ReplicaSets & Deployments

(The Underrated, OG Operators)

Why do we care about ReplicaSets (formerly ReplicaControllers)?

Redundancy

Multiple running instances means failure can be tolerated.

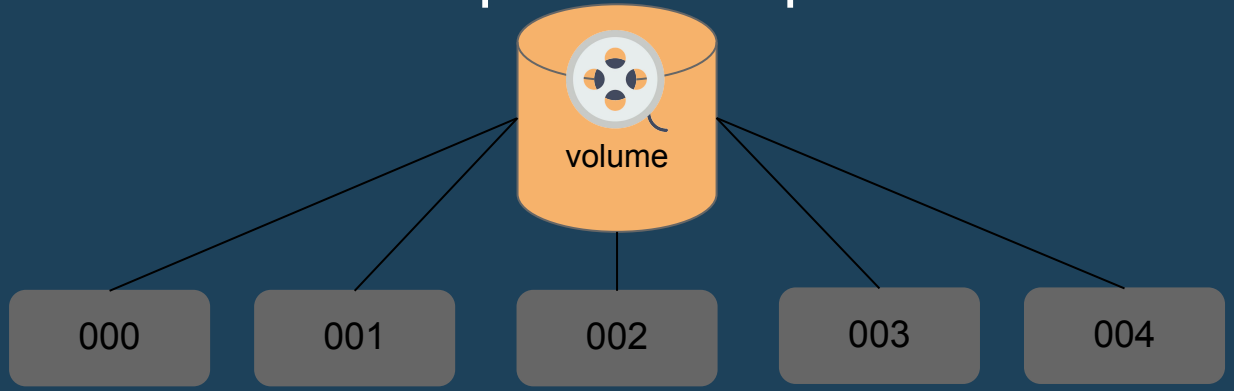
Scale

Multiple running instances mean more requests can be handled.

Red Hat

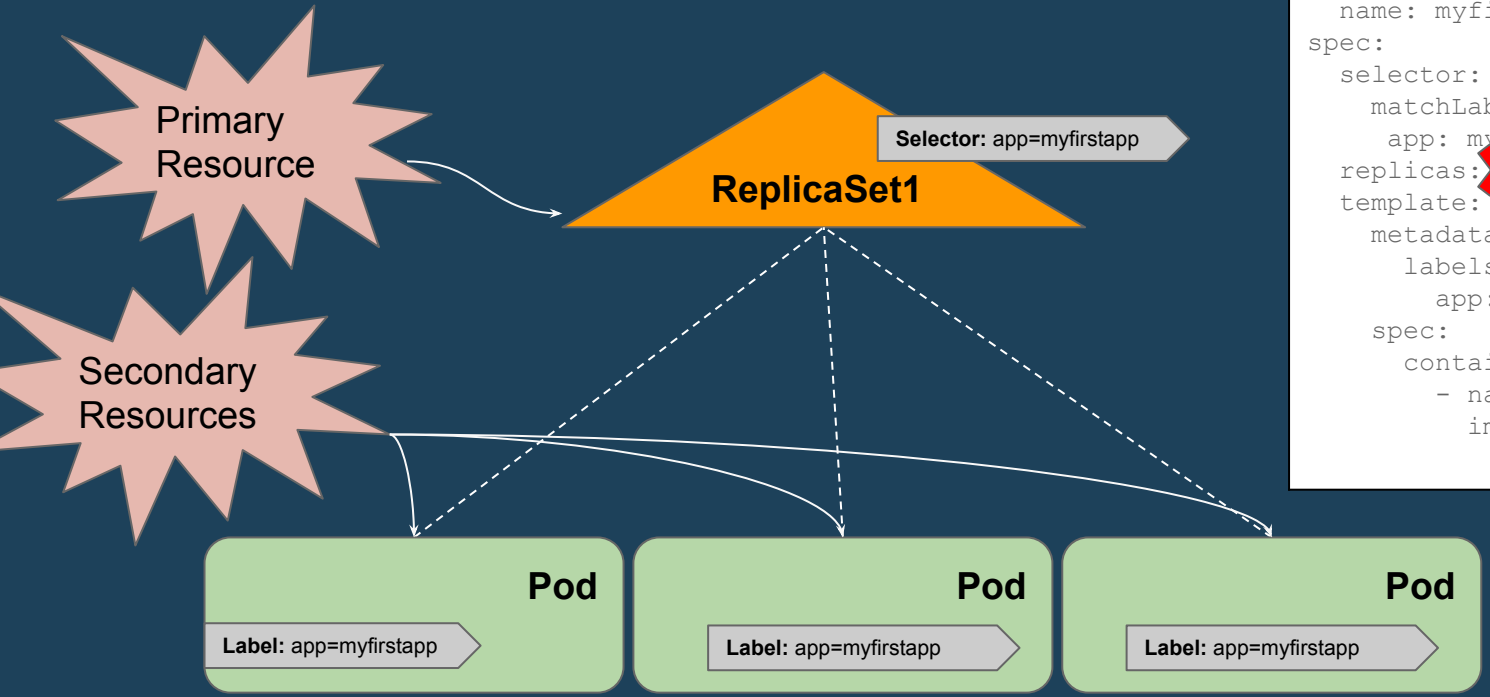
Sharding

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



ReplicaSets in Action!

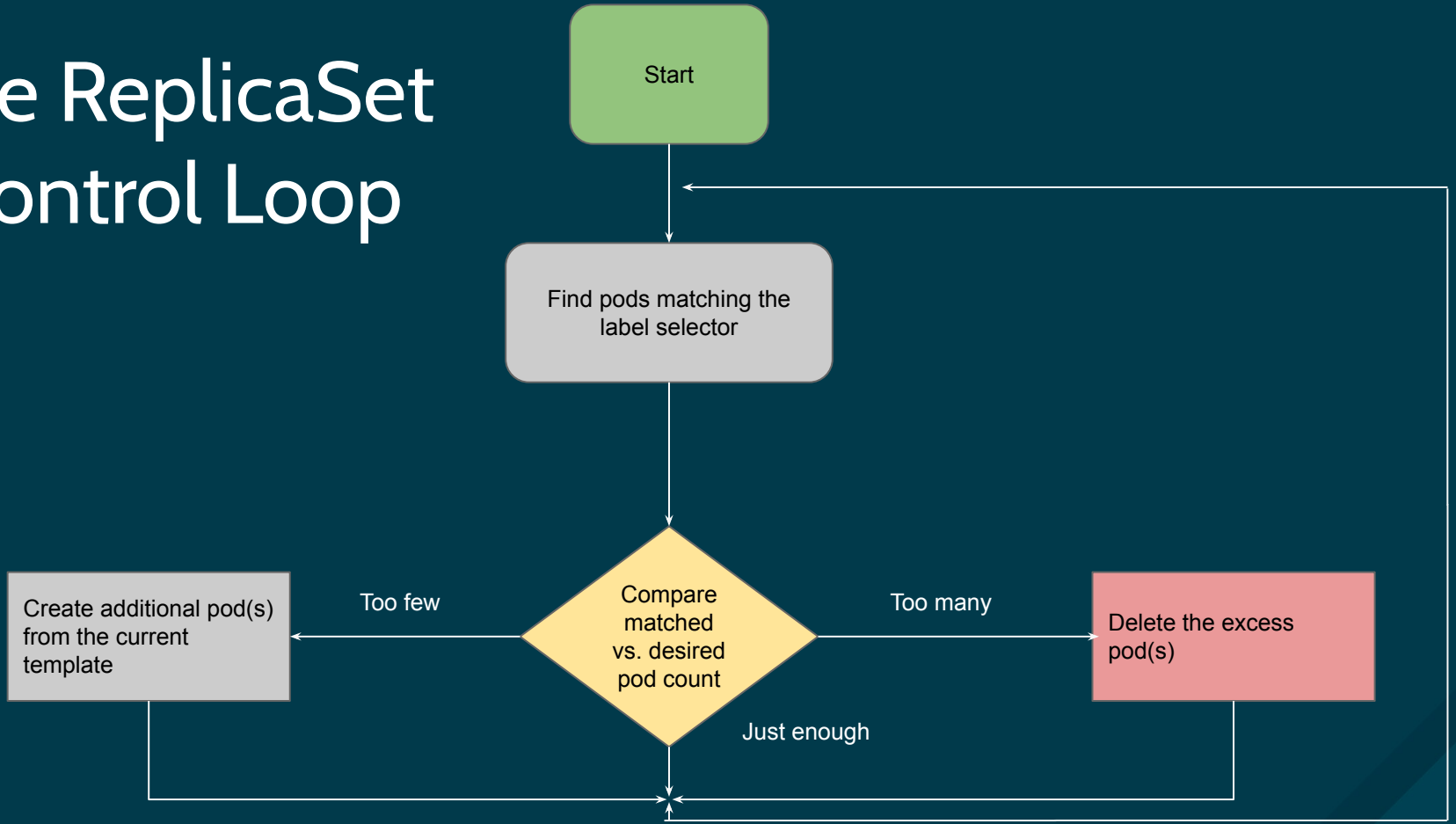
```
kubectl create -f  myfirstreplicaset.yaml

kubectl scale replicaset myfirstreplicaset --replicas=3
```



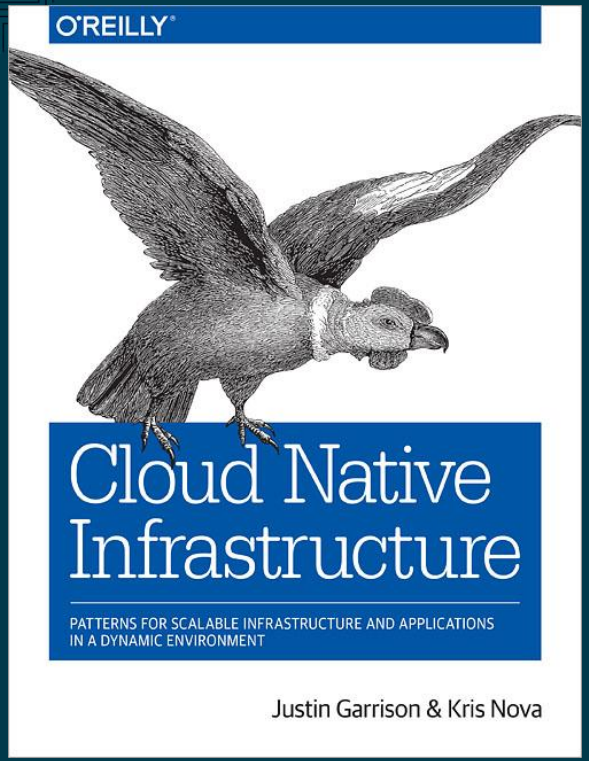

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```

Red Hat

The ReplicaSet Control Loop

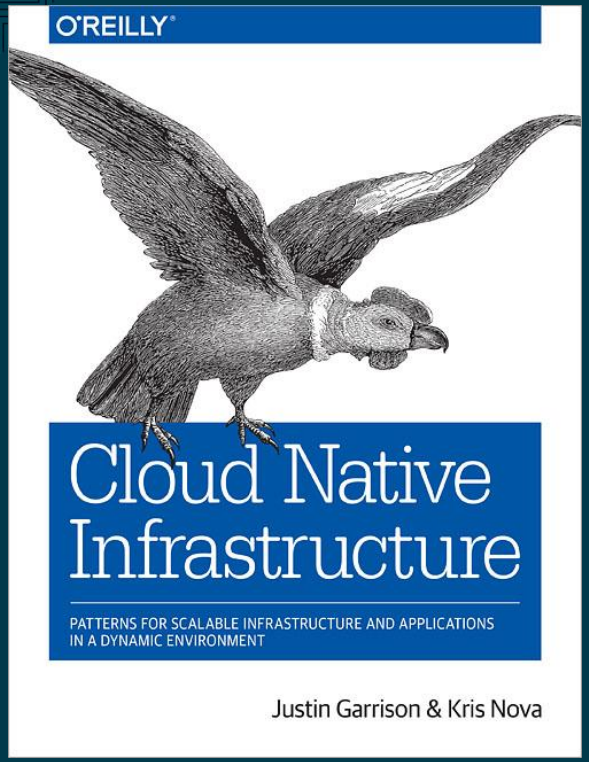


How do we accomplish this?



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.*



*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
```

0 < spec.replicas?

1 < spec.replicas?

2 < spec.replicas?

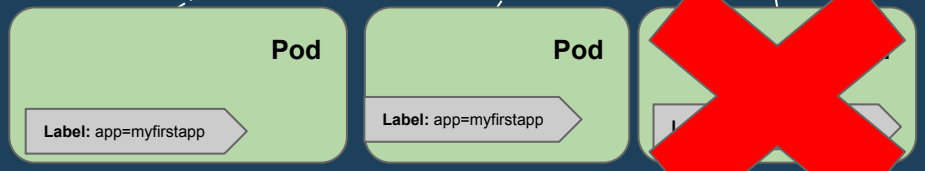
3 < spec.replicas?

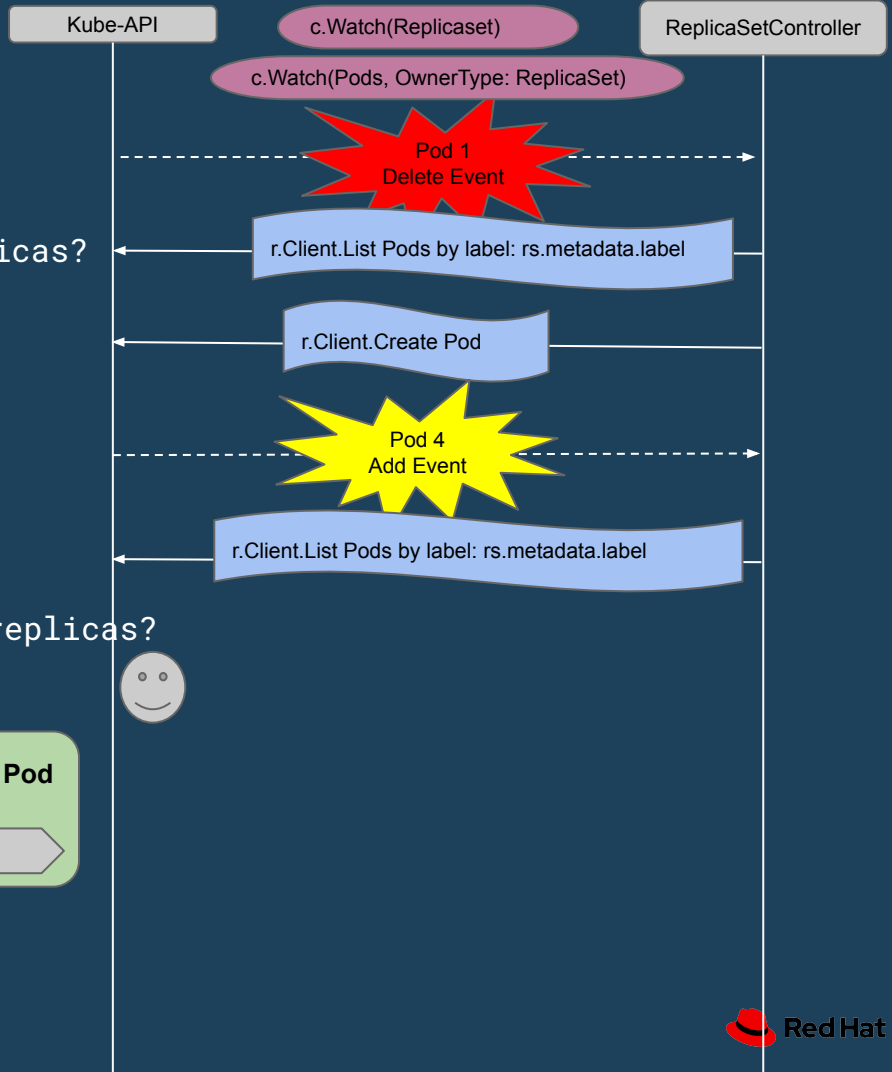
Red Hat

ReplicaSets in Action!

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

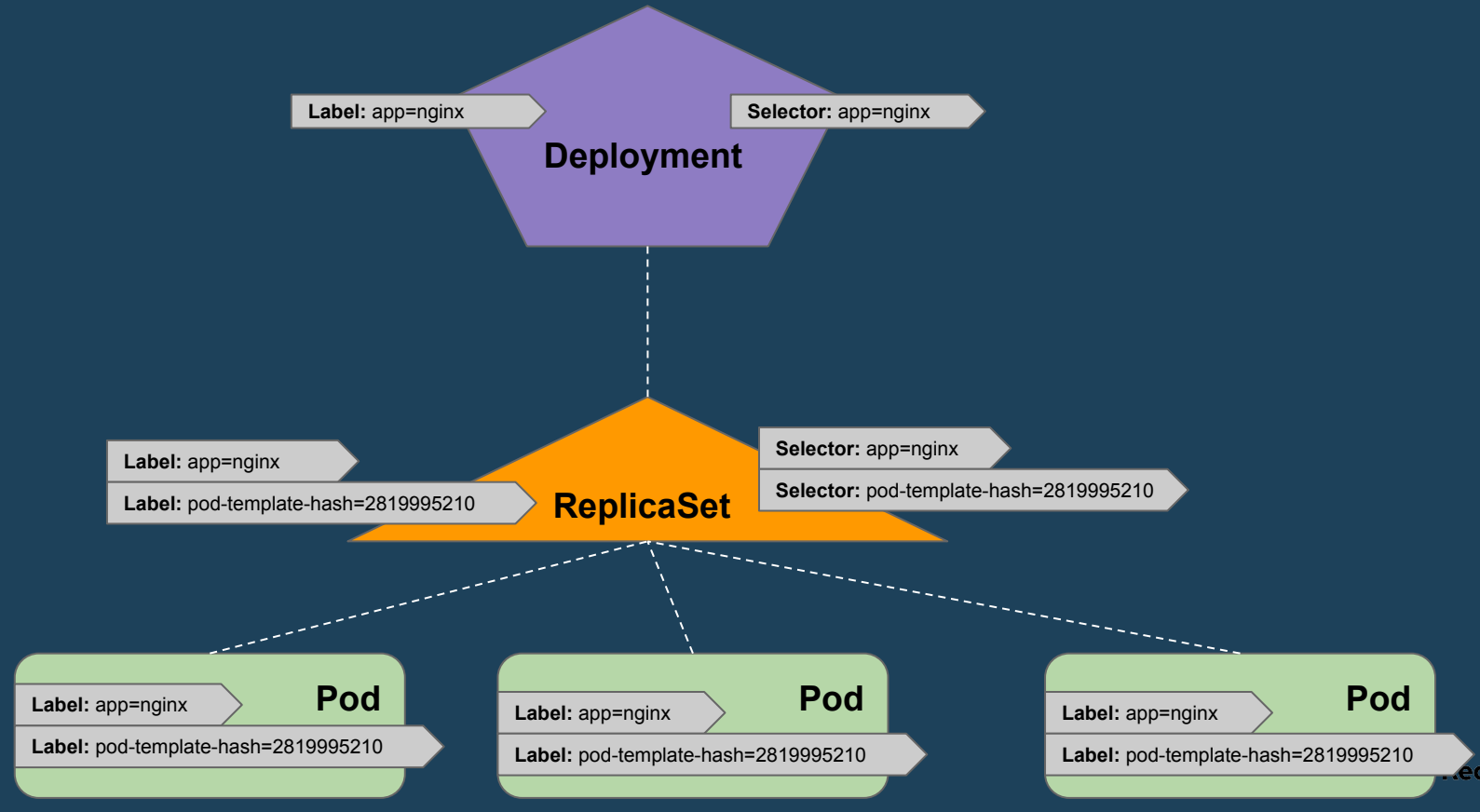
```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```

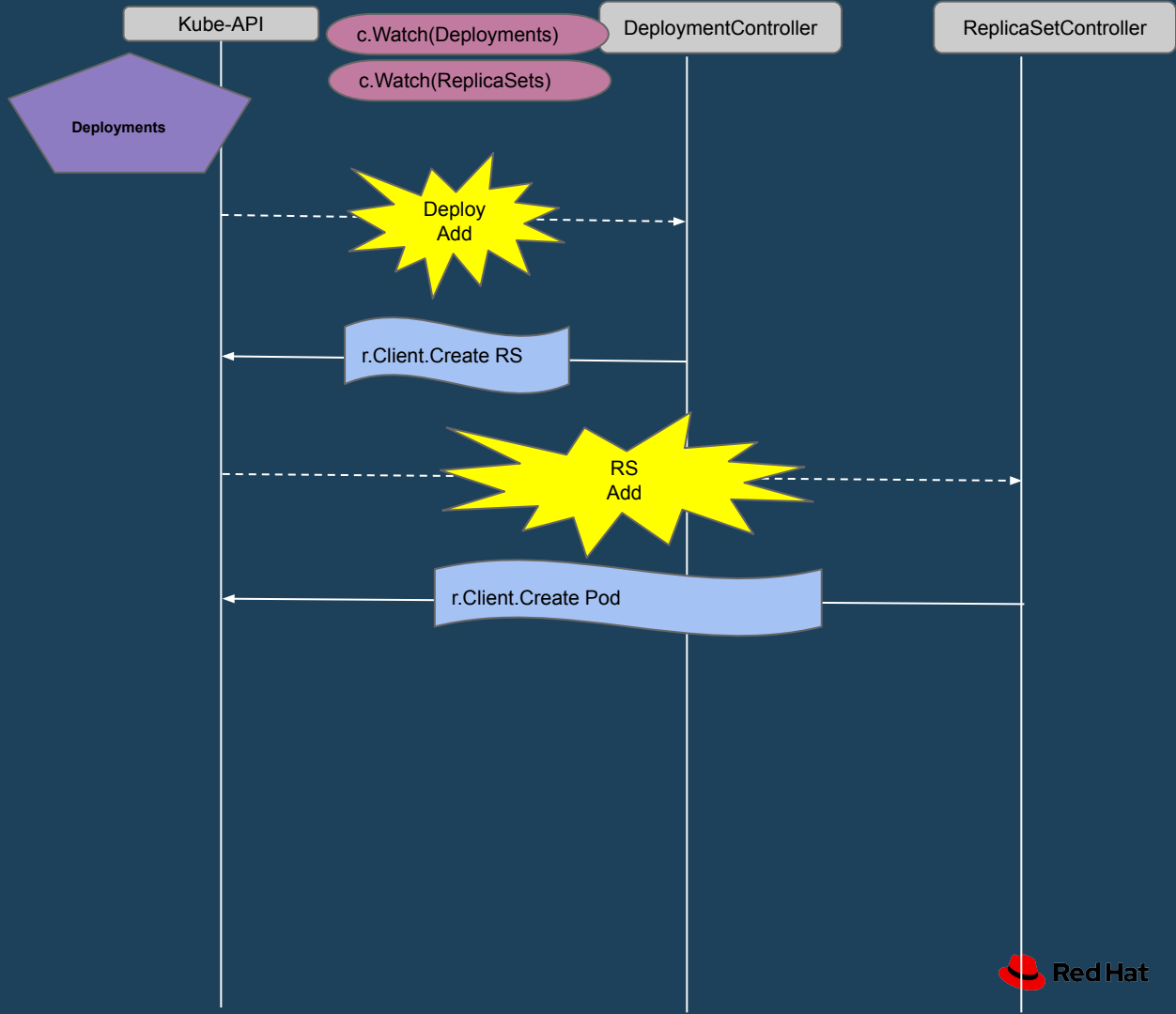




Red Hat

Deployments!



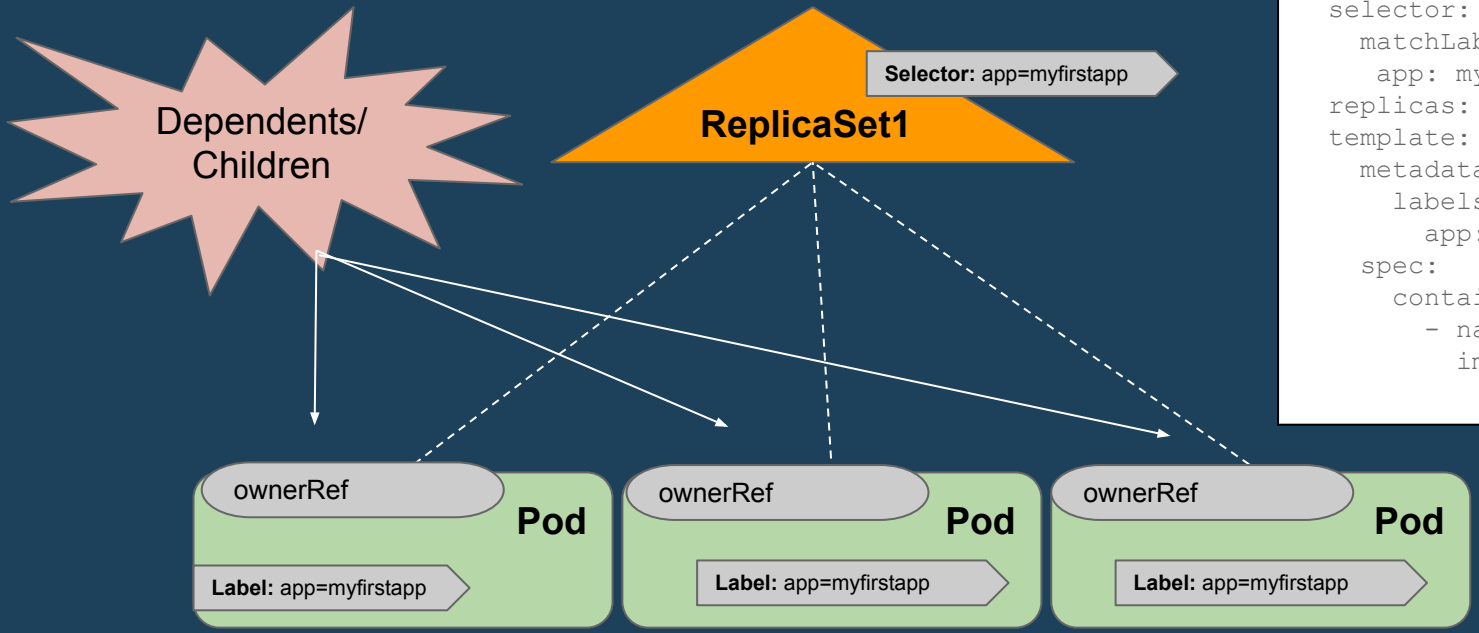

Kube-API
c.Watch(Deployments)
DeploymentController
ReplicaSetController
c.Watch(ReplicaSets)
Deployments
Deploy
Add
r.Client.Create RS
RS
Add
r.Client.Create Pod

Garbage Collection (GC)

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

OwnerReferences

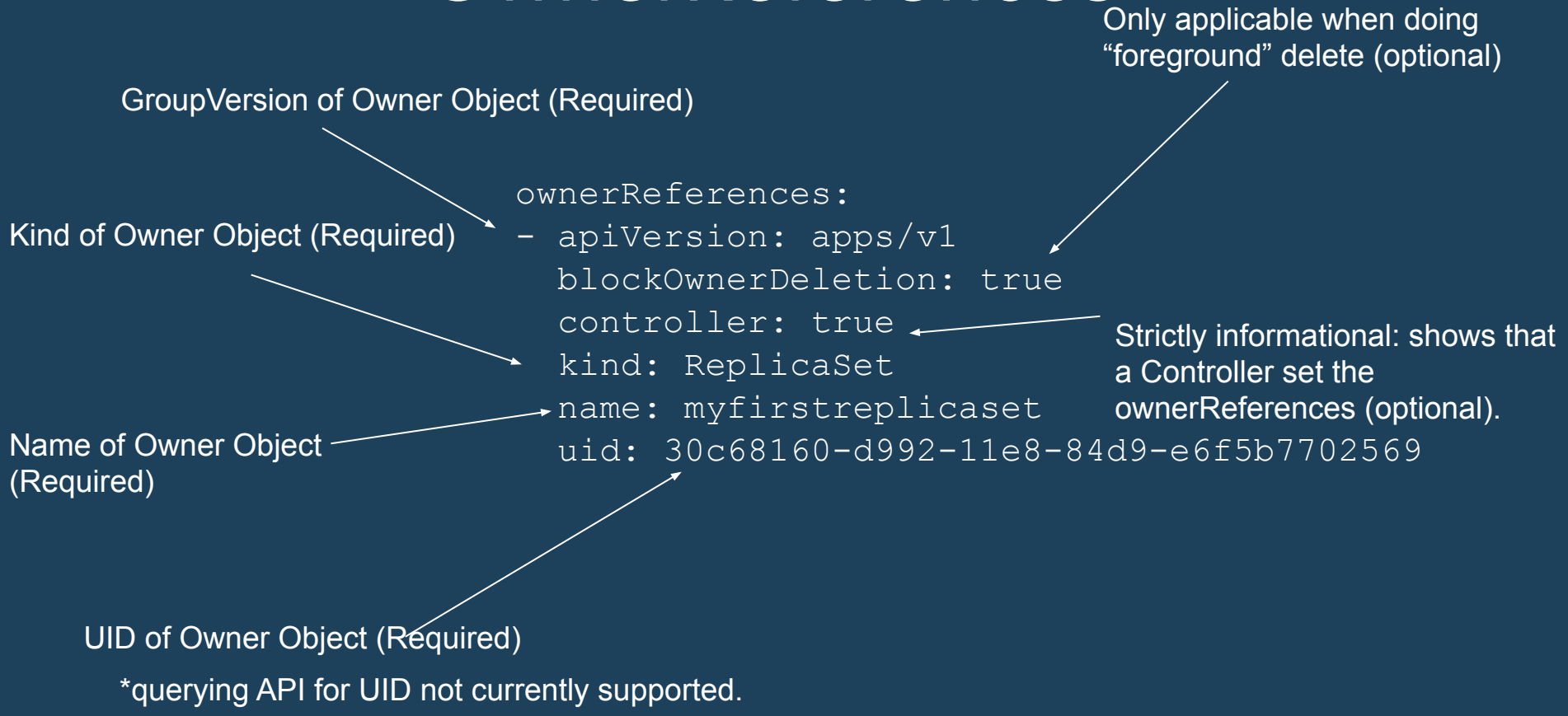
```
kubectl create -f  myfirstreplicaset.yaml
```




```
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
```

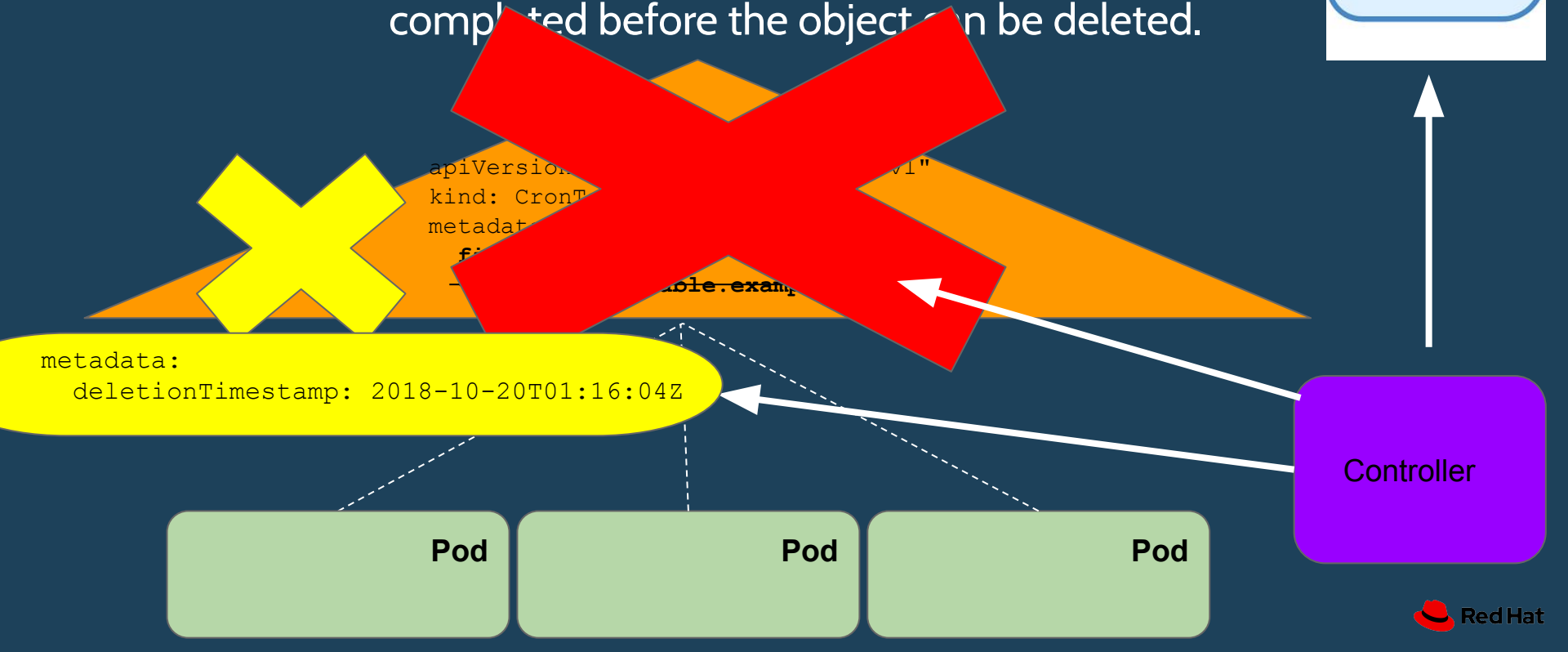
Red Hat

OwnerReferences



Finalizers

Allows controllers to implement conditions that must be comp[illegible]ted before the object [illegible]n be deleted.



Red Hat