Exercise 7.1: Working with ReplicaSets

Overview

Understanding and managing the state of containers is a core Kubernetes task. In this lab we will first explore the API objects used to manage groups of containers. The objects available have changed as Kubernetes has matured, so the Kubernetes version in use will determine which are available. Our first object will be a `ReplicaSet`, which does not include newer management features found with `Deployments`. A `Deployment` operator manages `ReplicaSet` operators for you. We will also work with another object and watch loop called a `DaemonSet` which ensures a container is running on newly added node.

Then we will update the software in a container, view the revision history, and roll-back to a previous version.

A `ReplicaSet` is a next-generation of a `Replication Controller`, which differs only in the selectors supported. The only reason to use a `ReplicaSet` anymore is if you have no need for updating container software or require update orchestration which won't work with the typical process.

1. View any current `ReplicaSets`. If you deleted resources at the end of a previous lab, you should have none reported in the `default` namespace.

```
student@cp:~$ kubectl get rs
No resources found in default namespace.
```

2. Create a YAML file for a simple `ReplicaSet`. The `apiVersion` setting depends on the version of Kubernetes you are using. The object is stable using the `apps/v1` apiVersion. We will use an older version of **nginx** then update to a newer version later in the exercise.

```
student@cp:~$ vim rs.yaml
```

rs.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-one
spec:
  replicas: 2
  selector:
    matchLabels:
      system: ReplicaOne
  template:
    metadata:
      labels:
        system: ReplicaOne
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.1
        ports:
        - containerPort: 80
```

3. Create the `ReplicaSet`:

```
student@cp:~$ kubectl create -f rs.yaml
```

```
replicaset.apps/rs-one created
```

4. View the newly created `ReplicaSet`:

```
student@cp:~$ kubectl describe rs rs-one
```

```
Name:           rs-one
Namespace:      default
Selector:       system=ReplicaOne
Labels:         <none>
Annotations:    <none>
Replicas:       2 current / 2 desired
Pods Status:    2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:       system=ReplicaOne
  Containers:
   nginx:
    Image:          nginx:1.15.1
    Port:           80/TCP
    Host Port:      0/TCP
    Environment:    <none>
    Mounts:         <none>
  Volumes:          <none>
Events:             <none>
```

5. View the Pods created with the `ReplicaSet`. From the yaml file created there should be two Pods. You may see a `Completed` busybox which will be cleared out eventually.

```
student@cp:~$ kubectl get pods
```

```
NAME           READY     STATUS    RESTARTS   AGE
rs-one-2p9x4   1/1       Running   0          5m4s
rs-one-3c6pb   1/1       Running   0          5m4s
```

6. Now we will delete the `ReplicaSet`, but not the Pods it controls.

```
student@cp:~$ kubectl delete rs rs-one --cascade=orphan
```

```
replicaset.apps "rs-one" deleted
```

7. View the `ReplicaSet` and Pods again:

```
student@cp:~$ kubectl describe rs rs-one
```

```
Error from server (NotFound): replicasets.apps "rs-one" not found
```

```
student@cp:~$ kubectl get pods
```

```
NAME           READY     STATUS    RESTARTS   AGE
rs-one-2p9x4   1/1       Running   0          7m
rs-one-3c6pb   1/1       Running   0          7m
```

8. Create the `ReplicaSet` again. As long as we do not change the `selector` field, the new `ReplicaSet` should take ownership. Pod software versions cannot be updated this way.

```
student@cp:~$ kubectl create -f rs.yaml
```

```
replicaset.apps/rs-one created
```

9. View the age of the ReplicaSet and then the Pods within:

```
student@cp:~$ kubectl get rs
NAME      DESIRED   CURRENT   READY     AGE
rs-one    2         2         2         46s

student@cp:~$ kubectl get pods
NAME           READY     STATUS    RESTARTS   AGE
rs-one-2p9x4   1/1       Running   0          8m
rs-one-3c6pb   1/1       Running   0          8m
```

10. We will now isolate a Pod from its ReplicaSet. Begin by editing the label of a Pod. We will change the system: parameter to be IsolatedPod.

```
student@cp:~$ kubectl edit pod rs-one-3c6pb

....
  labels:
    system: IsolatedPod   #<-- Change from ReplicaOne
managedFields:
....
```

11. View the number of pods within the ReplicaSet. You should see two running.

```
student@cp:~$ kubectl get rs
NAME      DESIRED   CURRENT   READY     AGE
rs-one    2         2         2         4m
```

12. Now view the pods with the label key of system. You should note that there are three, with one being newer than others. The ReplicaSet made sure to keep two replicas, replacing the Pod which was isolated.

```
student@cp:~$ kubectl get po -L system
NAME           READY     STATUS    RESTARTS   AGE       SYSTEM
rs-one-3c6pb   1/1       Running   0          10m       IsolatedPod
rs-one-2p9x4   1/1       Running   0          10m       ReplicaOne
rs-one-dq5xd   1/1       Running   0          30s       ReplicaOne
```

13. Delete the ReplicaSet, then view any remaining Pods.

```
student@cp:~$ kubectl delete rs rs-one
replicaset.apps "rs-one" deleted

student@cp:~$ kubectl get po
NAME           READY     STATUS        RESTARTS   AGE
rs-one-3c6pb   1/1       Running       0          14m
rs-one-dq5xd   0/1       Terminating   0          4m
```

14. In the above example the Pods had not finished termination. Wait for a bit and check again. There should be no ReplicaSets, but one Pod.

```
student@cp:~$ kubectl get rs
No resources found in default namespaces.
```

```
student@cp:~$ kubectl get pod
```

```
NAME           READY     STATUS    RESTARTS   AGE
rs-one-3c6pb   1/1       Running   0          16m

```

15. Delete the remaining Pod using the label.

```
student@cp:~$ kubectl delete pod -l system=IsolatedPod
```

```
pod "rs-one-3c6pb" deleted
```