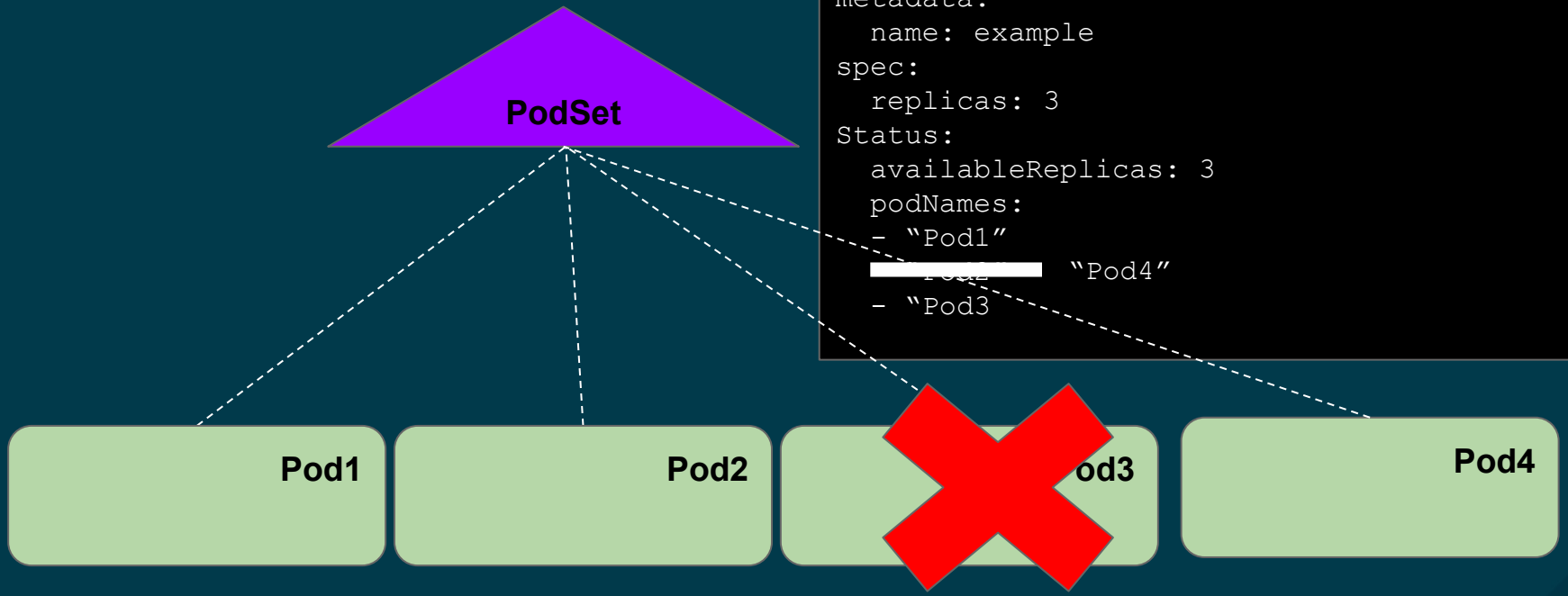
Pod Set Operator Challenge

Let’s Create a Simple Operator.

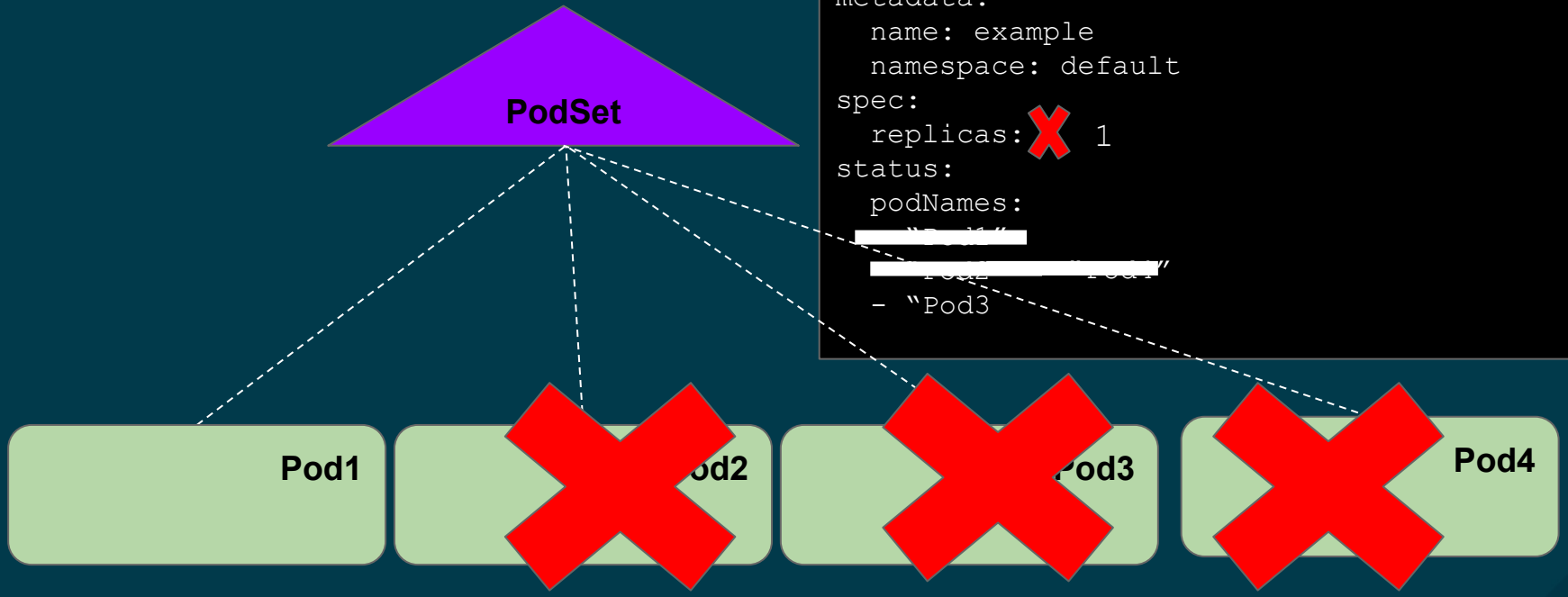
It’s Called a “Pod Set”.

A Simple Controller that Manages Pods.



A Pod Set Allows You to Scale Up/Down.

```
apiVersion: podset.redhat.com/v1alpha1
kind: PodSet
metadata:
  name: example
  namespace: default
spec:
  replicas: 1
status:
  podNames:
  - "Pod1"
  - "Pod2"
  - "Pod3"
```



Pod Set Requirements

- `Spec: replicas`
  - `Labels: app: frontend`
- `Status: podNames`

Always Run

`operator-sdk generate k8s`

after modifying `*_types.go`.

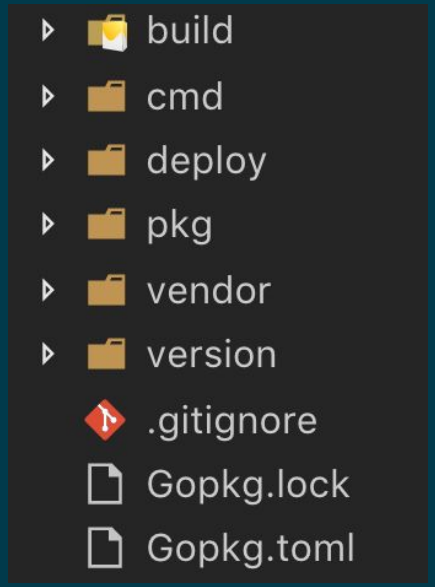
Hints Before You Begin

- **Identify Primary and Secondary Resources.**
  - These will determine what you Watch.
- **Use OwnerRefs**.
  - controllerutil.SetControllerReference to set OwnerRefs on all pods!
- **Use Labels** to manage the relationship between Primary and Secondary Resources.
  - Use `labels.SelectorFromSet` to assist in converting an existing map of labels to selectors.

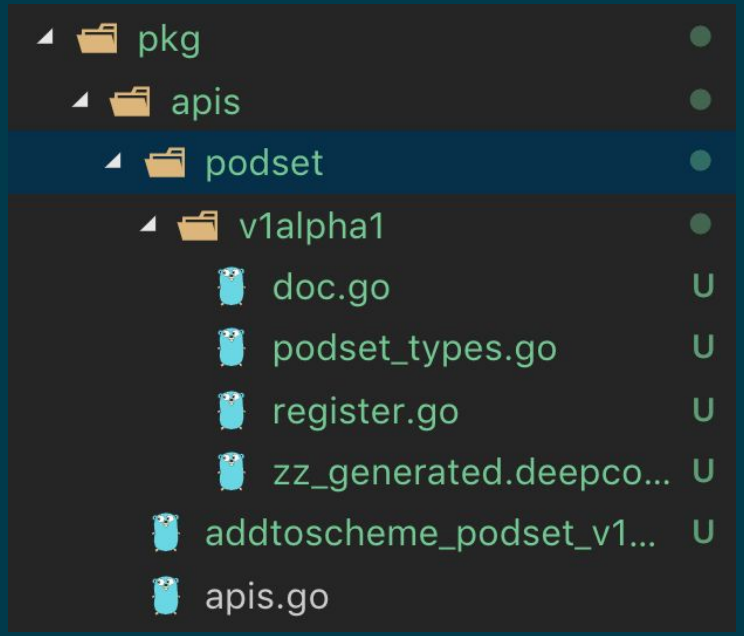
Pod Set Solution

Easy Stuff First.

operator-sdk new podset-operator



```
operator-sdk add api \
--api-version=app.example.com/v1alpha1 \ --kind=PodSet
```



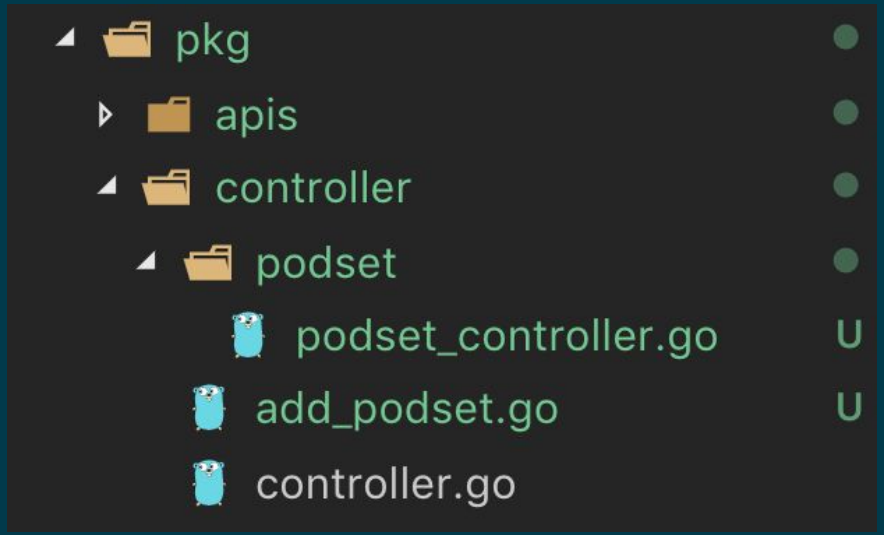
Modify Spec/Status in `podset_types.go`

```go
// PodSetSpec defines the desired state of PodSet
type PodSetSpec struct {
    // INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
    // Important: Run "operator-sdk generate k8s" to regenerate code after modifying this file
    Replicas int32 `json:"replicas"`
}

// PodSetStatus defines the observed state of PodSet
type PodSetStatus struct {
    // INSERT ADDITIONAL STATUS FIELD - define observed state of cluster
    // Important: Run "operator-sdk generate k8s" to regenerate code after modifying this file
    PodNames []string `json:"podNames"`
}
```

`operator-sdk generate k8s`

```
operator-sdk add controller \
--api-version=app.example.com/v1alpha1 --kind=PodSet
```



Identify our Primary and Secondary Resources to Watch.

The Core Component of the Controller is the `Watch` function.

1

```
Watch func(src source.Source, eventhandler
handler.EventHandler, predicates ...predicate.Predicate)
error
```

2

3

The Core Component of the Controller is the `Watch` function.

```
Watch func(src source.Source, eventhandler handler.EventHandler, predicates ...predicate.Predicate) error
```

1

2

3

1. Source of events ("sigs.k8s.io/controller-runtime/pkg/source")
   - One Kind: Your Custom GVK, Pods, Deployments, Routes, etc)
   - One Channel: Trigger reconcile based on something outside the cluster.

2. EventHandler ("sigs.k8s.io/controller-runtime/pkg/handler ")
   - 3 Types: EnqueueRequestForObject, EnqueueRequestForOwner, EnqueueRequestsFromMapFunc

3. Predicate: ("sigs.k8s.io/controller-runtime/pkg/predicate")
   - Filter events before enqueueing keys: i.e. Only enqueue Delete Events.

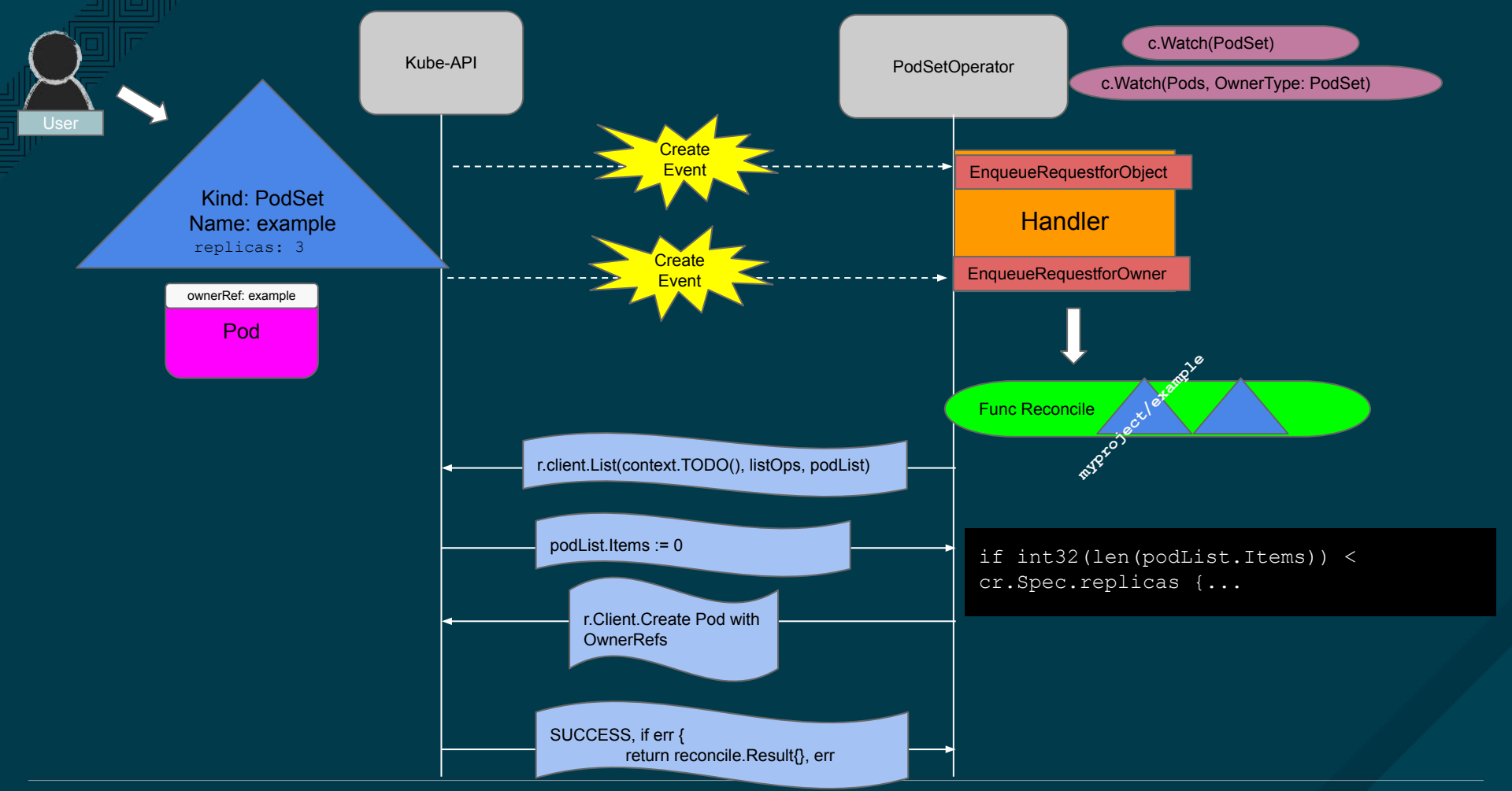
```go
// add adds a new Controller to mgr with r as the reconcile.Reconciler
func add(mgr manager.Manager, r reconcile.Reconciler) error {
    // Create a new controller
    c, err := controller.New("podset-controller", mgr, controller.Options{Reconciler: r})
    if err != nil {
        return err
    }

    // Watch for changes to primary resource PodSet
    err = c.Watch(&source.Kind{Type: &podsetv1alpha1.PodSet{}}, &handler.EnqueueRequestForObject{})
    if err != nil {
        return err
    }

    // TODO(user): Modify this to be the types you create that are owned by the primary resource
    // Watch for changes to secondary resource Pods and requeue the owner PodSet
    err = c.Watch(&source.Kind{Type: &corev1.Pod{}}, &handler.EnqueueRequestForOwner{
        IsController: true,
        OwnerType:    &podsetv1alpha1.PodSet{},
    })
    if err != nil {
        return err
    }

    return nil
}
```

Let’s Visualize the Watch.

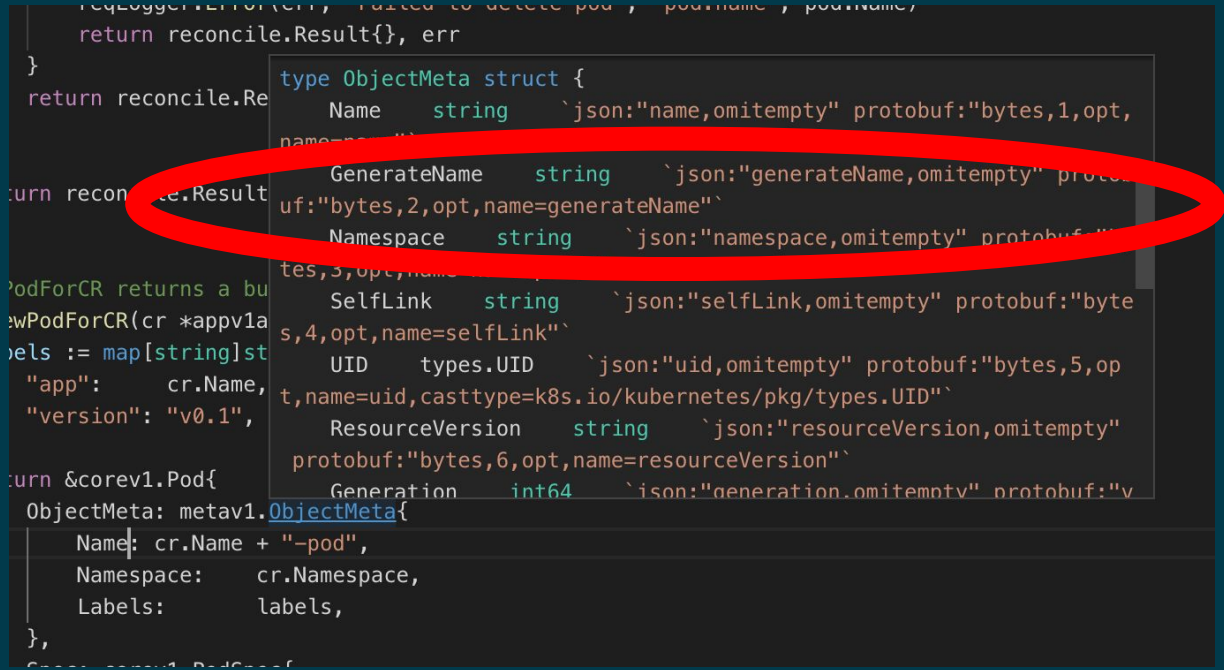

User
Kube-API
PodSetOperator
c.Watch(PodSet)
c.Watch(Pods, OwnerType: PodSet)
Kind: PodSet
Name: example
replicas: 3
ownerRef: example
Pod
Create Event
Create Event
EnqueueRequestforObject
Handler
EnqueueRequestforOwner
Func Reconcile
myproject/example
r.client.List(context.TODO(), listOps, podList)
podList.Items := 0
r.Client.Create Pod with OwnerRefs
SUCCESS, if err {
return reconcile.Result{}, err
if int32(len(podList.Items)) < cr.Spec.replicas {...

Create a Function for the Creation of the Pod.

func newPodForCR

```
128  // newPodForCR returns a busybox pod with the same name/namespace as the cr
129  func newPodForCR(cr *podsetv1alpha1.PodSet) *corev1.Pod {
130      labels := map[string]string{
131          "app": cr.Name,
132      }
133      return &corev1.Pod{
134          ObjectMeta: metav1.ObjectMeta{
135              Name:      cr.Name + "-pod",
136              Namespace: cr.Namespace,
137              Labels:    labels,
138          },
139          Spec: corev1.PodSpec{
140              Containers: []corev1.Container{
141                  {
142                      Name:    "busybox",
143                      Image:   "busybox",
144                      Command: []string{"sleep", "3600"},
145                  },
146              },
147          },
148      }
149  }
```

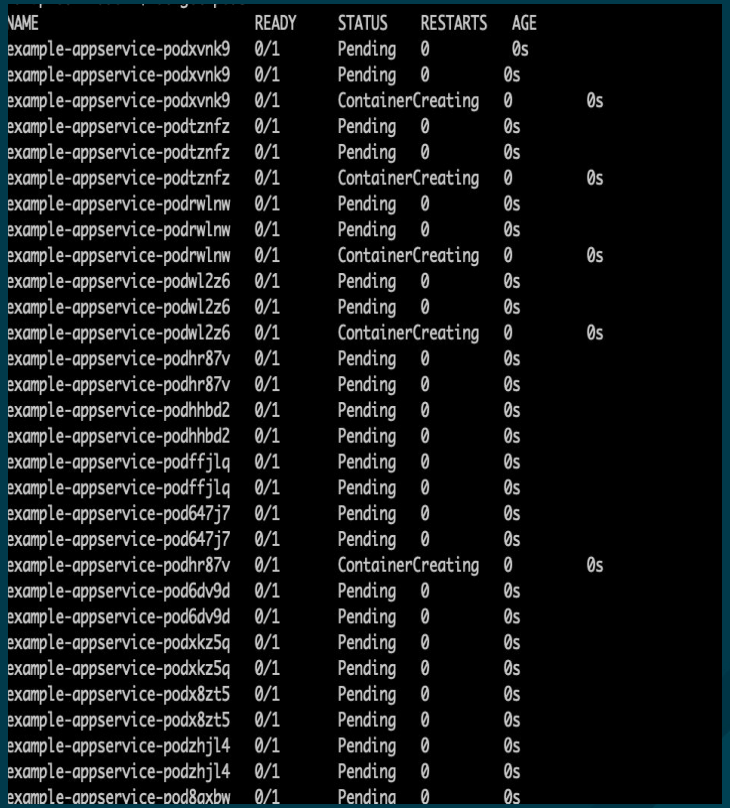
GenerateName



example-podset-podc6xlj

Red Hat

Watch Out for Hot Loops!



Red Hat

Listing Pods with List Options

```
103     // List all pods owned by this PodSet instance
104     podList := &corev1.PodList{}
105     lbs := map[string]string{
106         "app":     podSet.Name,
107         "version": "v0.1",
108     }
109     labelSelector := labels.SelectorFromSet(lbs)
110     listOps := &client.ListOptions{Namespace: podSet.Namespace, LabelSelector: labelSelector}
111     if err = r.client.List(context.TODO(), listOps, podList); err != nil {
112         return reconcile.Result{}, err
113     }
```

Be Careful When Listing Pods...

Some May Be in Terminating Status..

```
example-appservice-pod   0/1       Terminating   0          1m
```

Only Count Pods that are “Available”

```
115     // Count the pods that are pending or running as available
116     var available []corev1.Pod
117     for _, pod := range podList.Items {
118         if pod.ObjectMeta.DeletionTimestamp != nil {
119             continue
120         }
121         if pod.Status.Phase == corev1.PodRunning || pod.Status.Phase == corev1.PodPending {
122             available = append(available, pod)
123         }
124     }
125     numAvailable := int32(len(available))
126     availableNames := []string{}
127     for _, pod := range available {
128         availableNames = append(availableNames, pod.ObjectMeta.Name)
129     }
```