Kubernetes

Lab 4 – Deployments and Replica Sets

In this lab we will explore the nature of Kubernetes deployments and replica sets and how to work with them.

Deployments

A deployment provides declarative updates for pods and replica sets. You describe the desired state in a deployment object, and the deployment controller will change the actual state to the desired state at a controlled rate for you. You can define deployments to create new resources, or replace existing ones by new ones. Typical uses:

- bring up a replica set and (indirectly) its pods
- capturing the results and status of a deployment
- updating an existing deployment to recreate pods with a new image (rolling updates)
- rolling back to an earlier deployment revision if the current deployment isn't stable
- pausing and resuming a deployment

ReplicaSets

Replica sets (RS) supersede the older replication controller (RC) resource type. Replica sets support the set-based selectors as well as equality-based selector requirements (RCs only supported equality.) While replica sets can be used independently, they are mainly used by deployments as a mechanism to orchestrate pod creation, deletion, and updates. When you use deployments you don't have to worry about managing the replica sets that they create; deployments own and manage their replica sets.

ReplicaSets ensure that a specified number of pod "replicas" are running at all times. If there are too many, it will kill some. If there are too few, it will start more. Unlike in the case where a user directly created pods, a ReplicaSet replaces pods that are deleted or terminated for any reason, such as in the case of node failure or disruptive node maintenance (e.g. a kernel upgrade, etc.)

For this reason the Kubernetes team recommends that you use a Deployment/ReplicaSet even if your application requires only a single pod. ReplicaSets are like process supervisors in many ways but monitor processes on multiple nodes at once. A ReplicaSet delegates local container restarts to some agent on the node (e.g., Kubelet or Docker.)

A ReplicaSet is only appropriate for pods with *RestartPolicy* = *Always* (if the RestartPolicy is not set, the default value is *Always*.) A ReplicaSet will refuse to instantiate any pod that has a different restart policy.

A ReplicaSet will never terminate on its own, but it isn't expected to be as long-lived as services. Services may be composed of pods controlled by multiple ReplicaSets, and it is expected that many ReplicaSets may be created and destroyed over the lifetime of a service (for instance, to perform an update of pods that run the service.) Both services themselves and their clients should remain oblivious to the ReplicaSets that maintain the pods of the services.

Now to create some Deployments/ReplicaSets.

1. A Simple Deployment

As a first exploratory step lets create a simple deployment which stands up three nginx pods. Create a config file similar to the following to accomplish this task:

```
user@ubuntu:~$ cd ~

user@ubuntu:~$ mkdir ~/dep && cd ~/dep

user@ubuntu:~/dep$ nano mydep.yaml && cat mydep.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: website
  labels:
    bu: sales
spec:
  replicas: 3
```

```
  selector:
    matchLabels:
      appname: webserver
      targetenv: demo
  template:
    metadata:
      labels:
        appname: webserver
        targetenv: demo
    spec:
      containers:
      - name: podweb
        image: nginx:1.7.9
        ports:
        - containerPort: 80

user@ubuntu:~/dep$
```

Deployments were promoted to the apps/v1 API in K8s 1.9 but were added to Kubernetes 1.2 and are the go forward solution for deploying replicated pods. The spec for Replication Controllers (part of the v1 API) is almost the same as the spec for Deployments though deployments add a few key features such as the ability to specify upgrades declaratively. The specification for Deployments can be found here:

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#deployment-v1-apps

Now create the Deployment using the `kubectl apply` subcommand and verify that the Deployment, its ReplicaSet and pods are up with the `get` subcommand:

```
user@ubuntu:~/dep$ kubectl apply -f mydep.yaml

deployment.apps/website created

user@ubuntu:~/dep$
```

List the deployments, replicasets and pods. You can abbreviate certain APi objects when calling them, as shown below:

```
user@ubuntu:~/dep$ kubectl get deploy,rs,po

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/website   3/3     3            3           15s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/website-5577f87457   3         3         3       15s

NAME                            READY   STATUS    RESTARTS   AGE
pod/website-5577f87457-6gxrf   1/1     Running   0          15s
pod/website-5577f87457-fswcn   1/1     Running   0          15s
pod/website-5577f87457-vv96d   1/1     Running   0          15s

user@ubuntu:~/dep$
```

While everything appears to be running we can verify that there are no scheduling cycles or fail/restart activities by examining the system events. We have viewed resource specific events in the past using the `kubectl describe` subcommand. This time we'll use the `kubectl get events` subcommand to view cluster wide events:

```
user@ubuntu:~/dep$ kubectl get events --sort-by='{metadata.creationTimestamp}' | grep website

3m16s       Normal    SuccessfulCreate          replicaset/website-5577f87457   Created pod:
website-5577f87457-fswcn
<unknown>   Normal    Scheduled                 pod/website-5577f87457-6gxrf    Successfully
assigned default/website-5577f87457-6gxrf to ubuntu
3m16s       Normal    SuccessfulCreate          replicaset/website-5577f87457   Created pod:
website-5577f87457-vv96d
3m16s       Normal    SuccessfulCreate          replicaset/website-5577f87457   Created pod:
website-5577f87457-6gxrf
<unknown>   Normal    Scheduled                 pod/website-5577f87457-vv96d    Successfully
```

```
assigned default/website-5577f87457-vv96d to ubuntu
<unknown>   Normal    Scheduled                 pod/website-5577f87457-fswcn    Successfully
assigned default/website-5577f87457-fswcn to ubuntu
3m16s       Normal    ScalingReplicaSet         deployment/website              Scaled up
replica set website-5577f87457 to 3
3m14s       Normal    Pulling                   pod/website-5577f87457-fswcn    Pulling image
"nginx:1.7.9"
3m14s       Normal    Pulling                   pod/website-5577f87457-6gxrf    Pulling image
"nginx:1.7.9"
3m14s       Normal    Pulling                   pod/website-5577f87457-vv96d    Pulling image
"nginx:1.7.9"
3m5s        Normal    Pulled                    pod/website-5577f87457-6gxrf    Successfully
pulled image "nginx:1.7.9"
3m5s        Normal    Created                   pod/website-5577f87457-6gxrf    Created
container podweb
3m5s        Normal    Started                   pod/website-5577f87457-6gxrf    Started
container podweb
3m4s        Normal    Created                   pod/website-5577f87457-vv96d    Created
container podweb
3m4s        Normal    Pulled                    pod/website-5577f87457-vv96d    Successfully
pulled image "nginx:1.7.9"
3m3s        Normal    Started                   pod/website-5577f87457-vv96d    Started
container podweb
3m2s        Normal    Started                   pod/website-5577f87457-fswcn    Started
container podweb
3m2s        Normal    Created                   pod/website-5577f87457-fswcn    Created
container podweb
3m2s        Normal    Pulled                    pod/website-5577f87457-fswcn    Successfully
pulled image "nginx:1.7.9"

user@ubuntu:~/dep$
```

Checking the event log occasionally will help you identify normal cluster patterns and make it possible for you to spot anomalies more easily when debugging.

The replica set began with a scale of 3, causing 3 instances of the pod template to get scheduled. Replica sets ensure that some number of instances of the pod template are always running. Try deleting a pod (use the pod name displayed by `kubectl get pods`).

```
user@ubuntu:~/dep$ kubectl delete $(kubectl get pod -o name |tail -1)

pod "website-5577f87457-vv96d" deleted

user@ubuntu:~/dep$ kubectl get pods

NAME                       READY   STATUS    RESTARTS   AGE
website-5577f87457-6gxrf   1/1     Running   0          4m20s
website-5577f87457-fswcn   1/1     Running   0          4m20s
website-5577f87457-mb7cv   1/1     Running   0          20s

user@ubuntu:~/dep$
```

You might ask: "why would Kubernetes let someone delete the pod if it will just restart it?". There are many reasons you might want to delete a pod:

- The pod ran into problems and you want to generate a new replacement.
- The current node has problems and you want Kubernetes to reschedule this particular pod somewhere else.
- A change was made to another Kubernetes resource the pod relied on, like a configMap or volume mount, which require a pod to be recreated to take effect in many cases.

To actually terminate our pod permanently we must delete the deployment, the deployment controls the replica set, the replica set controls the pods.

When many resources are running on a cluster it can be advantageous to restrict output to a certain set of resources. The Kubernetes labeling system makes this easy. The `-l` switch can be used with the `kubectl get` subcommand to filter output by label.

Try listing all pods with the `--show-labels` options to show the pod labels:

```
user@ubuntu:~/dep$ kubectl get pods --show-labels --all-namespaces

NAMESPACE     NAME                              READY   STATUS    RESTARTS   AGE     LABELS
default       website-5577f87457-6gxrf          1/1     Running   0          4m40s
appname=webserver,pod-template-hash=5577f87457,targetenv=demo
default       website-5577f87457-fswcn          1/1     Running   0          4m40s
appname=webserver,pod-template-hash=5577f87457,targetenv=demo
default       website-5577f87457-mb7cv          1/1     Running   0          40s
appname=webserver,pod-template-hash=5577f87457,targetenv=demo
kube-system   coredns-5644d7b6d9-b4rnz          1/1     Running   1          80m     k8s-
app=kube-dns,pod-template-hash=5644d7b6d9
kube-system   coredns-5644d7b6d9-lxdqv          1/1     Running   1          80m     k8s-
app=kube-dns,pod-template-hash=5644d7b6d9
kube-system   etcd-ubuntu                       1/1     Running   1          79m
component=etcd,tier=control-plane
kube-system   kube-apiserver-ubuntu             1/1     Running   1          79m
component=kube-apiserver,tier=control-plane
kube-system   kube-controller-manager-ubuntu    1/1     Running   1          79m
component=kube-controller-manager,tier=control-plane
kube-system   kube-proxy-npxks                  1/1     Running   1          80m     controller-
revision-hash=57cd7f4c65,k8s-app=kube-proxy,pod-template-generation=1
kube-system   kube-scheduler-ubuntu             1/1     Running   1          79m
component=kube-scheduler,tier=control-plane
kube-system   weave-net-rvhvk                   2/2     Running   3          64m     controller-
revision-hash=7f54576664,name=weave-net,pod-template-generation=1

user@ubuntu:~/dep$
```

Now try filtering by the "appname" label key we assigned to all of our pods in the pod template metadata, searching all namespaces:

```
user@ubuntu:~/dep$ kubectl get pods --show-labels --all-namespaces -l appname

NAMESPACE   NAME                        READY   STATUS    RESTARTS   AGE     LABELS
default     website-5577f87457-6gxrf    1/1     Running   0          5m11s
appname=webserver,pod-template-hash=5577f87457,targetenv=demo
default     website-5577f87457-fswcn    1/1     Running   0          5m11s
appname=webserver,pod-template-hash=5577f87457,targetenv=demo
default     website-5577f87457-mb7cv    1/1     Running   0          71s
appname=webserver,pod-template-hash=5577f87457,targetenv=demo

user@ubuntu:~/dep$
```

You can also filter by key and value:

```
user@ubuntu:~/dep$ kubectl get pods --all-namespaces -l targetenv=demo

NAMESPACE   NAME                        READY   STATUS    RESTARTS   AGE
default     website-5577f87457-6gxrf    1/1     Running   0          5m35s
default     website-5577f87457-fswcn    1/1     Running   0          5m35s
default     website-5577f87457-mb7cv    1/1     Running   0          95s

user@ubuntu:~/dep$
```

You can filter by pod name:

```
user@ubuntu:~/dep$ kubectl get $(kubectl get pods -o name | head -1)

NAME                        READY   STATUS    RESTARTS   AGE
website-5577f87457-6gxrf    1/1     Running   0          5m58s

user@ubuntu:~/dep$
```

Our pod has labels we have added and the Kubernetes infrastructure may add labels as well:

```
user@ubuntu:~/dep$ kubectl describe $(kubectl get pods -o name | head -1) | grep -A2 -i label

Labels:         appname=webserver
                pod-template-hash=5577f87457
                targetenv=demo

user@ubuntu:~/dep$
```

Unfortunately describe doesn't allow for JSON output. Good news, though, `get` does.

```
user@ubuntu:~/dep$ kubectl get $(kubectl get pods -o name | head -1) -o json | jq
.metadata.labels
```

```
{
  "appname": "webserver",
  "pod-template-hash": "5577f87457",
  "targetenv": "demo"
}
```

```
user@ubuntu:~/dep$
```

- Why do each of the filters above work or not work?
- Enter a command to display all of the pods with either the “demo” or “prod” value for targetenv
- Find all pods other than those with the “demo” or “prod” value for targetenv
- Enter a command to display all of the pods with either the “demo” or “prod” value for targetenv and the appname key set to webserver

2. Checking status of a Deployment

We have seen previously how to check the status of a deployment.

```
user@ubuntu:~/dep$ kubectl get deploy

NAME      READY   UP-TO-DATE   AVAILABLE   AGE
website   3/3     3            3           6m43s

user@ubuntu:~/dep$
```

Now we take an slightly more application-centric view.

```
user@ubuntu:~/dep$ kubectl rollout status deploy/website

deployment "website" successfully rolled out

user@ubuntu:~/dep$
```

Rollouts are used to update a given set of Pods, the ones controlled by this Deployment's replica set. It reports success when all the currently deployed Pods match what is expected in the current deployment. In k8s technical terms these conditions are all true:

- .status.observedGeneration >= .metadata.generation
- .status.updatedReplicas == .spec.replicas
- .spec.availableReplicas >= minimum required

3. Updating a Deployment

We are using nginx 1.7.9 in our example, lets update to 1.9.1.

```
user@ubuntu:~/dep$ kubectl set image deploy/website podweb=nginx:1.9.1 --record

deployment.apps/website image updated

user@ubuntu:~/dep$
```

Alternative is to use `kubectl edit deployment/website`

Check the status of the rollout (if you're not fast you may not see these updates):

```
user@ubuntu:~/dep$ kubectl rollout status deploy/website

Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "website" rollout to finish: 1 old replicas are pending termination...
deployment "website" successfully rolled out

user@ubuntu:~/dep$
```

```
user@ubuntu:~/dep$ kubectl get deploy/website

NAME      READY   UP-TO-DATE   AVAILABLE   AGE
website   3/3     3            3           7m29s

user@ubuntu:~/dep$
```

Look at the Replica Sets & Pods:

```
user@ubuntu:~/dep$ kubectl get rs,pod

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/website-5577f87457   0         0         0       7m44s
replicaset.apps/website-769bf6f999   3         3         3       36s

NAME                           READY   STATUS    RESTARTS   AGE
pod/website-769bf6f999-6m54n   1/1     Running   0          24s
pod/website-769bf6f999-78nlg   1/1     Running   0          36s
pod/website-769bf6f999-rjt2g   1/1     Running   0          22s

user@ubuntu:~/dep$
```

All the pods bear different suffixes, and should be seconds old at this point. When a rollout is performed, the original replicaSet controlling the pods are scaled down to zero, and a new replicaSet bearing the new spec (in this case, a new container image tag) is created and scaled up to replace it.

By describing the deployment we can inspect the events that occurred during the rollout:

```
user@ubuntu:~/dep$ kubectl describe deploy/website | grep -A 10 Events

Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  8m    deployment-controller  Scaled up replica set website-
5577f87457 to 3
  Normal  ScalingReplicaSet  52s   deployment-controller  Scaled up replica set website-
769bf6f999 to 1
  Normal  ScalingReplicaSet  40s   deployment-controller  Scaled down replica set website-
5577f87457 to 2
```

```
  Normal  ScalingReplicaSet  40s   deployment-controller  Scaled up replica set website-
769bf6f999 to 2
  Normal  ScalingReplicaSet  38s   deployment-controller  Scaled down replica set website-
5577f87457 to 1
  Normal  ScalingReplicaSet  38s   deployment-controller  Scaled up replica set website-
769bf6f999 to 3
  Normal  ScalingReplicaSet  36s   deployment-controller  Scaled down replica set website-
5577f87457 to 0

user@ubuntu:~/dep$
```

Note that the rollout was a smooth transition from one set of Pods controlled by our original ReplicaSet to our second set of Pods controlled by the RS.

4. Manually rolling back a deployment

Lets manually revert back to nginx 1.7.9 and check the status:

```
user@ubuntu:~/dep$ kubectl set image deploy/website podweb=nginx:1.7.9 --record

deployment.apps/website image updated

user@ubuntu:~/dep$
```

```
user@ubuntu:~/dep$ kubectl rollout status deploy/website

...

Waiting for rollout to finish: 1 old replicas are pending termination...
Waiting for rollout to finish: 1 old replicas are pending termination...
deployment "website" successfully rolled out

user@ubuntu:~/dep$
```

```
user@ubuntu:~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY   AGE
website-5577f87457   3         3         3       9m6s
website-769bf6f999   0         0         0       11

user@ubuntu:~/dep$
```

Notice which RS hash in the pod name is being used.

```
user@ubuntu:~/dep$ kubectl get pods

NAME                       READY   STATUS    RESTARTS   AGE
website-5577f87457-4585h   1/1     Running   0          55s
website-5577f87457-vjf2m   1/1     Running   0          57s
website-5577f87457-vwpcs   1/1     Running   0          5

user@ubuntu:~/dep$
```

Confirm your observations once again in the event log.

```
user@ubuntu:~/dep$ kubectl describe deploy/website | grep -A 15 Events

Events:
  Type    Reason             Age                From                   Message
  ----    ------             ----               ----                   -------
  Normal  ScalingReplicaSet  2m29s                deployment-controller  Scaled up replica set
```

```
website-769bf6f999 to 1
  Normal  ScalingReplicaSet  2m17s               deployment-controller  Scaled down replica set
website-5577f87457 to 2
  Normal  ScalingReplicaSet  2m17s               deployment-controller  Scaled up replica set
website-769bf6f999 to 2
  Normal  ScalingReplicaSet  2m15s               deployment-controller  Scaled down replica set
website-5577f87457 to 1
  Normal  ScalingReplicaSet  2m15s               deployment-controller  Scaled up replica set
website-769bf6f999 to 3
  Normal  ScalingReplicaSet  2m13s               deployment-controller  Scaled down replica set
website-5577f87457 to 0
  Normal  ScalingReplicaSet  73s                 deployment-controller  Scaled up replica set
website-5577f87457 to 1
  Normal  ScalingReplicaSet  72s                 deployment-controller  Scaled down replica set
website-769bf6f999 to 2
  Normal  ScalingReplicaSet  70s (x2 over 9m37s)  deployment-controller  Scaled up replica set
website-5577f87457 to 3
  Normal  ScalingReplicaSet  69s (x3 over 72s)    deployment-controller  (combined from similar
events): Scaled down replica set website-769bf6f999 to 0

user@ubuntu:~/dep$
```

5. Checking rollout history of a Deployment

We can use the *rollout history* subcommand to see what we have been doing to trigger these rollouts:

```
user@ubuntu:~/dep$ kubectl rollout history deploy/website

deployment.apps/website
REVISION  CHANGE-CAUSE
2         kubectl set image deploy/website podweb=nginx:1.9.1 --record=true
3         kubectl set image deploy/website podweb=nginx:1.7.9 --record=true

user@ubuntu:~/dep$
```

Take a detailed look at a previous deployment version:

```
user@ubuntu:~/dep$ kubectl rollout history deploy/website --revision=2

deployment.apps/website with revision #2
Pod Template:
  Labels:       appname=webserver
        pod-template-hash=769bf6f999
        targetenv=demo
  Annotations:  kubernetes.io/change-cause: kubectl set image deploy/website podweb=nginx:1.9.1
--record=true
  Containers:
   podweb:
    Image:      nginx:1.9.1
    Port:       80/TCP
    Host Port:  0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>

user@ubuntu:~/dep$
```

6. Rolling back to a previous Deployment

Confirm the current version of a container is 1.7.9.

```
user@ubuntu:~/dep$ kubectl get pods -o json | jq .items[0].spec.containers[0].image -r

nginx:1.7.9
```

```
user@ubuntu:~/dep$
```

Revert to previous version/revision, tracking the rollout status by chaining the `kubectl rollout status` command:

```
user@ubuntu:~/dep$ kubectl rollout undo deploy/website && kubectl rollout status deploy/website

deployment.apps/website rolled back
Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "website" rollout to finish: 1 old replicas are pending termination...
deployment "website" successfully rolled out

user@ubuntu:~/dep$
```

Alternative to above is `kubectl rollout undo deployment/website --to-revision=2`

```
user@ubuntu:~/dep$ kubectl get deploy/website

NAME      READY   UP-TO-DATE   AVAILABLE   AGE
website   3/3     3            3           9m49s

user@ubuntu:~/dep$
```

```
user@ubuntu:~/dep$ kubectl describe deploy/website | grep -A 15 Events

Events:
  Type    Reason             Age                  From                   Message
  ----    ------             ----                 ----                   -------
  Normal  ScalingReplicaSet  4m17s                deployment-controller  Scaled down replica set
website-5577f87457 to 2
  Normal  ScalingReplicaSet  4m15s                deployment-controller  Scaled down replica set
website-5577f87457 to 1
  Normal  ScalingReplicaSet  4m13s                deployment-controller  Scaled down replica set
website-5577f87457 to 0
  Normal  ScalingReplicaSet  3m13s                deployment-controller  Scaled up replica set
website-5577f87457 to 1
  Normal  ScalingReplicaSet  3m12s                deployment-controller  Scaled down replica set
website-769bf6f999 to 2
  Normal  ScalingReplicaSet  3m10s (x2 over 11m)  deployment-controller  Scaled up replica set
website-5577f87457 to 3
  Normal  ScalingReplicaSet  43s (x2 over 4m29s)  deployment-controller  Scaled up replica set
website-769bf6f999 to 1
  Normal  ScalingReplicaSet  41s (x2 over 4m17s)  deployment-controller  Scaled up replica set
website-769bf6f999 to 2
  Normal  ScalingReplicaSet  39s (x2 over 4m15s)  deployment-controller  Scaled up replica set
website-769bf6f999 to 3
  Normal  ScalingReplicaSet  37s (x6 over 3m12s)  deployment-controller  (combined from similar
events): Scaled down replica set website-5577f87457 to 0

user@ubuntu:~/dep$
```

Confirm the container image version has been reverted to 1.9.1:

```
user@ubuntu:~/dep$ kubectl get pods -o json | jq .items[0].spec.containers[0].image -r

nginx:1.9.1

user@ubuntu:~/dep$
```

7. Pausing and resuming a Deployment

In a larger installation, we may be deploying dozens of pods. For our small test it is hard to pause in time, so we chain the commands to hopefully catch it in the act.

```
user@ubuntu:~/dep$ kubectl set image deploy/website podweb=nginx:1.7.9; kubectl rollout pause
deploy/website

deployment.apps/website image updated
deployment.apps/website paused

user@ubuntu:~/dep$
```

```
user@ubuntu:~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY   AGE
website-5577f87457   1         1         1       12m
website-769bf6f999   3         3         3       5m15s

user@ubuntu:~/dep$
```

It worked! You can see that the original 1.7.9 replicaSet and the 1.9.1 replicaSets both have pods running, despite running the `kubectl set` command.

Check the deployment status with `kubectl rollout`. Since `kubectl rollout` will watch for the change, you will need to back out of the command with CTRL C:

```
user@ubuntu:~/dep$ kubectl rollout status deploy/website

Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...

^C

user@ubuntu:~/dep$
```

To resume the rollout, you can use the `resume` subcommand for `kubectl rollout`:

```
user@ubuntu:~/dep$ kubectl rollout resume deploy/website && kubectl rollout status
deploy/website

deployment.apps/website resumed
Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "website" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "website" rollout to finish: 1 old replicas are pending termination...
deployment "website" successfully rolled out

user@ubuntu:~/dep$
```

Now check the replicaSets to see if they have successfully replaced each other:

```
user@ubuntu:~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY     AGE
website-5577f87457   3         3         3       13m
website-769bf6f999   0         0         0       6m7s
```

```
user@ubuntu:~/dep$
```

There are no more nginx pods running version 1.9.1, only 1.7.6.

Delete your deployment:

```
user@ubuntu:~/dep$ kubectl delete deploy website

deployment.apps "website" deleted

user@ubuntu:~/dep$
```

8. Health Checks

In this step we will create a pod with a health check. Enter and run the following config (*hc.yaml*):

```
user@ubuntu:~/dep$ mkdir ~/hc && cd ~/hc

user@ubuntu:~/hc$ nano hc.yaml && cat hc.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      name: nginx
  template:
    metadata:
      labels:
        name: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        livenessProbe: # An HTTP health check
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 30
          timeoutSeconds: 1

user@ubuntu:~/hc$
```

Now run the deployment:

```
user@ubuntu:~/hc$ kubectl apply -f hc.yaml

deployment.apps/nginx created

user@ubuntu:~/hc$
```

View your deployment:

```
user@ubuntu:~/hc$ kubectl get deploy,rs,pods

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
```

```
deployment.apps/nginx   3/3     3            3           13s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-8cdb45866   3         3         3       13s

NAME                          READY   STATUS    RESTARTS   AGE
pod/nginx-8cdb45866-9p8ws   1/1     Running   0          13s
pod/nginx-8cdb45866-lvbkl   1/1     Running   0          13s
pod/nginx-8cdb45866-r89pv   1/1     Running   0          13s

user@ubuntu:~/hc$
```

Note that our nginx service listens on port 80 and responds normally to requests for "/", so our health check is passing.

To trigger the health check repair logic, we need to simulate an error condition. By forcing nginx to report a 404, the *httpGet* livenessProbe will fail. We can do this by deleting the nginx configuration file in the nginx container.

Display the events for the first pod in the set:

```
user@ubuntu:~/hc$ kubectl get events --sort-by='{metadata.creationTimestamp}' \
| grep $(kubectl get pods -o name | head -1 | awk -F '/' '{print $2}')

32s         Normal    SuccessfulCreate          replicaset/nginx-8cdb45866      Created pod:
nginx-8cdb45866-9p8ws
<unknown>   Normal    Scheduled                 pod/nginx-8cdb45866-9p8ws       Successfully
assigned default/nginx-8cdb45866-9p8ws to ubuntu
29s         Normal    Pulling                   pod/nginx-8cdb45866-9p8ws       Pulling image
"nginx:latest"
23s         Normal    Started                   pod/nginx-8cdb45866-9p8ws       Started
container nginx
23s         Normal    Created                   pod/nginx-8cdb45866-9p8ws       Created
container nginx
23s         Normal    Pulled                    pod/nginx-8cdb45866-9p8ws       Successfully
pulled image "nginx:latest"

user@ubuntu:~/hc$
```

The status is good.

Now lets tell the nginx in the first pod to stop serving the root IRI by deleting the nginx default config:

```
user@ubuntu:~/hc$ kubectl exec -it $(kubectl get pods -o name | head -1 | awk -F '/' '{print
$2}') \
-- sh -c "rm /etc/nginx/conf.d/default.conf && nginx -s reload"

2020/01/08 21:46:40 [notice] 15#15: signal process started

user@ubuntu:~/hc$
```

Now redisplay the events for the pod:

```
user@ubuntu:~/hc$ kubectl get events --sort-by='{metadata.creationTimestamp}' \
| grep $(kubectl get pods -o name | head -1 | awk -F '/' '{print $2}')

66s         Normal    SuccessfulCreate          replicaset/nginx-8cdb45866      Created pod:
nginx-8cdb45866-9p8ws
<unknown>   Normal    Scheduled                 pod/nginx-8cdb45866-9p8ws       Successfully
assigned default/nginx-8cdb45866-9p8ws to ubuntu
63s         Normal    Pulling                   pod/nginx-8cdb45866-9p8ws       Pulling image
"nginx:latest"
57s         Normal    Pulled                    pod/nginx-8cdb45866-9p8ws       Successfully
pulled image "nginx:latest"
57s         Normal    Created                   pod/nginx-8cdb45866-9p8ws       Created
container nginx
57s         Normal    Started                   pod/nginx-8cdb45866-9p8ws       Started
container nginx
```

```
5s          Warning   Unhealthy                 pod/nginx-8cdb45866-9p8ws       Liveness probe
failed: Get http://10.32.0.6:80/: dial tcp 10.32.0.6:80: connect: connection refused

user@ubuntu:~/hc$
```

What happened?

Events reported by the event stream are not as granular as those provided by the describe, try it:

```
user@ubuntu:~/hc$ kubectl describe pod \
$(kubectl get pods -o name | head -1 | awk -F '/' '{print $2}') | grep -A 15 Events

Events:
  Type     Reason     Age                 From               Message
  ----     ------     ----                ----               -------
  Normal   Scheduled  <unknown>           default-scheduler  Successfully assigned default/nginx-
8cdb45866-9p8ws to ubuntu
  Warning  Unhealthy  19s (x3 over 39s)   kubelet, ubuntu    Liveness probe failed: Get
http://10.32.0.6:80/: dial tcp 10.32.0.6:80: connect: connection refused
  Normal   Killing    19s                 kubelet, ubuntu    Container nginx failed liveness
probe, will be restarted
  Normal   Pulling    18s (x2 over 87s)   kubelet, ubuntu    Pulling image "nginx:latest"
  Normal   Pulled     17s (x2 over 81s)   kubelet, ubuntu    Successfully pulled image
"nginx:latest"
  Normal   Created    17s (x2 over 81s)   kubelet, ubuntu    Created container nginx
  Normal   Started    17s (x2 over 81s)   kubelet, ubuntu    Started container nginx

user@ubuntu:~/hc$
```

As you can see the Liveness probe failed at one point, but recovered. The nginx container in the pod was created, started, found unhealthy, killed, created and started again.

Remove the related resources:

```
user@ubuntu:~/jobs$ kubectl delete deploy/nginx

deployment.apps "nginx" deleted

user@ubuntu:~/jobs$
```

9. Creating a Job

In a previous lab we saw that running a pod standalone works but without an RS the pod will not restart if it crashes. Unfortunately, if we run a batch job in a pod with an RS and the pod completes the task, the RS will start the pod again.

What if we want a pod that runs only once, however, if it or the node it is running on fails before the pod completes successfully, we want the pod to be started again until it does complete successfully. Kubernetes provides a **Job** type for this scenario.

A Job is like an RC/RS that ensures that a pod runes once to completion. Imagine we want to calculate Pi. Not twice, not half of a time, but precisely once. A job would be the perfect way to run a container that calculates Pi. Enter this sample job config to compute Pi:

```
user@ubuntu:~/hc$ cd ~

user@ubuntu:~$ mkdir ~/jobs && cd ~/jobs/

user@ubuntu:~/jobs$ nano myjob.yaml && cat myjob.yaml

apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
```

```
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never

user@ubuntu:~/jobs$
```

The config uses apiVersion “batch/v1". The kind of object we will create is a Job. The Job will have the name “pi”, as per the metadata. The template for the pod the Job we'll create must have a name pi.

The spec for the pod uses a single perl container which will run the command that computes pi. We also set the restart policy to Never.

Now try running your Job:

```
user@ubuntu:~/jobs$ kubectl apply -f myjob.yaml

job.batch/pi created

user@ubuntu:~/jobs$
```

Examine the job:

```
user@ubuntu:~/jobs$ kubectl describe job/pi

Name:           pi
Namespace:      default
Selector:       controller-uid=bc8c99dc-2484-4879-aeae-0b00bccaa29c
Labels:         controller-uid=bc8c99dc-2484-4879-aeae-0b00bccaa29c
                job-name=pi
Annotations:    kubectl.kubernetes.io/last-applied-configuration:
                  {"apiVersion":"batch/v1","kind":"Job","metadata":{"annotations":
{},"name":"pi","namespace":"default"},"spec":{"template":{"metadata":{"nam...
Parallelism:    1
Completions:    1
Start Time:     Wed, 08 Jan 2020 13:48:18 -0800
Pods Statuses:  1 Running / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=bc8c99dc-2484-4879-aeae-0b00bccaa29c
           job-name=pi
  Containers:
   pi:
    Image:      perl
    Port:       <none>
    Host Port:  <none>
    Command:
      perl
      -Mbignum=bpi
      -wle
      print bpi(2000)
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  7s    job-controller  Created pod: pi-2l6j6

user@ubuntu:~/jobs$
```

The `kubectl apply` subcommand processes the job request and runs our pod. Displaying the Job description shows us the name of the pod that ran the Job. We can now dump the logs for the pod to see the result:

```
user@ubuntu:~/jobs$ kubectl logs $(kubectl get jobs -o name)

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211
70679821480865132823066470938446095505822317253594081284811174502841027019385211055596446229489
54930381964428810975665933446128475648233786783165271201909145648566923460348610454326648213393 6
07260249141273724587006606315588174881520920962829254091715364367892590360011330530548820466521 3
84146951941511609433057270365759591953092186117381932611793105118548074462379962749567351885752 7
24891227938183011949129833673362440656643086021394946395224737190702179860943702770539217176293 1
76752384674818467669405132000568127145263560827785771342757789609173637178721468440901224953430 1
46549585371050792279689258923542019956112129021960864034418159813629774771309960518707211349999 9
98372978049951059731732816096318595024459455346908302642522308253344685035261931188171010003137 8
38752886587533208381420617177669147303598253490428755468731159562863882353787593751957781857780 5
32171226806613001927876611195909216420198938095257201065485863278865936153381827968230301952035 3
01852968995773622599413891249721775283479131515574857242454150695950829533116861727855889075098 3
81754637464939319255060400927701671139009848824012858361603563707660104710181942955596198946767 8
37449448255379774726847104047534646208046684259069491293313677028989152104752162056966024058038 1
50193511253382430035587640247496473263914199272604269922796782354781636009341721641219924586315 0
30286182974555706749838505494588586926995690927210797509302955321165344987202755960236480665499 1
19881834797753566369807426542527862551818417574672890977772793800081647060016145249192173217214 7
72350141441973568548161361157352552133475741849468438523323907394143334547762416862518983569485 5
62099219222184272550254256887671790494601653466804988627232791786085784383827967976681454100953 8
83786360950680064225125205117392984896084128488626945604241965285022210661186306744278622039194 9
45047123713786960956364371917287467764657573962413890865832645995813390478027590 1

user@ubuntu:~/jobs$
```

By default, a Job is complete when one Pod runs to successful completion. You can also specify that this needs to happen multiple times by specifying Job spec key *"completions"* with a value greater than 1. You can suggest how many pods should run concurrently by setting Job spec key *"parallelism"* to the number of pods you would like to have running concurrently (the value defaults to "completions".) The parallelism key is just a hint and the Job may run fewer or more concurrent pods.

Jobs are complementary to Deployments. A Deployment manages pods which are not expected to terminate (e.g. web servers,) and a Job manages pods that are expected to terminate (e.g. batch jobs.)

Complete job pods and the jobs themselves remain after completion, so their logs can be called at any time.

Check the pods and jobs on your cluster:

```
user@ubuntu:~/jobs$ kubectl get pods,jobs

NAME           READY   STATUS      RESTARTS   AGE
pod/pi-2l6j6   0/1     Completed   0          74s

NAME           COMPLETIONS   DURATION   AGE
job.batch/pi   1/1           52s        74s

user@ubuntu:~/jobs$
```

It is up to you to remove the job at your discretion. The only way to remove the job pod is by removing the job itself, so When you are finished exploring remove the Job:

```
user@ubuntu:~/jobs$ kubectl delete job pi

job.batch "pi" deleted

user@ubuntu:~/jobs$
```

```
user@ubuntu:~/jobs$ kubectl get deploy,rs,pods,job

No resources found in default namespace.

user@ubuntu:~/jobs$ cd ~

user@ubuntu:~$
```

Congratulations you have completed the lab!