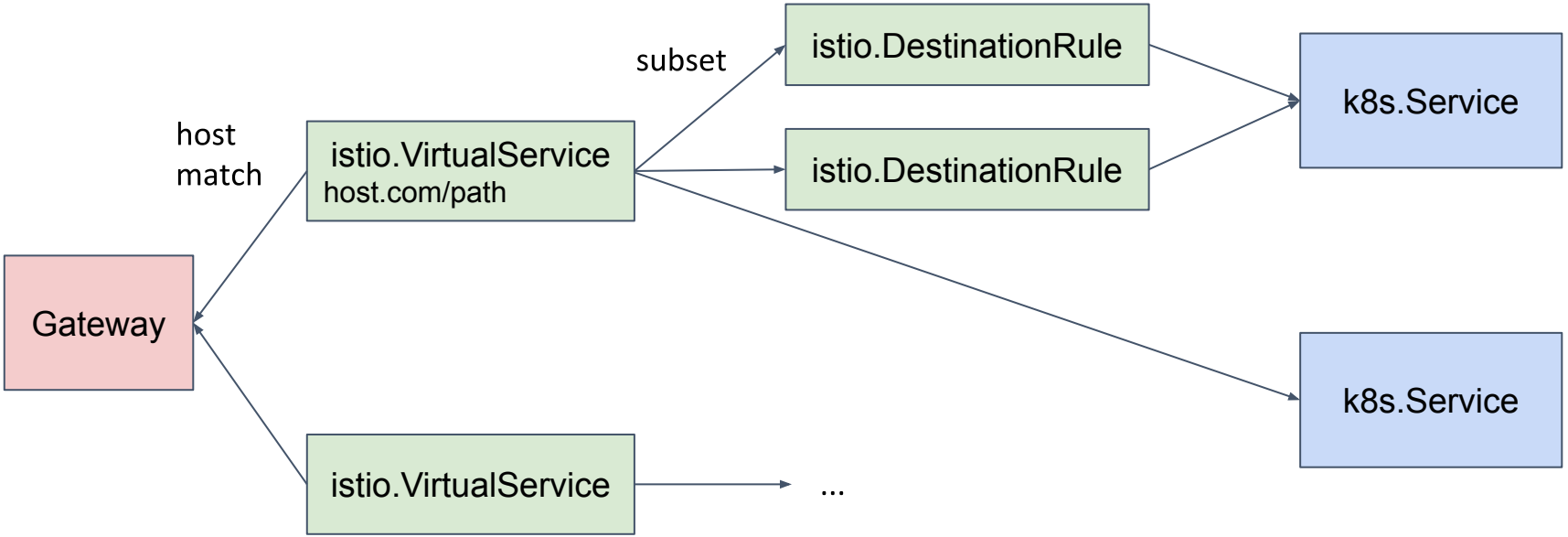
Key attributes: self-service, single-role, simple

API models: Ingress GA (proposed)



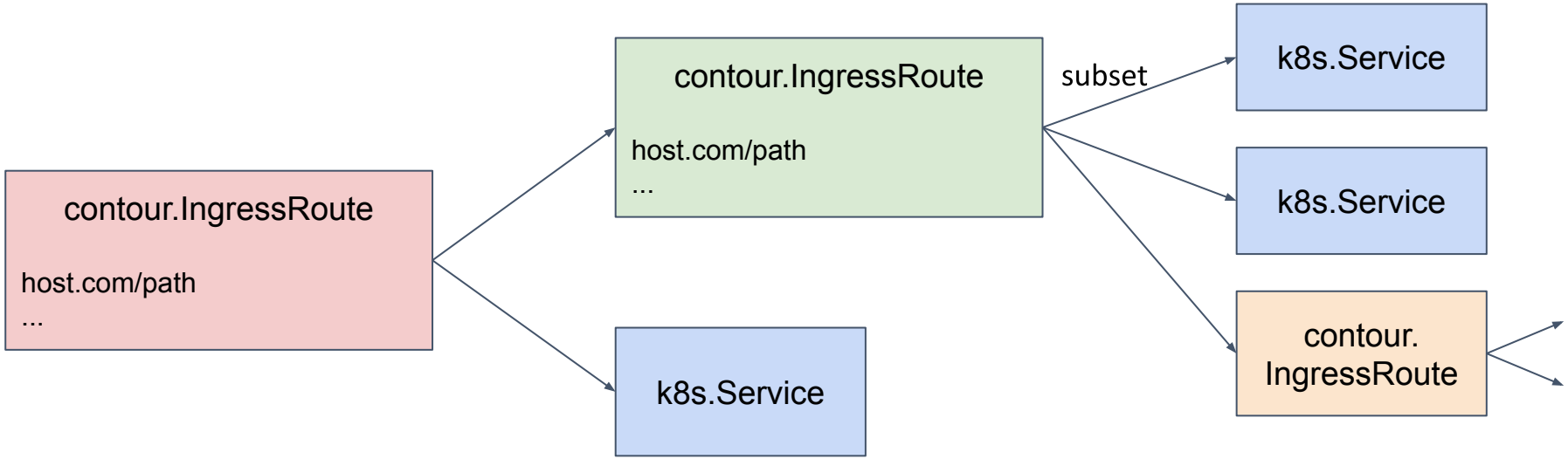
Key attributes: “flavor” of Ingress via class, heterogenous backends.

API models: Istio



Key attributes: separate roles for proxy infrastructure, application definition; rich feature set based on Envoy

API models: IngressRoute



Key attributes: Separate roles and recursive delegation, composability, additional features such as traffic splitting.

API models: Others

Lots of annotations for features (Everybody)

Decorators on k8s.Service instead of k8s.Ingress (Ambassador)

Use Custom Resources (Gloo)

What have we learned?

Multi-role environments

- Infrastructure vs App dev

How much portability?

- Should be a user choice.

Support future API growth

- Claim: providers/features will converge over time (but quite slowly).

A modest proposal

Warning: this is very early…

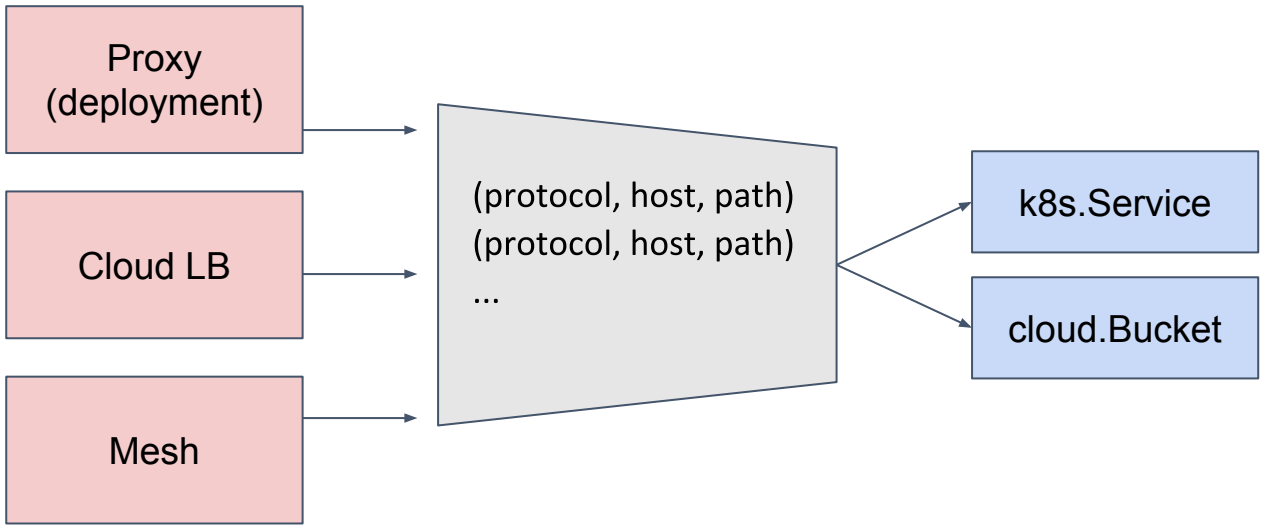
Taking from existing work...

Shape of the resources (model)

Addressing portability and extensibility

Future directions

A modest proposal: the model

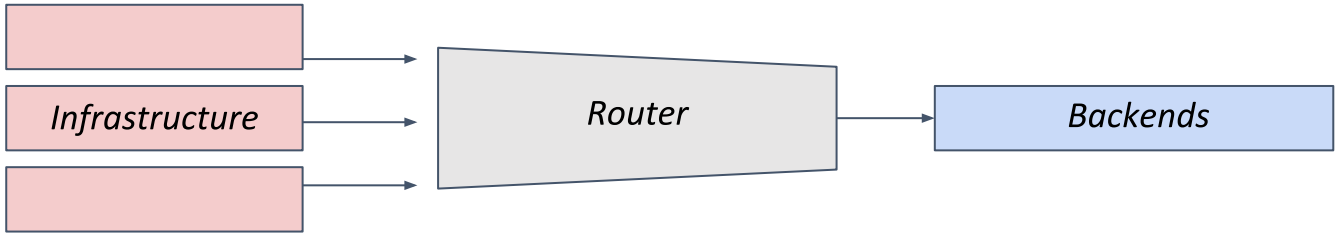


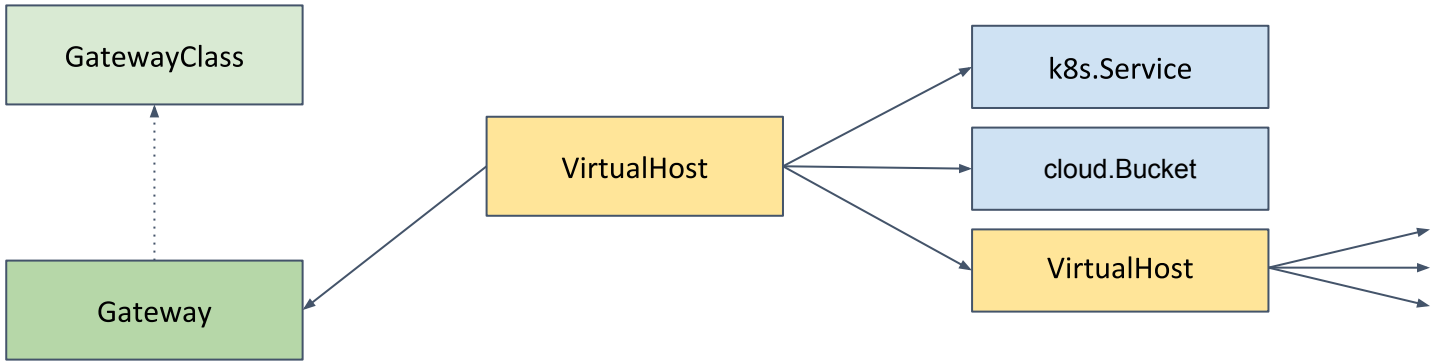
Infrastructure
[LB implementation]

Router
[Application description]

Backends
[Application implementation]

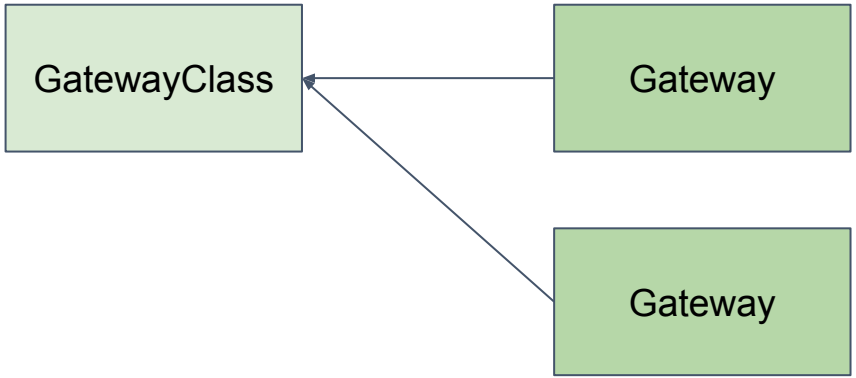
A modest proposal: the model





*Names are temporary

Gateway



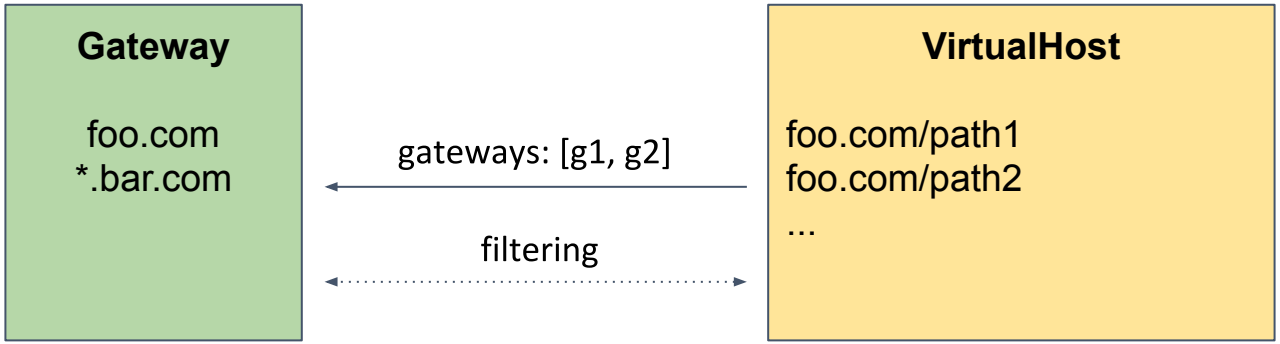
Deployment specific options - e.g. mergeable

Abstracts available implementations

Represents actual instance of LB/proxy infrastructure, capacity

Protocol termination (<IP:port>, TLS)

Gateway ↔ VirtualHost



How to match Gateway and VirtualHost?

- VirtualHost attaches to a Gateway
- Gateway filters VirtualHosts -- wildcard allows for self-service.

Portability

Core

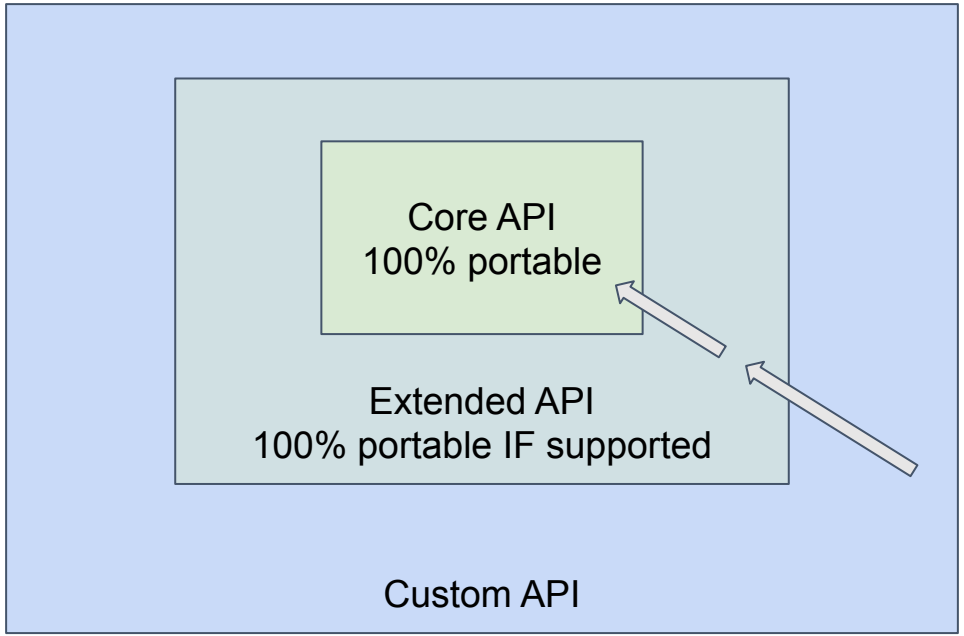
MUST be supported.

Extended

Feature by feature.
MAYBE supported, but
MUST be portable.
Part of k8s API schema.

Custom

No guarantee for portability,
No k8s API schema.

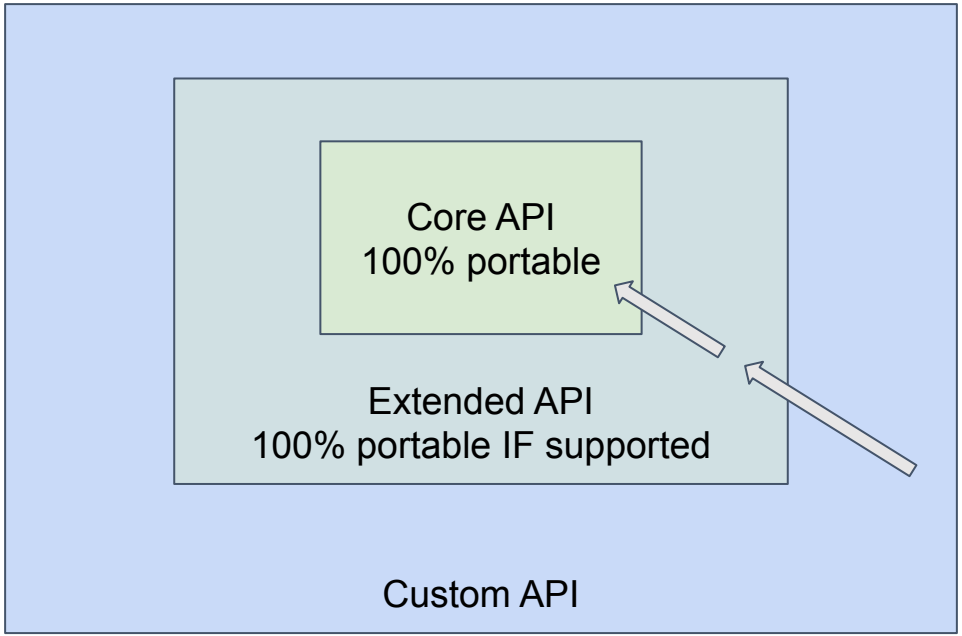


Portability

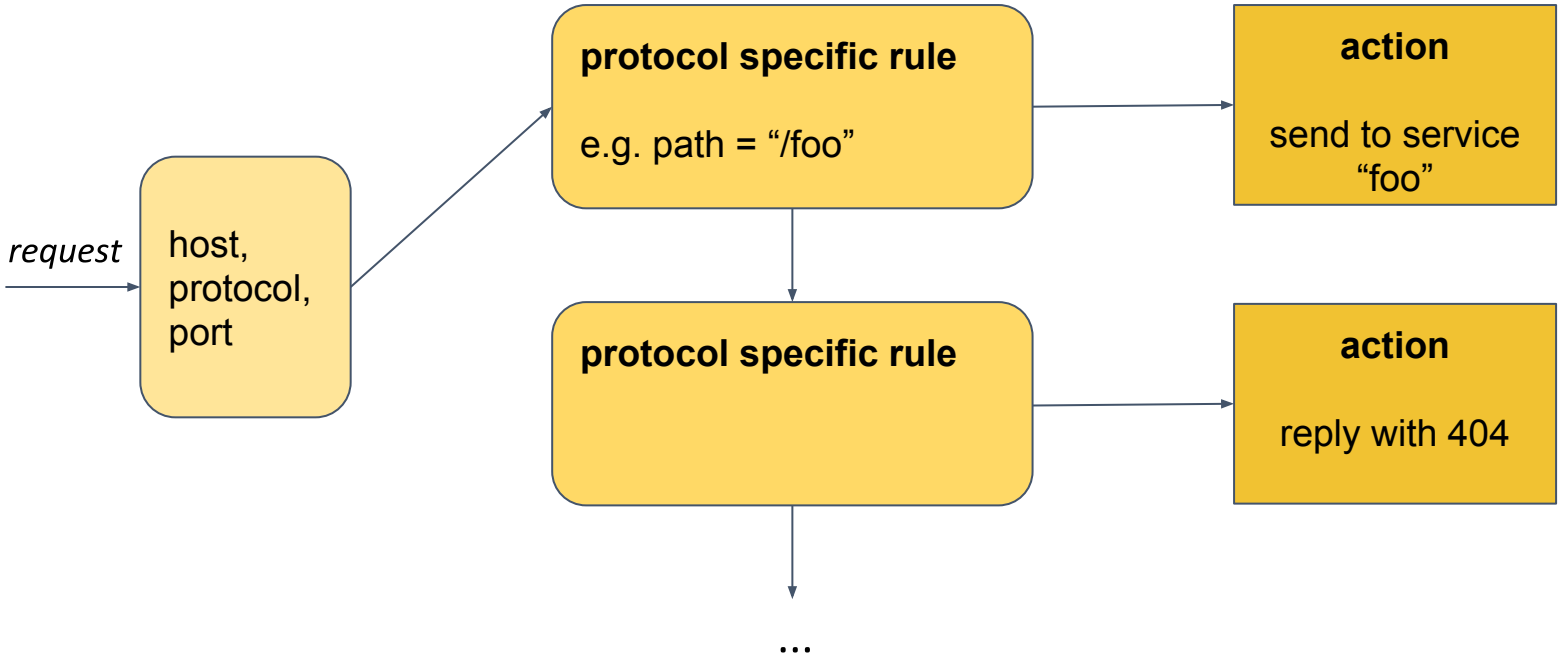
Enforcement by conformance tests

Extended feature definition requires self-contained conformance.

Require all extended features be checked statically?



VirtualHost request matching



VirtualHost (warning: sketch)


```
type VirtualHost struct {
     Routes []Route
}
type Route struct {
     HTTP *HTTPRoute
     TCP  *TCPRoute
}
type HTTPRoute struct {
     Host string
     Rules []HTTPRouteRule
}
type HTTPRouteRule struct {
     // Matchers, with room for extension
     Path     string
     // Actions, with room for extensions
     Backend    *HTTPRouteActionBackend
     StatusCode *HTTPRouteActionStatusCode
}
```

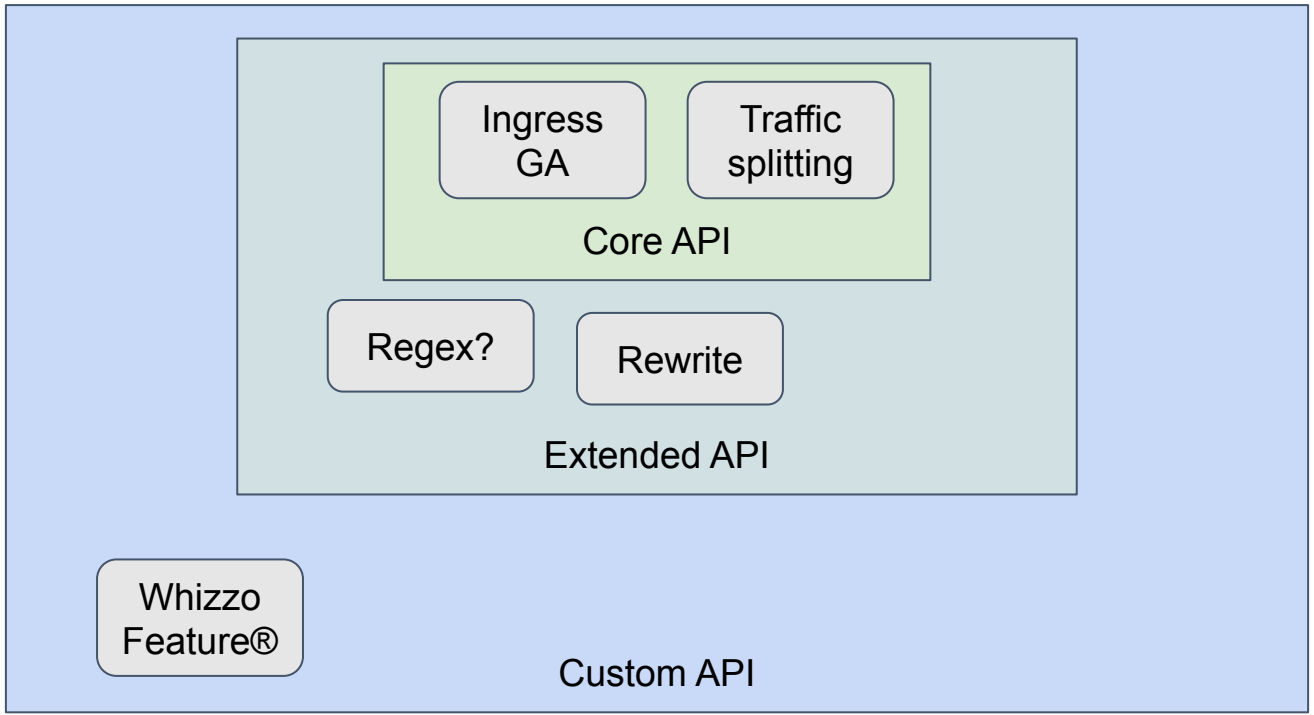
```
# VirtualHost
spec:
  routes:
  - http:
      host: "foo.com"
      rules:
      - path: "/"
        backend:
          service: foo-app
          servicePort: www
  - tcp:
      port: 9000
      rules:
      - backend:
          service: tcp-app
          servicePort: my-protocol
```

VirtualHost extensibility

Complex syntax tree -- needs a decorator pattern

- Better API machinery?
- String/String vs Raw Objects (inline CRDs) vs CRD link

Portability



Future directions

Interesting alternative backends:

- Storage bucket
- Multi-cluster Services

Multicluster Service

Multi-Cluster Services: Use Cases

- Canary - Deploy new version of application in one Kubernetes cluster in one locality before rolling it out worldwide.
- Low latency - Users get routed to the application backends that are closest to them geographically.
- High availability - Users requests are served even if one cluster holding application backends is completely down.

Multi-Cluster Services: User Journey

Persona: Network Admin

- I want **L7 load balancing across** a global service that is deployed on Kubernetes clusters.
- I want the ability to add / remove clusters from this load balancing based on business requirements.
- I want teams in my organization (service owners) to manage their own ingress traffic but maintain a clear boundary between the “admin” and the “Kubernetes end user”.
- I want my teams to use a native Kubernetes API to leverage default features (e.g namespacing, labels) & to reduce the learning curve.

Multi-Cluster Services: APIs


```
kind: MultiClusterIngress
metadata:
  name: my-mci
spec:
  template:
    spec:
      backend:
        serviceName: my-mc-service
        servicePort: 80
      rules:
      - host: foo.bar.com
        http:
          paths:
          - backend:
              serviceName: my-mc-service
              servicePort: 80
```

```
kind: MultiClusterService
metadata:
  name: my-mc-service
spec:
  template:
    selector:
      app: shopping
    ports:
    - name: web
      protocol: TCP
      port: 80
      targetPort: 80
  clusters:
  - selector:
    matchLabels:
      region: us
```

Multi-Cluster Services: APIs


```
kind: MultiClusterIngress
metadata:
  name: my-mci
spec:
  template:
    spec:
      backend:
        serviceName: my-mc-service
        servicePort: 80
      rules:
      - host: foo.bar.com
        http:
          paths:
          - backend:
              serviceName: my-mc-service
              servicePort: 80
```

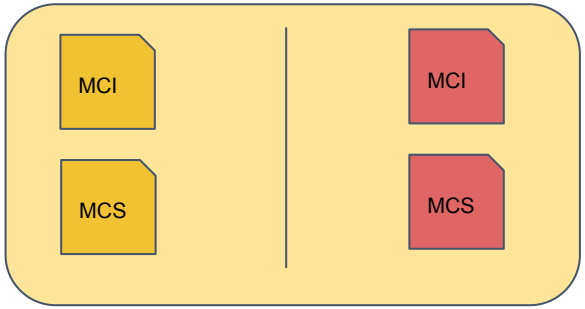
```
kind: MultiClusterService
metadata:
  name: my-mc-service
spec:
  template:
    selector:
      app: shopping
    ports:
    - name: web
      protocol: TCP
      port: 80
      targetPort: 80
  clusters:
  - selector:
    matchLabels:
      region: us
```

Multi-Cluster Services: APIs


```
kind: MultiClusterIngress
metadata:
  name: my-mci
spec:
  template:
    spec:
      backend:
        serviceName: my-mc-service
        servicePort: 80
      rules:
      - host: foo.bar.com
        http:
          paths:
          - backend:
              serviceName: my-mc-service
              servicePort: 80
```

```
kind: MultiClusterService
metadata:
  name: my-mc-service
spec:
  template:
    selector:
      app: shopping
    ports:
    - name: web
      protocol: TCP
      port: 80
      targetPort: 80
  clusters:
  - selector:
    matchLabels:
      region: us
```

Multi-Cluster Services: Workflow

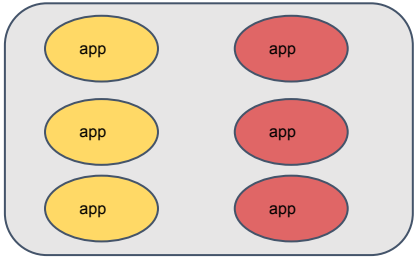


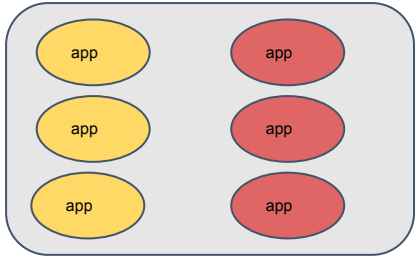
"Config Source"

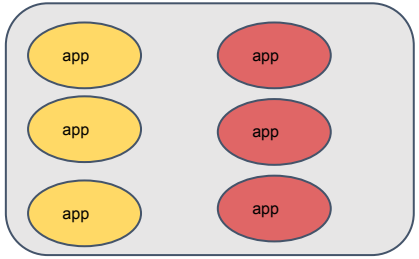
Admin

Cluster turnup / turndown

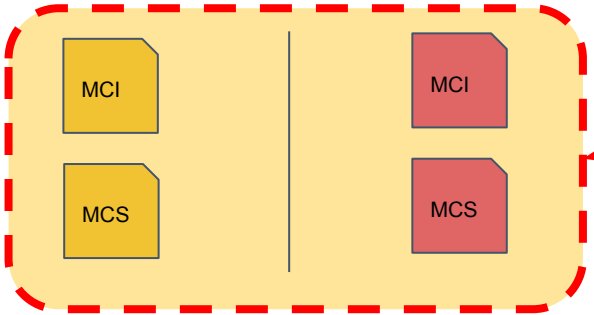
Developer Teams







Multi-Cluster Services: Workflow

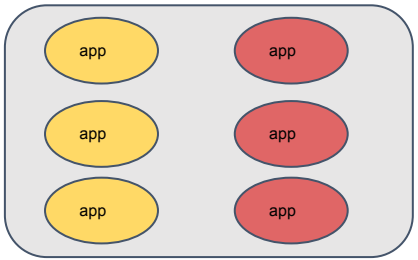


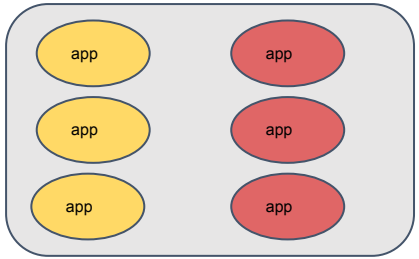
"Config Source"

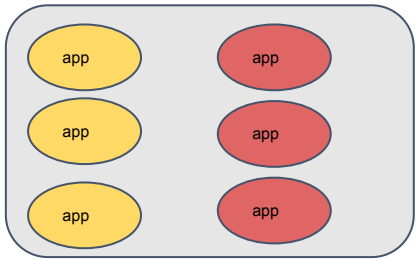
Admin

Cluster turnup / turndown

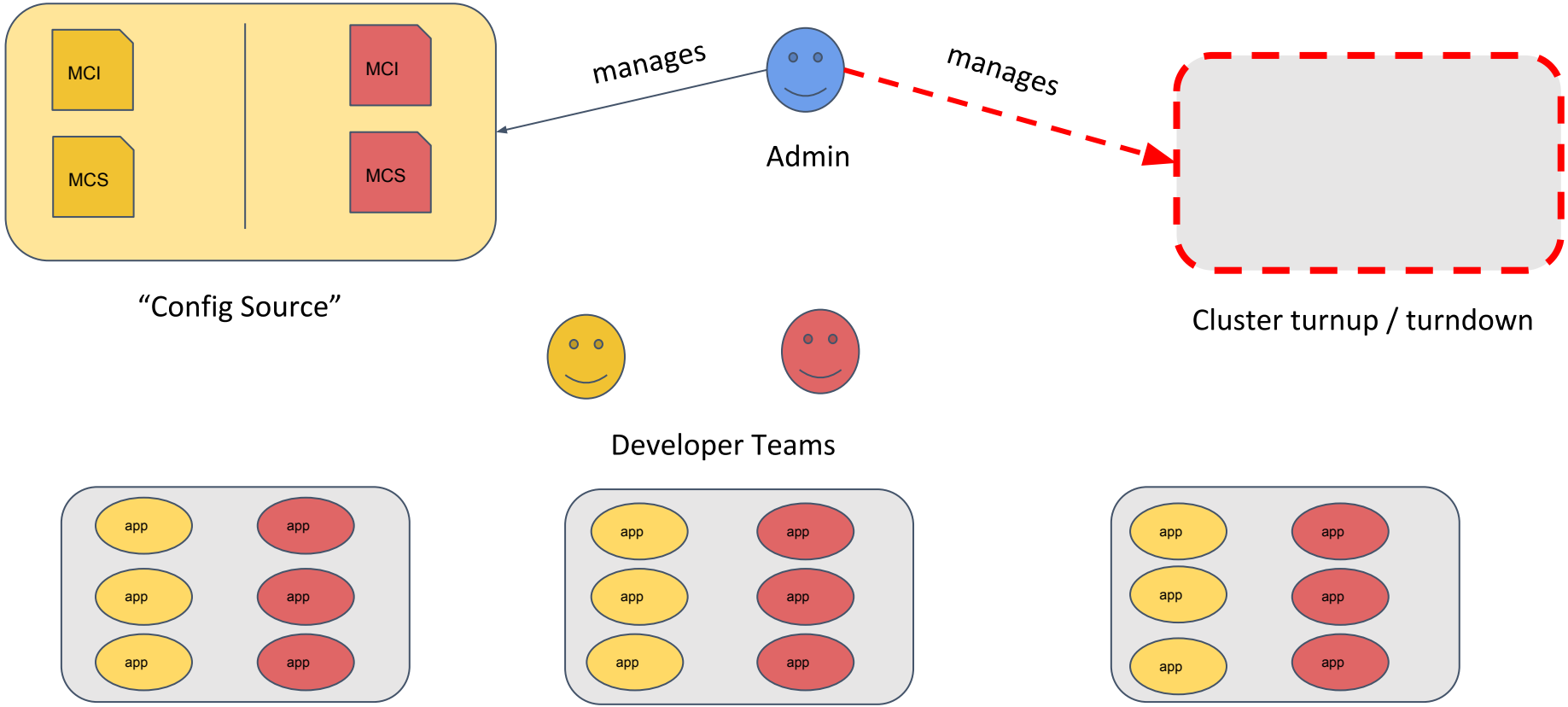
Developer Teams



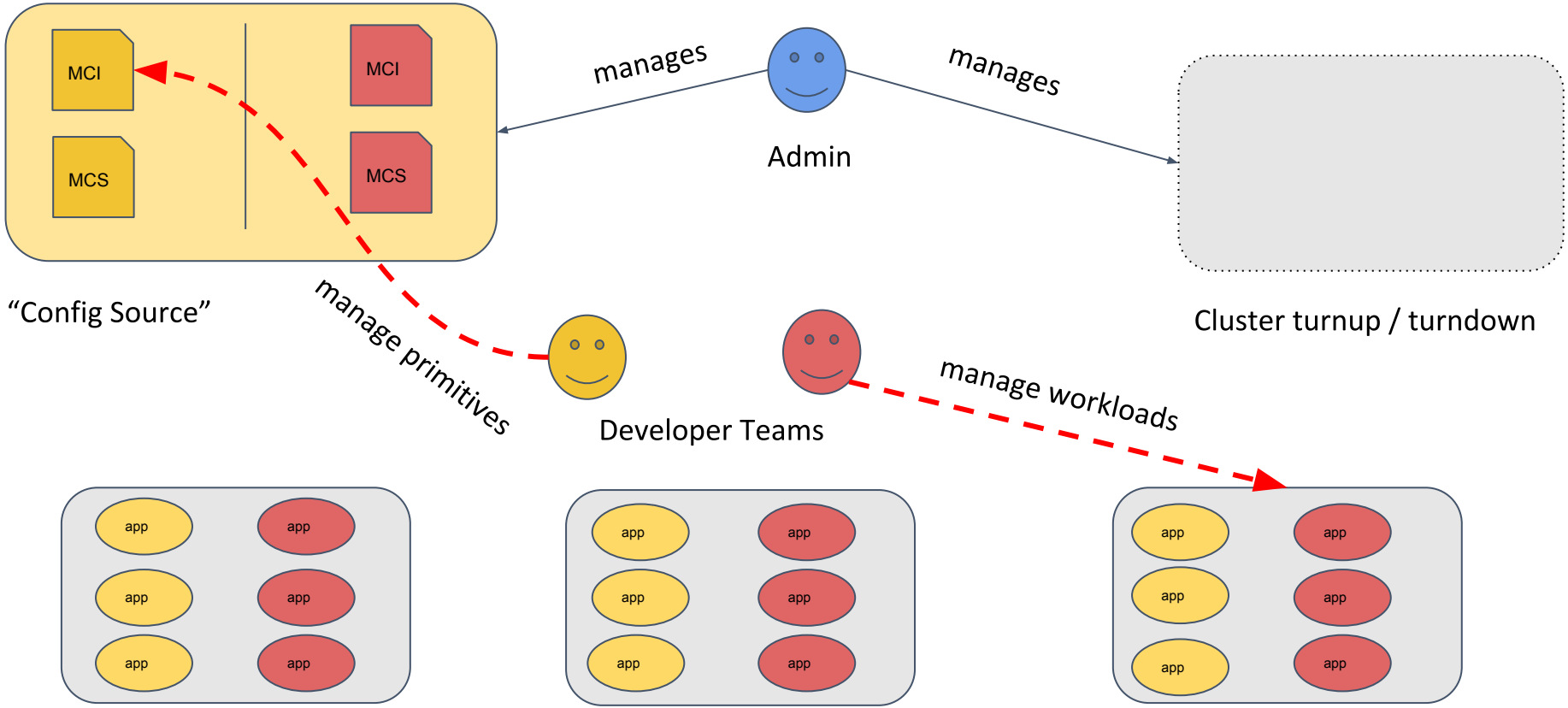




Multi-Cluster Services: Workflow



Multi-Cluster Services: Workflow



Multi-Cluster Services: FAQ

- *Why not have a MultiClusterDeployment? That alleviates the need for having to do the manual work of replicating their workload across clusters.*

  - Do not want to be opinionated on how users deploy their workloads. It's not really the spirit of what the API is trying to provide.

  - Best practices in this space are divided, no one-size fits all approach.

Multi-Cluster Services: FAQ

- *What about Federation v2?*
  - Multi-cluster models & assumptions are slightly different.
    - E.g FederatedIngress is replicated across clusters.
  - Federation v2 is opinionated on cross-cluster DNS & SD, deployment of workloads, etc.
  - In general, Federation v2 is trying to target a much wider set of use cases.

Multi-Cluster Services: FAQ

- Where does Istio's Multicluster story fit?
  - Potential for existing API's to support use case.

Conclusion

- Expect the KEP proposal to come shortly
- Feedback from community is key
- Reference implementation

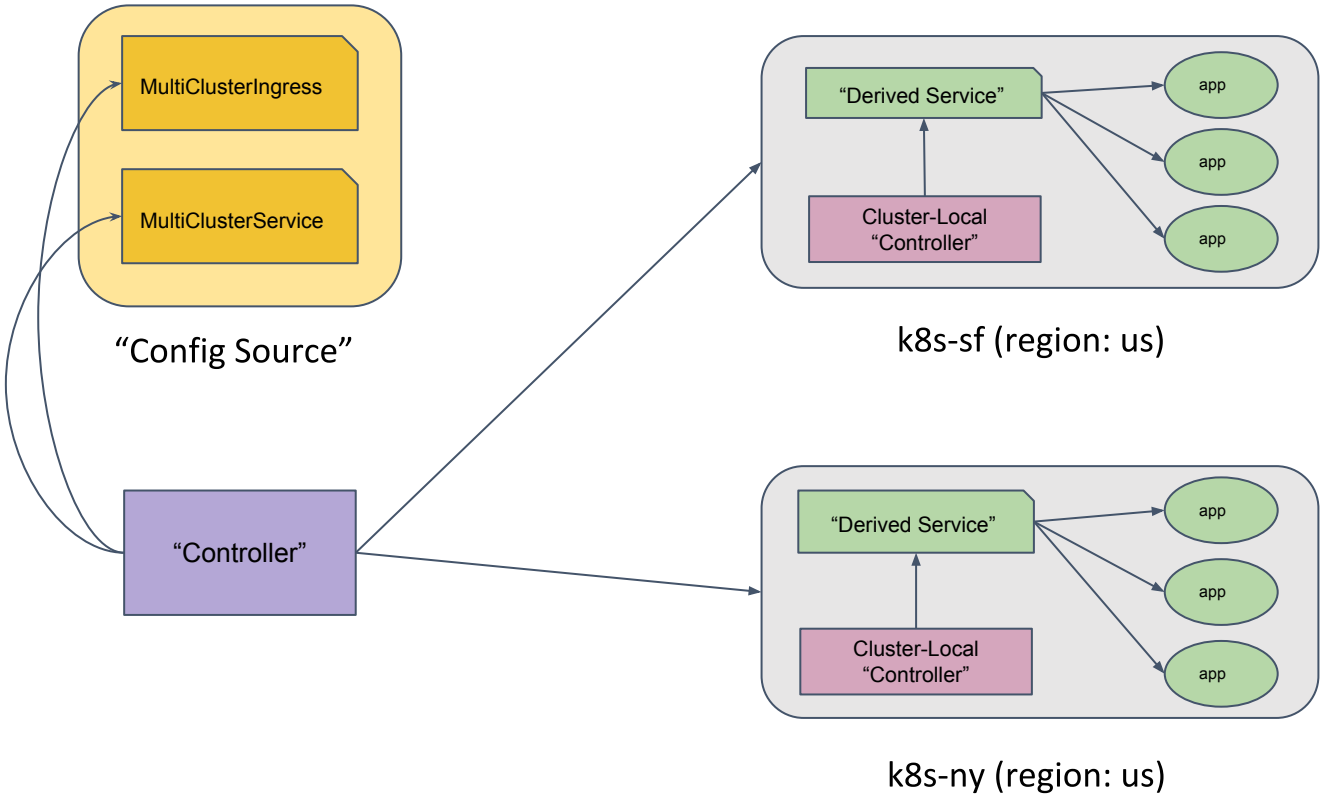
Thanks!

Rohit Ramkumar (rramkumar1@)

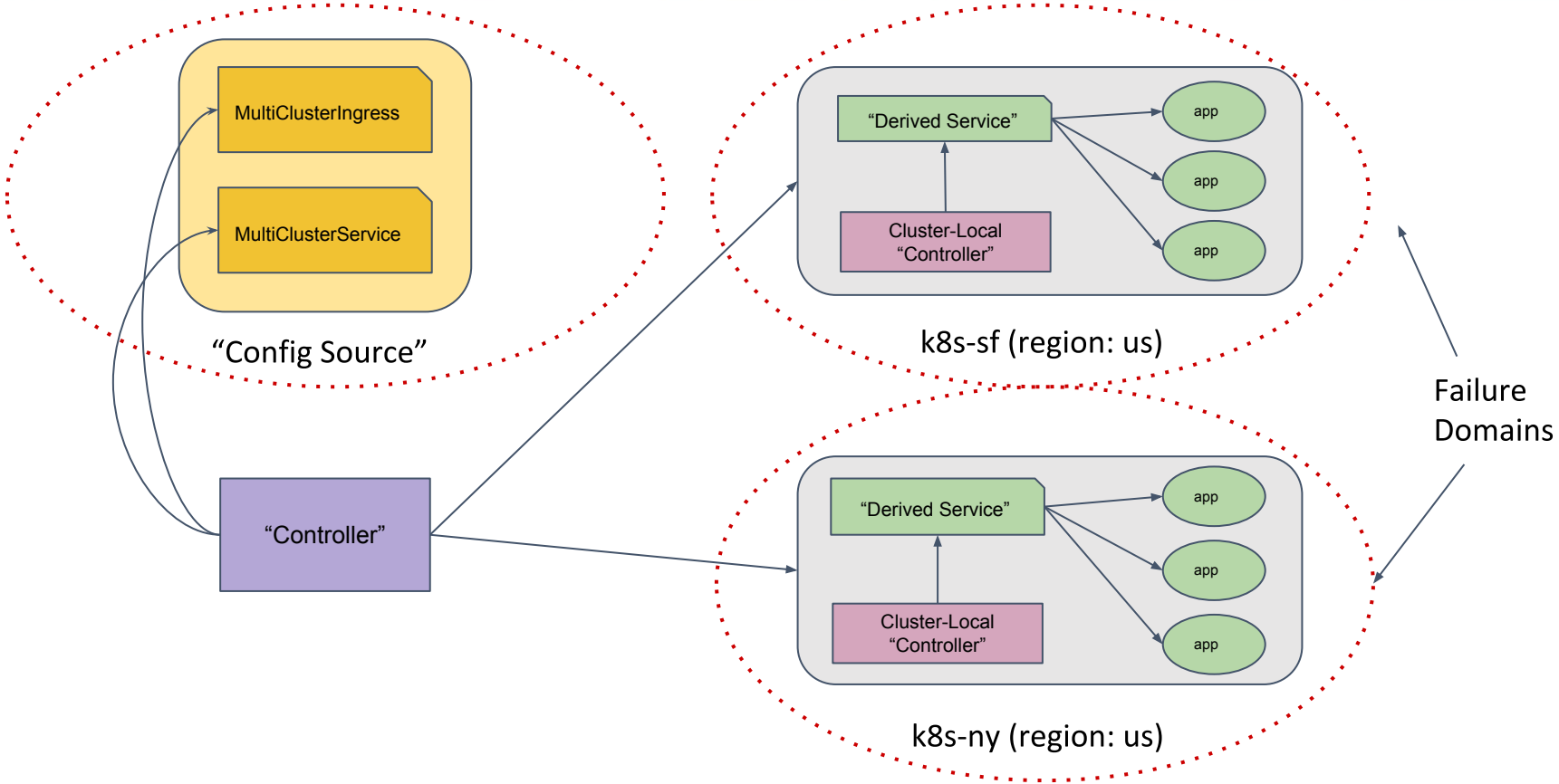
Bowei Du (bowei@)

Backup

Multi-Cluster Services: Reference Implementation



Multi-Cluster Services: Reference Implementation



Multi-Cluster Services: Reference Implementation