Lab: Creating a ReplicaSet

We can use the ReplicaSet spec that we used in the previous lecture:

```yaml
# replicaset.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  labels:
    app: myapp
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: web
          image: "nginx"
```

We can apply this to the cluster using `kubectl` as follows:

```
kubectl apply -f replicaset.yaml
```

View the pods that have been created for us:

```
kubectl get pods
```

Notice how the ReplicaSet automatically used unique names for the pods.

Testing pod recreation

If we delete one of the pods, the ReplicaSet will recreate it based on the template that it has:

```
kubectl get pods
kubectl delete pods <pod-name>
kubectl get pods # you should see a new pod with a new name created
and running
```

Describing a ReplicaSet

```
kuebctl describe myapp
```

Finding which ReplicaSet manages which pods

```
kuebctl get pods # to get the pods
kubectl get pods <pod-name> -
o=jsonpath='{.metadata.ownerReference[0].name}'
```

Finding which pods are managed by the ReplicaSet

First, we need to know which labels our ReplicaSet is using, we can do this with the `kubectl describe` command that we mentioned earlier.

Then, we can use those labels to list the pods:

```
kubectl get pods -l app=myapp
```

Scaling ReplicaSets

There are two ways to scale a ReplicaSet: imperative and declarative.

The imperative way is useful when we want to quicly apply an action. For example, there is a sudden spike in the load and we don't have time to find which manifest is responsible for this ReplicaSet, change it, save it, then apply it to the cluster. A quicker way would be to use a comand like the following:

```
kubectl scale replicasets myapp --replicas=4
```

If we check the number of pods now we'd see that they've been scaled out to `4`.

But we must remember to always update the ReplicaSet manifest so that it matches our business needs. Otherwise, we could accidentally revert to an undesirable state by applying the outdated manifest.

To do this, we can just update the `replicaCount` part of the manifest and set it to `4` then apply the updaed manifest to the cluster.

Autoscaling pods

Kubernetes distinguishes between two types of scaling:

Vertical scaling

Increasing the resources of the pods (CPU, and memory)

Horizontal scaling

Increasing the number of pods with the same resources.

Cluster autoscaling

Can be used in a cloud environment or if you have a means to automatically add or remove machines from the cluster (for example, through VMWare). In this scenario, we add/remove nodes from the cluster depending on the current load.

To use the HPA, we need to install a metrics server that measures the amount of CPU and memory consumed by a ReplicaSet and increases to decreases the `replicaCount` based on the load.

Deleting the ReplicaSet

Deleting the ReplicaSet by default deletes all the pods that it manages unless you use the `--cascase=false` switch:

```
kubectl delete rs myapp --cascade=false
```

Notice that, now, any pods that were managed by this ReplicaSet and *orphaned*, meaning that they are not managed by a ReplicaSet and it one of them dies, the ReplicaSet will not bring it back.