

ReplicaSets & Deployments

(The Underrated, OG Operators)

Why do we care about ReplicaSets
(formerly ReplicaControllers)?

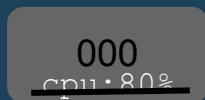
Redundancy

Multiple running instances means failure can be tolerated.

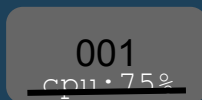


Scale

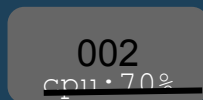
Multiple running instances mean more requests can be handled.



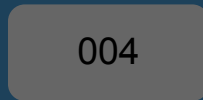
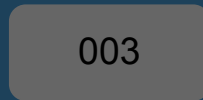
cpu:20%



cpu:25%

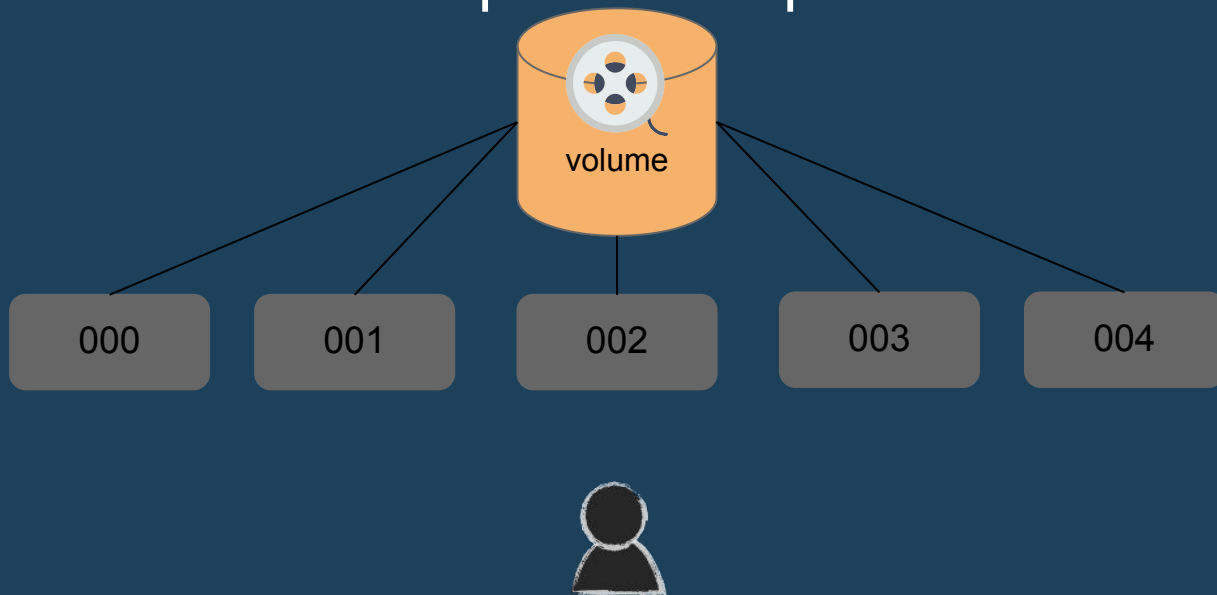


cpu:23%



Sharding

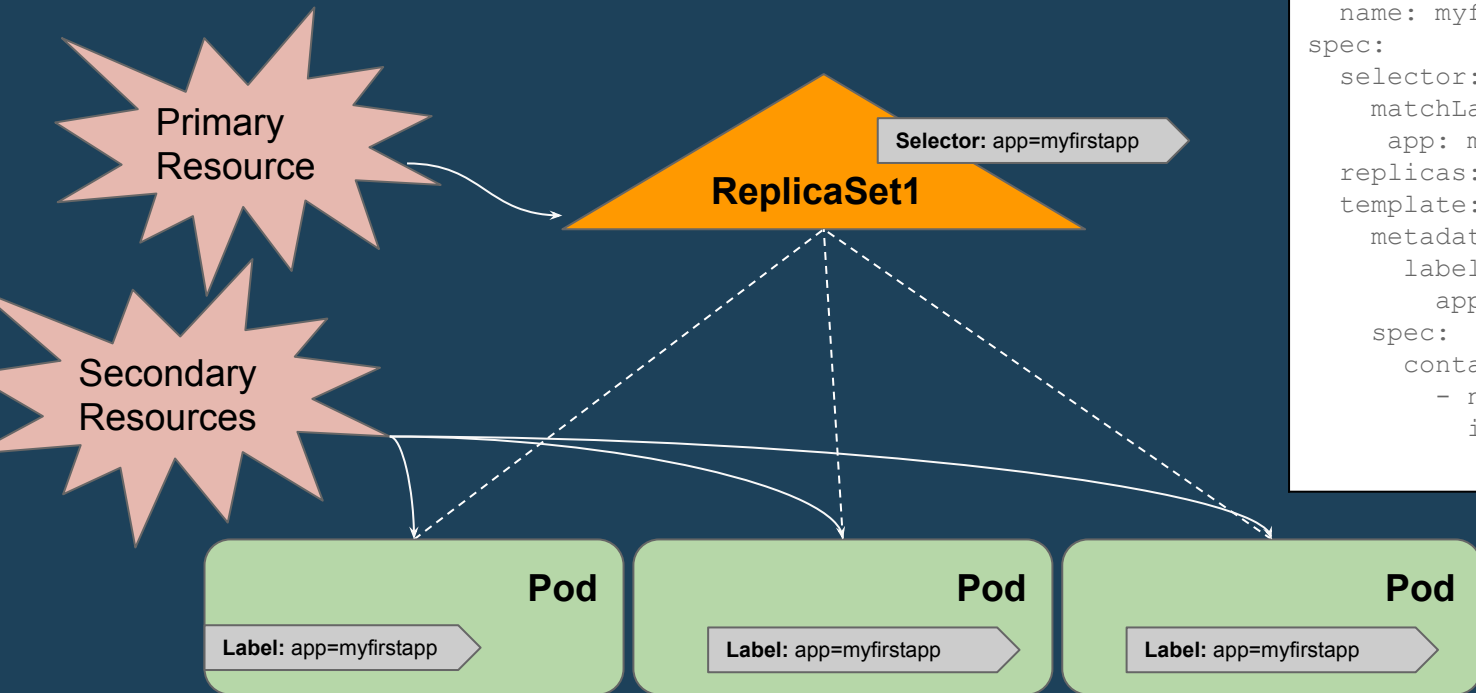
Multiple running instances can handle different parts of a computation in parallel.



ReplicaSets in Action!

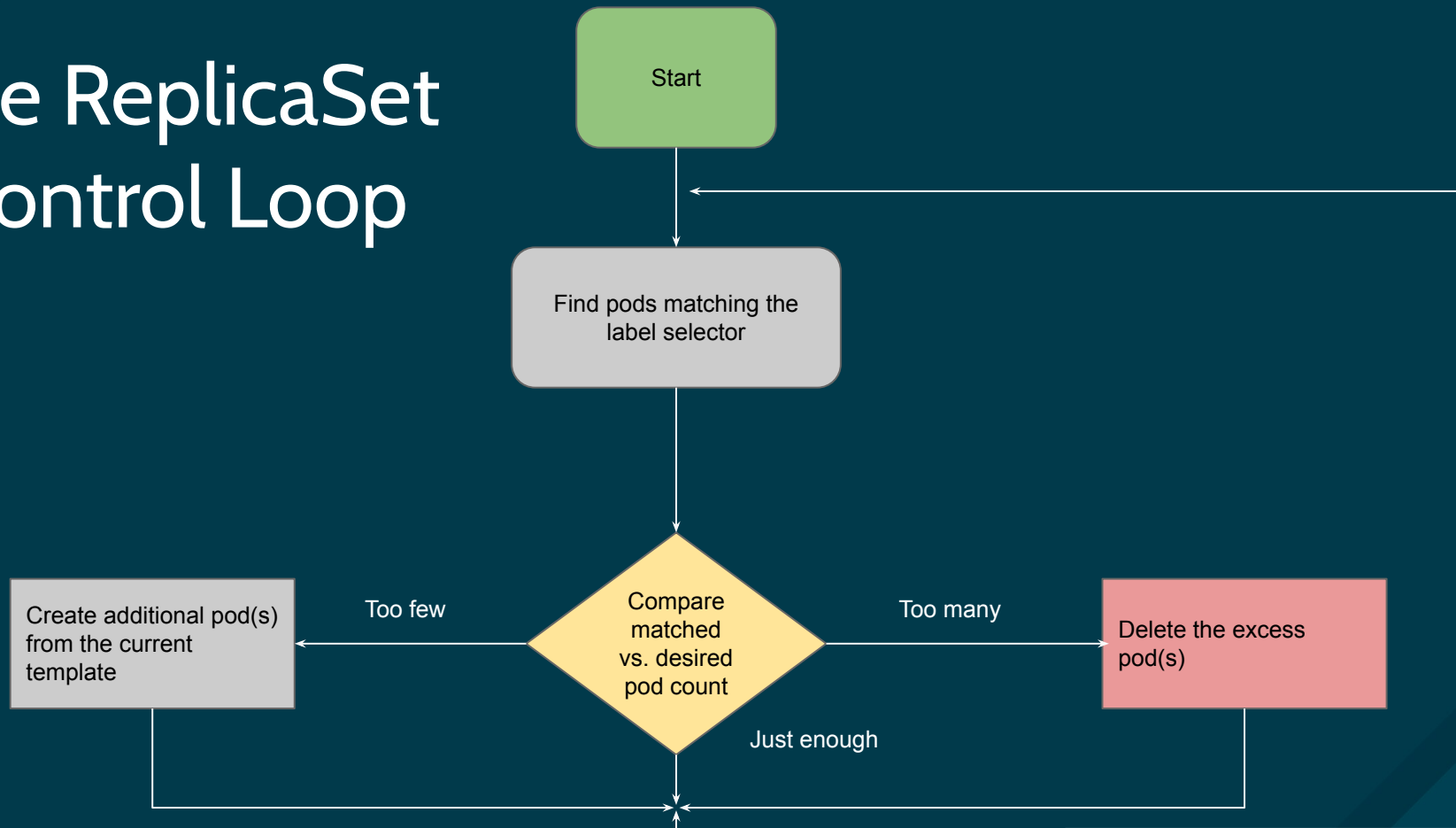
```
kubectl create -f myfirstreplicaset.yaml
```

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

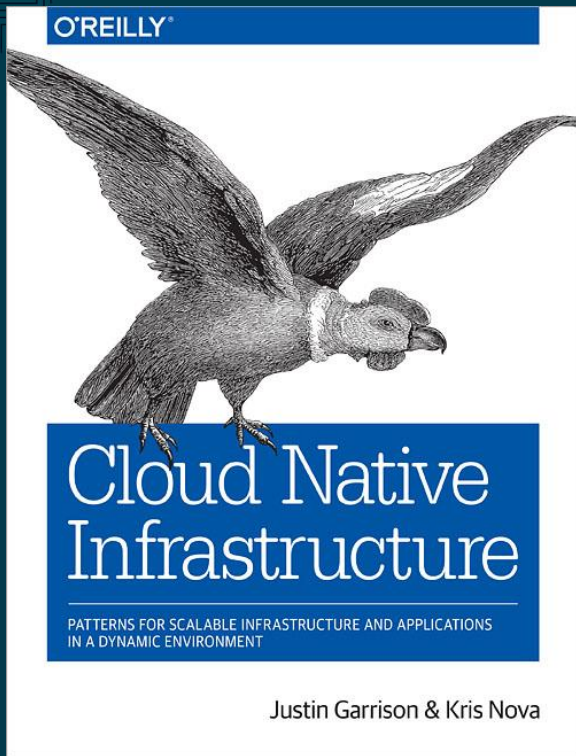


```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
      app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```

The ReplicaSet Control Loop



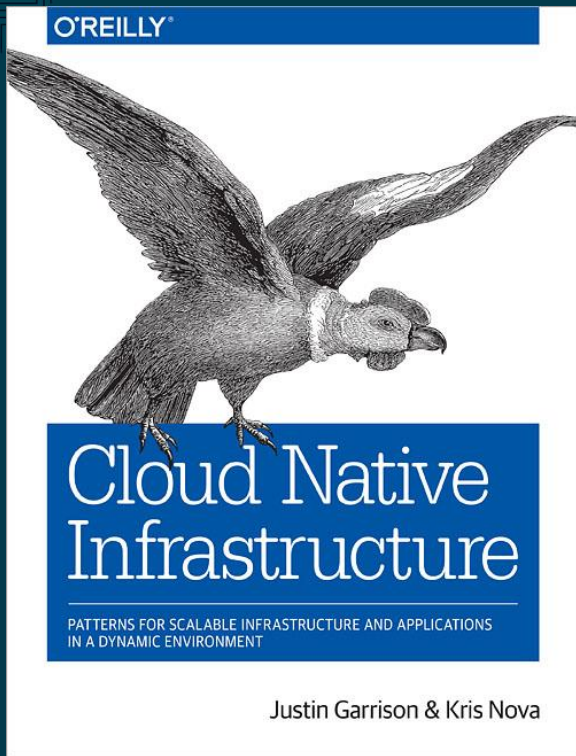
How do we accomplish this?



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?

Selector: app=myfirstapp

ReplicaSet1

2 < spec.replicas?

3 < spec.replicas?

Pod

Pod

Pod

Label:
app=myfirstapp

Label: app=myfirstapp

Label:
app=myfirstapp

Kube-API

c.Watch(ReplicaSet)

ReplicaSetController

c.Watch(Pods, OwnerType: ReplicaSet)

ReplicaSet
Add Event

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

r.Client.Create Pod 1

Pod 1
Add Event

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

r.Client.Create Pod 2

Pod 2
Add Event

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

r.Client.Create Pod 3

Pod 3
Add Event

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



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
```

2 < spec.replicas?

Selector: app=myfirstapp

ReplicaSet1

3 < spec.replicas?

Pod

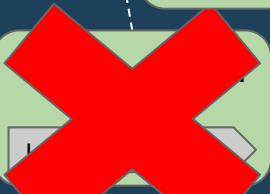
Label: app=myfirstapp

Pod

Label: app=myfirstapp

Pod

Label: app=myfirstapp



Kube-API

c.Watch(Replicaset)

ReplicaSetController

c.Watch(Pods, OwnerType: ReplicaSet)

Pod 1
Delete Event

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

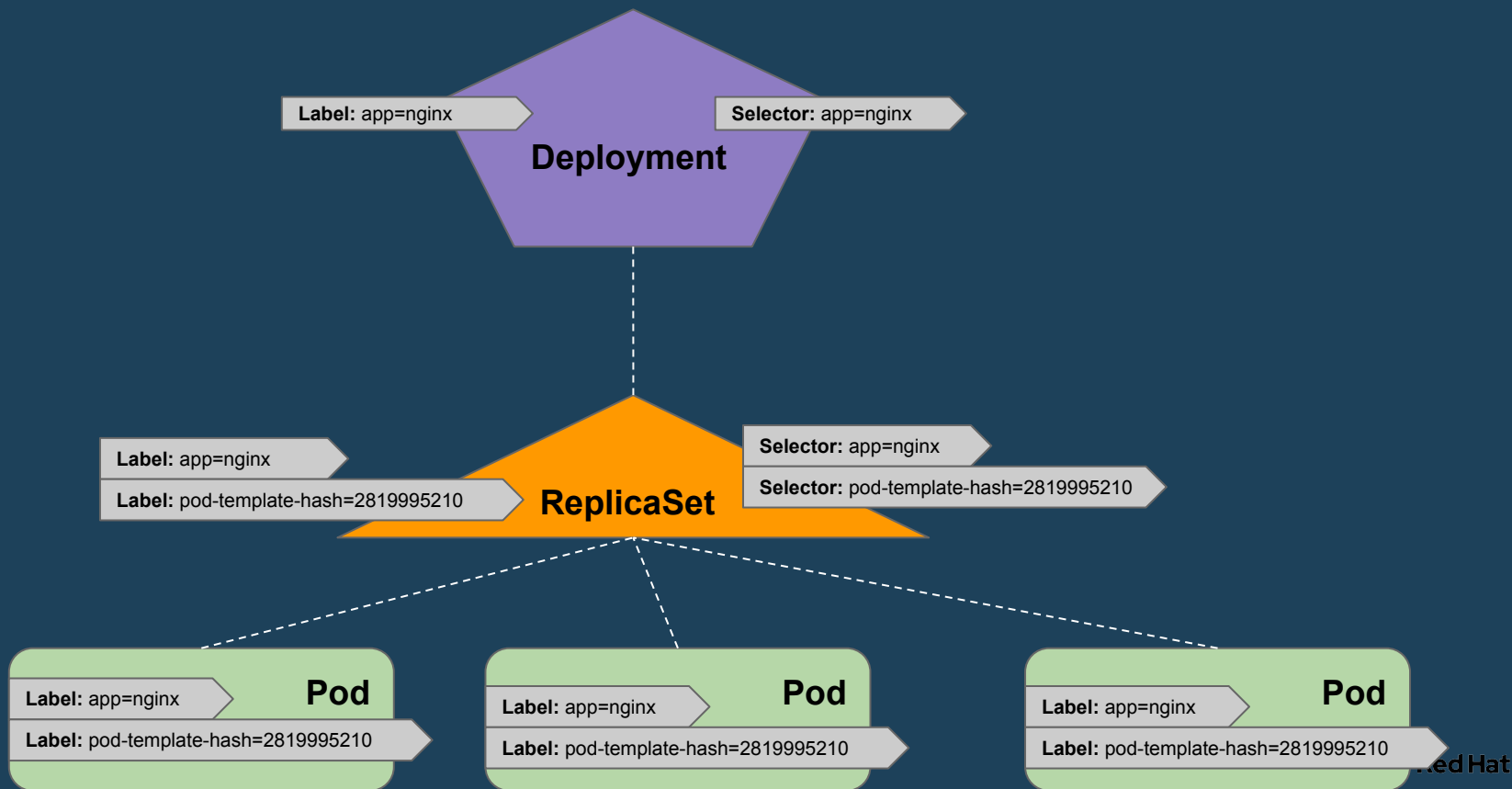
r.Client.Create Pod

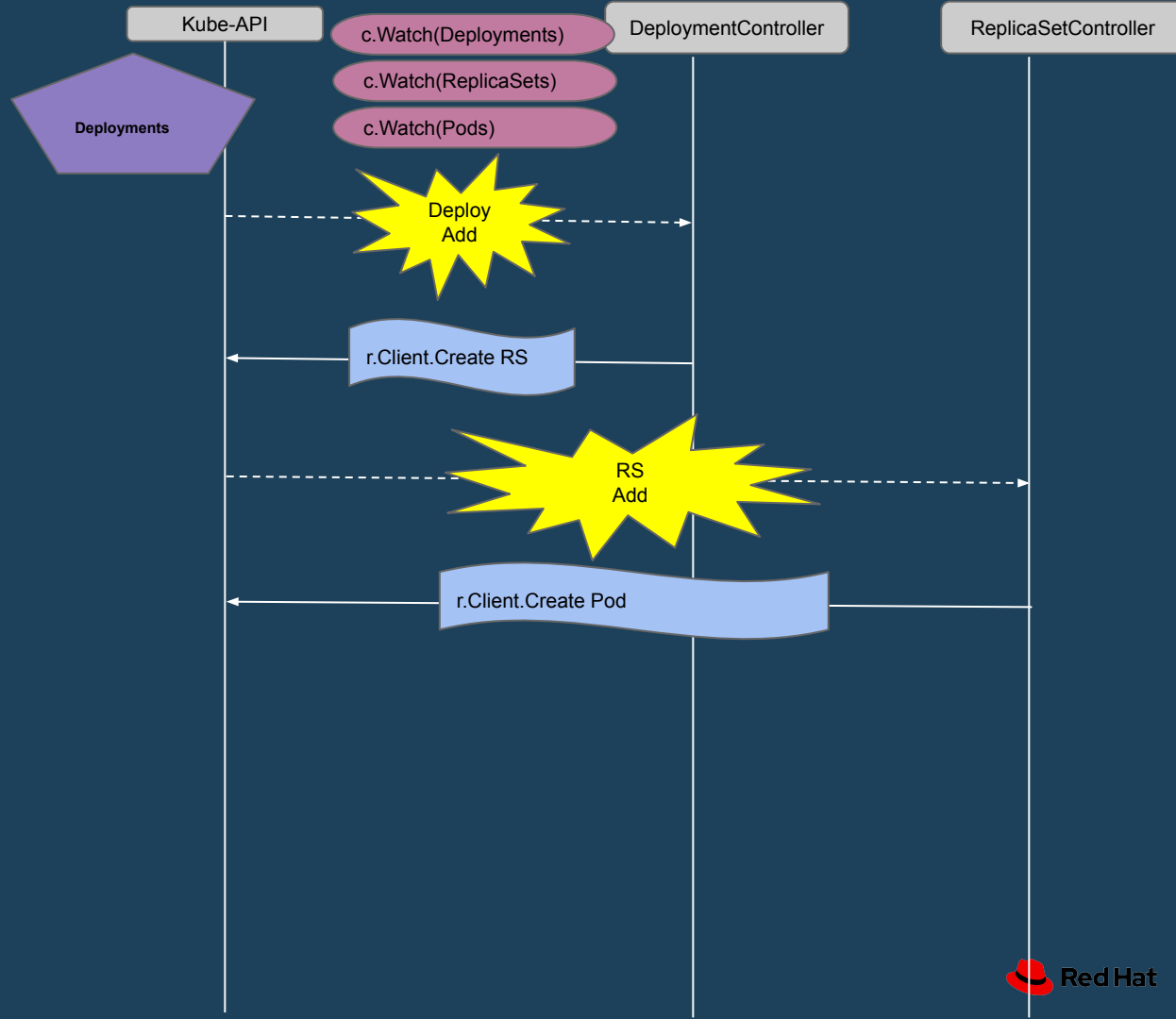
Pod 4
Add Event

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



Deployments!





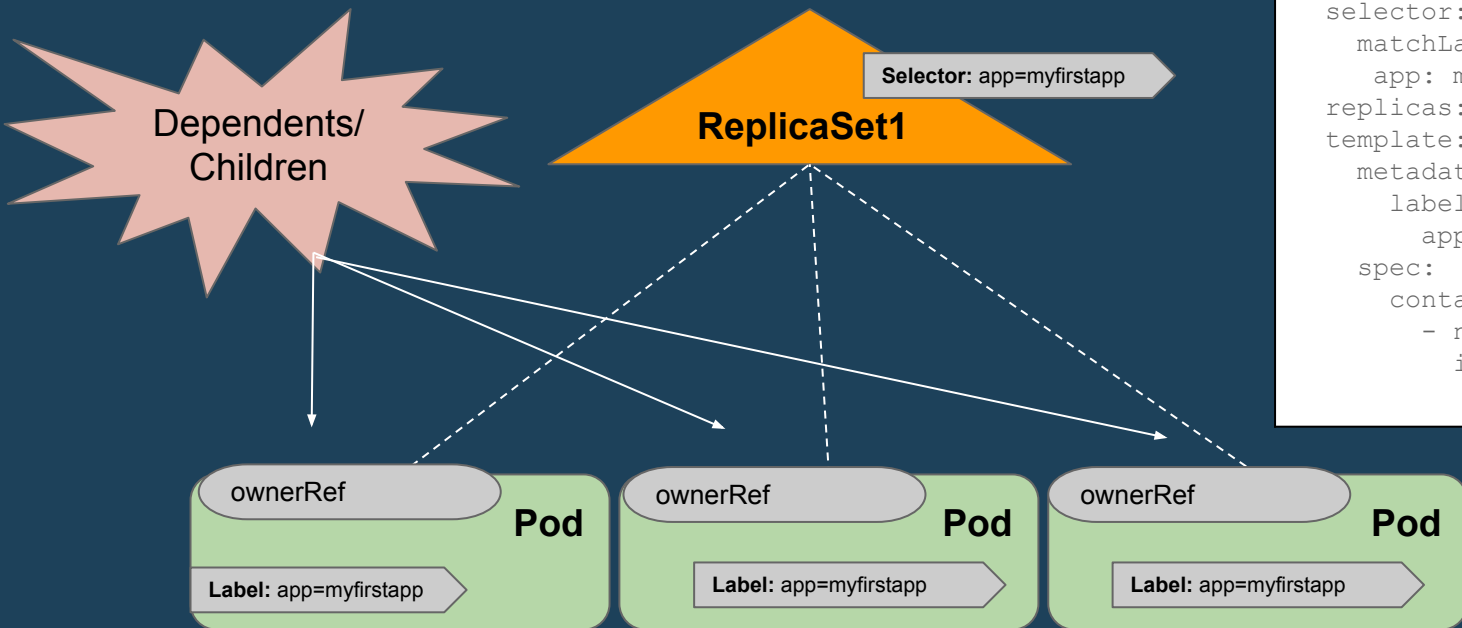
Garbage Collection (GC)

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
```



OwnerReferences

Only applicable when doing
“foreground” delete (optional)

GroupVersion of Owner Object (Required)

Kind of Owner Object (Required)

Name of Owner Object
(Required)

UID 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).

*querying API for UID not currently supported.

Finalizers

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



Controller

apiVersion: v1
kind: CronJob
metadata:
 name: example

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

Pod

Pod

Pod