Kubernetes Cluster Federation: How To Write A Federated Controller

Nikhil Jindal, Google
Madhusudan C.S., Google

Overview

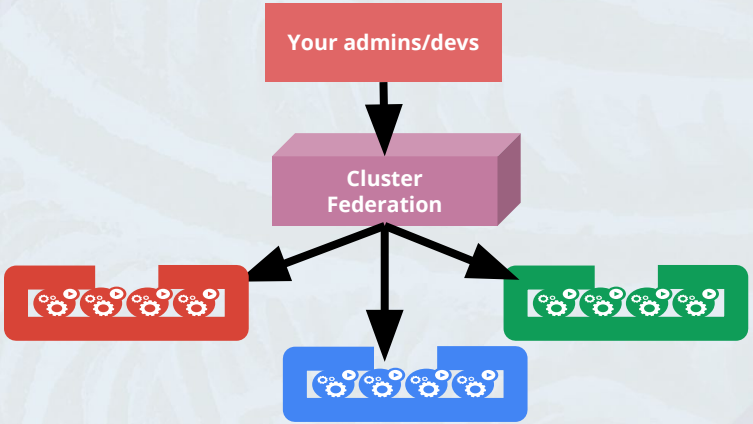
Cluster federation makes it easy to manage multiple clusters.

Why multiple clusters?

- Serving users from clusters closest to them
- High availability
- Scalability
- Avoiding vendor lock-in

Why federation?

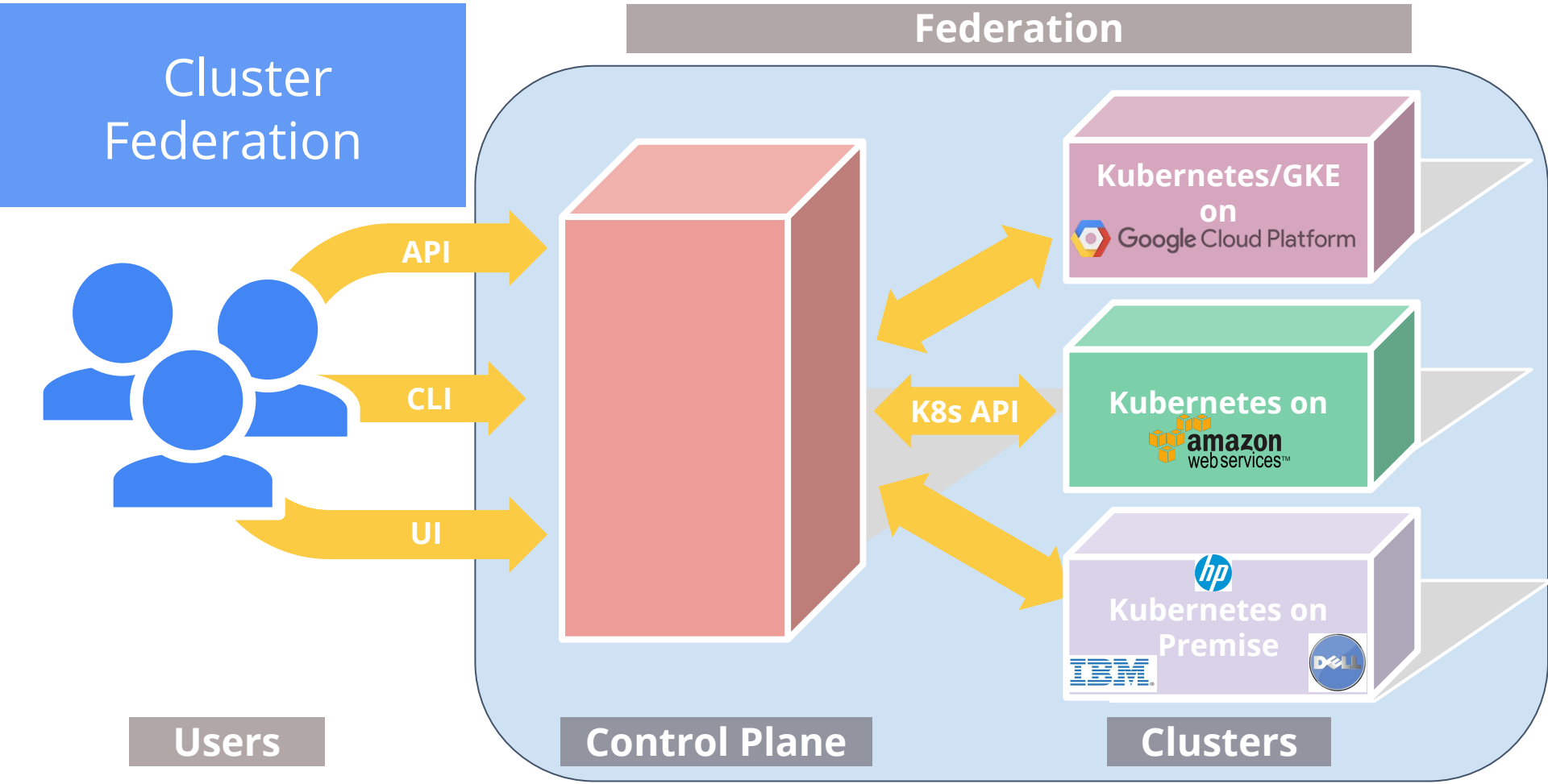
- Keep your app synced across clusters
- Configure network resources (services, ingress) to route traffic across clusters
- Single place to apply policies



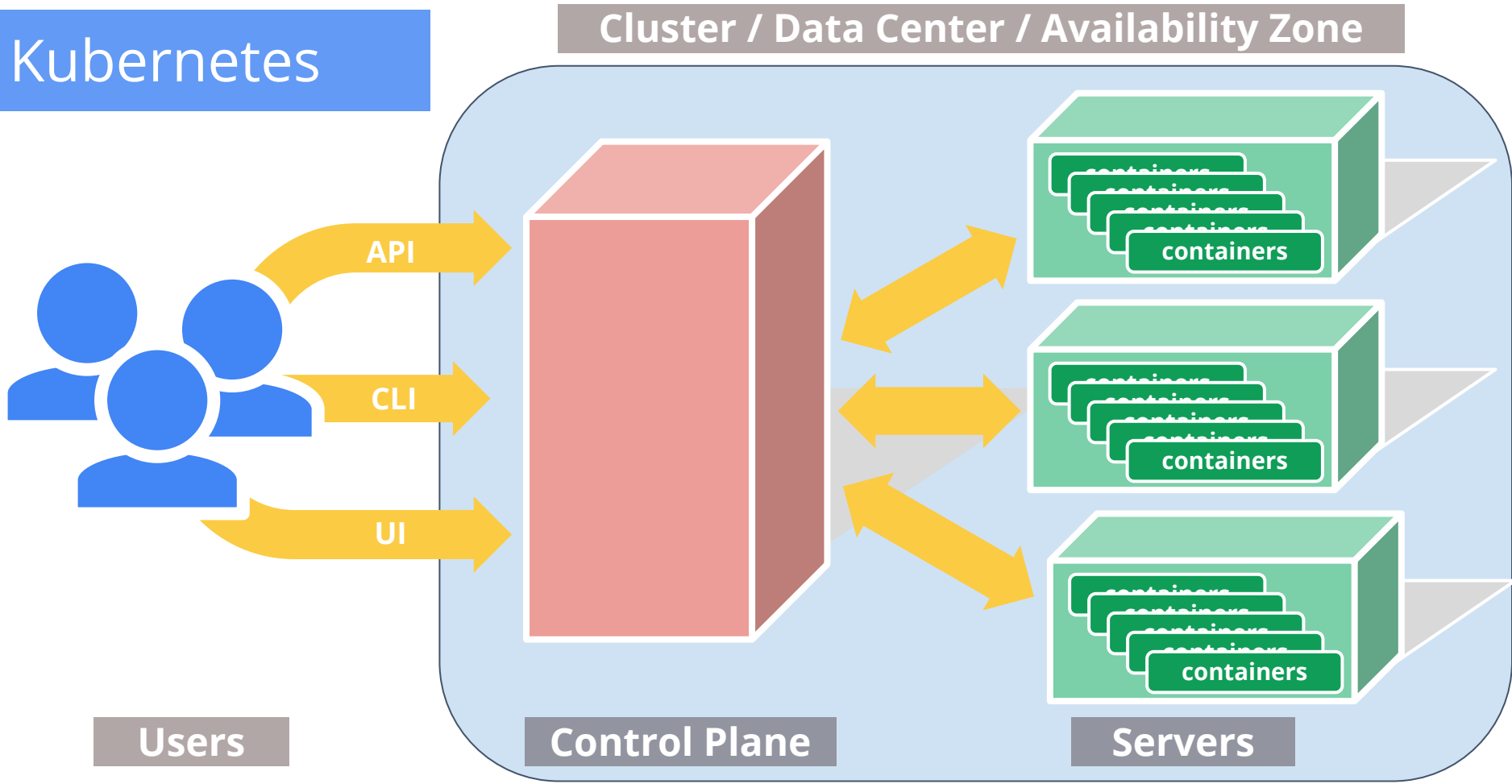
How it works

- Join clusters
- Create deployments, configmaps - spreads them across clusters and keeps them in sync
- Create services, ingress - configures them to route traffic across clusters
- 100% API compatibility with kubernetes

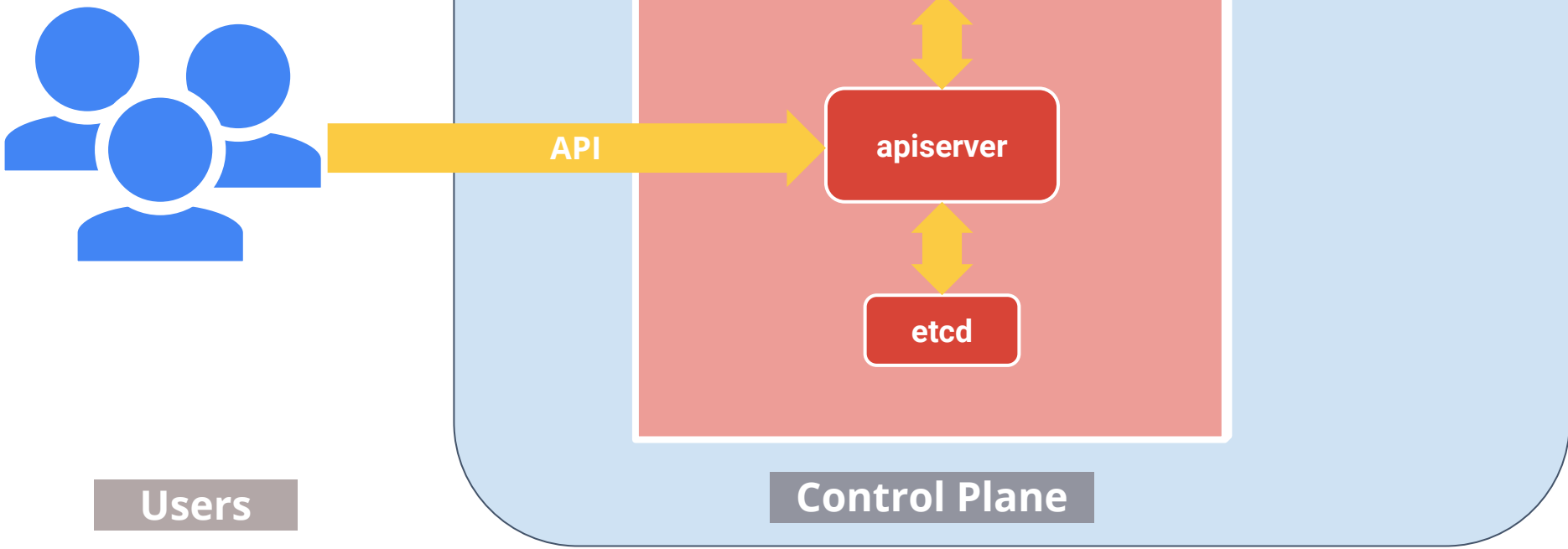
Architecture

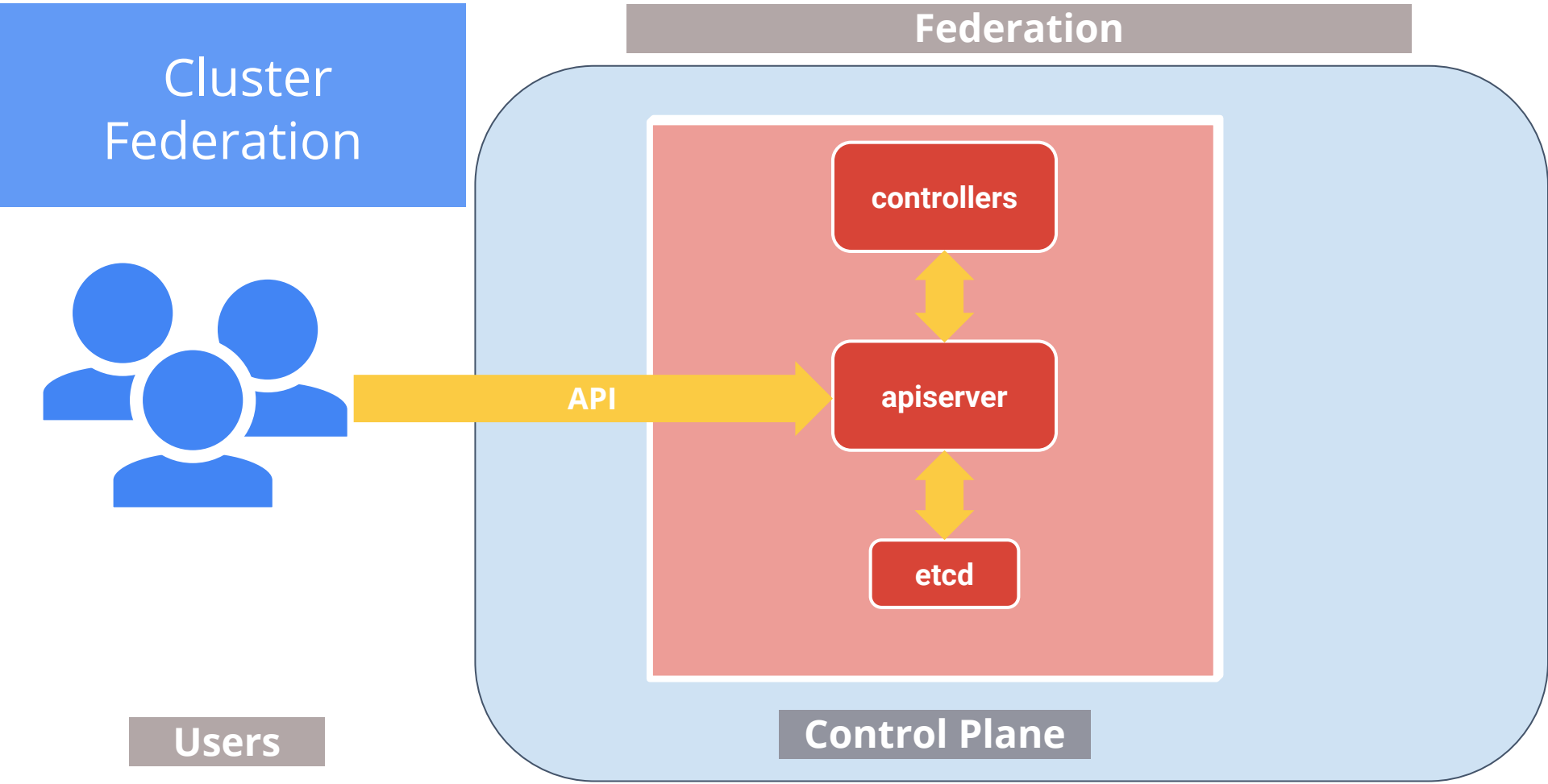

Cluster Federation
Federation
API
CLI
UI
K8s API
Kubernetes/GKE on Google Cloud Platform
Kubernetes on amazon web services
Kubernetes on Premise
IBM
DELL
Users
Control Plane
Clusters

Kubernetes


Cluster / Data Center / Availability Zone
API
CLI
UI
containers
containers
containers
Users
Control Plane
Servers

Kubernetes


Cluster
controllers
API
apiserver
etcd
Users
Control Plane


Cluster Federation
Federation
controllers
apiserver
etcd
API
Users
Control Plane

Federation Controllers

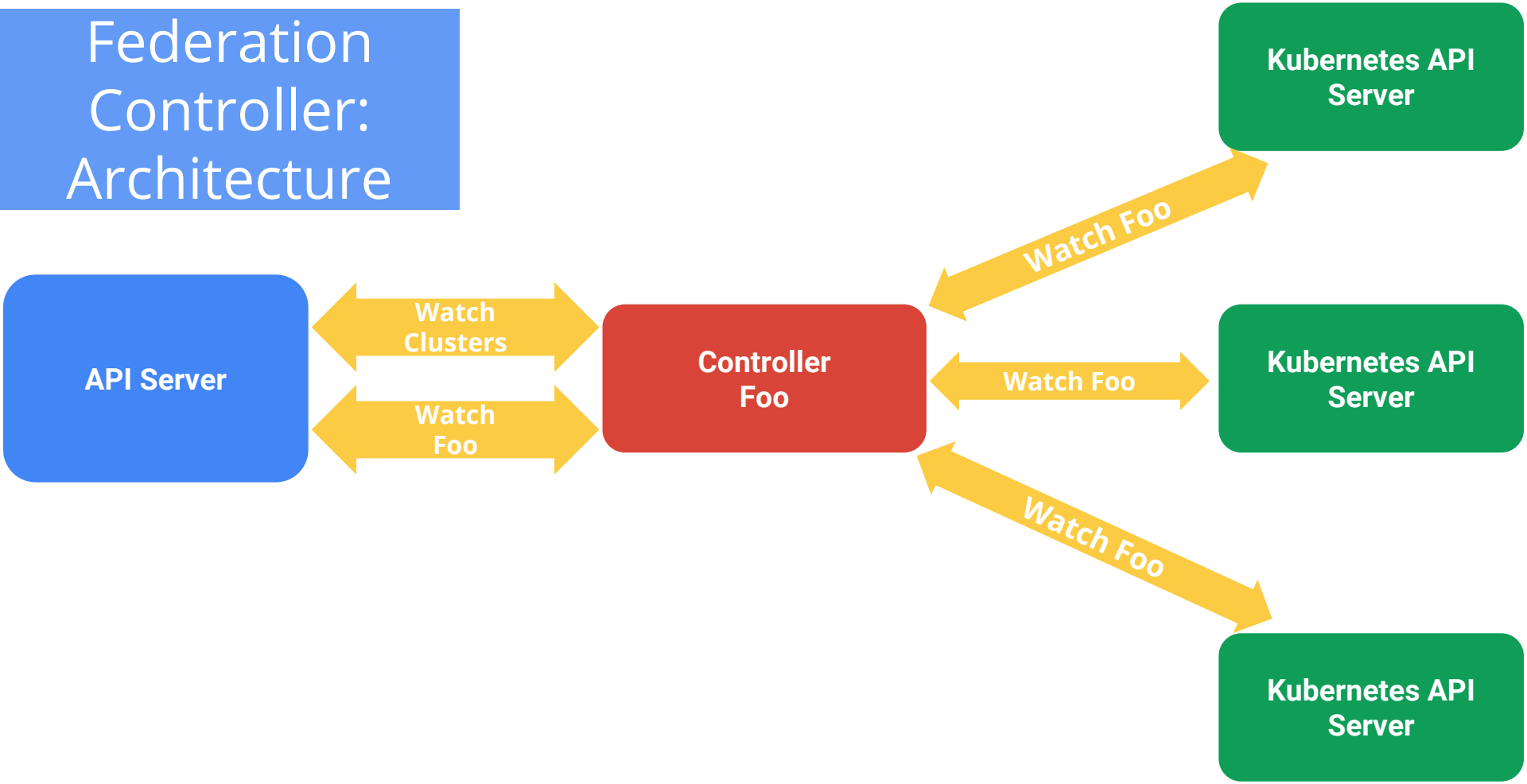
Why do you want to write your own controller?

- Augment existing functionality
  - Extend available controllers

Why do you want to write your own controller?

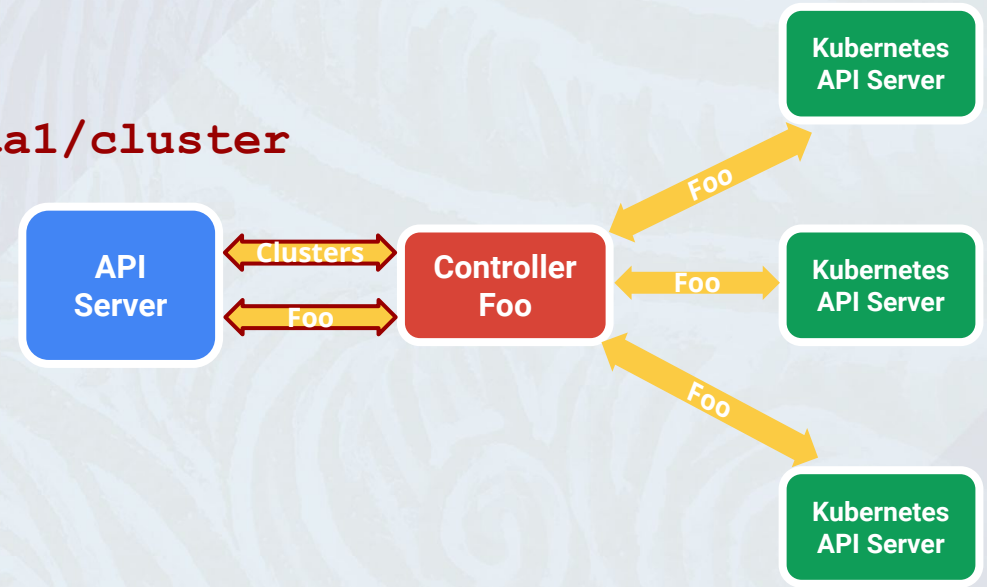
- Augment existing functionality
  - Extend available controllers
- Customize behavior
  - Replace existing controllers with custom implementations

Federation Controller: Architecture



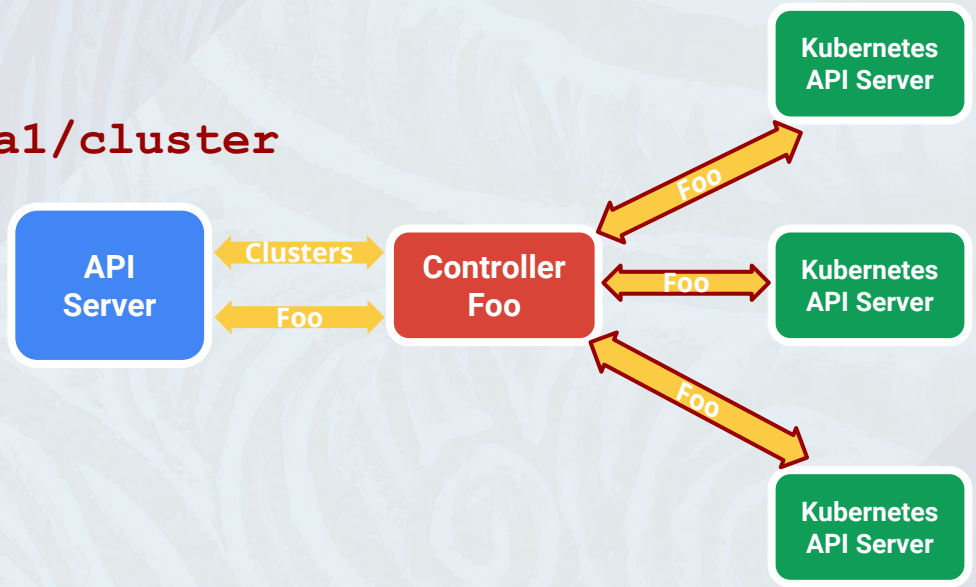
Federation Controller: Architecture

- Watches Federation API Server
  - Clusters - `federation/v1beta1/cluster`
  - API Resources - `v1/foo`



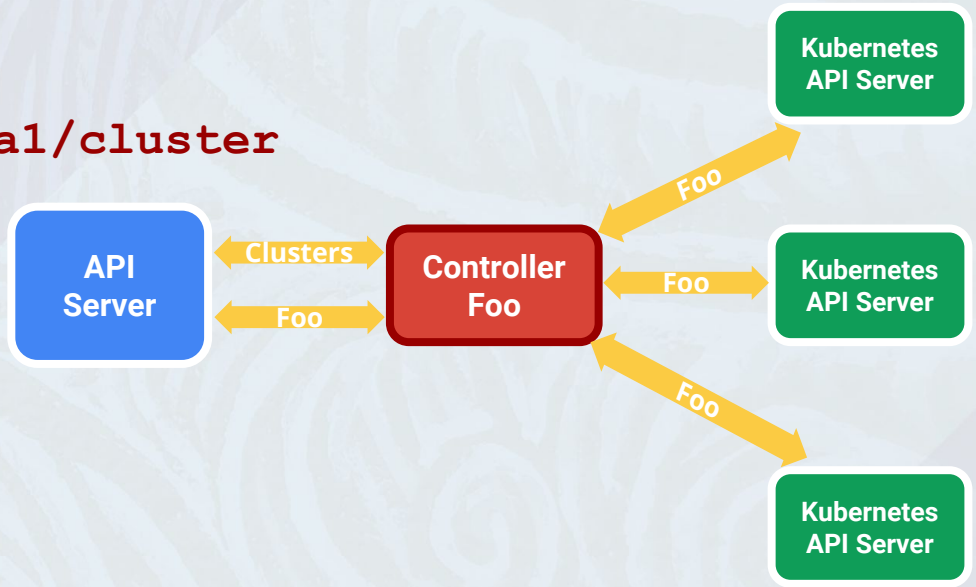
Federation Controller: Architecture

- Watches Federation API Server
  - Clusters - `federation/v1beta1/cluster`
  - API Resources - `v1/foo`
- Watches All Kubernetes Clusters
  - API Resources - `v1/foo`



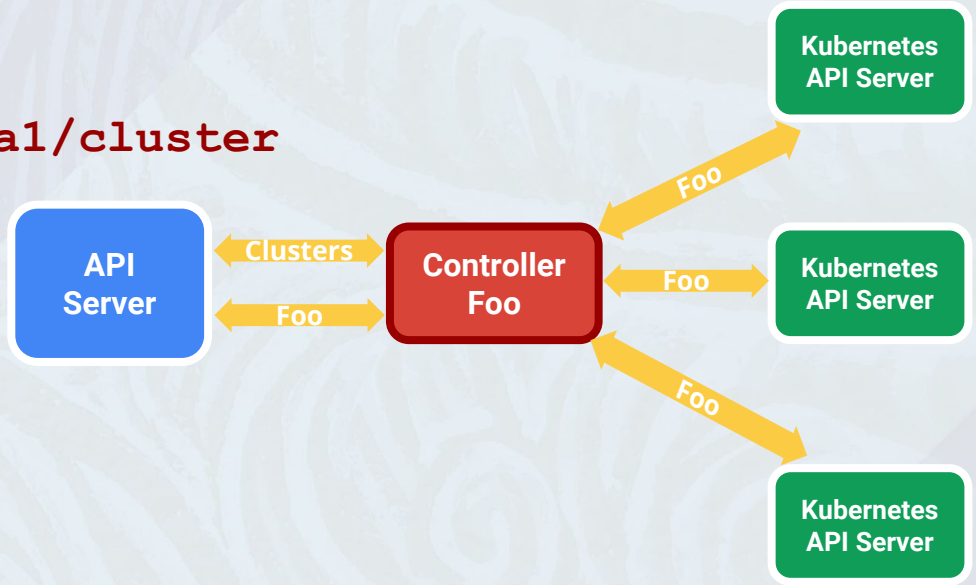
Federation Controller: Architecture

- Watches Federation API Server
  - Clusters - `federation/v1beta1/cluster`
  - API Resources - `v1/foo`
- Watches All Kubernetes Clusters
  - API Resources - `v1/foo`
- Reconciles
  - Compare and update



Federation Controller: Architecture

- Watches Federation API Server
  - Clusters - `federation/v1beta1/cluster`
  - API Resources - `v1/foo`
- Watches All Kubernetes Clusters
  - API Resources - `v1/foo`
- Reconciles
  - Compare and update
- Handles cascading deletion



Federation Controller: Implementation

- Informer to watch Federation API Resource - **v1/foo**

```
// Start informer on federated API servers on foos that should be federated.
foocontroller.fooInformerStore, foocontroller.fooInformerController = cache.NewInformer(
      &cache.ListWatch{
            ListFunc: func(options metav1.ListOptions) (pkgruntime.Object, error) {
                  return client.Core().Foos(metav1.NamespaceAll).List(options)
            },
            WatchFunc: func(options metav1.ListOptions) (watch.Interface, error) {
                  return client.Core().Foos(metav1.NamespaceAll).Watch(options)
            },
      },
      &apiv1.Foo{},
      controller.NoResyncPeriodFunc(),
      util.NewTriggerOnAllChanges(func(obj pkgruntime.Object) {
            foocontroller.deliverFooObj(obj, 0, false)
      }),
)
```

- Federated Informer to watch
  - Kubernetes API Resource in all clusters - **v1/foo**
  - Kubernetes clusters - **federation/v1beta1/cluster**

```
// Federated informer on foos in members of federation.
foocontroller.fooFederatedInformer = util.NewFederatedInformer(
        client,
        func(cluster *federationapi.Cluster, targetClient kubeclientset.Interface) (cache.Store, cache.Controller) {
                return cache.NewInformer(
                        &cache.ListWatch{
                                ...
                        },
                        &apiv1.Foo{},
                        controller.NoResyncPeriodFunc(),
                        // Trigger reconciliation whenever something in federated cluster is changed. In most cases it
                        // would be just confirmation that some foo operation succeeded.
                        util.NewTriggerOnAllChanges(
                                func(obj pkgruntime.Object) {
                                        foocontroller.deliverFooObj(obj, foocontroller.fooReviewDelay, false)
                                },
                        ))
        },
        &util.ClusterLifecycleHandlerFuncs{
                ClusterAvailable: func(cluster *federationapi.Cluster) {
                        // When new cluster becomes available process all the foos again.
                        foocontroller.clusterDeliverer.DeliverAt(allClustersKey, nil, time.Now().Add(foocontroller.clusterAvailableDelay))
                },
        },
)
```

- DeletionHelper to set up cascading deletion

```
foocontroller.deletionHelper = deletionhelper.NewDeletionHelper(
      foocontroller.hasFinalizerFunc,
      foocontroller.removeFinalizerFunc,
      foocontroller.addFinalizerFunc,
      // objNameFunc
      func(obj pkgruntime.Object) string {
            foo := obj.(*apiv1.Foo)
            return foo.Name
      },
      foocontroller.updateTimeout,
      foocontroller.eventRecorder,
      foocontroller.fooFederatedInformer,
      foocontroller.federatedUpdater,
)
```

- Connecting the dots...

```
func (foocontroller *FooController) Run(stopChan <-chan struct{}) {
	go foocontroller.fooInformerController.Run(stopChan)
	foocontroller.fooFederatedInformer.Start()
	go func() {
		<-stopChan
		foocontroller.fooFederatedInformer.Stop()
	}()
	foocontroller.fooDeliverer.StartWithHandler(func(item *util.DelayingDelivererItem) {
		foo := item.Value.(*types.NamespacedName)
		foocontroller.reconcileFoo(*foo)
	})
	foocontroller.clusterDeliverer.StartWithHandler(func(_ *util.DelayingDelivererItem) {
		foocontroller.reconcileFoosOnClusterChange()
	})
	util.StartBackoffGC(foocontroller.fooBackoff, stopChan)
}
```

- Reconcile on federation membership change

```
// The function triggers reconciliation of all federated foos.
func (foocontroller *FooController) reconcileFoosOnClusterChange() {
      if !foocontroller.isSynced() {
            glog.V(4).Infof("Foo controller not synced")
            foocontroller.clusterDeliverer.DeliverAt(allClustersKey, nil,
time.Now().Add(foocontroller.clusterAvailableDelay))
      }
      for _, obj := range foocontroller.fooInformerStore.List() {
            foo := obj.(*apiv1.Foo)
            foocontroller.deliverFoo(types.NamespacedName{Namespace: foo.Namespace, Name: foo.Name},
                  foocontroller.smallDelay, false)
      }
}
```

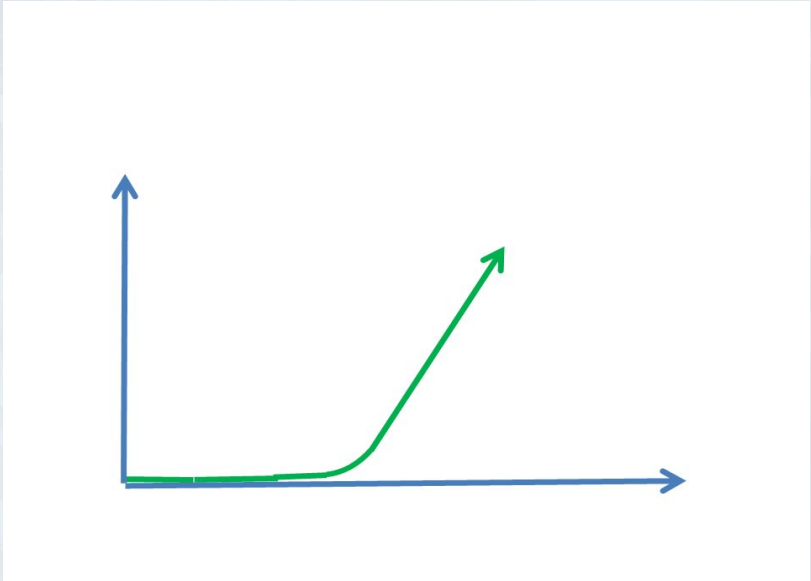
- Reconcile on any change

```
func (foocontroller *FooController) reconcileFoo(nsFoo types.NamespacedName) {
	...
	for _, cluster := range clusters {
		if !found {
			operations = append(operations, util.FederatedOperation{
				Type:        util.OperationTypeAdd,
				Obj:         desiredFoo,
				ClusterName: cluster.Name,
			})
		} else {
			// Update existing foo, if needed.
			if !util.FooEquivalent(desiredFoo, clusterFoo) {
				operations = append(operations, util.FederatedOperation{
					Type:        util.OperationTypeUpdate,
					Obj:         desiredFoo,
					ClusterName: cluster.Name,
				})
			}
		}
	}
	...
	err = foocontroller.federatedUpdater.UpdateWithOnError(operations, foocontroller.updateTimeout,
		func(op util.FederatedOperation, operror error) {...})
	...
```

Demo

We're moving fast...

- **Q3 2016 - Some killer features:**
  - Federated **Ingress**,
  - **ReplicaSets**,
  - **Namespaces**,
  - **Secrets**,
  - **Events**
- **Q4 2016 - Easier installation, expanded API:**
  - Federated **Deployments**,
  - **Daemonsets**,
  - **ConfigMaps**
  - `kubefed` **"alpha"**
- **Q1 2017 - Stabilization and paying debt:**
  - `kubefed` **"beta"**



The Future... 2017 and beyond

- **Policy-based Resource Placement (and...)**
  - Federation overlays policy over application requirements
- **Improved Identity and Access Management**
  - Especially integration with external IAM providers, across multiple cloud providers
- **Stateful Apps and Federated Persistent Storage**
  - Cross-cluster data replication, snapshot+restore, etc...
- **Hybrid Cloud Federated Ingress**
  - Smart cross-cloud L7 load balancing
- **Private Federated Services**
  - Private IP's and DNS
- **GUI/Visualization**, etc, etc...

This is a community effort! Feel free to join.

- **SIG Federation**
  - kubernetes-sig-federation@groups.google.com
  - groups.google.com/kubernetes-sig-federation
- **Working Group**
  - tinyurl.com/ubernetes-wg-notes
- **Us:**
  - Nikhil
    - nikhiljindal@google.com
    - nikhiljindal@github
  - Madhu
    - madhusudancs@google.com
    - madhusudancs@github

HUAWEI

CoreOS

redhat.

Google

ebay

Thank you!