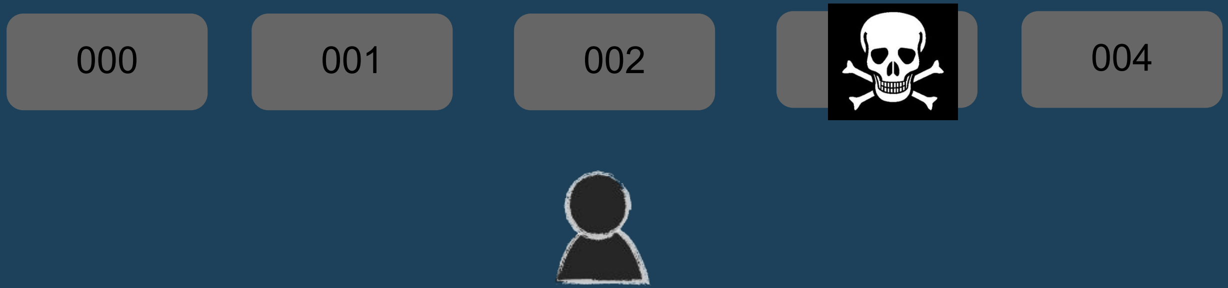
ReplicaSets & Deployments

(The Underrated, OG Operators)

Why do we care about ReplicaSets (formerly ReplicationControllers)?

Redundancy

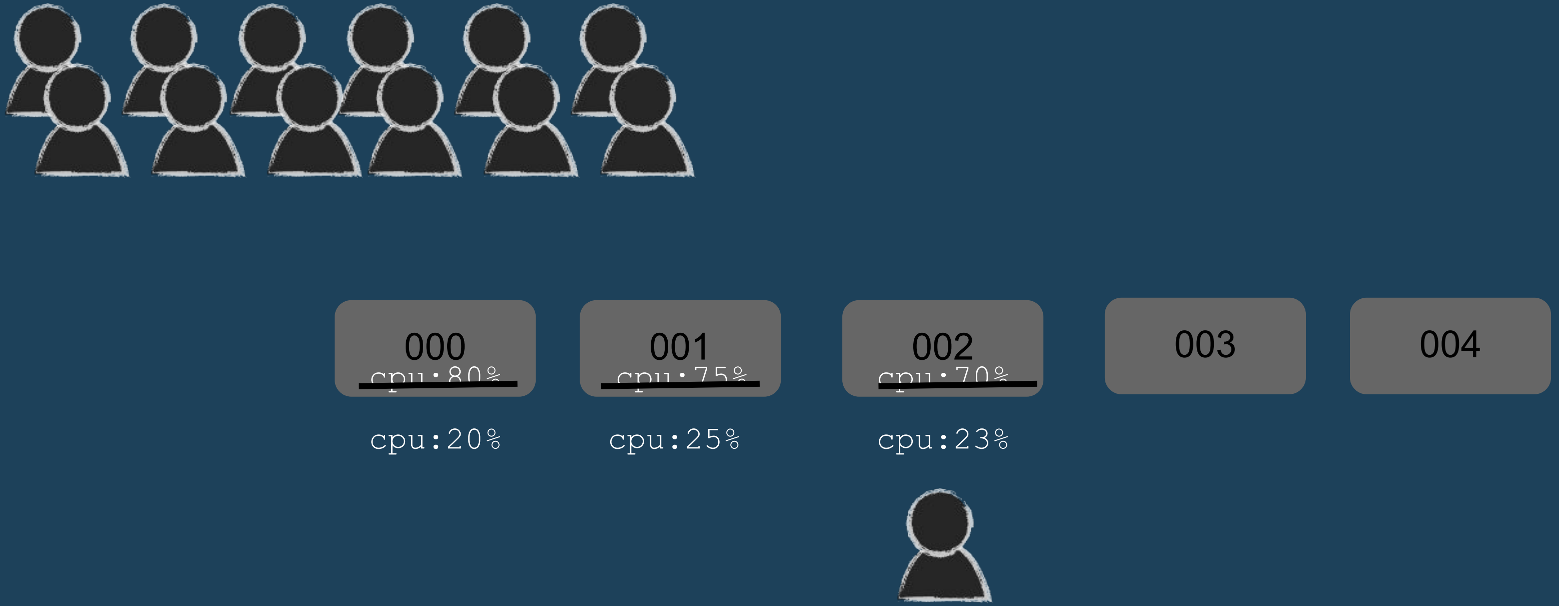
Multiple running instances means failure can be tolerated.



000 001 002 004

Scale

Multiple running instances mean more requests can be handled.



000 ~~cpu:80%~~

cpu:20%

001 ~~cpu:75%~~

cpu:25%

002 ~~cpu:70%~~

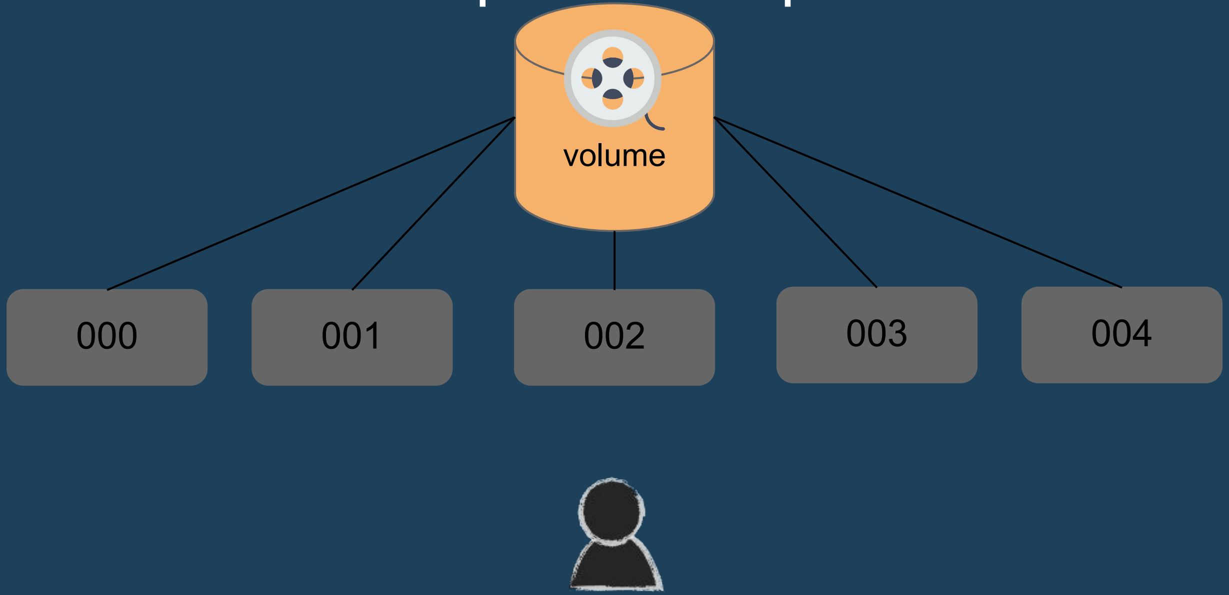
cpu:23%

003

004

Sharding

Multiple running instances can handle different parts of a computation in parallel.

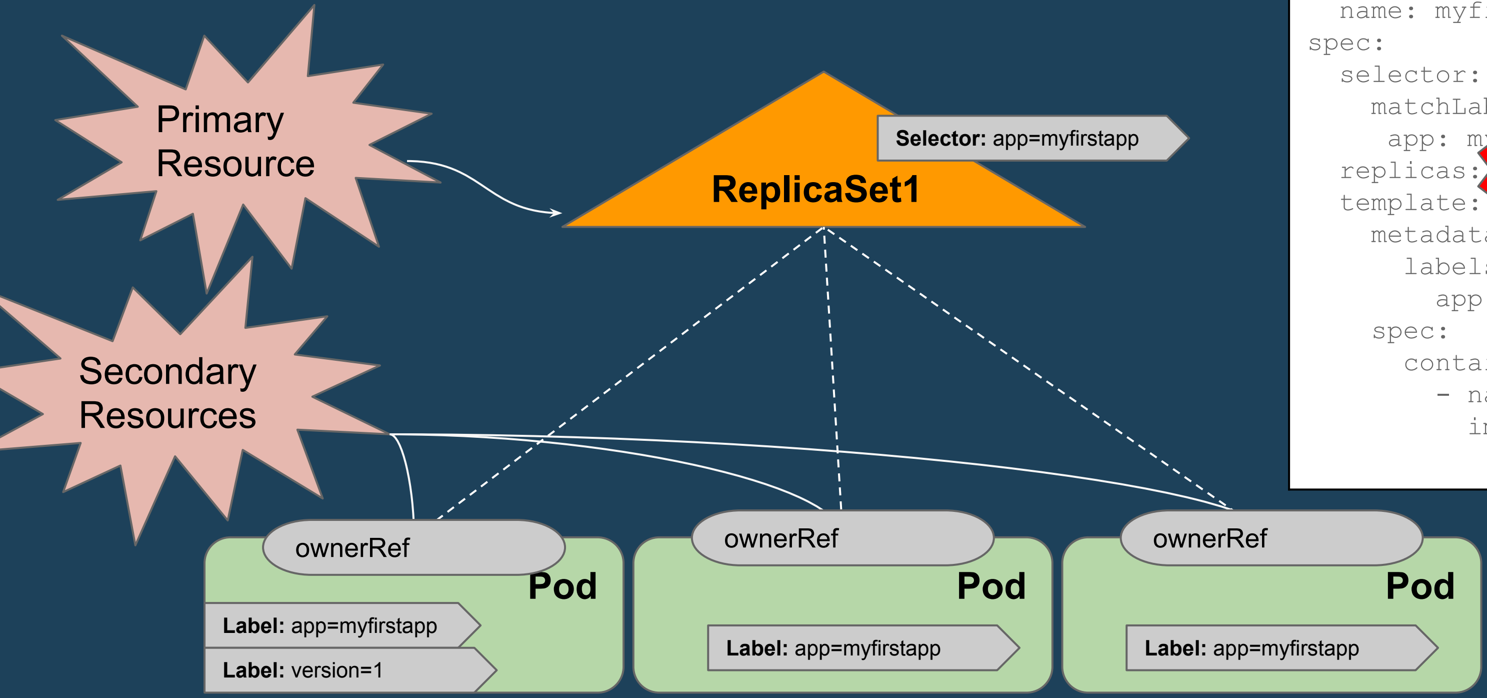


ReplicaSets in Action!

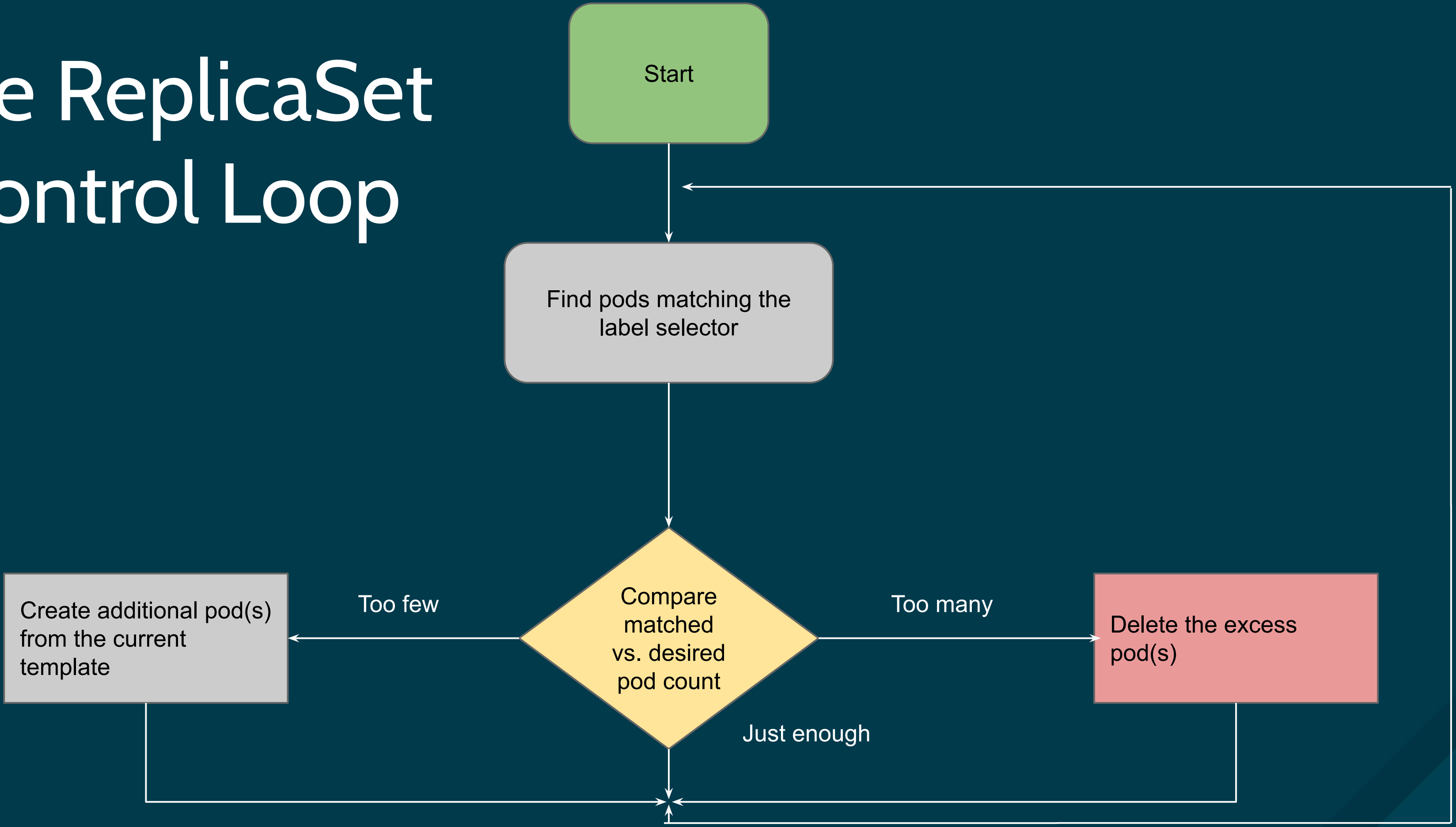
```
kubectl run myfirstapp --image quay.io/coreostrainme/hello-whoami:2.0.1  --restart=Never -l app=myfirstapp,version=1
```

```
kubectl create -f  myfirstreplicaset.yaml
kubectl scale replicaset myfirstreplicaset --replicas=3
```

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: ✗ 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```



The ReplicaSet Control Loop



Start

Find pods matching the label selector

Compare matched vs. desired pod count

Too few

Create additional pod(s) from the current template

Too many

Delete the excess pod(s)

Just enough

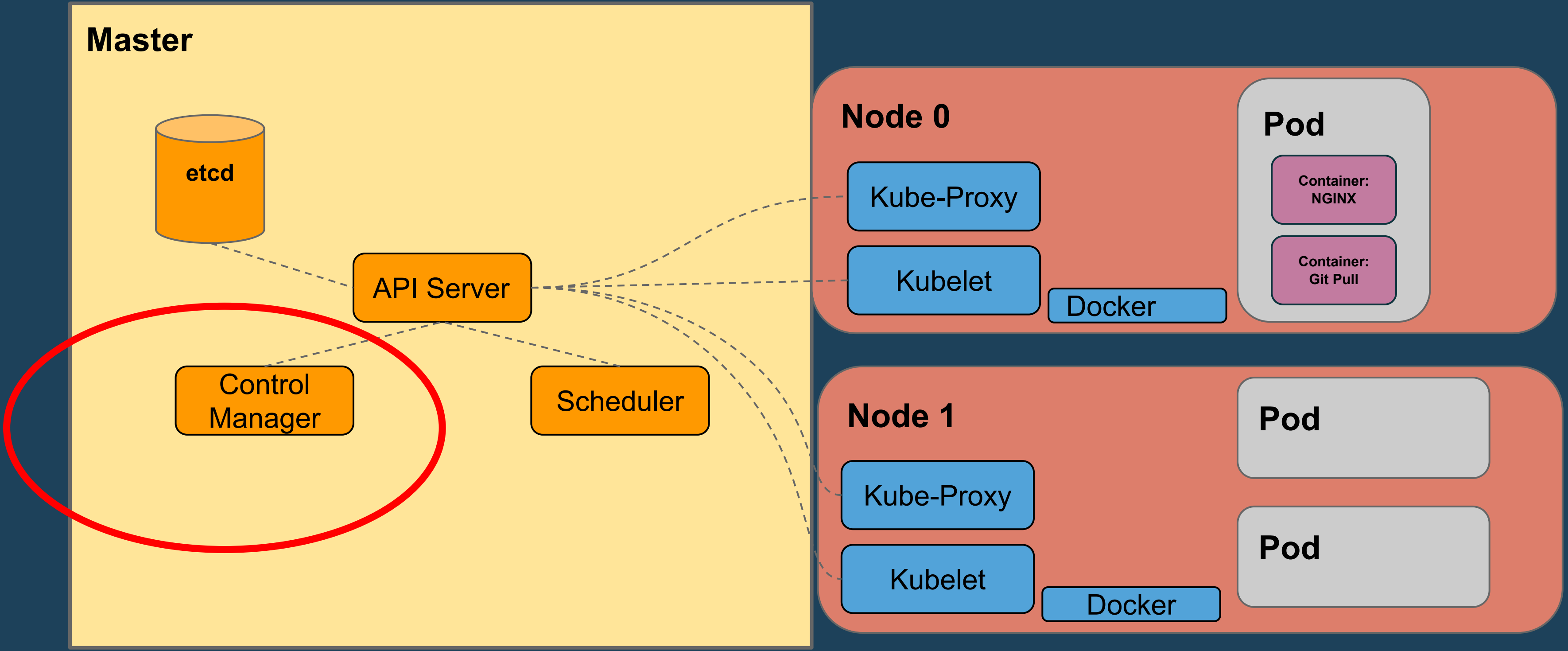
How do we accomplish this?

kube-controller-manager

kube-controller-manager

- Daemon that embeds the core control loops shipped with Kubernetes.
- The control loop is non-terminating loop that regulates the state of the system.
- In Kubernetes, there are multiple control loops running at all times that watch the shared state of the cluster through the apiserver and make changes attempting to move the current state towards the desired state.
- Examples include: ReplicaSets, Endpoints, Deployments, DaemonSets, etc.

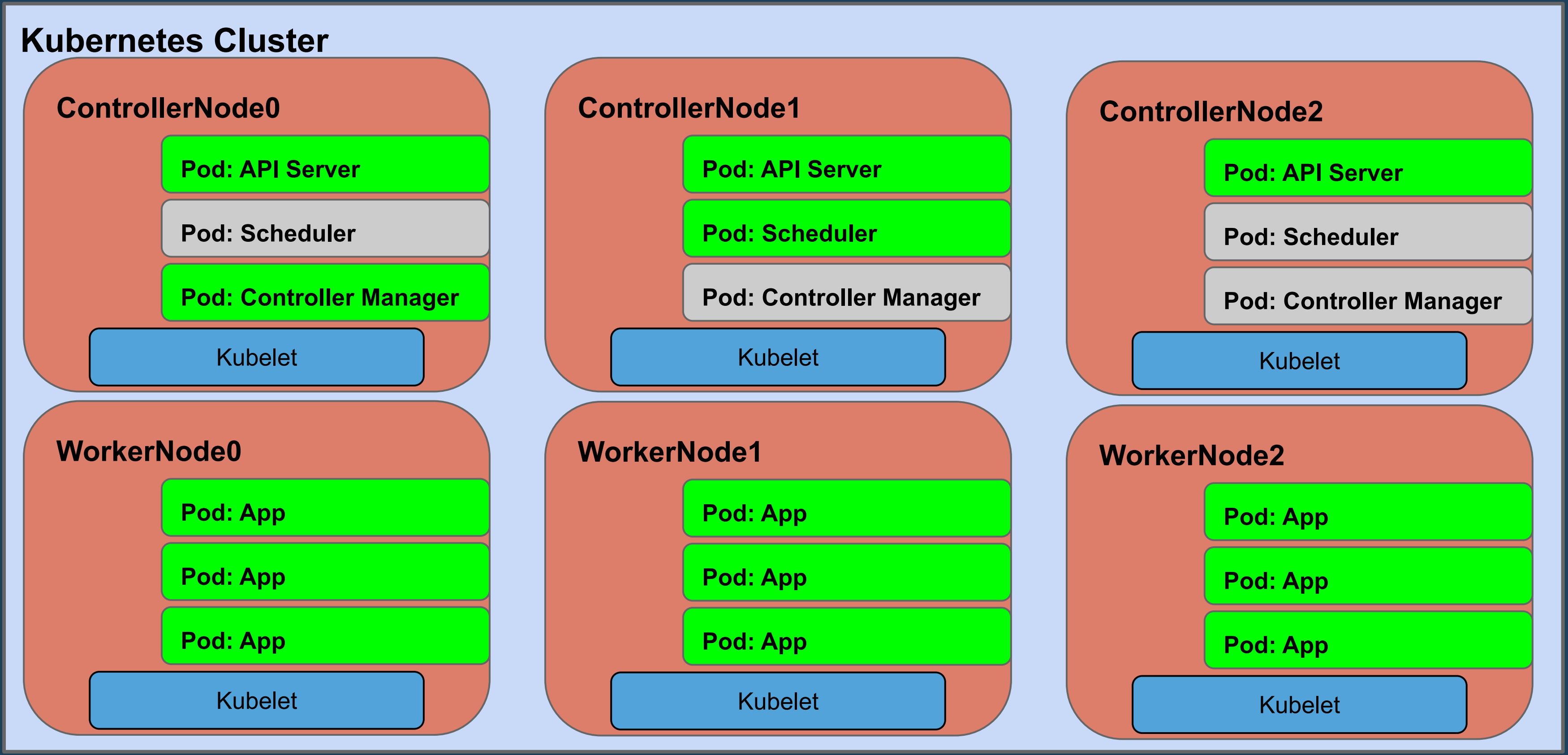
Kubernetes Cluster



Self-Hosted Kubernetes

🟩 Active

⬜ Passive/Standby



Kubernetes Cluster

ControllerNode0
🟩 Pod: API Server
⬜ Pod: Scheduler
🟩 Pod: Controller Manager
Kubelet

ControllerNode1
🟩 Pod: API Server
🟩 Pod: Scheduler
⬜ Pod: Controller Manager
Kubelet

ControllerNode2
🟩 Pod: API Server
⬜ Pod: Scheduler
⬜ Pod: Controller Manager
Kubelet

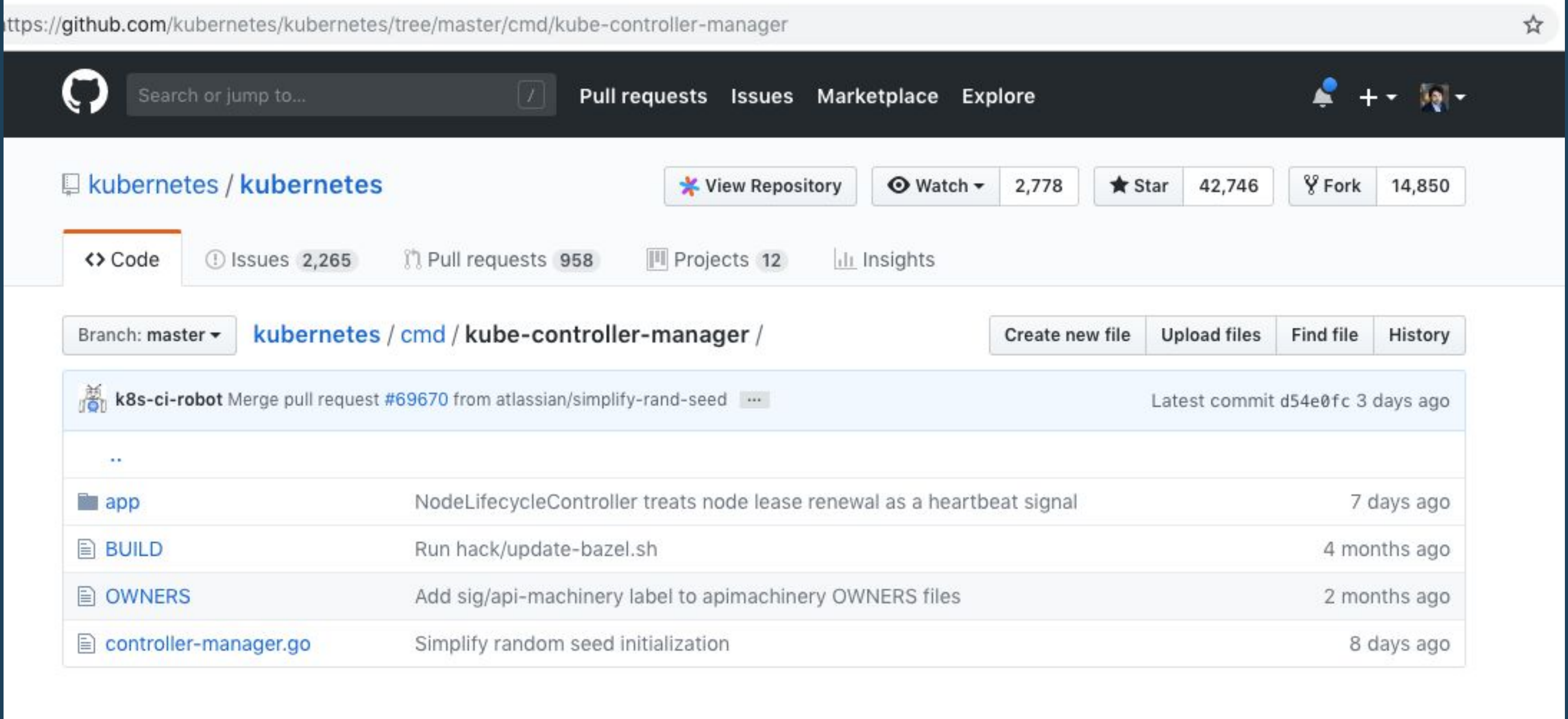
WorkerNode0
🟩 Pod: App
🟩 Pod: App
🟩 Pod: App
Kubelet

WorkerNode1
🟩 Pod: App
🟩 Pod: App
🟩 Pod: App
Kubelet

WorkerNode2
🟩 Pod: App
🟩 Pod: App
🟩 Pod: App
Kubelet

```
oc -n kube-system logs kube-controller-manager-localhost -f
```

Let’s head to GitHub.



ttps://github.com/kubernetes/kubernetes/tree/master/cmd/kube-controller-manager

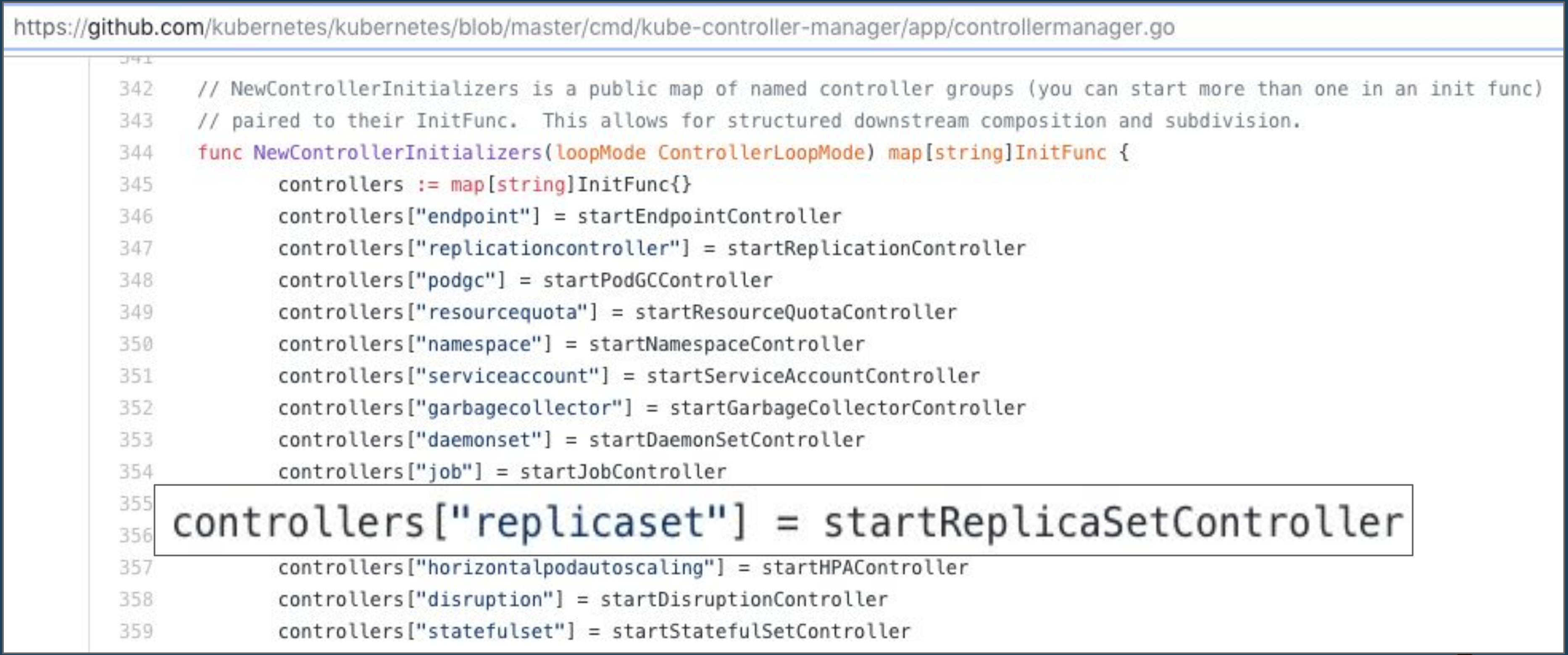
kubernetes / kubernetes

Branch: master ▾ kubernetes / cmd / kube-controller-manager /

k8s-ci-robot Merge pull request #69670 from atlassian/simplify-rand-seed … Latest commit d54e0fc 3 days ago

..		
app	NodeLifecycleController treats node lease renewal as a heartbeat signal	7 days ago
BUILD	Run hack/update-bazel.sh	4 months ago
OWNERS	Add sig/api-machinery label to apimachinery OWNERS files	2 months ago
controller-manager.go	Simplify random seed initialization	8 days ago

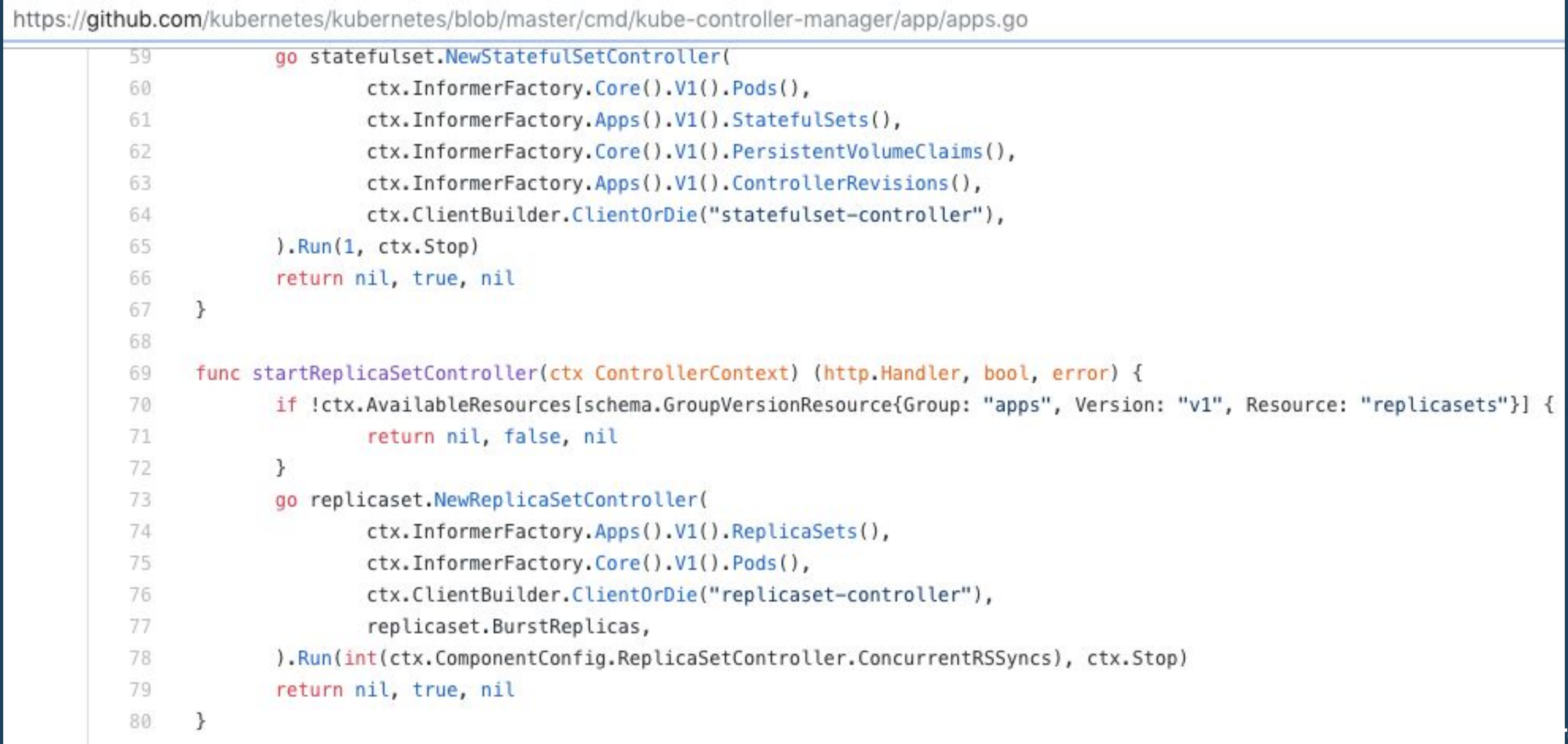
The ReplicaSet Controller Initialization.



https://github.com/kubernetes/kubernetes/blob/master/cmd/kube-controller-manager/app/controllermanager.go

```go
// NewControllerInitializers is a public map of named controller groups (you can start more than one in an init func)
// paired to their InitFunc.  This allows for structured downstream composition and subdivision.
func NewControllerInitializers(loopMode ControllerLoopMode) map[string]InitFunc {
	controllers := map[string]InitFunc{}
	controllers["endpoint"] = startEndpointController
	controllers["replicationcontroller"] = startReplicationController
	controllers["podgc"] = startPodGCController
	controllers["resourcequota"] = startResourceQuotaController
	controllers["namespace"] = startNamespaceController
	controllers["serviceaccount"] = startServiceAccountController
	controllers["garbagecollector"] = startGarbageCollectorController
	controllers["daemonset"] = startDaemonSetController
	controllers["job"] = startJobController
controllers["replicaset"] = startReplicaSetController
	controllers["horizontalpodautoscaling"] = startHPAController
	controllers["disruption"] = startDisruptionController
	controllers["statefulset"] = startStatefulSetController
```

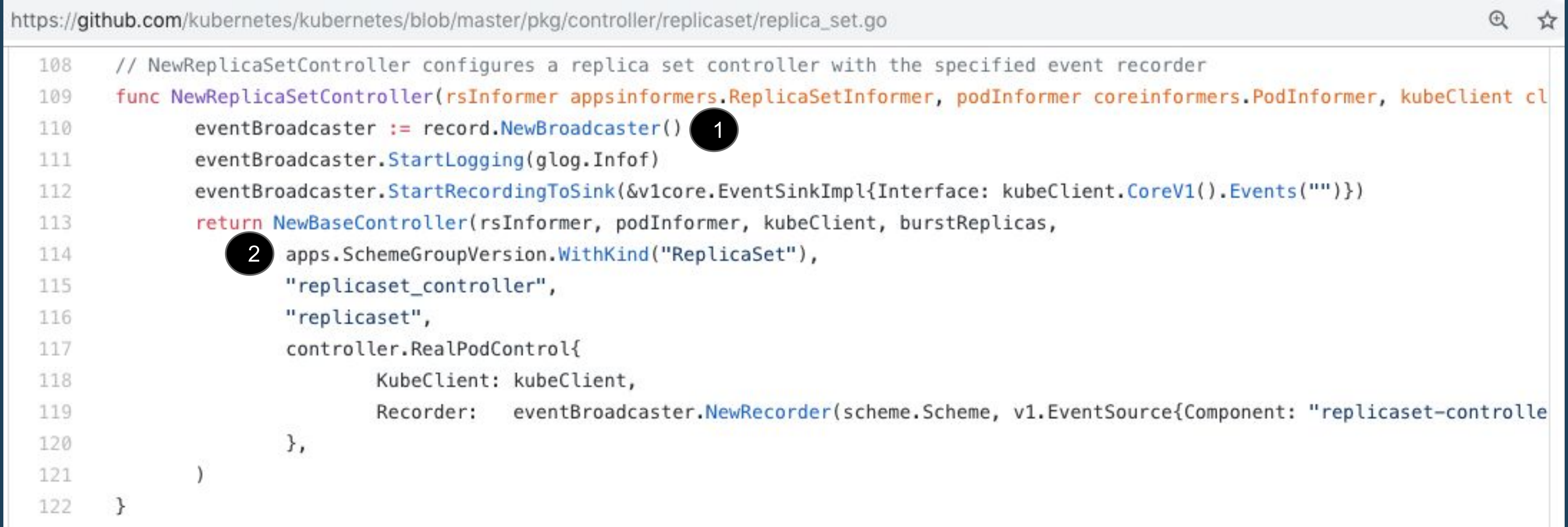
startReplicaSetController



https://github.com/kubernetes/kubernetes/blob/master/cmd/kube-controller-manager/app/apps.go

```go
	go statefulset.NewStatefulSetController(
		ctx.InformerFactory.Core().V1().Pods(),
		ctx.InformerFactory.Apps().V1().StatefulSets(),
		ctx.InformerFactory.Core().V1().PersistentVolumeClaims(),
		ctx.InformerFactory.Apps().V1().ControllerRevisions(),
		ctx.ClientBuilder.ClientOrDie("statefulset-controller"),
	).Run(1, ctx.Stop)
	return nil, true, nil
}

func startReplicaSetController(ctx ControllerContext) (http.Handler, bool, error) {
	if !ctx.AvailableResources[schema.GroupVersionResource{Group: "apps", Version: "v1", Resource: "replicasets"}] {
		return nil, false, nil
	}
	go replicaset.NewReplicaSetController(
		ctx.InformerFactory.Apps().V1().ReplicaSets(),
		ctx.InformerFactory.Core().V1().Pods(),
		ctx.ClientBuilder.ClientOrDie("replicaset-controller"),
		replicaset.BurstReplicas,
	).Run(int(ctx.ComponentConfig.ReplicaSetController.ConcurrentRSSyncs), ctx.Stop)
	return nil, true, nil
}
```

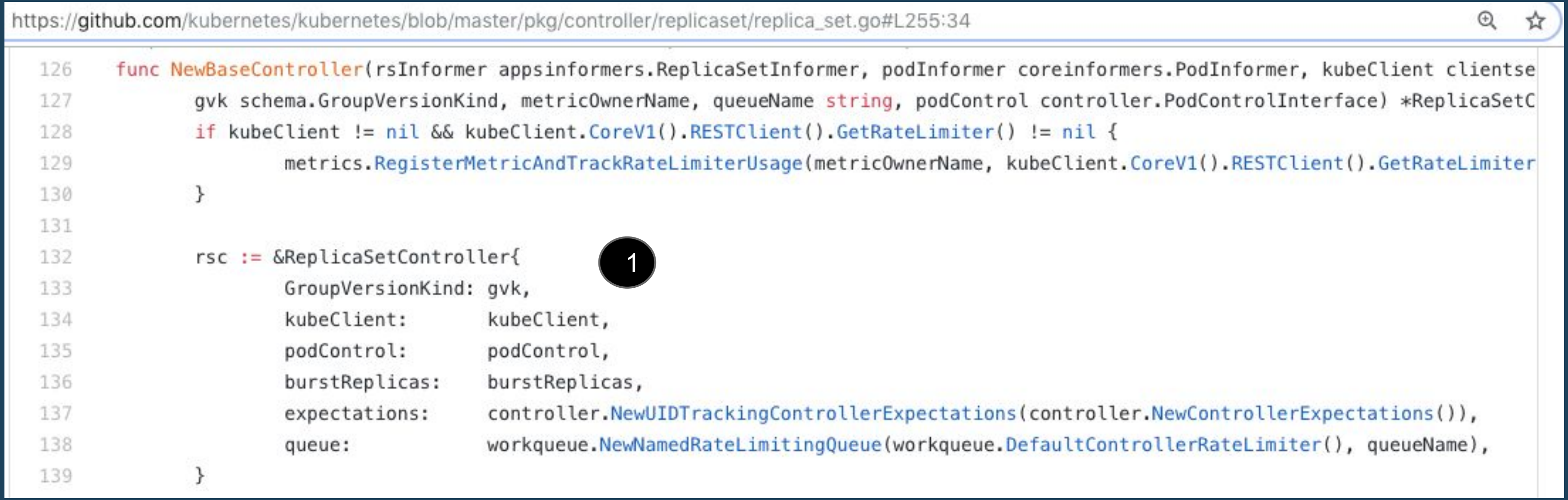
newReplicaSetController



https://github.com/kubernetes/kubernetes/blob/master/pkg/controller/replicaset/replica_set.go

```go
// NewReplicaSetController configures a replica set controller with the specified event recorder
func NewReplicaSetController(rsInformer appsinformers.ReplicaSetInformer, podInformer coreinformers.PodInformer, kubeClient cl
        eventBroadcaster := record.NewBroadcaster()
        eventBroadcaster.StartLogging(glog.Infof)
        eventBroadcaster.StartRecordingToSink(&v1core.EventSinkImpl{Interface: kubeClient.CoreV1().Events("")})
        return NewBaseController(rsInformer, podInformer, kubeClient, burstReplicas,
                apps.SchemeGroupVersion.WithKind("ReplicaSet"),
                "replicaset_controller",
                "replicaset",
                controller.RealPodControl{
                        KubeClient: kubeClient,
                        Recorder:   eventBroadcaster.NewRecorder(scheme.Scheme, v1.EventSource{Component: "replicaset-controlle
                },
        )
}
```

newBaseController

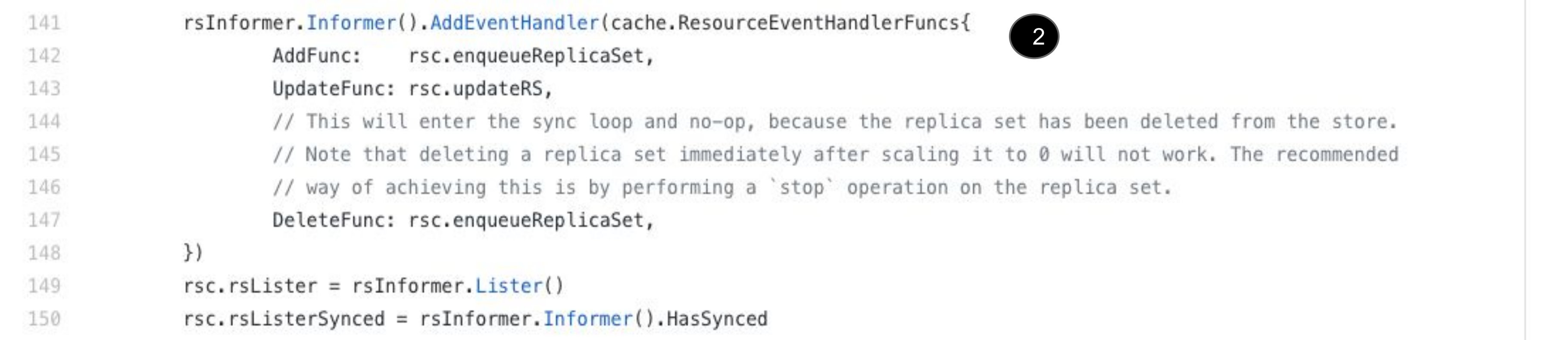


https://github.com/kubernetes/kubernetes/blob/master/pkg/controller/replicaset/replica_set.go#L255:34

```
func NewBaseController(rsInformer appsinformers.ReplicaSetInformer, podInformer coreinformers.PodInformer, kubeClient clientse
	gvk schema.GroupVersionKind, metricOwnerName, queueName string, podControl controller.PodControlInterface) *ReplicaSetC
	if kubeClient != nil && kubeClient.CoreV1().RESTClient().GetRateLimiter() != nil {
		metrics.RegisterMetricAndTrackRateLimiterUsage(metricOwnerName, kubeClient.CoreV1().RESTClient().GetRateLimiter
	}

	rsc := &ReplicaSetController{
		GroupVersionKind: gvk,
		kubeClient:       kubeClient,
		podControl:       podControl,
		burstReplicas:    burstReplicas,
		expectations:     controller.NewUIDTrackingControllerExpectations(controller.NewControllerExpectations()),
		queue:            workqueue.NewNamedRateLimitingQueue(workqueue.DefaultControllerRateLimiter(), queueName),
	}
```

1

newBaseController

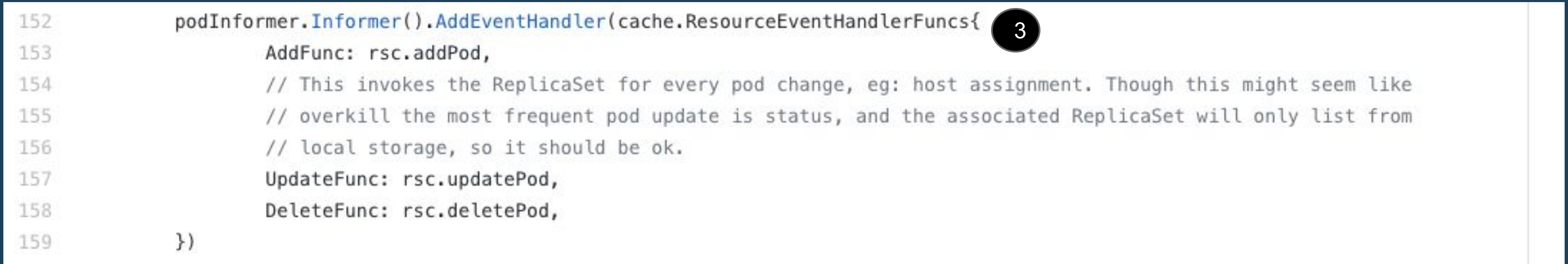


```
	rsInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
		AddFunc:    rsc.enqueueReplicaSet,
		UpdateFunc: rsc.updateRS,
		// This will enter the sync loop and no-op, because the replica set has been deleted from the store.
		// Note that deleting a replica set immediately after scaling it to 0 will not work. The recommended
		// way of achieving this is by performing a `stop` operation on the replica set.
		DeleteFunc: rsc.enqueueReplicaSet,
	})
	rsc.rsLister = rsInformer.Lister()
	rsc.rsListerSynced = rsInformer.Informer().HasSynced
```

2

Watching Primary Resource: kind:ReplicaSet

newBaseController

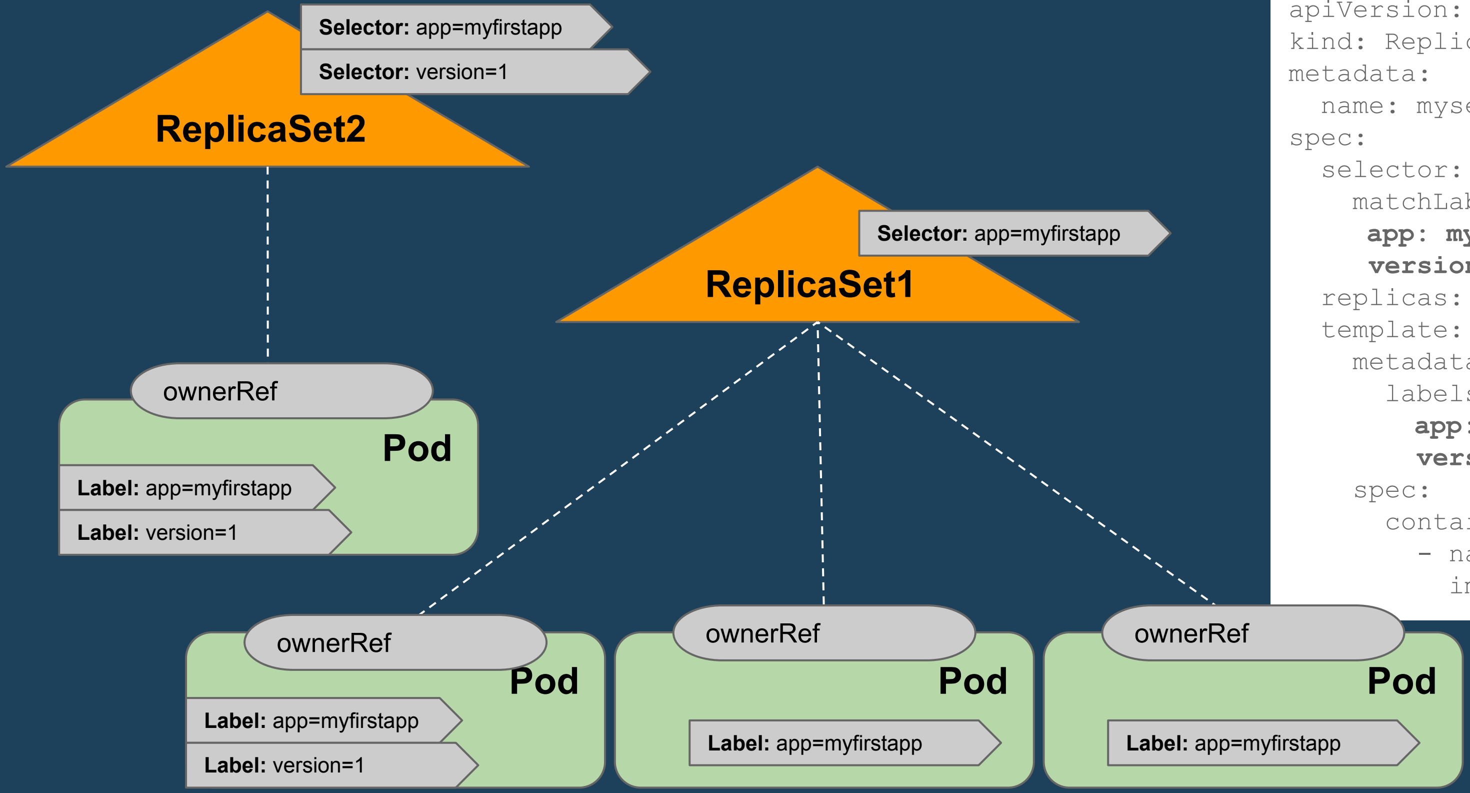


```
podInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
        AddFunc: rsc.addPod,
        // This invokes the ReplicaSet for every pod change, eg: host assignment. Though this might seem like
        // overkill the most frequent pod update is status, and the associated ReplicaSet will only list from
        // local storage, so it should be ok.
        UpdateFunc: rsc.updatePod,
        DeleteFunc: rsc.deletePod,
})
```

Watching Secondary Resource: kind:Pod

Creating another ReplicaSet

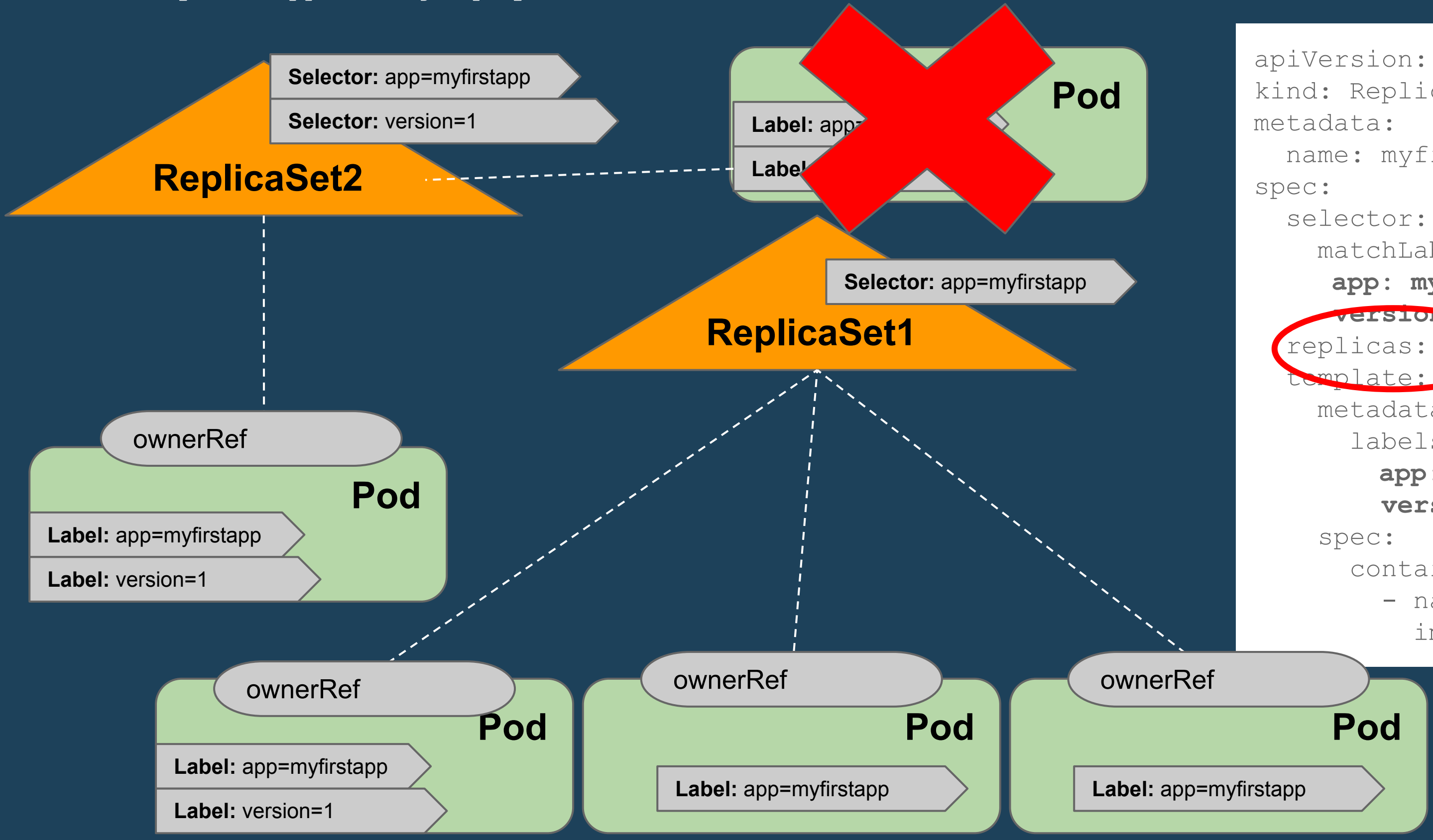
```
kubectl create -f  mysecondreplicaset.yaml
```



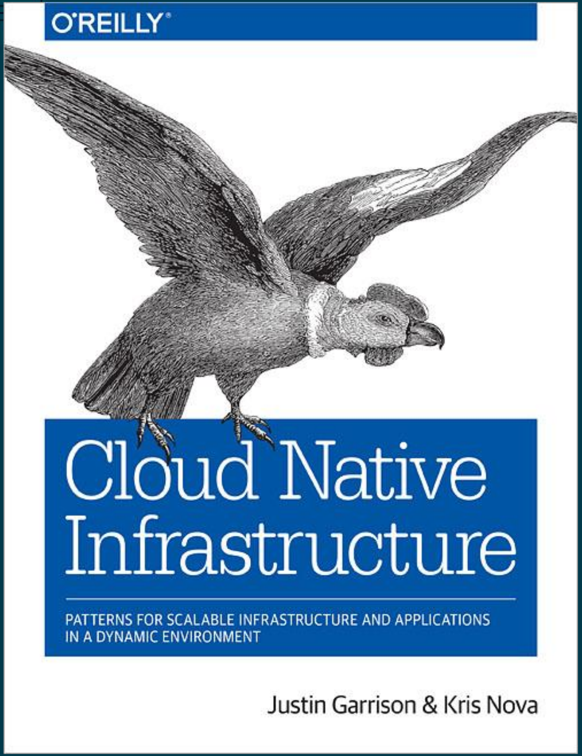
```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: mysecondreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
     version: 1
  replicas: 1
  template:
    metadata:
      labels:
        app: myfirstapp
        version: 1
    spec:
      containers:
        - name: nodejs
          image: myimage
```

Creating Another Orphan Pod

```
kubectl run myfirstapp --image quay.io/coreostrainme/hello-whoami:2.0.1  --restart=Never -l app=myfirstapp,version=1
```

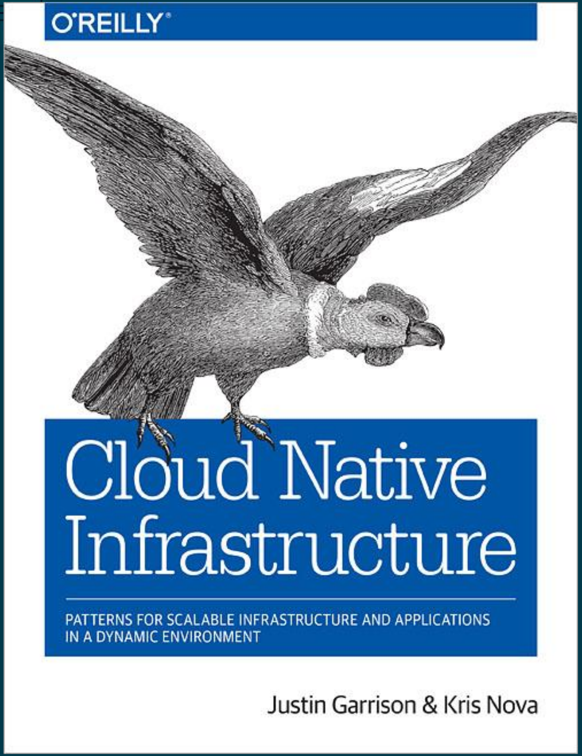


```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
     version: 1
  replicas: 1
  template:
    metadata:
      labels:
        app: myfirstapp
        version: 1
    spec:
      containers:
        - name: nodejs
          image: myimage
```


O'REILLY
Cloud Native Infrastructure
PATTERNS FOR SCALABLE INFRASTRUCTURE AND APPLICATIONS IN A DYNAMIC ENVIRONMENT
Justin Garrison & Kris Nova

Chapter 4
Designing Infrastructure Applications

*The **reconciler pattern** is a software pattern that can be used or expanded upon for managing cloud native infrastructure. The pattern enforces the idea of having two representations of the infrastructure—the first being the actual state of the infrastructure, and the second being the expected state of the infrastructure.*



O'REILLY

Cloud Native Infrastructure

PATTERNS FOR SCALABLE INFRASTRUCTURE AND APPLICATIONS IN A DYNAMIC ENVIRONMENT

Justin Garrison & Kris Nova

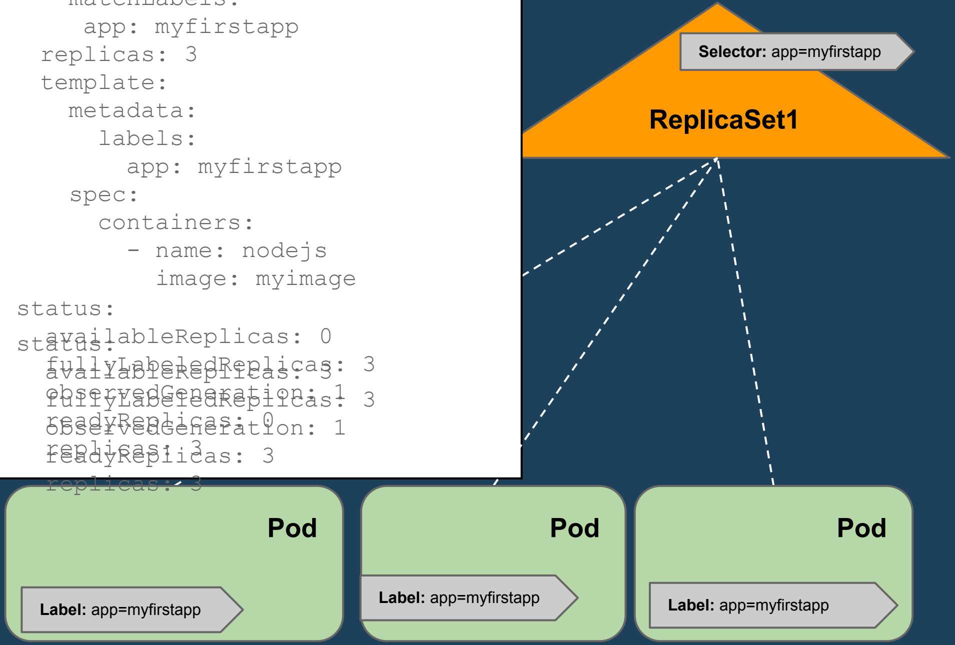
*The **reconciler pattern** will force the engineer to have two independent avenues for getting either of these representations, as well as to implement a solution to reconcile the actual state into the expected state.*

ReplicaSets in Action!

```
kubectl create -f  myfirstreplicaset.yaml
```

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
status:
  availableReplicas: 0
  fullyLabeledReplicas: 3
  observedGeneration: 1
  readyReplicas: 0
  replicas: 3
```

```
status:
  availableReplicas: 3
  fullyLabeledReplicas: 3
  observedGeneration: 1
  readyReplicas: 3
  replicas: 3
```



Selector: app=myfirstapp

ReplicaSet1

Pod

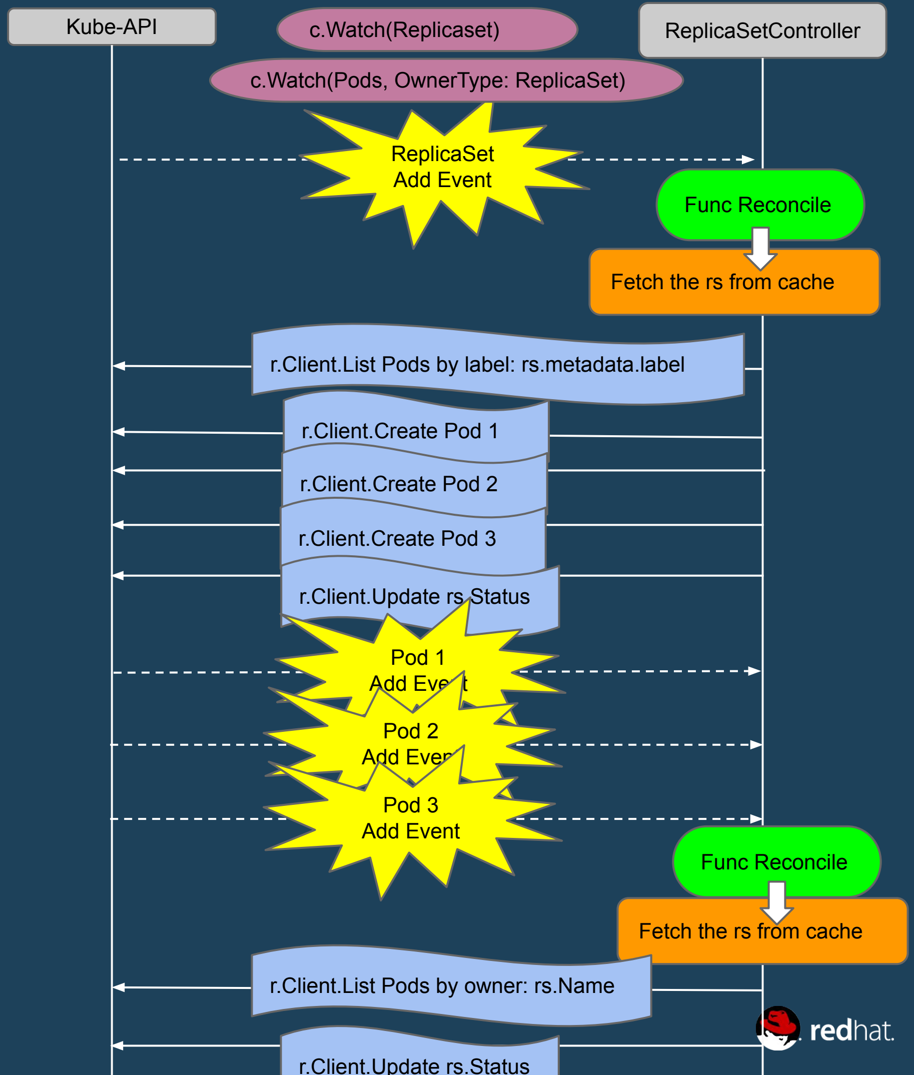
Label: app=myfirstapp

Pod

Label: app=myfirstapp

Pod

Label: app=myfirstapp



Kube-API

c.Watch(Replicaset)

ReplicaSetController

c.Watch(Pods, OwnerType: ReplicaSet)

ReplicaSet Add Event

Func Reconcile

Fetch the rs from cache

r.Client.List Pods by label: rs.metadata.label

r.Client.Create Pod 1

r.Client.Create Pod 2

r.Client.Create Pod 3

r.Client.Update rs.Status

Pod 1 Add Event

Pod 2 Add Event

Pod 3 Add Event

Func Reconcile

Fetch the rs from cache

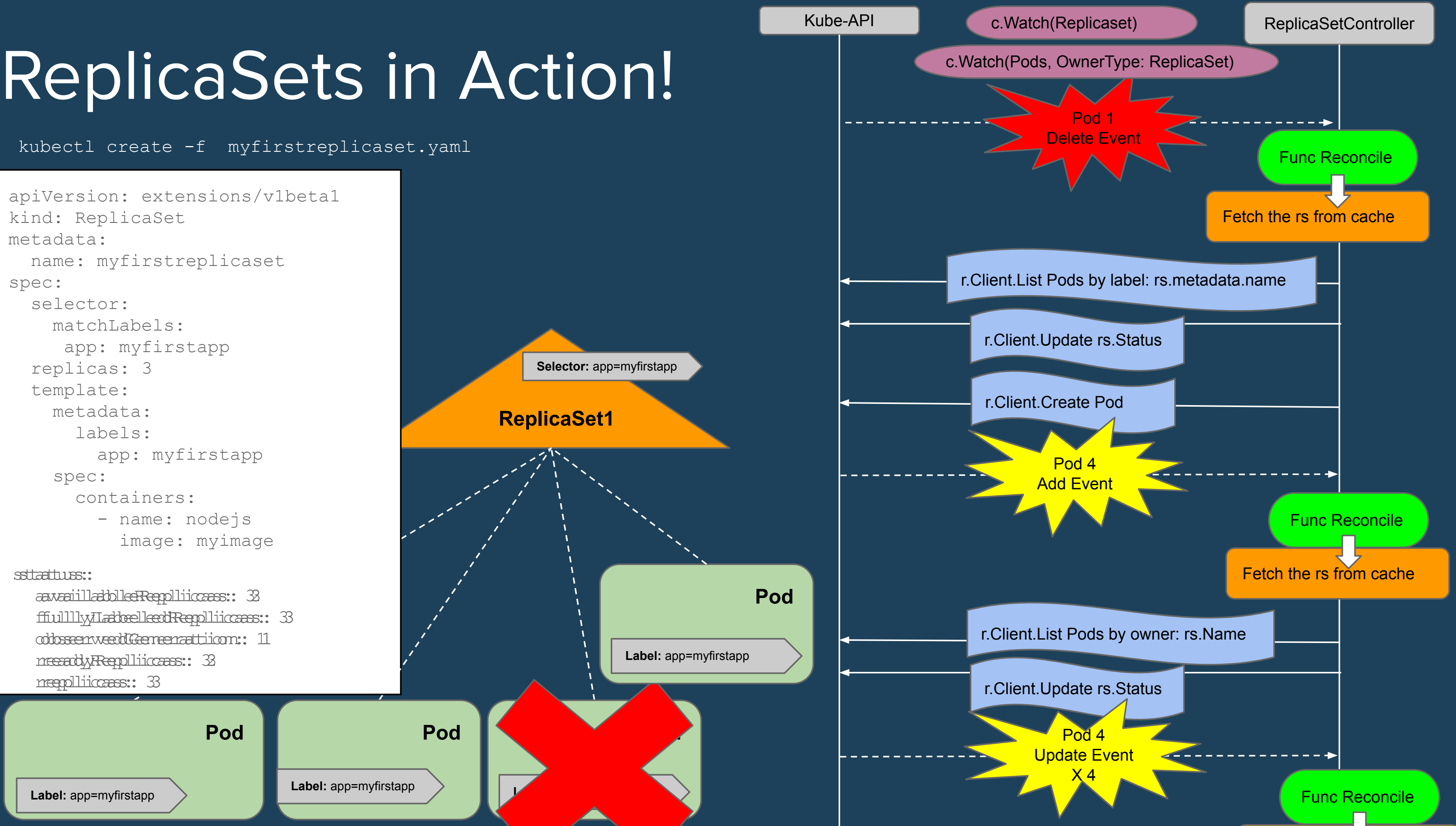
r.Client.List Pods by owner: rs.Name

r.Client.Update rs.Status

ReplicaSets in Action!

```
kubectl create -f  myfirstreplicaset.yaml
```

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage

status:
  availableReplicas: 3
  fullyLabeledReplicas: 3
  observedGeneration: 1
  readyReplicas: 3
  replicas: 3
```

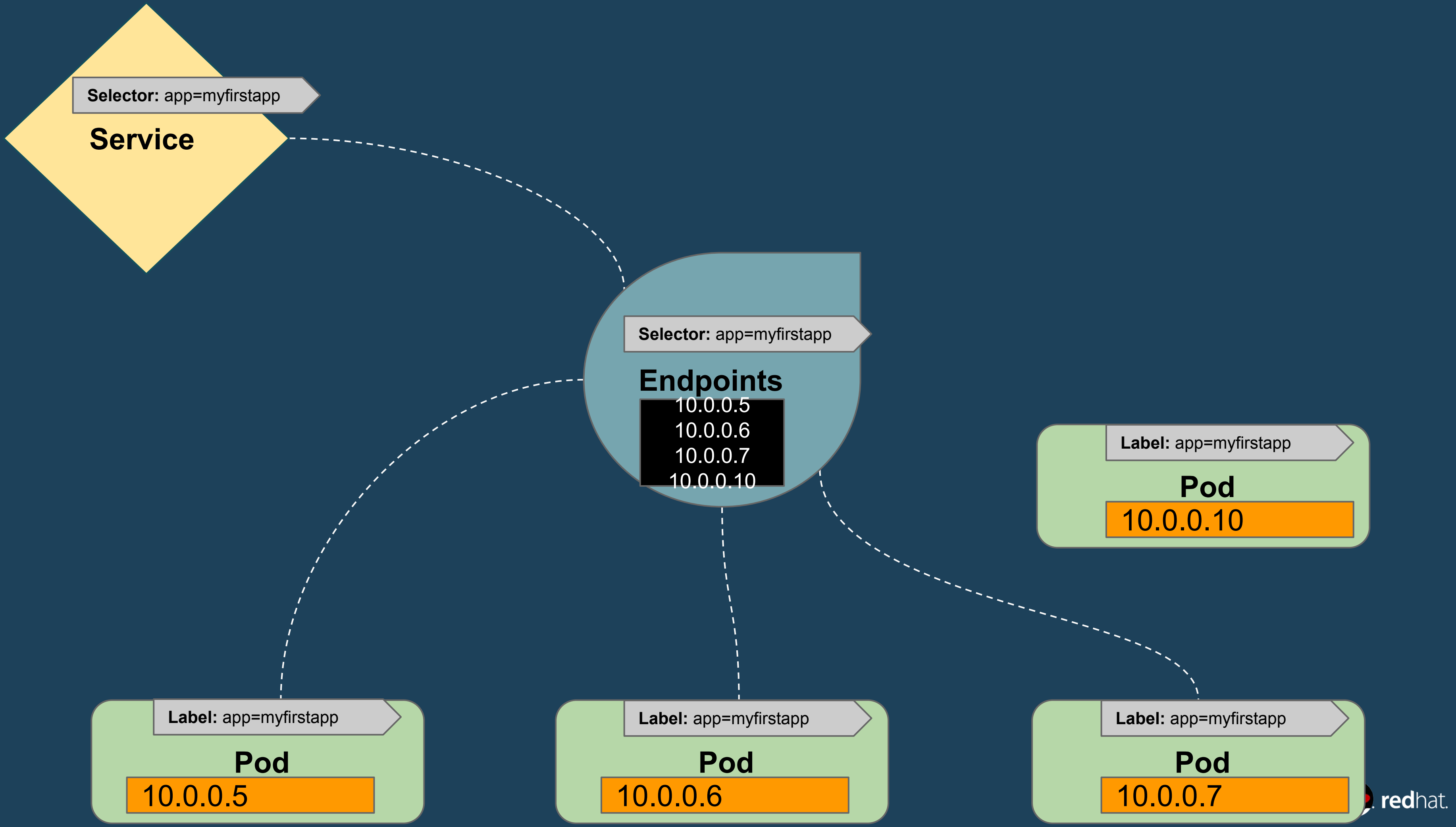


Let’s Identify Primary/Secondary Resources for Existing Kubernetes Controllers!

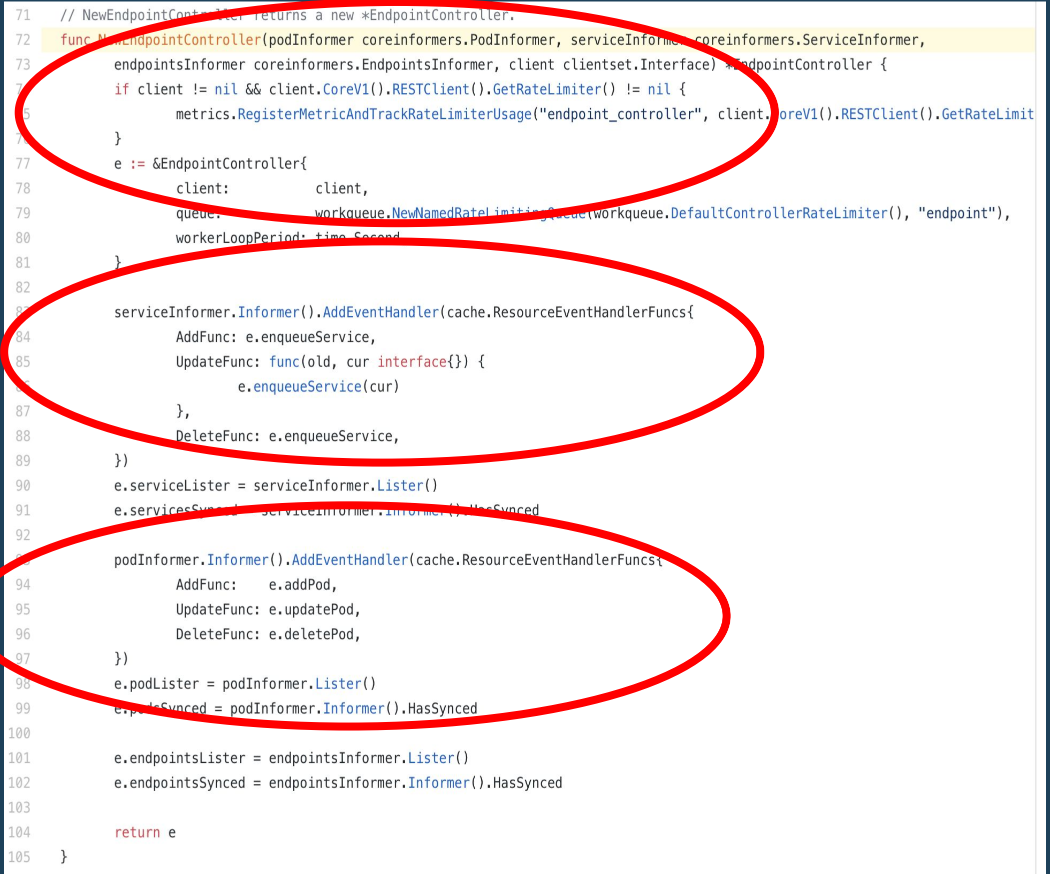
(without looking at the code!)

redhat.

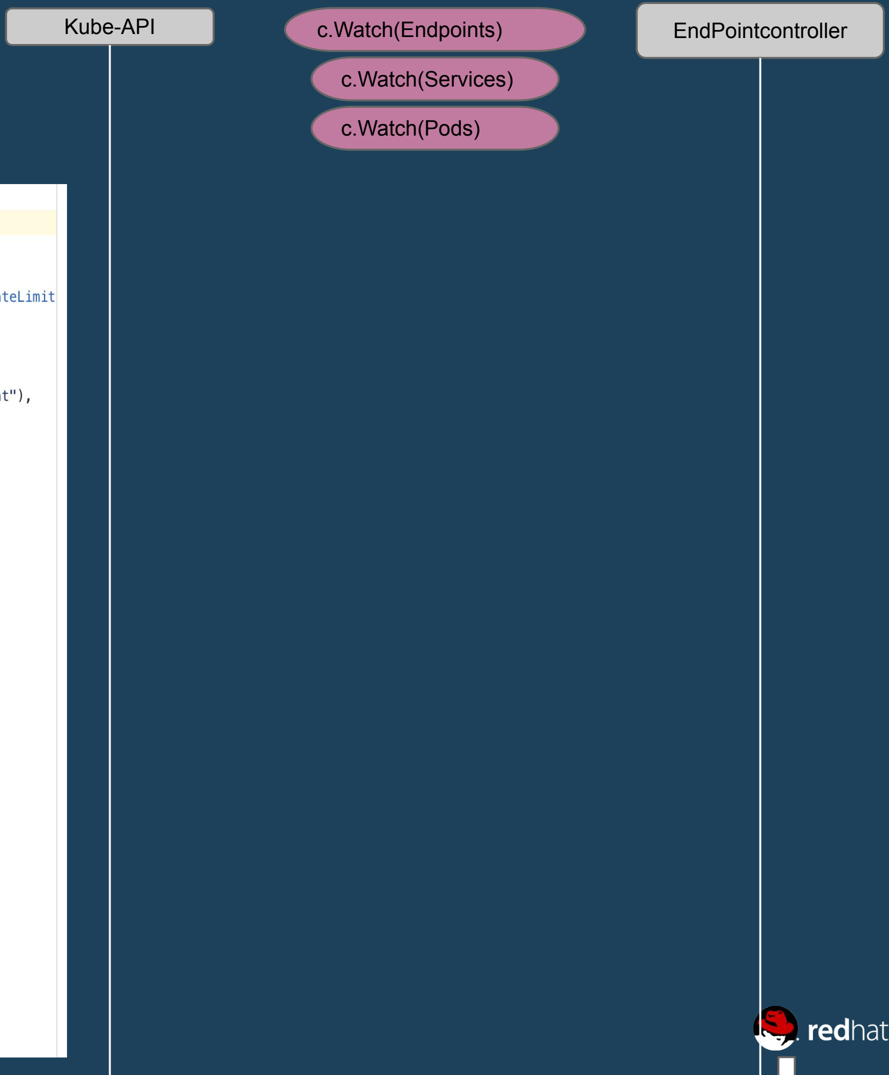
Endpoints


Selector: app=myfirstapp
Service
Selector: app=myfirstapp
Endpoints
10.0.0.5
10.0.0.6
10.0.0.7
10.0.0.10
Label: app=myfirstapp
Pod
10.0.0.10
Label: app=myfirstapp
Pod
10.0.0.5
Label: app=myfirstapp
Pod
10.0.0.6
Label: app=myfirstapp
Pod
10.0.0.7
redhat.

EndPoint Controller

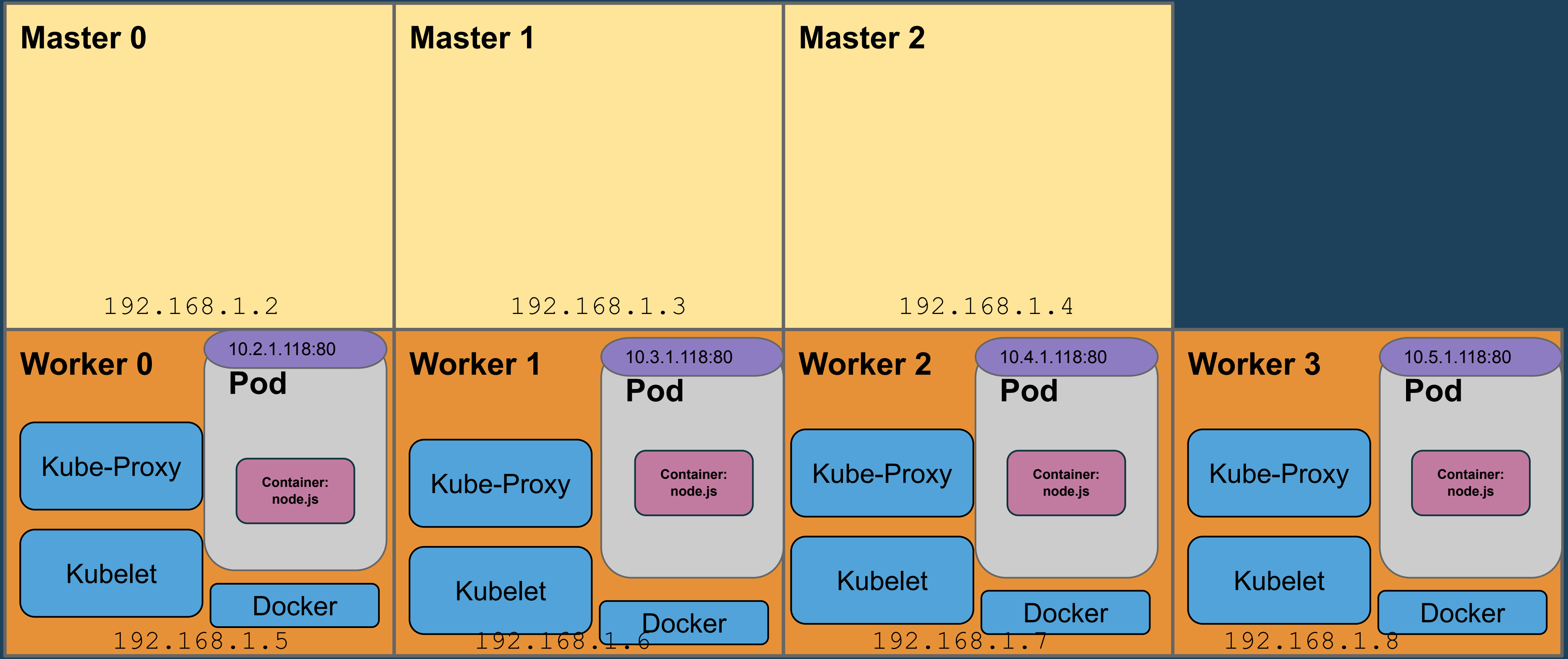


```
// NewEndpointController returns a new *EndpointController.
func NewEndpointController(podInformer coreinformers.PodInformer, serviceInformer coreinformers.ServiceInformer,
	endpointsInformer coreinformers.EndpointsInformer, client clientset.Interface) *EndpointController {
	if client != nil && client.CoreV1().RESTClient().GetRateLimiter() != nil {
		metrics.RegisterMetricAndTrackRateLimiterUsage("endpoint_controller", client.CoreV1().RESTClient().GetRateLimit
	}
	e := &EndpointController{
		client:           client,
		queue:            workqueue.NewNamedRateLimitingQueue(workqueue.DefaultControllerRateLimiter(), "endpoint"),
		workerLoopPeriod: time.Second,
	}

	serviceInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
		AddFunc: e.enqueueService,
		UpdateFunc: func(old, cur interface{}) {
			e.enqueueService(cur)
		},
		DeleteFunc: e.enqueueService,
	})
	e.serviceLister = serviceInformer.Lister()
	e.servicesSynced = serviceInformer.Informer().HasSynced

	podInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
		AddFunc:    e.addPod,
		UpdateFunc: e.updatePod,
		DeleteFunc: e.deletePod,
	})
	e.podLister = podInformer.Lister()
	e.podsSynced = podInformer.Informer().HasSynced

	e.endpointsLister = endpointsInformer.Lister()
	e.endpointsSynced = endpointsInformer.Informer().HasSynced

	return e
}
```

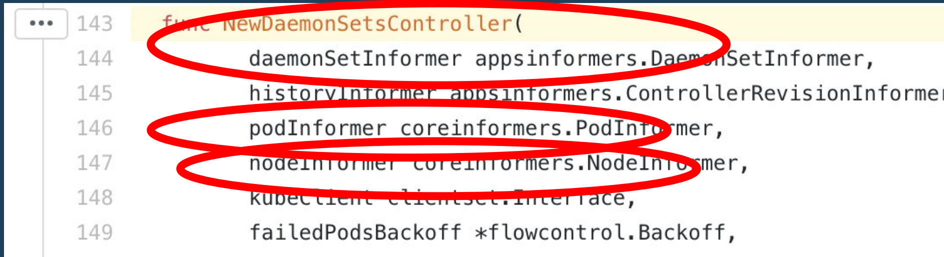


DaemonSets

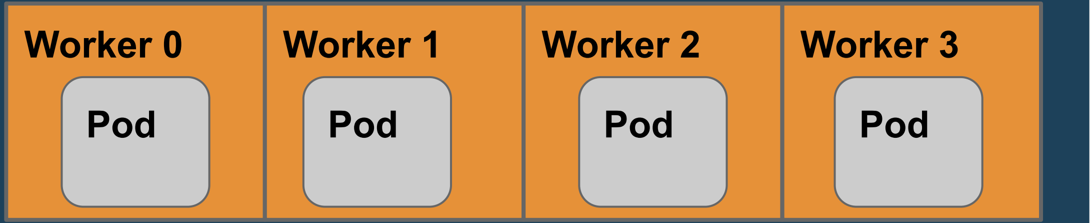
DaemonSets


Master 0
192.168.1.2
Master 1
192.168.1.3
Master 2
192.168.1.4
Worker 0
10.2.1.118:80
Pod
Kube-Proxy
Container:
node.js
Kubelet
Docker
192.168.1.5
Worker 1
10.3.1.118:80
Pod
Kube-Proxy
Container:
node.js
Kubelet
Docker
192.168.1.6
Worker 2
10.4.1.118:80
Pod
Kube-Proxy
Container:
node.js
Kubelet
Docker
192.168.1.7
Worker 3
10.5.1.118:80
Pod
Kube-Proxy
Container:
node.js
Kubelet
Docker
192.168.1.8

DaemonSetController



```
func NewDaemonSetsController(
        daemonSetInformer appsinformers.DaemonSetInformer,
        historyInformer appsinformers.ControllerRevisionInformer,
        podInformer coreinformers.PodInformer,
        nodeInformer coreinformers.NodeInformer,
        kubeClient clientset.Interface,
        failedPodsBackoff *flowcontrol.Backoff,
```



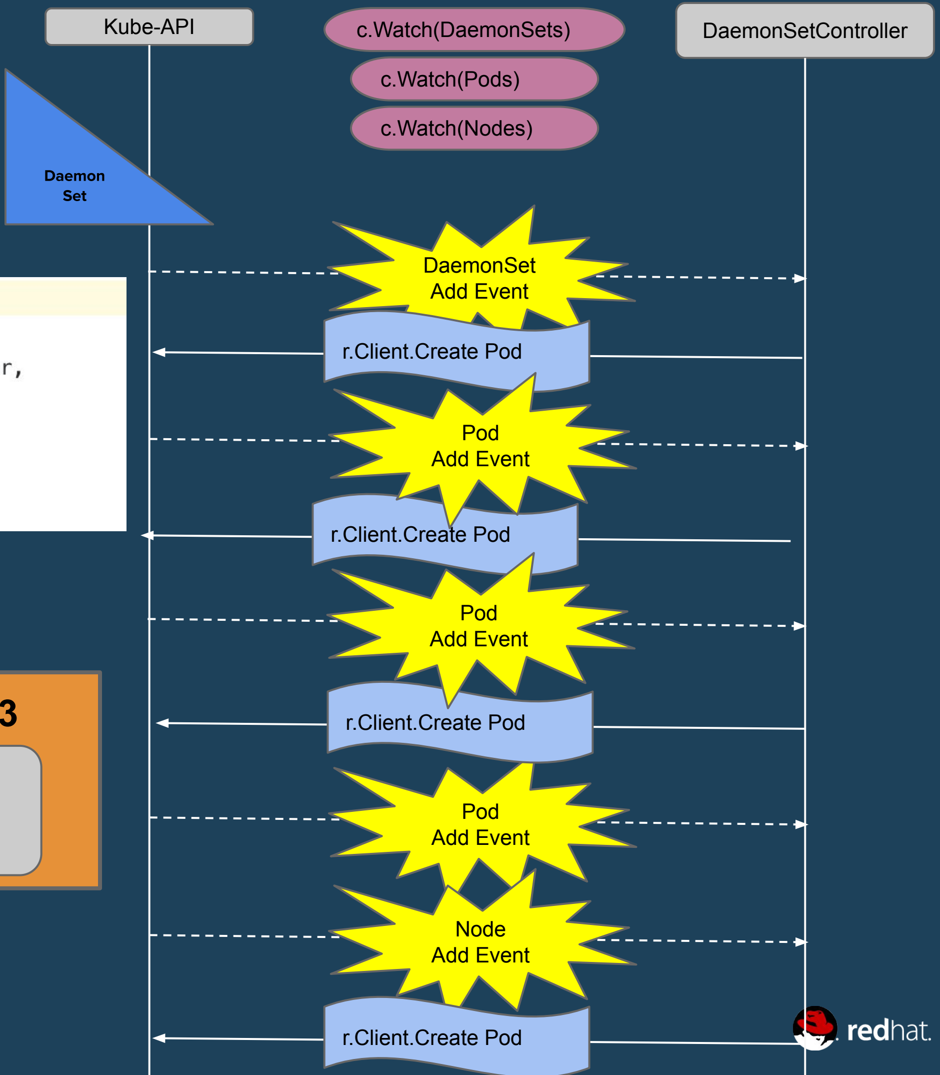
Worker 0 Pod

Worker 1 Pod

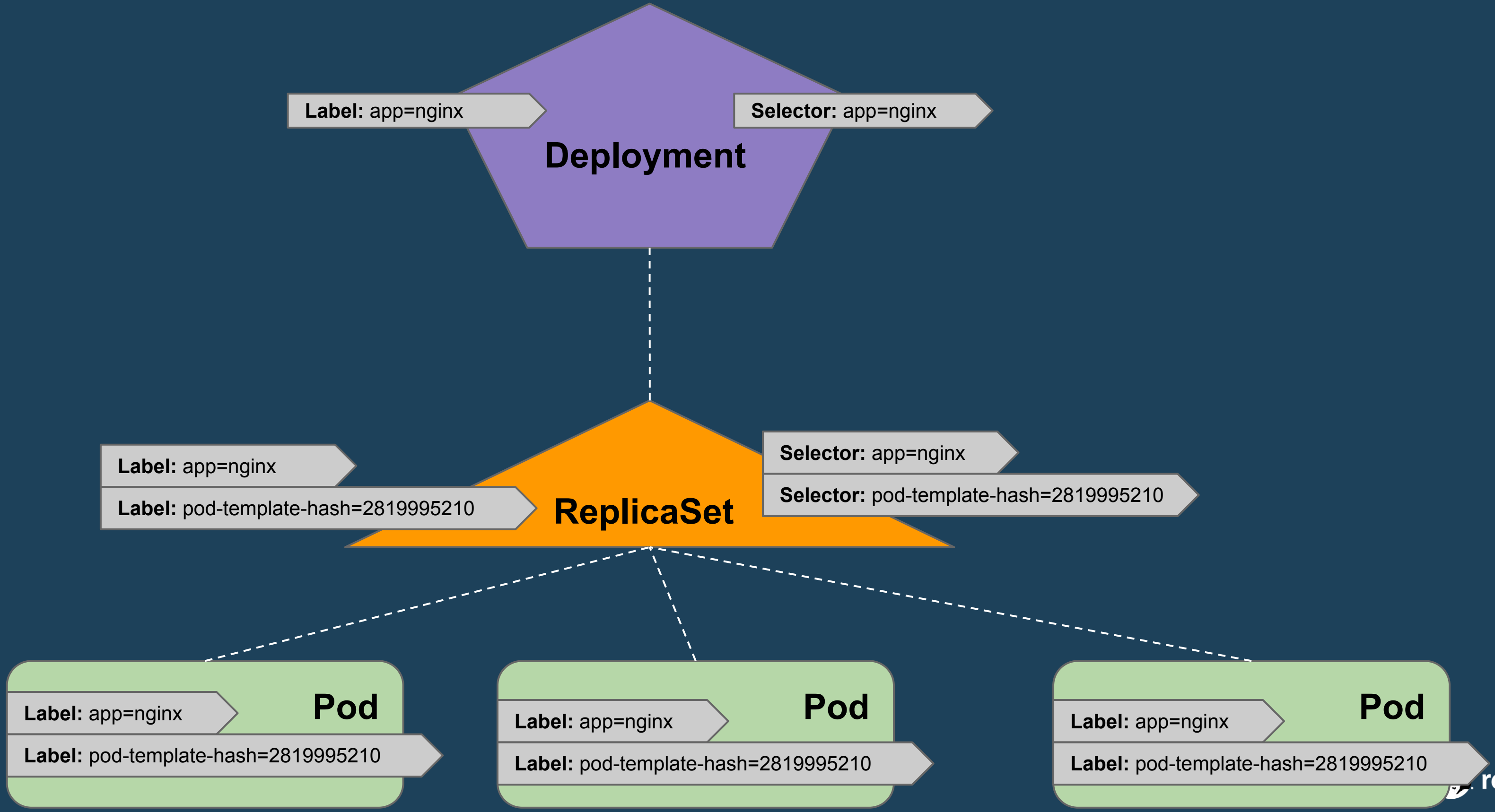
Worker 2 Pod

Worker 3 Pod

Administrator adds a new node!



Deployments



Label: app=nginx

Selector: app=nginx

Deployment

Label: app=nginx

Label: pod-template-hash=2819995210

ReplicaSet

Selector: app=nginx

Selector: pod-template-hash=2819995210

Pod

Label: app=nginx

Label: pod-template-hash=2819995210

Pod

Label: app=nginx

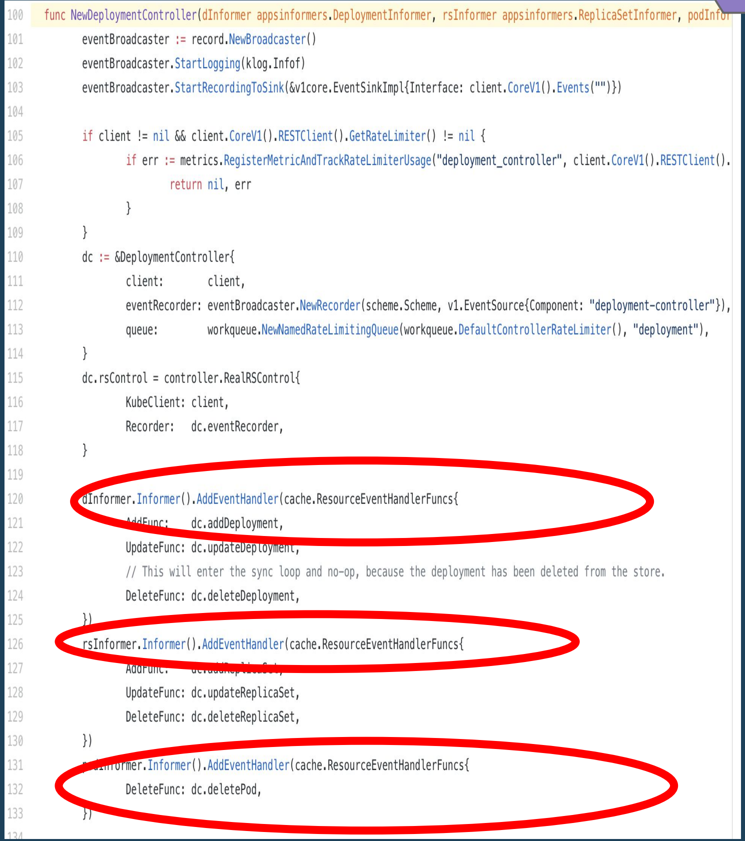
Label: pod-template-hash=2819995210

Pod

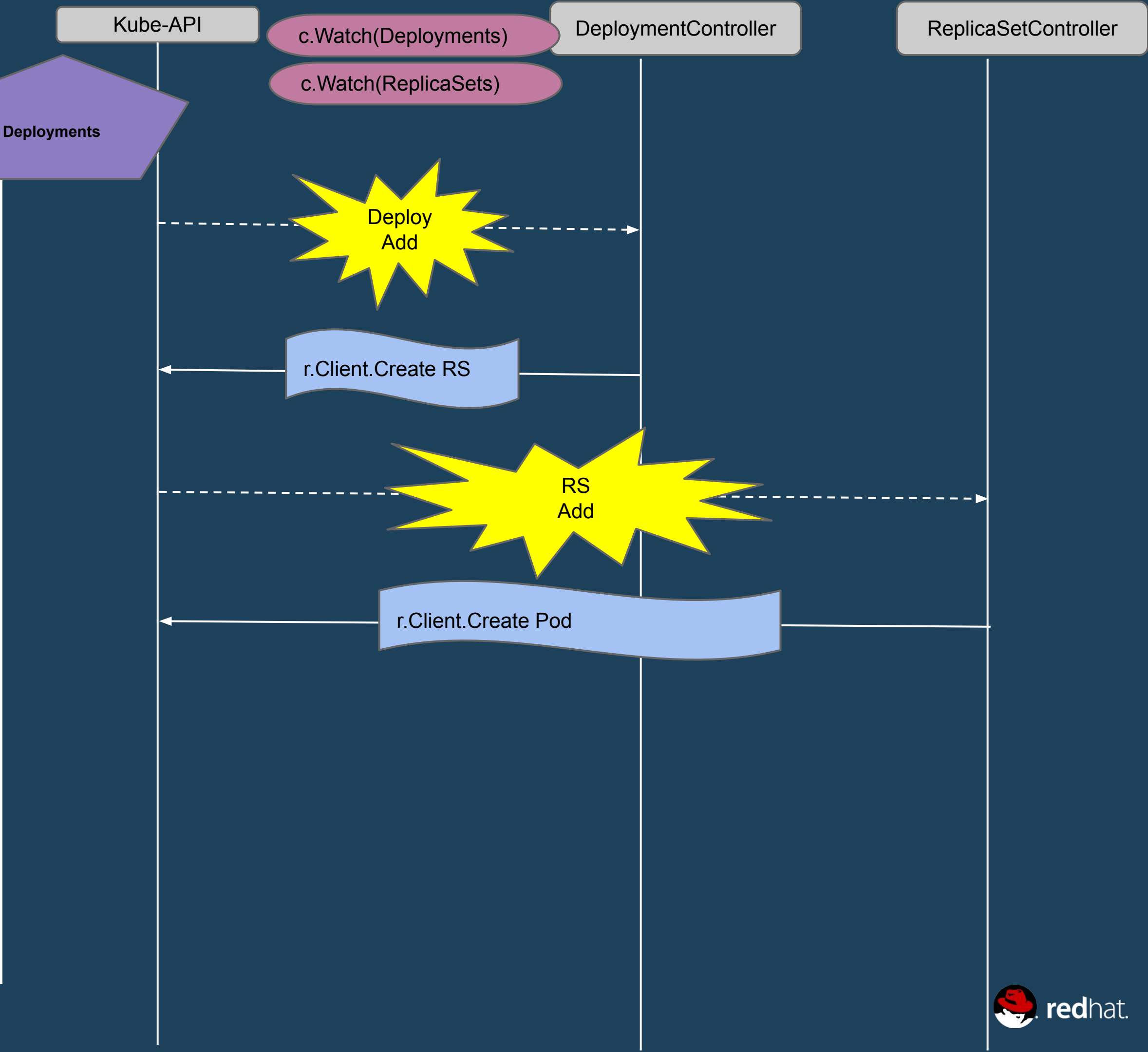
Label: app=nginx

Label: pod-template-hash=2819995210

DeploymentControll



```
func NewDeploymentController(dInformer appsinformers.DeploymentInformer, rsInformer appsinformers.ReplicaSetInformer, podInfor
        eventBroadcaster := record.NewBroadcaster()
        eventBroadcaster.StartLogging(klog.Infof)
        eventBroadcaster.StartRecordingToSink(&v1core.EventSinkImpl{Interface: client.CoreV1().Events("")})

        if client != nil && client.CoreV1().RESTClient().GetRateLimiter() != nil {
                if err := metrics.RegisterMetricAndTrackRateLimiterUsage("deployment_controller", client.CoreV1().RESTClient().
                        return nil, err
                }
        }
        dc := &DeploymentController{
                client:        client,
                eventRecorder: eventBroadcaster.NewRecorder(scheme.Scheme, v1.EventSource{Component: "deployment-controller"}),
                queue:         workqueue.NewNamedRateLimitingQueue(workqueue.DefaultControllerRateLimiter(), "deployment"),
        }
        dc.rsControl = controller.RealRSControl{
                KubeClient: client,
                Recorder:   dc.eventRecorder,
        }

        dInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
                AddFunc:    dc.addDeployment,
                UpdateFunc: dc.updateDeployment,
                // This will enter the sync loop and no-op, because the deployment has been deleted from the store.
                DeleteFunc: dc.deleteDeployment,
        })
        rsInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
                AddFunc:    dc.addReplicaSet,
                UpdateFunc: dc.updateReplicaSet,
                DeleteFunc: dc.deleteReplicaSet,
        })
        podInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
                DeleteFunc: dc.deletePod,
        })
```

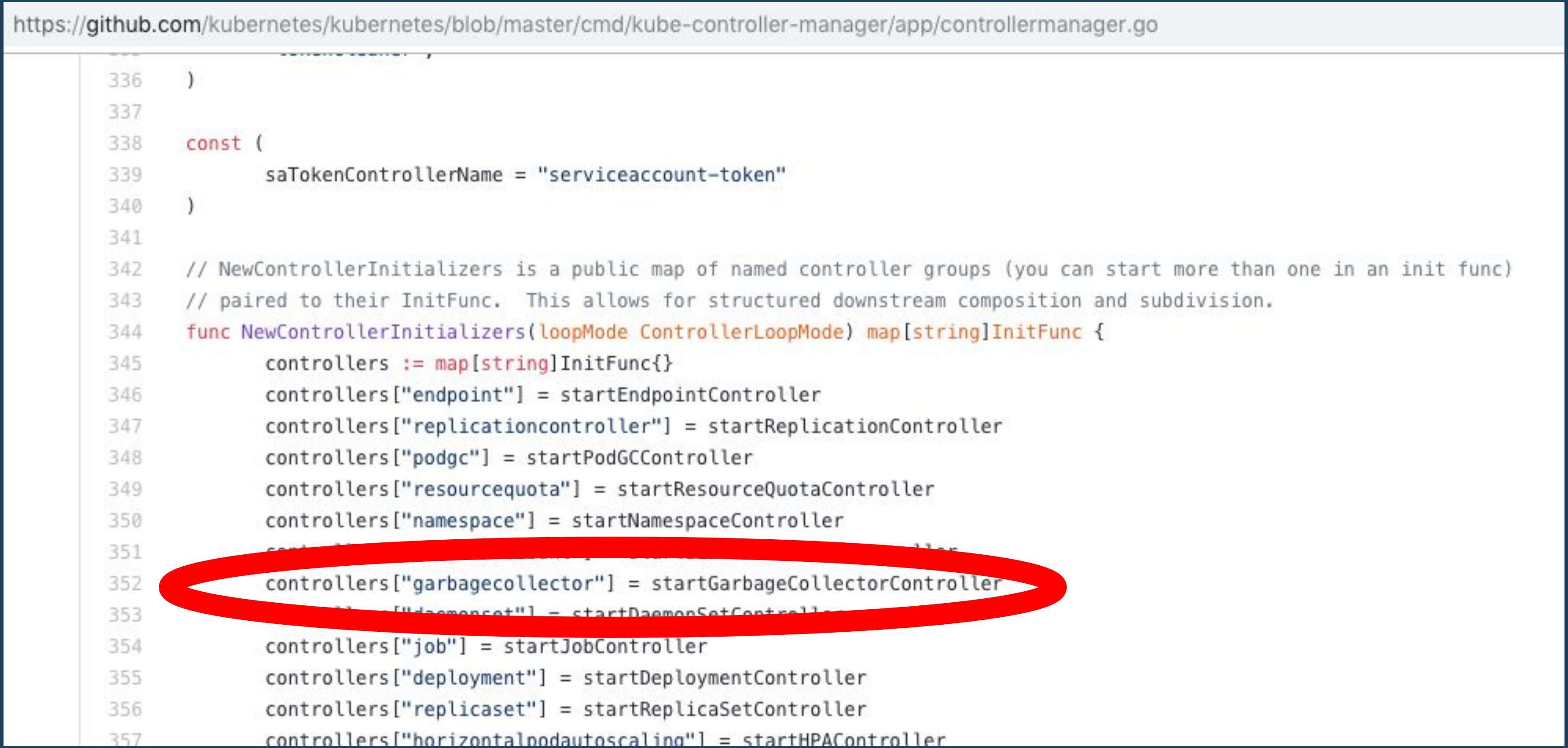


Let’s Identify Primary/Secondary Resources for Existing Kubernetes Controllers!

Garbage Collection (GC)

Garbage Collection assists in deleting objects that have an owner that **no longer** exists.

controllermanager.go



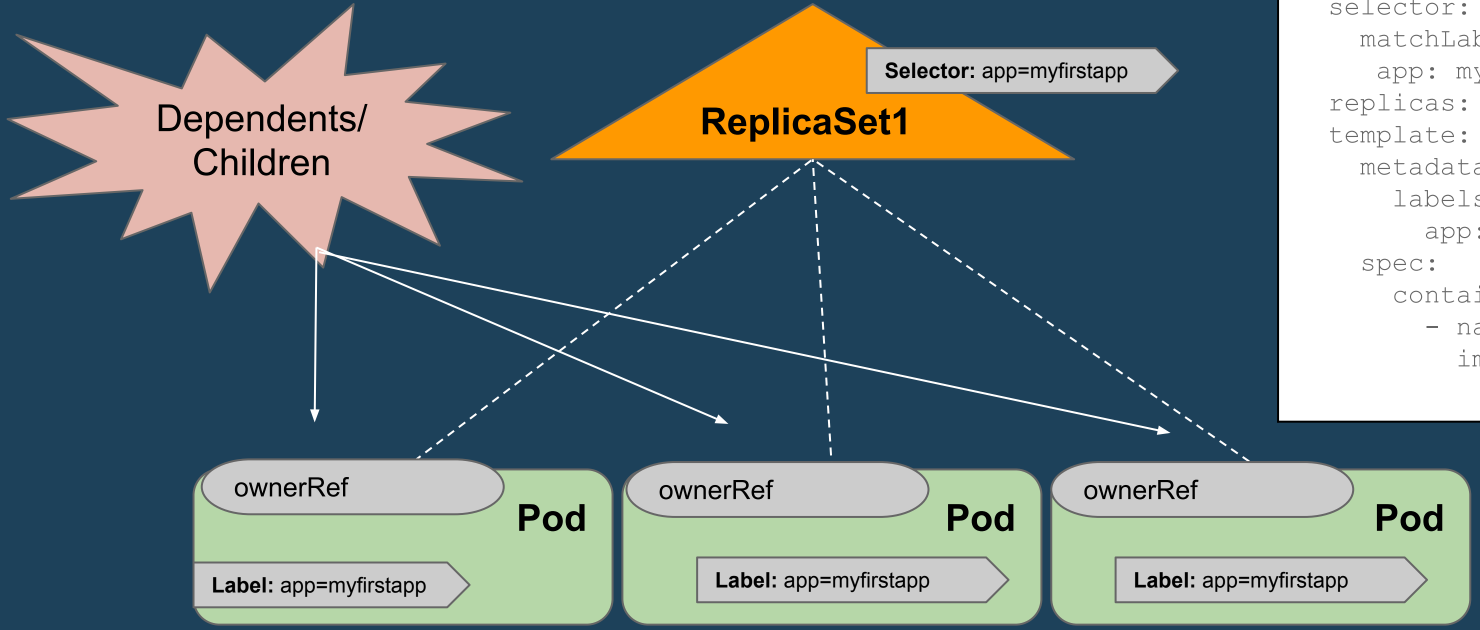
https://github.com/kubernetes/kubernetes/blob/master/cmd/kube-controller-manager/app/controllermanager.go

```go
)

const (
	saTokenControllerName = "serviceaccount-token"
)

// NewControllerInitializers is a public map of named controller groups (you can start more than one in an init func)
// paired to their InitFunc.  This allows for structured downstream composition and subdivision.
func NewControllerInitializers(loopMode ControllerLoopMode) map[string]InitFunc {
	controllers := map[string]InitFunc{}
	controllers["endpoint"] = startEndpointController
	controllers["replicationcontroller"] = startReplicationController
	controllers["podgc"] = startPodGCController
	controllers["resourcequota"] = startResourceQuotaController
	controllers["namespace"] = startNamespaceController
	[illegible]
	controllers["garbagecollector"] = startGarbageCollectorController
	controllers["daemonset"] = startDaemonSetController
	controllers["job"] = startJobController
	controllers["deployment"] = startDeploymentController
	controllers["replicaset"] = startReplicaSetController
	controllers["horizontalpodautoscaling"] = startHPAController
```

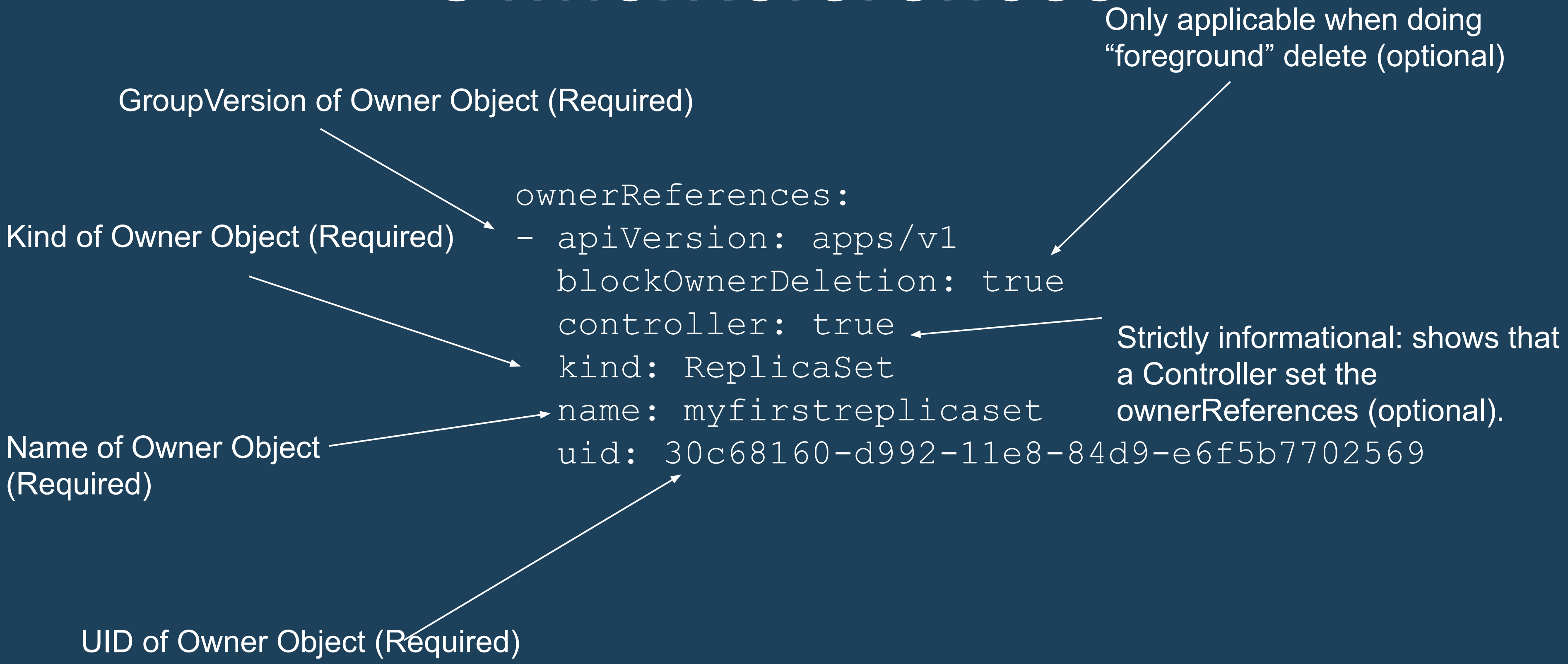
OwnerReferences

```
kubectl create -f  myfirstreplicaset.yaml
```

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```


Dependents/ Children
ReplicaSet1
Selector: app=myfirstapp
ownerRef
Pod
Label: app=myfirstapp
ownerRef
Pod
Label: app=myfirstapp
ownerRef
Pod
Label: app=myfirstapp

OwnerReferences



Only applicable when doing “foreground” delete (optional)

GroupVersion of Owner Object (Required)

Kind of Owner Object (Required)

```
ownerReferences:
- apiVersion: apps/v1
  blockOwnerDeletion: true
  controller: true
  kind: ReplicaSet
  name: myfirstreplicaset
  uid: 30c68160-d992-11e8-84d9-e6f5b7702569
```

Strictly informational: shows that a Controller set the ownerReferences (optional).

Name of Owner Object (Required)

UID of Owner Object (Required)

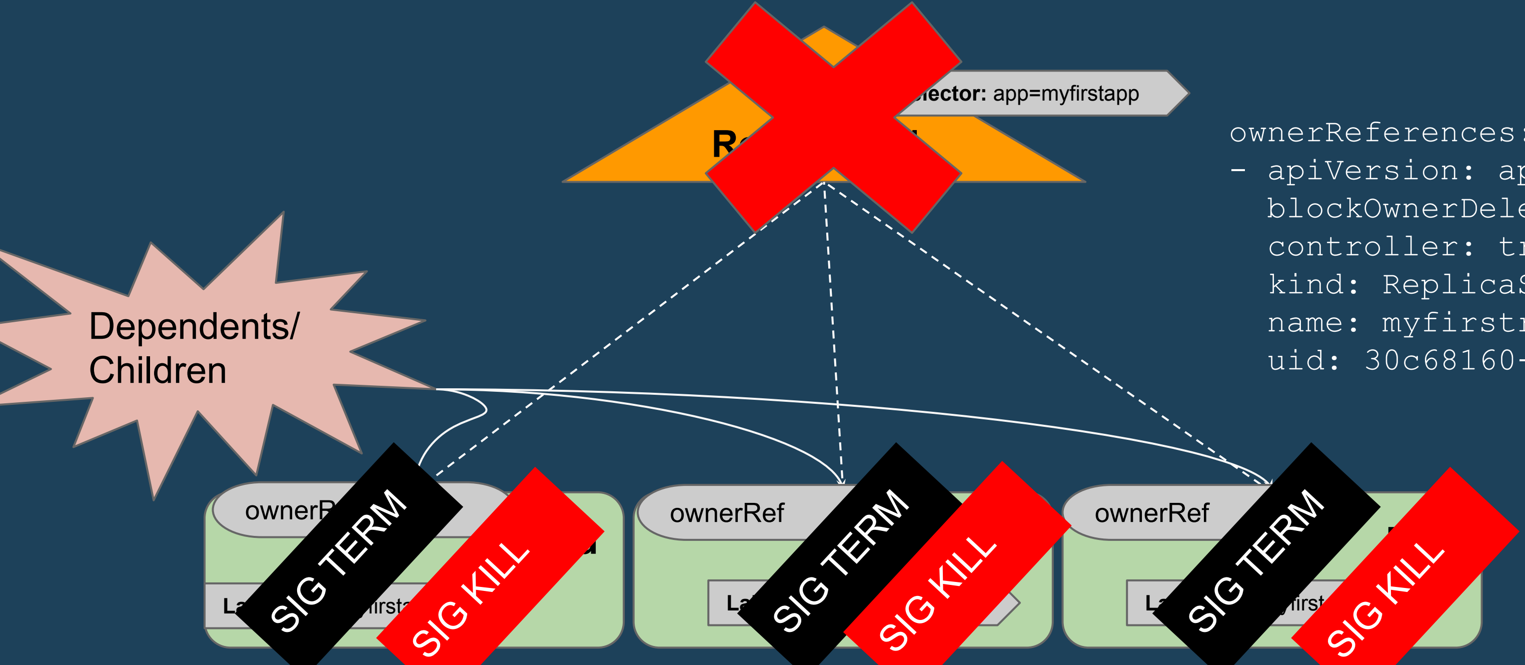
*querying API for UID not currently supported.

Background Delete

```
oc delete -f myfirstreplicaset
```

```
curl -X DELETE localhost:8080/apis/apps/v1/namespaces/default/replicasets/my-repset \
-d '{"kind":"DeleteOptions","apiVersion":"v1","propagationPolicy":"Background"}' \
-H "Content-Type: application/json"
```

30s..



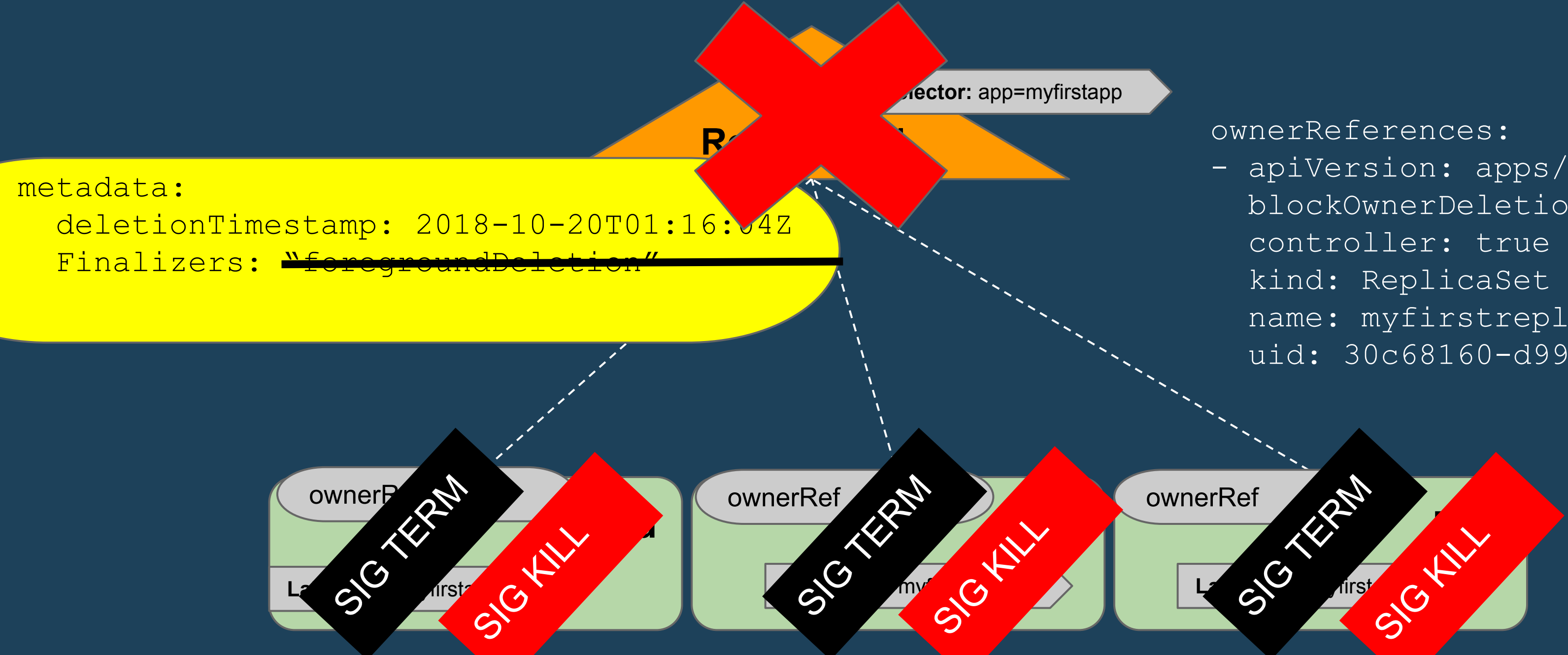
```
ownerReferences:
- apiVersion: apps/v1
  blockOwnerDeletion: true
  controller: true
  kind: ReplicaSet
  name: myfirstreplicaset
  uid: 30c68160-d992-11e8-84d9-e6f5b
```

Foreground Delete

```
oc delete -f myfirstreplicaset
```

```
curl -X DELETE localhost:8080/apis/apps/v1/namespaces/default/replicasets/my-repset \
-d '{"kind":"DeleteOptions","apiVersion":"v1","propagationPolicy":"Foreground"}' \
-H "Content-Type: application/json"
```

30s..

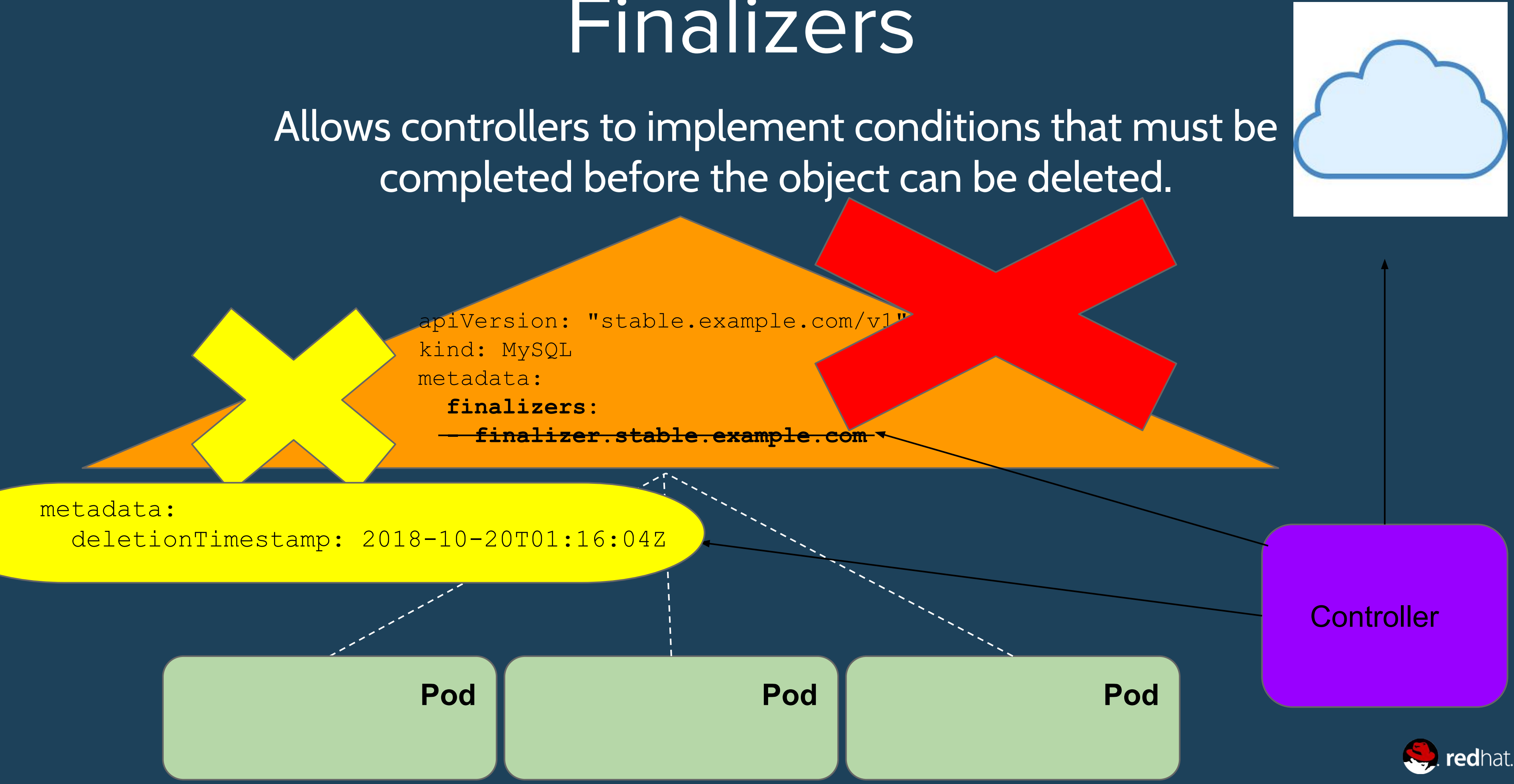


```
metadata:
  deletionTimestamp: 2018-10-20T01:16:04Z
  Finalizers: ~~“foregroundDeletion”~~
```

```
ownerReferences:
- apiVersion: apps/v1
  blockOwnerDeletion: true
  controller: true
  kind: ReplicaSet
  name: myfirstreplicaset
  uid: 30c68160-d992-11e8-84d9-e6f5b
```

Finalizers

Allows controllers to implement conditions that must be completed before the object can be deleted.



```
apiVersion: "stable.example.com/v1"
kind: MySQL
metadata:
  finalizers:
  ~~- finalizer.stable.example.com~~
```

```
metadata:
  deletionTimestamp: 2018-10-20T01:16:04Z
```