Kubernetes

Deployments, ReplicaSets, & Labs

In this lab we will explore the nature of Kubernetes Deployments and ReplicaSets and how to work with them.

Deployments

A Deployment controller provides declarative updates for pods and ReplicaSets. 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. Typical uses:

- Bring up a ReplicaSet and (indirectly) its pods
- Capture the results and status of an application deployment
- Update an existing deployment to recreate pods with a new app version image (rolling update)
- Roll back to an earlier deployment revision if the current deployment isn't stable
- Pause and resume a deployment's rollout/rollback

ReplicaSets

ReplicaSets (RS) ensure that a specified number of pod "replicas" are running at all times. If there are too many, it will delete some. If there are too few, it will start more. Unlike the case where a user directly creates pods, a ReplicaSet replaces pods that are deleted or terminated, in cases like node failure or disruptive node maintenance (e.g. a kernel upgrade, etc.). A ReplicaSet is recommended even if your application requires only a single pod.

ReplicaSets can be used independently, however, they are mainly used by Deployment controllers as a mechanism to orchestrate pod creation, deletion, updates, and rollbacks. When you use Deployments you do not have to worry about managing the ReplicaSets that they create; Deployment controllers own and manage their ReplicaSets.

A Deployment/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.

In this lab we will create some Deployments/ReplicaSets, trigger a rolling update to understand controller behaviors, and examine the one-to-many relationship between Deployments and 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. First create a base manifest file in yaml with the following imperative command:

```
~$ mkdir -p ~/dep ; cd ~/dep

~/dep$
```

Unlike pods, deployments are created using `kubectl create deploy`.

Prepare the deployment manifest. If you created it with `kubectl create deployment`, make the following changes:

- Add the `targetenv: demo` label to the deployment metadata
- Add the `targetenv: demo` label to the deployment selector and pod template spec
- Name the container `podweb`

```
~/dep$ nano mydep.yaml && cat $_
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: website
    targetenv: demo          # Add this
  name: website
spec:
  replicas: 3
  selector:
    matchLabels:
      app: website
      targetenv: demo
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: website
        targetenv: demo      # Add this
    spec:
      containers:
```

```
      - image: nginx:1.7.9
        name: podweb          # Change this
        ports:
        - containerPort: 80
        resources: {}
status: {}
```

```
~/dep$
```

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

```
~/dep$ kubectl apply -f mydep.yaml

deployment.apps/website created

~/dep$
```

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

```
~/dep$ kubectl get deploy,rs,po

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/website   3/3     3            3           33s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/website-74fcfd87c8   3         3         3       33s

NAME                           READY   STATUS    RESTARTS   AGE
pod/website-74fcfd87c8-5hq7b   1/1     Running   0          33s
pod/website-74fcfd87c8-fpf67   1/1     Running   0          33s
pod/website-74fcfd87c8-zm8wz   1/1     Running   0          33s

~/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:

```
~/dep$ kubectl get events --sort-by=metadata.creationTimestamp | grep
website
```

```
6s          Normal   SuccessfulCreate    replicaset/website-86895ff58d
Created pod: website-86895ff58d-4h6dz
6s          Normal   Scheduled           pod/website-86895ff58d-dz9wd
Successfully assigned default/website-86895ff58d-dz9wd to ip-172-31-4-161
6s          Normal   Scheduled           pod/website-86895ff58d-r7c5m
Successfully assigned default/website-86895ff58d-r7c5m to ip-172-31-4-161
6s          Normal   Scheduled           pod/website-86895ff58d-4h6dz
Successfully assigned default/website-86895ff58d-4h6dz to ip-172-31-4-161
6s          Normal   SuccessfulCreate    replicaset/website-86895ff58d
Created pod: website-86895ff58d-r7c5m
6s          Normal   SuccessfulCreate    replicaset/website-86895ff58d
Created pod: website-86895ff58d-dz9wd
6s          Normal   ScalingReplicaSet   deployment/website
Scaled up replica set website-86895ff58d to 3
5s          Normal   Pulling             pod/website-86895ff58d-4h6dz
Pulling image "nginx:1.7.9"
5s          Normal   Pulling             pod/website-86895ff58d-dz9wd
Pulling image "nginx:1.7.9"
5s          Normal   Pulling             pod/website-86895ff58d-r7c5m
Pulling image "nginx:1.7.9"

~/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 ReplicaSet 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 by using the pod displayed on the last line of `kubectl get pods -o name`:

```
~/dep$ kubectl delete $(kubectl get po -o name -l app=website | tail -1)

pod "website-86895ff58d-r7c5m" deleted

~/dep$ kubectl get po

NAME                       READY   STATUS    RESTARTS   AGE
website-86895ff58d-4h6dz   1/1     Running   0          57s
website-86895ff58d-dz9wd   1/1     Running   0          57s
website-86895ff58d-hw4jf   1/1     Running   0          39s

~/dep$
```

You might ask: "why would Kubernetes let someone delete a 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

To actually terminate our pod permanently we must delete the Deployment which controls the ReplicaSet, which in turn, controls the pods.

2. Labels

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:

```
~/dep$ kubectl get po --show-labels

NAME                       READY   STATUS    RESTARTS   AGE   LABELS
website-74fcfd87c8-5hq7b   1/1     Running   0          74s
app=website,pod-template-hash=74fcfd87c8,targetenv=demo
website-74fcfd87c8-bf5g6   1/1     Running   0          23s
app=website,pod-template-hash=74fcfd87c8,targetenv=demo
website-74fcfd87c8-fpf67   1/1     Running   0          74s
app=website,pod-template-hash=74fcfd87c8,targetenv=demo

~/dep$
```

To illustrate how labels work, run pods with the same label keys but different values:

```
~/dep$ kubectl run cache-pod -l app=cache --image redis

pod/cache-pod created

~/dep$ kubectl run dev-pod -l targetenv=dev --image httpd:2.2

pod/dev-pod created

~/dep$
```

Now try filtering by the "`app`" label key we assigned to all of our pods in the pod template metadata:

```
~/dep$ kubectl get po --show-labels -l app

NAME                       READY   STATUS    RESTARTS   AGE   LABELS
cache-pod                  1/1     Running   0          12s   app=cache
website-86895ff58d-4h6dz   1/1     Running   0          82s
app=website,pod-template-hash=86895ff58d,targetenv=demo
website-86895ff58d-dz9wd   1/1     Running   0          82s
app=website,pod-template-hash=86895ff58d,targetenv=demo
```

```
website-86895ff58d-hw4jf   1/1     Running   0          64s
app=website,pod-template-hash=86895ff58d,targetenv=demo

~/dep$
```

Now try the "`targetenv`" key:

```
~/dep$ kubectl get po --show-labels -l targetenv

NAME                       READY   STATUS    RESTARTS   AGE    LABELS
dev-pod                    1/1     Running   0          24s
targetenv=dev
website-86895ff58d-4h6dz   1/1     Running   0          102s
app=website,pod-template-hash=86895ff58d,targetenv=demo
website-86895ff58d-dz9wd   1/1     Running   0          102s
app=website,pod-template-hash=86895ff58d,targetenv=demo
website-86895ff58d-hw4jf   1/1     Running   0          84s
app=website,pod-template-hash=86895ff58d,targetenv=demo

~/dep$
```

You can also filter pods by the label key and a specific value:

```
~/dep$ kubectl get po -l app=website

NAME                       READY   STATUS    RESTARTS   AGE
website-86895ff58d-4h6dz   1/1     Running   0          111s
website-86895ff58d-dz9wd   1/1     Running   0          111s
website-86895ff58d-hw4jf   1/1     Running   0          93s

~/dep$ kubectl get po -l targetenv=demo

NAME                       READY   STATUS    RESTARTS   AGE
website-86895ff58d-4h6dz   1/1     Running   0          115s
website-86895ff58d-dz9wd   1/1     Running   0          115s
website-86895ff58d-hw4jf   1/1     Running   0          97s

~/dep$
```

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

```
~/dep$ kubectl describe $(kubectl get po -l app=website -o name | head -1)
| grep -A2 -i label

Labels:           app=website
                  pod-template-hash=86895ff58d
                  targetenv=demo
```

```
~/dep$
```

Unfortunately describe doesn't allow for JSON output. Remember though, `get` does:

```
~/dep$ kubectl get $(kubectl get po -l app=website -o name | head -1) \
-o jsonpath='{.metadata.labels}' ; echo

{"app":"website","pod-template-hash":"86895ff58d","targetenv":"demo"}

~/dep$
```

Delete the pods before continuing:

```
~/dep$ kubectl delete po cache-pod dev-pod

pod "cache-pod" deleted
pod "dev-pod" deleted

~/dep$
```

3. Checking status of a Deployment

We have seen previously how to check the status of a deployment using `kubectl get`:

```
~/dep$ kubectl get deploy

NAME      READY   UP-TO-DATE   AVAILABLE   AGE
website   3/3     3            3           4m18s

~/dep$
```

Now we take an slightly more application-centric view using `kubectl rollout`:

```
~/dep$ kubectl rollout status deployment website

deployment "website" successfully rolled out

~/dep$
```

Rollouts are used to update a given set of Pods, the ones controlled by this Deployment's ReplicaSet. 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`

4. Updating a Deployment

We are using `nginx:1.7.9` in our example, let's update the container named `podweb` in our deployment to `nginx:1.9.1`:

```
~/dep$ kubectl set image deploy website podweb=nginx:1.9.1

deployment.apps/website image updated

~/dep$
```

Imperative commands like `kubectl set` are useful for making ad-hoc changes to resources. An alternative is to use `kubectl edit deployment/website` and edit the `spec.template.spec.containers.image` key.

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

```
~/dep$ kubectl rollout status deployment 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: 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

~/dep$
```

In a rolling update strategy, pods are gradually replaced with new versions. Check the deployment for any changes:

```
~/dep$ kubectl get deploy website

NAME      READY   UP-TO-DATE   AVAILABLE   AGE
website   3/3     3            3           4m1s

~/dep$
```

No obvious changes or replacements made to the deployment. Now look at the Replica Sets & Pods:

```
~/dep$ kubectl get rs,po

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/website-76bc5bdf5d   3         3         3       38s
replicaset.apps/website-86895ff58d   0         0         0       3m26s

NAME                           READY   STATUS    RESTARTS   AGE
pod/website-76bc5bdf5d-q9vsz   1/1     Running   0          23s
pod/website-76bc5bdf5d-x8d2j   1/1     Running   0          22s
pod/website-76bc5bdf5d-zxz2x   1/1     Running   0          38s

~/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:

```
~/dep$ kubectl describe deploy website | grep -A 15 Events

Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  3m37s  deployment-controller  Scaled up
replica set website-86895ff58d to 3
  Normal  ScalingReplicaSet  49s    deployment-controller  Scaled up
replica set website-76bc5bdf5d to 1
  Normal  ScalingReplicaSet  34s    deployment-controller  Scaled down
replica set website-86895ff58d to 2 from 3
  Normal  ScalingReplicaSet  34s    deployment-controller  Scaled up
replica set website-76bc5bdf5d to 2 from 1
  Normal  ScalingReplicaSet  33s    deployment-controller  Scaled down
replica set website-86895ff58d to 1 from 2
  Normal  ScalingReplicaSet  33s    deployment-controller  Scaled up
replica set website-76bc5bdf5d to 3 from 2
  Normal  ScalingReplicaSet  32s    deployment-controller  Scaled down
replica set website-86895ff58d to 0 from 1
```

```
~/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 a different RS.

5. Manually rolling back a Deployment

Let's manually revert back to `nginx:1.7.9` with `kubectl set image`:

```
~/dep$ kubectl set image deploy website podweb=nginx:1.7.9

deployment.apps/website image updated

~/dep$
```

And check the status with `kubectl rollout status`:

```
~/dep$ kubectl rollout status deployment website

deployment "website" successfully rolled out

~/dep$
```

And see what happened to the ReplicaSets:

```
~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY   AGE
website-76bc5bdf5d   0         0         0       76s
website-86895ff58d   3         3         3       4m4s

~/dep$
```

Notice which ReplicaSet hash in the pod name is being used. Since you used this image before, no new ReplicaSet is created. The nginx:1.9.1 ReplicaSet is scaled down, and the 1.7.9 ReplicaSet is scaled back up.

Check the pods now (using the `-l app=website` option to filter your results to just the deployment's pods):

```
~/dep$ kubectl get po -l app=website

NAME                       READY   STATUS    RESTARTS   AGE
website-74fcfd87c8-7b4vj   1/1     Running   0          18s
website-74fcfd87c8-jw8lm   1/1     Running   0          16s
```

```
website-74fcfd87c8-r7cxh   1/1     Running   0          13s

~/dep$
```

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

Confirm your observations once again in the event log:

```
~/dep$ kubectl describe deploy website | grep -A 15 Events

Events:
  Type    Reason             Age                From                   Message
  ----    ------             ----               ----                   ---
  Normal  ScalingReplicaSet  4m16s              deployment-controller  Scaled up replica set website-86895ff58d to 3
  Normal  ScalingReplicaSet  88s                deployment-controller  Scaled up replica set website-76bc5bdf5d to 1
  Normal  ScalingReplicaSet  73s                deployment-controller  Scaled down replica set website-86895ff58d to 2 from 3
  Normal  ScalingReplicaSet  73s                deployment-controller  Scaled up replica set website-76bc5bdf5d to 2 from 1
  Normal  ScalingReplicaSet  72s                deployment-controller  Scaled down replica set website-86895ff58d to 1 from 2
  Normal  ScalingReplicaSet  72s                deployment-controller  Scaled up replica set website-76bc5bdf5d to 3 from 2
  Normal  ScalingReplicaSet  71s                deployment-controller  Scaled down replica set website-86895ff58d to 0 from 1
  Normal  ScalingReplicaSet  19s                deployment-controller  Scaled up replica set website-86895ff58d to 1 from 0
  Normal  ScalingReplicaSet  18s                deployment-controller  Scaled down replica set website-76bc5bdf5d to 2 from 3
  Normal  ScalingReplicaSet  16s (x4 over 18s)  deployment-controller  (combined from similar events): Scaled down replica set website-76bc5bdf5d to 0 from 1

~/dep$
```

So when the image in a controller's template changes, the replicasets are merely scaled up or down according to the demands of the deployment spec.

6. Checking rollout history of a Deployment

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

```
~/dep$ kubectl rollout history deploy website

deployment.apps/website
```

```
REVISION  CHANGE-CAUSE
2         <none>
3         <none>

~/dep$
```

The rollout history, by default, retains 10 versions of a resource (the default value of `revisionHistoryLimit` in a Deployment spec). You will only get the `CHANGE-CAUSE` next to a revision if you add an annotation with the `kubernetes.io/change-cause` key. It is a useful tool for performing an immediate rollback of an update, but for more significant changes (such as rolling back several previous versions) a proper CI/CD pipeline and version control integration is more appropriate.

To gain insight on what changed in a revision, you can use the `--revision` flag to inspect a specific revision.

Take a detailed look at a previous deployment version:

```
~/dep$ kubectl rollout history deploy website --revision=2

deployment.apps/website with revision #2
Pod Template:
  Labels:       app=website
        pod-template-hash=76bc5bdf5d
        targetenv=demo
  Containers:
   podweb:
    Image:      nginx:1.9.1
    Port:       80/TCP
    Host Port:  0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>

~/dep$
```

Each rollout history entry has a pod and container spec within it. If a change causes the overall state of a the deployment to match one of the previous rollout states, then Kubernetes uses that revision.

7. Rolling back to a previous Deployment

We previously performed a manual rollback using `kubectl set`. Another method is to use the `kubectl rollout undo` subcommand, which automatically selects the previous revision present in a resource's rollout history to roll back to.

Confirm the current version of a container is 1.7.9 by feeding the name of a pod (`kubectl get po -l app=website -o name | tail -1`) to a second `get` command that parses the pod's manifest for its image (`kubectl get pod/<podname> -o jsonpath='{.spec.containers[].image}`):

```
~/dep$ kubectl get $(kubectl get po -l app=website -o name | tail -1) \
-o jsonpath='{.spec.containers[].image}' ; echo

nginx:1.7.9

~/dep$
```

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

```
~/dep$ kubectl rollout undo deploy website && kubectl rollout status
deployment 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

~/dep$
```

You can specify a revision to roll back to using the `--to-revision` flag. If you use that flag, your command would be: `kubectl rollout undo deployment/website --to-revision=2`

```
~/dep$ kubectl get deploy website

NAME      READY   UP-TO-DATE   AVAILABLE   AGE
website   3/3     3            3           6m57s

~/dep$
```

Check the events in the website deployment's history:

```
~/dep$ kubectl describe deploy website | grep -A 15 Events

Events:
  Type    Reason             Age                From                   
Message
  ----    ------             ----               ----                   --
-----
  Normal  ScalingReplicaSet  5m19s              deployment-controller  
Scaled up replica set website-86895ff58d to 3
  Normal  ScalingReplicaSet  2m31s              deployment-controller  
Scaled up replica set website-76bc5bdf5d to 1
  Normal  ScalingReplicaSet  2m16s              deployment-controller  
Scaled down replica set website-86895ff58d to 2 from 3
  Normal  ScalingReplicaSet  2m16s              deployment-controller  
Scaled up replica set website-76bc5bdf5d to 2 from 1
  Normal  ScalingReplicaSet  2m15s              deployment-controller  
Scaled up replica set website-76bc5bdf5d to 3 from 2
  Normal  ScalingReplicaSet  2m14s              deployment-controller  
Scaled down replica set website-86895ff58d to 0 from 1
  Normal  ScalingReplicaSet  82s                deployment-controller  
Scaled up replica set website-86895ff58d to 1 from 0
  Normal  ScalingReplicaSet  81s                deployment-controller  
Scaled down replica set website-76bc5bdf5d to 2 from 3
  Normal  ScalingReplicaSet  9s (x2 over 2m15s)  deployment-controller  
Scaled down replica set website-86895ff58d to 1 from 2
  Normal  ScalingReplicaSet  8s (x9 over 81s)    deployment-controller  
(combined from similar events): Scaled down replica set website-86895ff58d
to 0 from 1

~/dep$
```

Confirm the container image version has been reverted to 1.9.1:

```
~/dep$ kubectl get $(kubectl get po -l app=website -o name | tail -1) \
-o jsonpath='{.spec.containers[].image}' && echo

nginx:1.9.1

~/dep$
```

The ability to perform rollbacks to a previous version or last known good version is an essential ability that will keep your services and applications in top production shape.

8. Pausing and resuming a Deployment

In a larger installation, we may be deploying dozens of pods. We have the option of pausing a rollout, which allows multiple versions of a deployment to exist. This enables users to perform rudimentary canary testing by enabling some amount of traffic to go to a new pod and observing the behavior. If the behavior is acceptable, the rollout may be unpaused and completed.

Try to pause a rollout. For our small test it is hard to pause in time, so chain the commands to hopefully catch it in the act:

```
~/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

~/dep$
```

```
~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY   AGE
website-76bc5bdf5d   3         3         3       2m52s
website-86895ff58d   1         1         1       5m40s

~/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:

```
~/dep$ kubectl rollout status deployment website

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

^C

~/dep$
```

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

```
~/dep$ kubectl rollout resume deploy website && kubectl rollout status
deployment 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: 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

~/dep$
```

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

```
~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY   AGE
website-76bc5bdf5d   0         0         0       3m18s
website-86895ff58d   3         3         3       6m6s

~/dep$
```

There are no more nginx pods running version 1.9.1, only 1.7.9.

Delete your deployment:

```
~/dep$ kubectl delete deploy website

deployment.apps "website" deleted

~/dep$
```

9. Clean Up

Use `kubectl get all` to view any remaining deployments or pods and make sure to delete all of those remaining resources. Only the `kubernetes` service should remain:

```
~/dep$ kubectl get all

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   143m

~/dep$
```

10.1. CHALLENGE: labels

Recreate the website deployment, the cache-pod, and dev-pod resources from earlier in the lab:

```
~/dep$ cd ~/dep

~/dep$ kubectl apply -f ~/dep/mydep.yaml

deployment.apps/website created

~/dep$ kubectl run cache-pod -l app=cache --image redis

pod/cache-pod created

~/dep$ kubectl run dev-pod -l targetenv=dev --image httpd:2.2

pod/dev-pod created

~/dep$
```

- Run a pod named `labelpod` with the label `targetenv=prod` and a container from `nginx:1.7.9`
- 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 app key set to website
- Delete the pods you created for this challenge at the end

10.2. Challenge: Deployments

- Create a deployment named `webserver-special`
    - Ensure the deployment is labeled with `app=commerce`, `vendor=student-tech`, `component=webserver`
    - The deployment should maintain 3 pods that have the labels `component=webserver` and `server=nginx` present
    - The containers of pods in the deployment should be created from the `nginx:1.20.1` image
- After creating the deployment, modify it so its pods:
    - run from the `nginx:1.23.2` image
    - Have the environment variable `HTTPD_PROXY=main`
    - Have 5 copies instead of 3
- What is different about the pods now in `kubectl get`?
- Delete the deployment when finished

Congratulations you have completed the lab!