[Shared externally]

[SIG-NETWORK]

CompositeBackend proposal [DRAFT]

Status: Draft
Last updated: Oct 27, 2025

TL;DR

Provide a new resource (CompositeBackend) to enable flexible multi-cluster load balancing with Gateway API.

Goals

- Simplify multi-cluster load balancing with Gateway API.
- Support new multi-cluster use cases where xRoute objects target non-Service backends, such as InferencePool.

Non-goals

API compatibility with the Multicluster Services API (MCS).

Overview

We introduce a new API resource, CompositeBackend, which can be used in between an xRoute object and a backend object. Conceptually:

- CompositeBackend wraps one or more backends (via its backendRefs field).
- CompositeBackend is itself a backend (can be referenced via a backendRef field).
- CompositeBackend objects with the same namespaced name across a group of clusters are logically aggregated.

Consider the following example, where a Gateway and HTTPRoute are used to direct traffic to a Service in one cluster. We have Service objects in two other clusters, but they are not part of the load balancing configuration.

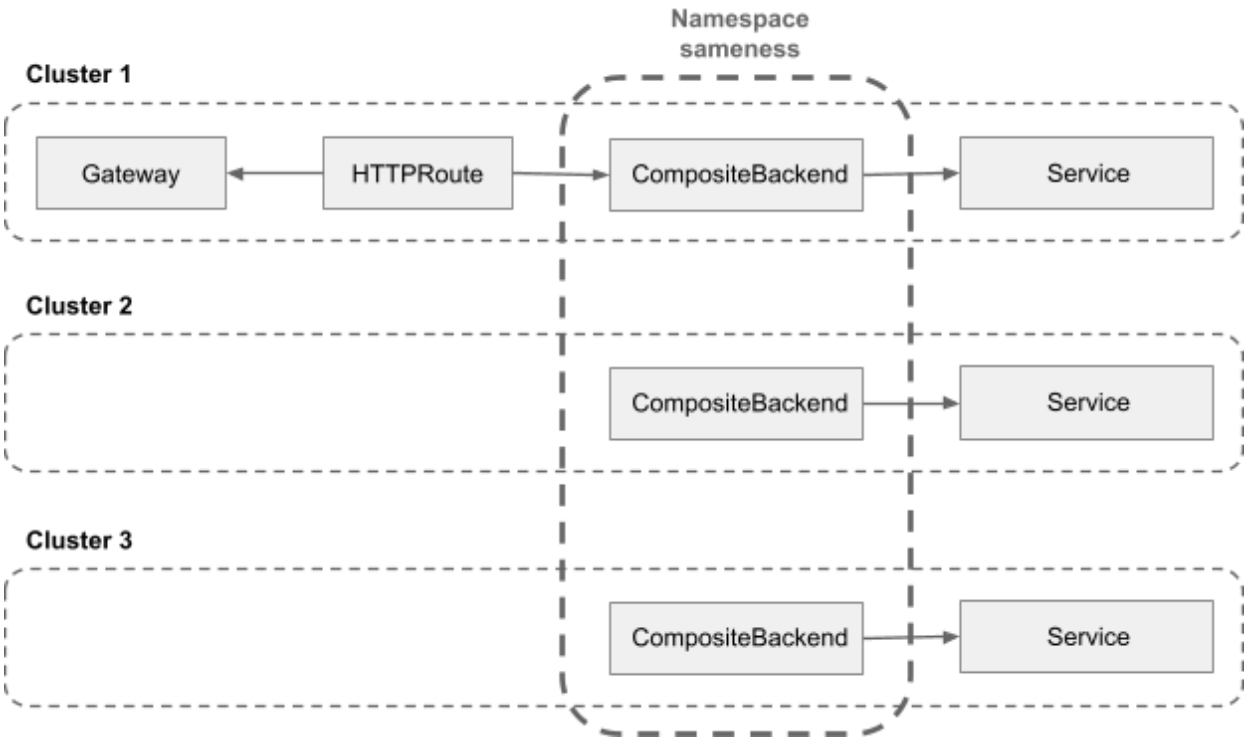
The YAML looks like this:

```yaml
kind: Gateway
metadata:
  name: bookinfo
spec:
  gatewayClassName: \
      some-class
  listeners:
  - name: bookinfo
    protocol: HTTP
    port: 8080
    allowedRoutes:
      kinds:
      - kind: HTTPRoute
```

```yaml
kind: HTTPRoute
metadata:
  name: bookinfo-route
spec:
  parentRefs:
  - name: bookinfo
  rules:
    backendRefs:
    - kind: Service
      name: bookinfo
      port: 8080
```

```yaml
kind: Service
metadata:
  name: bookinfo
spec:
  selector:
    app: bookinfo
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 8080
```

In order to enable multi-cluster load balancing, we create a CompositeBackend object in each cluster, which references the backend Service. In the first cluster, we modify the HTTPRoute to instead point at the CompositeBackend. Since namespace sameness is applied to the CompositeBackend objects, we get a load balancing configuration wired up that balances to all Services.



And the YAML looks like this:

```yaml
kind: Gateway
metadata:
  name: bookinfo
spec:
  gatewayClassName: \
      some-class
  listeners:
  - name: bookinfo
    protocol: HTTP
    port: 8080
    allowedRoutes:
      kinds:
      - kind: HTTPRoute
```

```yaml
kind: HTTPRoute
metadata:
  name: bookinfo-route
spec:
  parentRefs:
  - name: bookinfo
  rules:
    backendRefs:
    - kind: CompositeBackend
      name: bookinfo-backend
      port: 8080
```

```yaml
kind: CompositeBackend
metadata:
  name: bookinfo-backend
spec:
  targetRefs:
  - kind: Service
    name: bookinfo
```

```yaml
kind: Service
metadata:
  name: bookinfo
spec:
  selector:
    app: bookinfo
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 8080
```

Purpose

The purpose of this proposal is to enable a more flexible multi-cluster load balancing model within the Gateway API ecosystem. By itself, the CompositeBackend resource is not protocol-specific, does not control routing or DNS naming. It provides only an anchor which can implicitly aggregate backends across multiple clusters.

CompositeBackend is also backend-agnostic. It doesn't require the backend to be a Service. For example, it can work with InferencePool or EndpointSelector backends.

API

The spec and status of the CompositeBackend object are minimal:

```
None
apiVersion: gateway.networking.k8s.io/v1
kind: CompositeBackend
metadata:
  name: bookinfo
  namespace: bookinfo
spec:
  targetRefs:
  - group: v1
    kind: Service
    name: bookinfo
status:
  conditions:
  - type: Accepted
    status: "True"
    lastTransitionTime: "2025-10-09T20:35:10Z"
```

Note: CompositeBackend may **not** point at another CompositeBackend via backendRefs.

Conformance details

TODO

Standard graduation criteria

TODO

Alternatives

The main alternative considered is to extend MCS to work in a simpler way by:

- Supporting targeting non-Service backends explicitly.
- Making ClusterSetIP allocation optional.
- Making automatic ServiceImport object creation optional.

The reason we chose not to pursue this alternative is because this will lead to a larger MCS specification, with a more complex set of optional features which may not be provided by all implementations. Whereas we feel CompositeBackend can have a small and precise specification that is not hard to implement.

References

- Multicluster Services API (MCS)
- Namespace sameness
- [SIG-NETWORK] Multi-Cluster Inference Gateways
- [SIG-NETWORK] ClusterIP Gateway