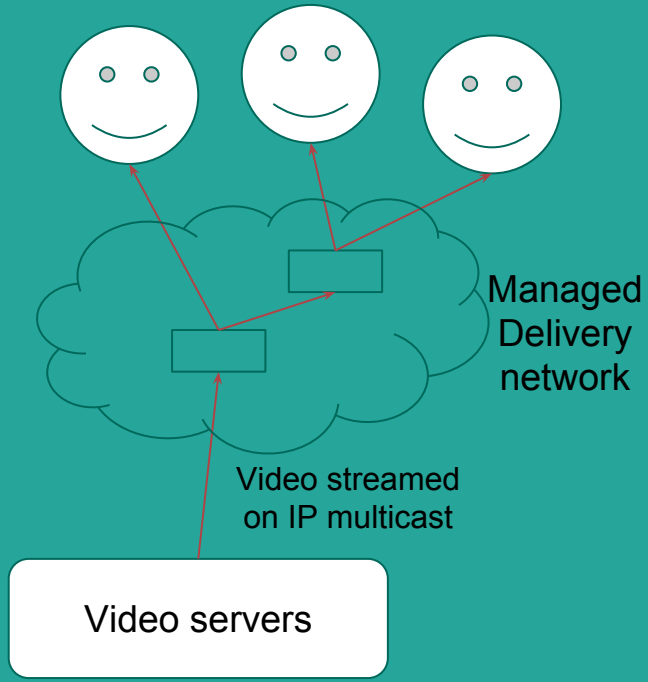
Writing a Kubernetes Operator to deploy a complex system

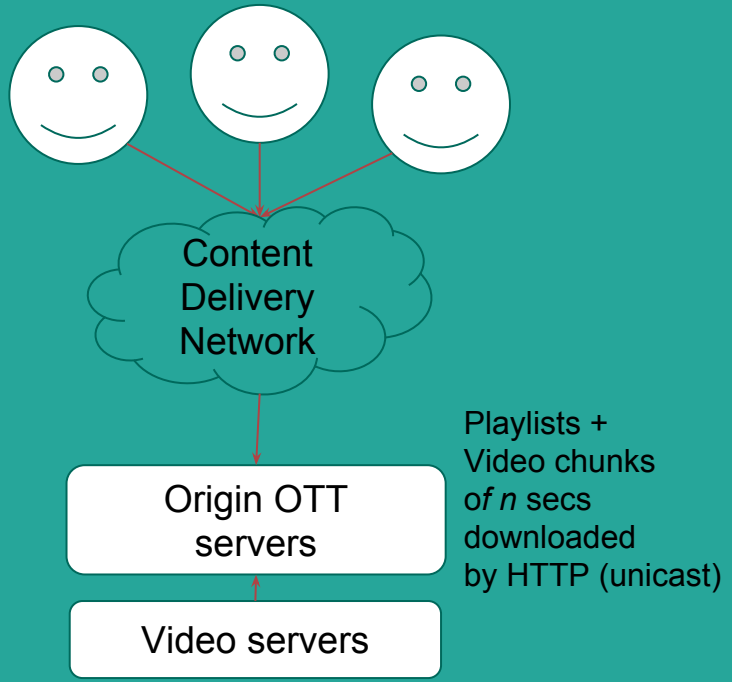
—

O'Reilly Velocity Conference
London 2018

Philippe Martin
Anevia

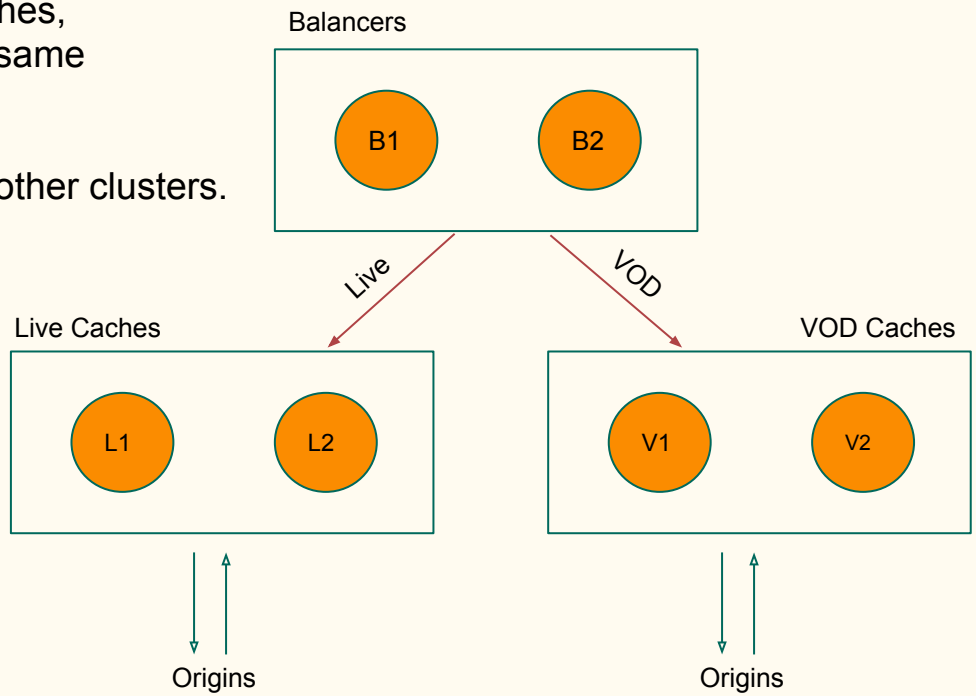
IPTV vs OTT


Managed
Delivery
network
Video streamed
on IP multicast
Video servers


Content
Delivery
Network
Playlists +
Video chunks
of *n* secs
downloaded
by HTTP (unicast)
Origin OTT
servers
Video servers

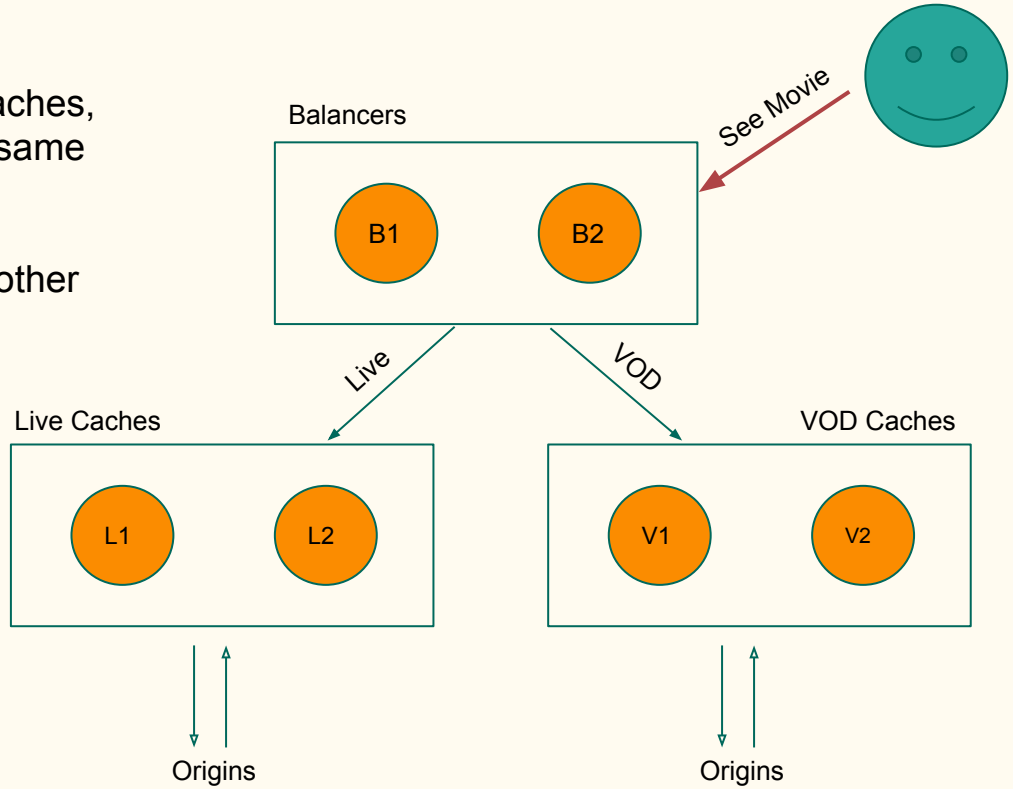
A use-case at Anevia

- Deploy clusters of Balancers or Caches, all instances of a cluster having the same configuration,
- Define some clusters as sources of other clusters.



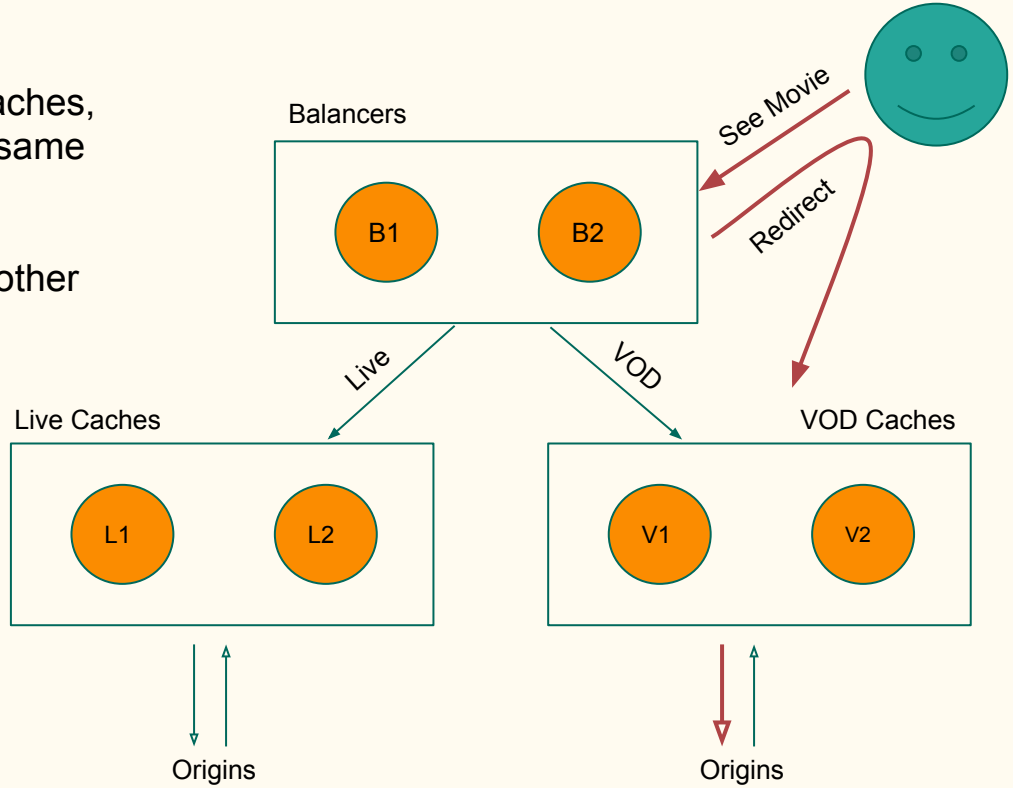
A use-case at Anevia

- Deploy clusters of Balancers and Caches, all instances of a cluster having the same configuration,
- Define some clusters as sources of other clusters.



A use-case at Anevia

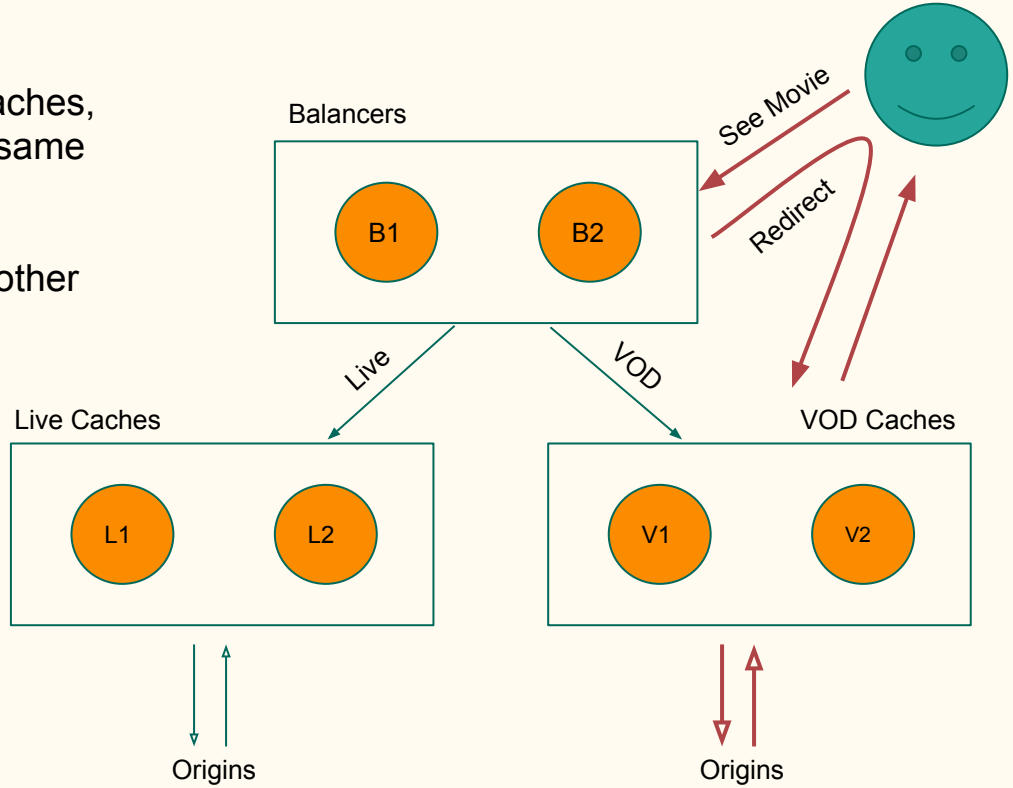
- Deploy clusters of Balancers and Caches, all instances of a cluster having the same configuration,
- Define some clusters as sources of other clusters.



A use-case at Anevia

- Deploy clusters of Balancers and Caches,
- all instances of a cluster having the same configuration,

- Define some clusters as sources of other clusters.



A solution based on a Kubernetes Operator

- **a Custom Resource**
  - Representing a Cluster of Balancers or Caches
  - Defining the list of Sources of this cluster
- **an Operator**
  - Reading resources
  - Deploying containers into a Kubernetes cluster
  - Configuring the containers depending on the sources

“Presenting complexity in familiar ways
- *Kubernetes CRDs”*

Kris Nova, this morning

using sample-controller

https://github.com/kubernetes/sample-controller

Using sample-controller

- **Benefits**
  - Easy way to begin with a functional and simple CRD and an operator

- **Limits**
  - How to extend?
  - How to write tests?
  - How to deploy, especially which RBAC auth to apply?

using kubebuilder

kubebuilder init --domain anevia.com

The resource

Creating a new `CdnCluster` Resource

Customizing the `CdnCluster` Resource

Creating `CdnCluster`s instances from client

Creating a new `CdnCluster` Resource

The Resource

Creating a new resource

Group	Version	Kind
core	v1	Pod
apps	v1	Deployment
cluster	v1	CdnCluster

```
$ kubebuilder create api \
  --group cluster \
  --version v1 \
  --kind CdnCluster
```

The Resource

Examining the created files

Generated sources in config/

```
$ make manifests
```

CustomResourceDefinition

- crds/cluster_v1_cdncluster.yaml

Custom Resource Definition

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
 creationTimestamp: null
 labels:
   controller-tools.k8s.io: "1.0"
 name: cdnclusters.cluster.anevia.com
spec:
 group: cluster.anevia.com
 names:
   kind: CdnCluster
   plural: cdnclusters
 scope: Namespaced
```

```yaml
spec:
 validation:
   openAPIV3Schema:
     properties:
       apiVersion:
         type: string
       kind:
         type: string
       metadata:
         type: object
       spec:
         type: object
       status:
         type: object
 version: v1
```

```yaml
status:
 acceptedNames:
   kind: ""
   plural: ""
 conditions: []
 storedVersions: []
```

The Resource

Examining the created files

Generated sources in config/

```
$ make manifests
```

CustomResourceDefinition

- crds/cluster_v1_cdncluster.yaml

ClusterRole & ClusterRoleBinding

- rbac/rbac_role.yaml
- rbac/rbac_role_binding.yaml

Cluster Role

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 creationTimestamp: null
 name: manager-role
rules:
```

The sample operator can work on Deployments and CdnClusters

```yaml
rules:
- apiGroups:
 - apps
 resources:
 - deployments
 verbs:
 - get
 - list
 - watch
 - create
 - update
 - patch
 - delete
```

```yaml
rules:
- apiGroups:
 - cluster.anevia.com
 resources:
 - cdnclusters
 verbs:
 - get
 - list
 - watch
 - create
 - update
 - patch
 - delete
```

The Resource

Examining the created files

Generated sources in config/

```
$ make manifests
```

CustomResourceDefinition

- crds/cluster_v1_cdncluster.yaml

ClusterRole & ClusterRoleBinding

- rbac/rbac_role.yaml
- rbac/rbac_role_binding.yaml

CdnCluster

- samples/cluster_v1_cdncluster.yaml

Sample CdnCluster

```yaml
apiVersion: cluster.anevia.com/v1
kind: CdnCluster
metadata:
 labels:
   controller-tools.k8s.io: "1.0"
 name: cdncluster-sample
spec:
 # Add fields here
 foo: bar
```

The Resource

Examining the created files

Go sources in pkg/apis/
of the Custom Resource Definition

```
├ addtoscheme_cluster_v1.go
├ apis.go
└ cluster
 ├ group.go
 └ v1
   ├ cdncluster_types.go # definition
   ├ cdncluster_types_test.go # tests
   ├ doc.go
   ├ register.go
   ├ v1_suite_test.go
   └ zz_generated.deepcopy.go
```

The Resource

The sample
Custom Resource Definition
sources

```
// CdnClusterSpec defines the desired state
type CdnClusterSpec struct {
}

// CdnClusterStatus defines the observed state
type CdnClusterStatus struct {
}

// CdnCluster is the Schema for the cdnclusters
type CdnCluster struct {
  metav1.TypeMeta   `json:",inline"`
  metav1.ObjectMeta `json:"metadata,omitempty"`

  Spec   CdnClusterSpec   `json:"spec,omitempty"`
  Status CdnClusterStatus `json:"status,omitempty"`
}
```

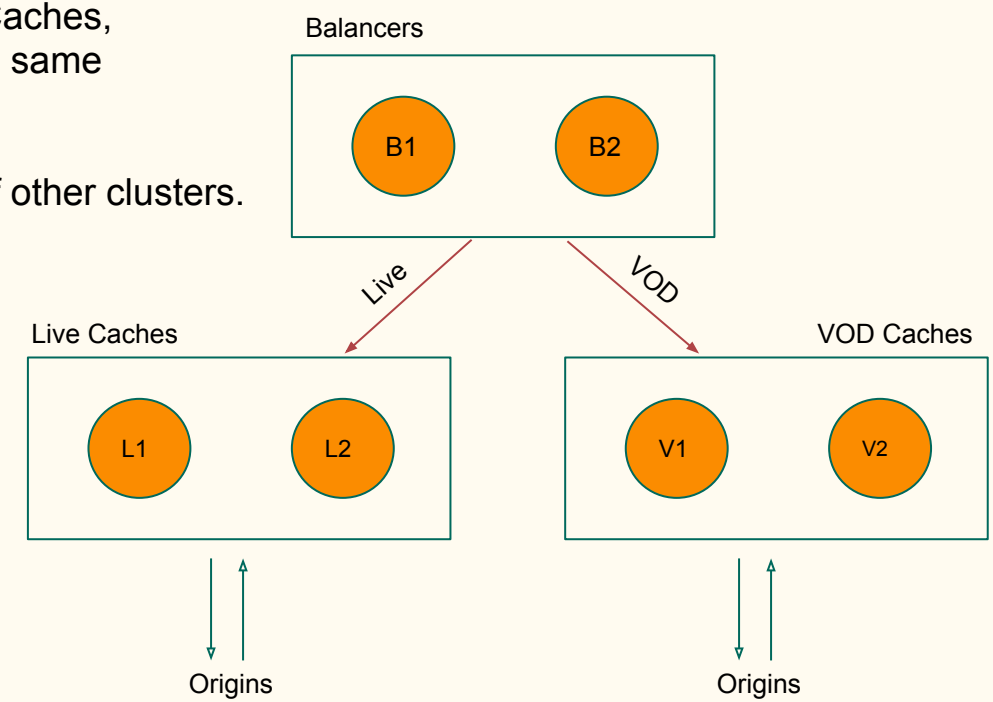
How are RBAC authorizations generated?

```
// In pkg/controller/cdncluster/cdncluster_controller.go

// Automatically generate RBAC rules to allow the Controller
// to read and write Deployments
// +kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;...

// Automatically generate RBAC rules to allow the Controller
// to read and write CdnClusters
// +kubebuilder:rbac:groups=cluster.anevia.com,resources=cdnclusters,verbs=get;list;..
```

Customizing the `CdnCluster` Resource

A use-case at Anevia

- Deploy clusters of Balancers and Caches, all instances of a cluster having the same configuration,
- Define some clusters as sources of other clusters.



The Resource

Customizing the
Custom Resource Definition
from sources

```go
// CdnClusterSpec defines the desired state
type CdnClusterSpec struct {
  // Role must be 'balancer' or 'cache'
  Role string `json:"role"`
  // Sources is the list of source clusters
  Sources []CdnClusterSource `json:"sources"`
}

// CdnClusterSource defines a source cluster
type CdnClusterSource struct {
  // The name of the source cluster
  Name string `json:"name"`
  // The path condition to enter this cluster,
  // can be omitted for the default source
  PathCondition string `json:"pathCondition,omitempty"`
}

// CdnClusterStatus defines the observed state
type CdnClusterStatus struct {
  // State of the CDN cluster
  State string `json:"state"`
}
```

```
$ make manifests
```

Creating CdnCluster instances

```yaml
apiVersion: cluster.anevia.com/v1
kind: CdnCluster
metadata:
 name: balancer
spec:
 role: balancer
 sources:
 - name: cache-live
   pathCondition: ^/live/
 - name: cache-vod
   pathCondition: ^/vod/

---
```

```yaml
apiVersion: cluster.anevia.com/v1
kind: CdnCluster
metadata:
 name: cache-live
spec:
 role: cache
 sources: [] # add origins here
---
apiVersion: cluster.anevia.com/v1
kind: CdnCluster
metadata:
 name: cache-vod
spec:
 role: cache
 sources: [] # add origins here
```

Deploying the `CdnCluster` Resource

The Resource

Deploying the sample
Custom Resource Definition
to augment the k8s API

```
$ kubectl get crds
No resources found.

$ make install(*)
[...]

$ kubectl get crds
NAME                                  AGE
cdnclusters.cluster.anevia.com        3s

(*) or
$ make manifests
$ kubectl apply -f config/crds
```

Creating `CdnCluster`s instances from clients

The Resource

Creating the sample CdnCluster instance with kubectl

```
$ kubectl get cdncluster
No resources found.

$ kubectl apply -f \
config/samples/cluster_v1_cdncluster.yaml
[...]

$ kubectl get cdncluster
NAME                AGE
cdncluster-sample   3s

$ kubectl delete \
cdncluster cdncluster-sample
```

The Resource

Generating CdnClusters with the controller-runtime client

```go
package main
import (
    "context"
    "fmt"
    "github.com/feloy/operator/pkg/apis"
    cdnclusterv1
      "github.com/feloy/operator/pkg/apis/cluster/v1"
    corev1 "k8s.io/api/core/v1"
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/client-go/kubernetes/scheme"
    "sigs.k8s.io/controller-runtime/pkg/client"
    "sigs.k8s.io/controller-runtime/pkg/client/config"
)
```

Using the controller-runtime client

```go
func main() {
  // Create the client
  config, _ := config.GetConfig()
  cl, err := client.New(config, client.Options{})
  if err != nil {
    fmt.Printf("%s\n", err.Error())
  }
  // Create a native ConfigMap
  cmap := &corev1.ConfigMap{
    ObjectMeta: metav1.ObjectMeta{
      Name: "cmap", Namespace: "default"
    },
  }
  err = cl.Create(context.Background(), cmap)
  if err != nil {
    fmt.Printf("%s\n", err.Error())
  }
  // Add Cluster scheme to global scheme
  apis.AddToScheme(scheme.Scheme)

  // Create a Custom Resource CdnCluster
  cdn := &cdnclusterv1.CdnCluster{
    ObjectMeta: metav1.ObjectMeta{
      Name: "cdn", Namespace: "default"},
    Spec: cdnclusterv1.CdnClusterSpec{
      Role:    "balancer",
      Sources: []cdnclusterv1.CdnClusterSource{},
    },
  }
  err = cl.Create(context.Background(), cdn)
  if err != nil {
    fmt.Printf("%s\n", err.Error())
  }
}
```

A solution based on a Kubernetes Operator

- **a Custom Resource**
  - Representing a Cluster of Balancers or Caches
  - Defining the list of Sources of this cluster
- **an Operator**
  - Reading resources
  - Deploying containers into a Kubernetes cluster
  - Configuring the containers depending on the sources

The Operator

The Reconcile function

When to call Reconcile

Customizing Reconcile

Setting Status, sending Events

The Reconcile function

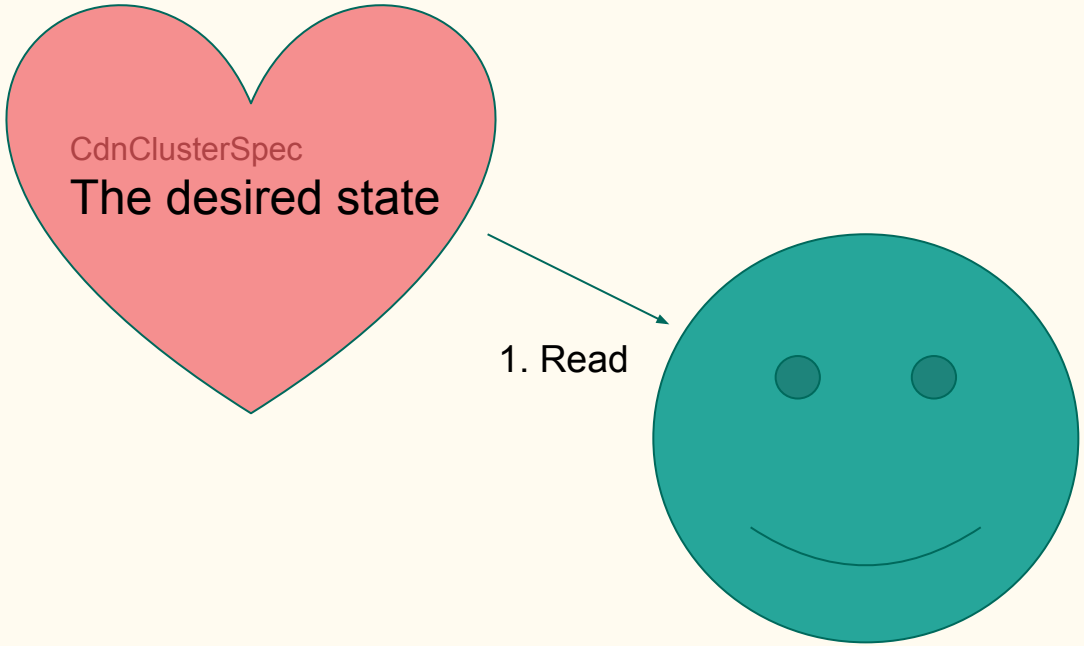
The Operator

The Reconcile function

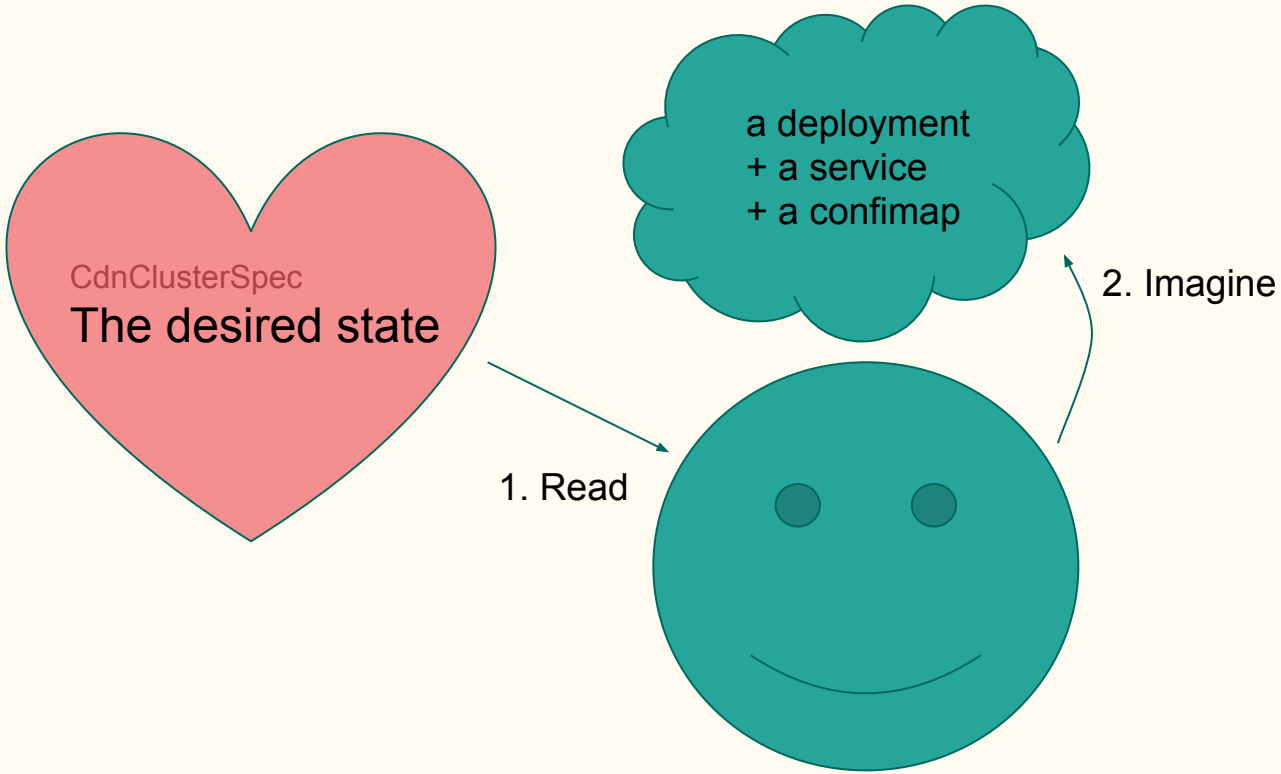
```
// cdncluster_controller.go

type ReconcileCdnCluster struct {
    client.Client
    scheme   *runtime.Scheme
}

func (r *ReconcileCdnCluster)
Reconcile(request reconcile.Request)
(reconcile.Result, error) {

}
```

The Reconcile function

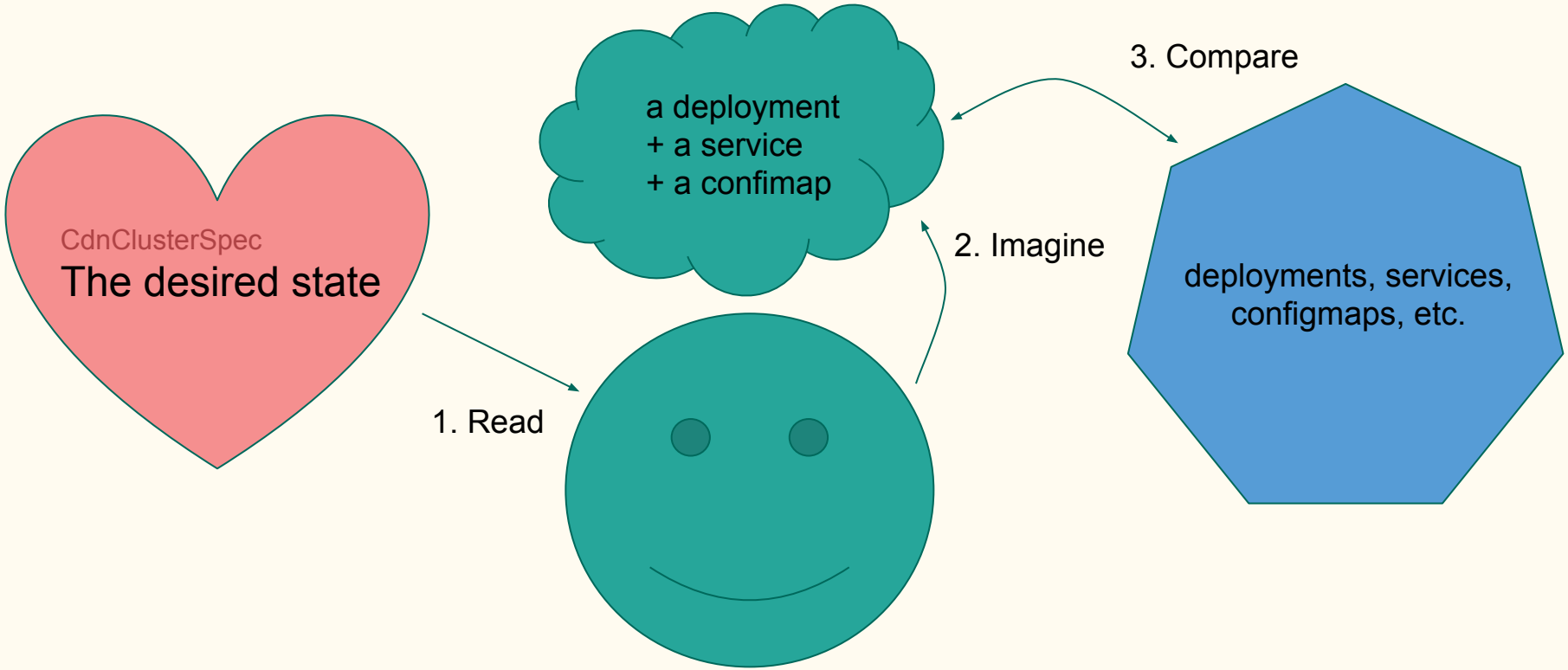
The Reconcile function



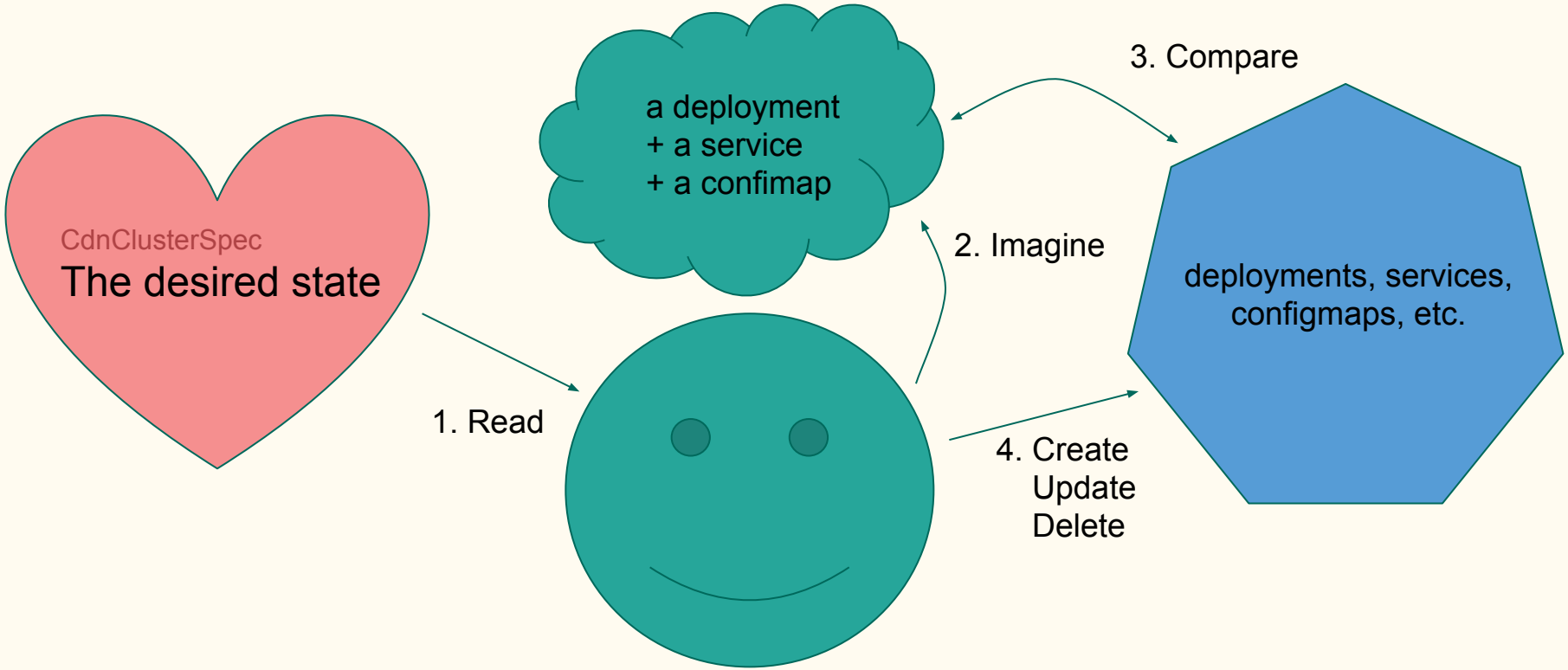
The Reconcile function



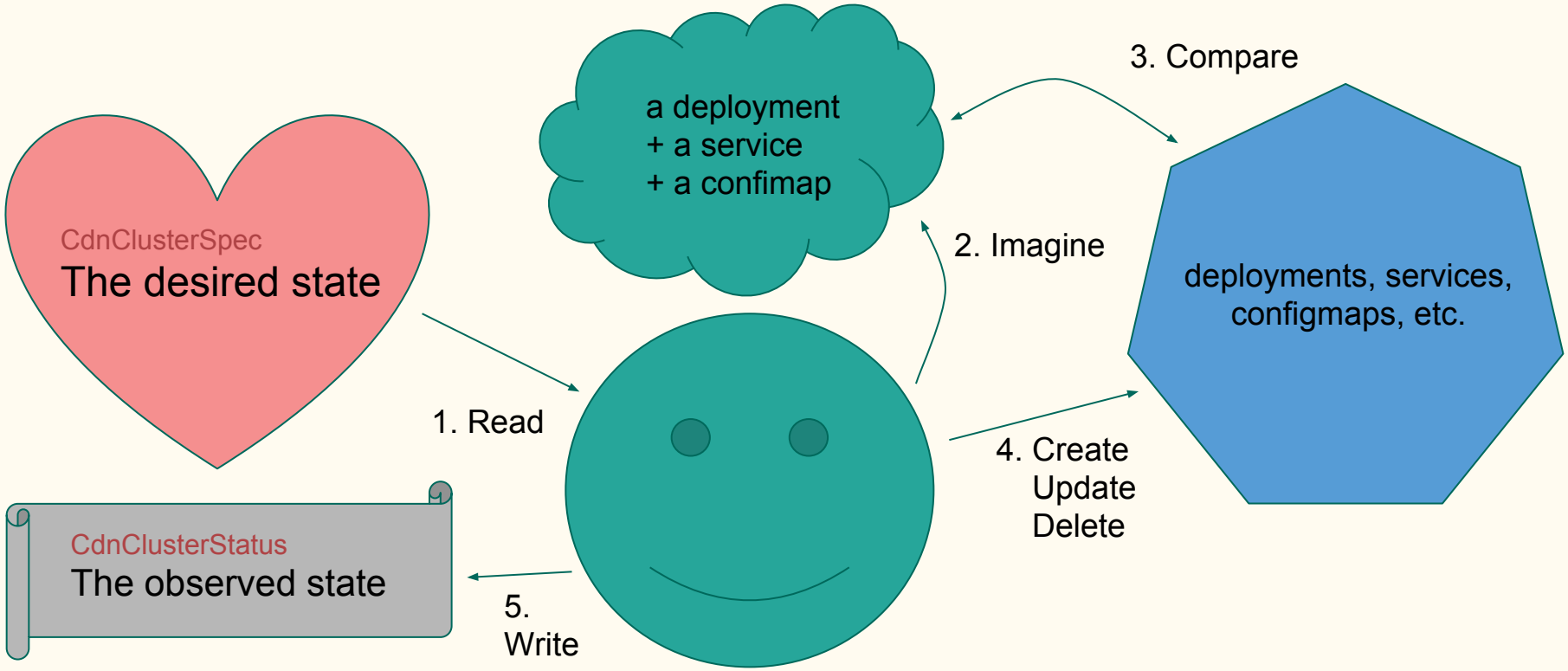
The Reconcile function



The Reconcile function



The Reconcile function



```
// 1. Get the CdnCluster instance in the `cdncluster` structure

cdncluster := &clusterv1.CdnCluster{}
err := r.Get(context.TODO(), request.NamespacedName, cdncluster)

if err != nil {
  if errors.IsNotFound(err) {
    // Object not found, return.  Created objects are automatically garbage collected.
    // For additional cleanup logic use finalizers.
    return reconcile.Result{}, nil
  }
  // Error reading the object - requeue the request.
  return reconcile.Result{}, err
}
```

```
// 2. Define the desired Deployment object in memory

deploy := &appsv1.Deployment{

  ObjectMeta: metav1.ObjectMeta{
    Name:      cdncluster.Name + "-deployment",
    Namespace: cdncluster.Namespace,
  },

  Spec: appsv1.DeploymentSpec{
    Selector: &metav1.LabelSelector{
      MatchLabels: map[string]string{"deployment": cdncluster.Name + "-deployment"},
    },
    Template: corev1.PodTemplateSpec{

      ObjectMeta: metav1.ObjectMeta{Labels: map[string]string{"deployment": cdncluster.Name +
"-deployment"}},

      Spec: corev1.PodSpec{Containers: []corev1.Container{{Name:  "nginx", Image: "nginx"}}}
    }
  },
}
```

```
// 2. Define the CdnCluster resource as the owner of
// the created deployment,
// so the garbage collector can handle the deletion of this object
// when the CdnCluster is deleted

err := controllerutil.SetControllerReference(cdncluster, deploy, r.scheme)

if err != nil {
  return reconcile.Result{}, err
}
```

```
// 3. Check if the Deployment already exists in the real world

found := &appsv1.Deployment{}
err = r.Get(context.TODO(), types.NamespacedName{
    Name: deploy.Name,
    Namespace: deploy.Namespace
  }, found)

if err != nil && errors.IsNotFound(err) {
  log.Printf("Creating Deployment %s/%s\n", deploy.Namespace, deploy.Name)
  err = r.Create(context.TODO(), deploy)
  if err != nil {
    return reconcile.Result{}, err
  }
} else if err != nil {
  return reconcile.Result{}, err
}
```

```
// 4. If there are any changes,
// update the found object and write the result back

if !reflect.DeepEqual(deploy.Spec, found.Spec) {
  found.Spec = deploy.Spec
  log.Printf("Updating Deployment %s/%s\n", deploy.Namespace, deploy.Name)
  err = r.Update(context.TODO(), found)
  if err != nil {
    return reconcile.Result{}, err
  }
}
```

When to call Reconcile

The Operator

On which events to call the Reconcile function

```
// Watch for changes to any
// CdnCluster resource

err = c.Watch(

  &source.Kind{
   Type: &clusterv1.CdnCluster{},
  },

  &handler.EnqueueRequestForObject{}

)
```

The Operator

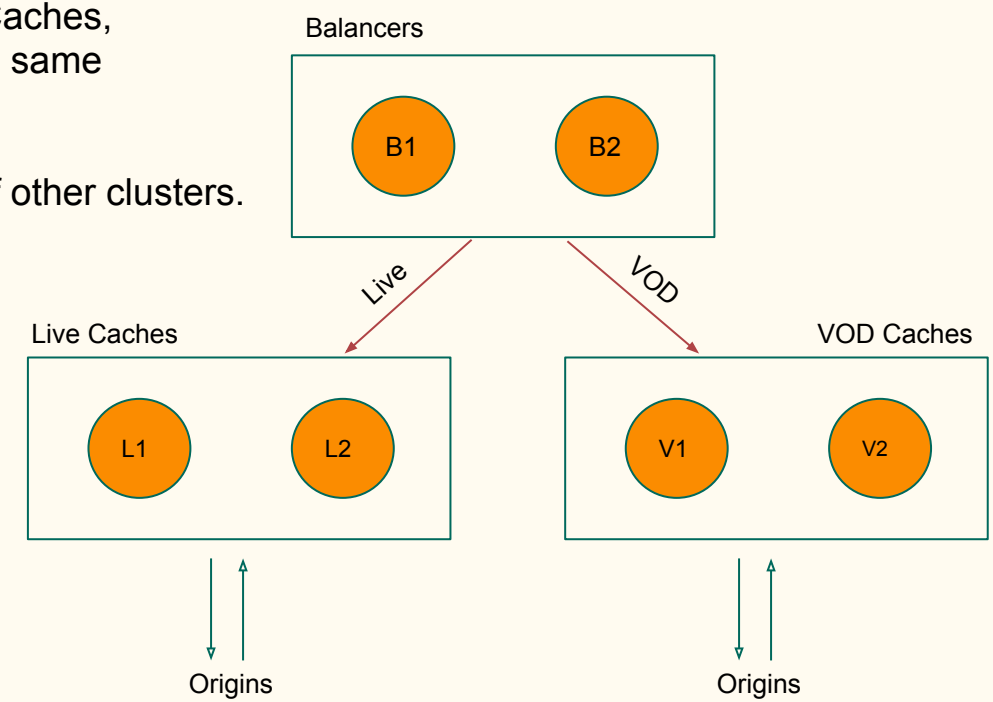
On which events to call the Reconcile function

```
// Watch for changes to any
// Deployment created by CdnCluster

err = c.Watch(

  &source.Kind{
    Type: &appsv1.Deployment{},
  },

  &handler.EnqueueRequestForOwner{
    IsController: true,
    OwnerType:    &clusterv1.CdnCluster{},
  }

)
```

Customizing the Reconcile function

A use-case at Anevia

- Deploy clusters of Balancers and Caches, all instances of a cluster having the same configuration,
- Define some clusters as sources of other clusters.



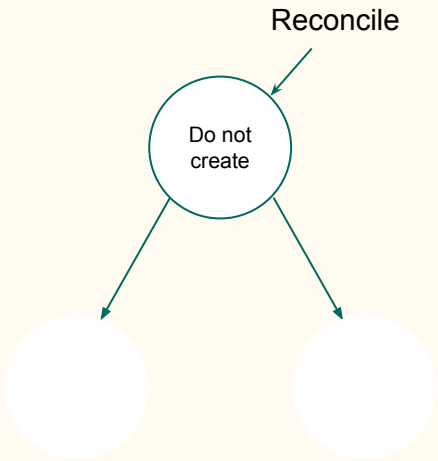
Customizing the Reconcile function

During initial deployment, sources must be created before the parent Cdn cluster can be deployed

- Store the dependencies between CdnClusters

- If some source does not exist, return from the Reconcile function

- When a Cdn Cluster is modified, call the Reconcile function for the parent CdnClusters



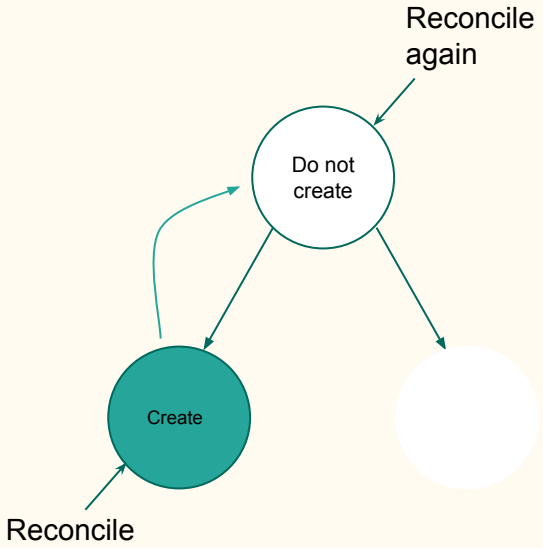
Customizing the Reconcile function

During initial deployment, sources must be created before the parent Cdn cluster can be deployed

- Store the dependencies between CdnClusters

- If some source does not exist, return from the Reconcile function

- When a Cdn Cluster is modified, call the Reconcile function for the parent CdnClusters



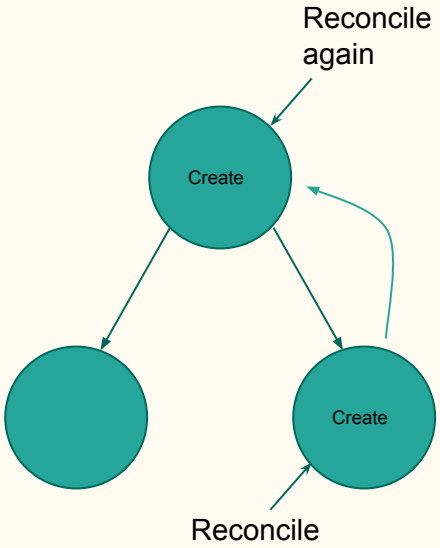
Customizing the Reconcile function

During initial deployment, sources must be created before the parent Cdn cluster can be deployed

- Store the dependencies between CdnClusters

- If some source does not exist, return from the Reconcile function

- When a Cdn Cluster is modified, call the Reconcile function for the parent CdnClusters



Customizing the Reconcile function

```
// Storing dependencies between CdnClusters

type ParentsList map[string][]string

func (o ParentsList) Add(source, parent string) {
  for _, p := range o[source] {
    if p == parent {
      return
    }
  }
  o[source] = append(o[source], parent)
}

var (
  // Parents is a map of parent CDN clusters for a CDN cluster
  Parents = ParentsList{}
)
```

Customizing the Reconcile function

```
// In Reconcile function

for _, source := range instance.Spec.Sources {
  Parents.Add(source.Name, instance.Name)
}

// Verify that all sources exist
// We do not continue until all sources exist
for _, source := range instance.Spec.Sources {
  sourceInstance := &clusterv1.CdnCluster{}
  err := r.Get(context.TODO(), types.NamespacedName{Name: source.Name, Namespace:
instance.Namespace}, sourceInstance)
  if err != nil {
    if errors.IsNotFound(err) {
      // Source not found, return.
      return reconcile.Result{}, nil
    }
    // Error reading the object - requeue the request.
    return reconcile.Result{}, err
  }
}
```

Customizing the Reconcile function

```
err = c.Watch(

  &source.Kind{Type: &clusterv1.CdnCluster{}},

  &handler.EnqueueRequestsFromMapFunc{
  ToRequests: handler.ToRequestsFunc(
    // Returns the list of parents of the CdnCluster in mapObject
    func(mapObject handler.MapObject) []reconcile.Request {
      v, ok := mapObject.Object.(*clusterv1.CdnCluster)
      if ok {
        var res = []reconcile.Request{}
        for _, parent := range Parents[v.Name] {
          res = append(res, reconcile.Request{
            NamespacedName: types.NamespacedName{Name: parent, Namespace: v.Namespace},
          })
        }
        return res
      }
      return nil
    }),
})
```

Setting Status

The Operator

Setting Status

```
// CdnClusterStatus defines the
observed state of CdnCluster
type CdnClusterStatus struct {

  // State of the CDN cluster
  State string `json:"state,omitempty"`

}
```

Setting Status

```
// setState changes the State of CDN cluster, if necessary
func (r *ReconcileCdnCluster) setState(cdncluster
*clusterv1.CdnCluster, newState string) error {
  if cdncluster.Status.State != newState {
    cdncluster.Status.State = newState
    return r.Update(context.TODO(), cdncluster)
  }
  return nil
}

[...]
err = r.setState(instance, "WaitingSource")
```

Sending Events

The Operator

Sending Events

```
$ kubectl describe cdncluster balancer

Name:         balancer
Namespace:    default
[...]
Events:
  Type    Reason          Age   From        Message
  ----    ------          ----  ----        -------
  Normal  SourceNotFound  3s    CdnCluster  Source
cache-live not found, will retry later
```

Sending Events

1. Add an EventRecorder to the ReconcileCluster structure

```
// ReconcileCdnCluster reconciles a CdnCluster object
type ReconcileCdnCluster struct {
  client.Client
  scheme   *runtime.Scheme
  recorder record.EventRecorder
}
```

Sending Events

2. Instantiates the recorder with manager.GetRecorder

```
func Add(mgr manager.Manager) error {
 return add(mgr, newReconciler(mgr, mgr.GetRecorder("CdnCluster")))
}

func newReconciler(mgr manager.Manager, recorder record.EventRecorder)
reconcile.Reconciler {
  return &ReconcileCdnCluster{
    Client:   mgr.GetClient(),
    scheme:   mgr.GetScheme(),
    recorder: recorder,
  }
}
```

Sending Events

3. Use Event/Eventf to send events

```
if errors.IsNotFound(err) {
  // Source not found, inform with an event and return.
  r.recorder.Eventf(instance, "Normal", "SourceNotFound",
        "Source %s not found, will retry later", source.Name)
  return reconcile.Result{}, nil
}
```

Testing

Testing creation of CdnClusters

```
$ git checkout 285d502

// cdncluster_types_test.go
created := &CdnCluster{
  ObjectMeta: metav1.ObjectMeta{
    Name: "foo", Namespace: "default"
  },
}


$ make test
[...]
spec.sources in body must be of type array: "null"
[...]
```

```
$ git checkout 671d726
created := &CdnCluster{
 ObjectMeta: metav1.ObjectMeta{
  Name: "foo", Namespace: "default"
 },
 Spec: CdnClusterSpec{
   Role: "balancer",
   Sources: []CdnClusterSource{
   { Name: "cache-live", PathCondition: "^/live/" },
   { Name: "cache-vod", PathCondition: "^/vod/" },
   },
 },
}

$ make test
ok github.com/feloy/operator/pkg/apis/cluster/v1
```

Testing the Reconcile function

```
// Prepare the environment

func TestReconcile(t *testing.T) {
  g := gomega.NewGomegaWithT(t)

  // Setup the Manager and Controller.
  // Wrap the Controller Reconcile function
  // so it writes each request to a channel when it is finished.
  mgr, err := manager.New(cfg, manager.Options{})
  g.Expect(err).NotTo(gomega.HaveOccurred())
  c := mgr.GetClient()

  recFn, requests := SetupTestReconcile(newReconciler(mgr))
  g.Expect(add(mgr, recFn)).NotTo(gomega.HaveOccurred())
  defer close(StartTestManager(mgr, g))

  [...]
}
```

Testing the Reconcile function

```
// Create an instance and test that the Reconcile function is called

instance := &clusterv1.CdnCluster{
  ObjectMeta: metav1.ObjectMeta{Name: "foo",Namespace: "default"},
  Spec:       clusterv1.CdnClusterSpec{Sources: []clusterv1.CdnClusterSource{}},
}

// Create the CdnCluster object
err = c.Create(context.TODO(), instance)
g.Expect(err).NotTo(gomega.HaveOccurred())
defer c.Delete(context.TODO(), instance)

// expect the Reconcile to be called with the instance namespace and name as parameter
var expectedRequest = reconcile.Request{
  NamespacedName: types.NamespacedName{Name: "foo",Namespace: "default"},
}
const timeout = time.Second * 5

g.Eventually(requests, timeout)
 .Should(gomega.Receive(gomega.Equal(expectedRequest)))
```

Testing the Reconcile function

```
// Then expect that a deployment is created with the expected name

deploy := &appsv1.Deployment{}

var depKey = types.NamespacedName{
  Name:      "foo-deployment",
  Namespace: "default",
}

g.Eventually(func() error {
  return c.Get(context.TODO(), depKey, deploy)
}, timeout).Should(gomega.Succeed())
```

Testing Status is set

```
// Get CDN cluster and expect the state to be "Deploying"

c.Get(context.TODO(), types.NamespacedName{
  Name:      "foo2",
  Namespace: "default",
}, instance)

g.Expect(instance.Status.State).To(gomega.Equal("Deploying"))
```

Testing Sending Events

```
// Prepare the environment

func TestReconcile(t *testing.T) {
  g := gomega.NewGomegaWithT(t)

  // Setup the Manager and Controller.
  // Wrap the Controller Reconcile function
  // so it writes each request to a channel when it is finished.
  mgr, err := manager.New(cfg, manager.Options{})
  g.Expect(err).NotTo(gomega.HaveOccurred())
  c := mgr.GetClient()

  eventRecorder := record.NewFakeRecorder(1024)
  recFn, requests := SetupTestReconcile(newReconciler(mgr, eventRecorder))
  g.Expect(add(mgr, recFn)).NotTo(gomega.HaveOccurred())
  defer close(StartTestManager(mgr, g))
```

Testing Sending Events

```
var eventReceived string

select {
case eventReceived = <-eventRecorder.Events:
}

g.Expect(eventReceived).To(gomega.Equal("Normal SourceNotFound
Source asource not found, will retry later"))
```

The END

Thanks!

Resources

Kubebuilder Documentation:

https://book.kubebuilder.io/

GitHub repository with complete source code and more documentation:

https://github.com/feloy/operator

My Medium blog (about Angular, Kubernetes):

https://medium.com/@feloy

Operator SDK (another way to build operators):

https://github.com/operator-framework/operator-sdk