RX-M Cloud Native Consulting

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:

```
ubuntu@ip-10-0-2-200:~$ cd ~

ubuntu@ip-10-0-2-200:~$ mkdir dep

ubuntu@ip-10-0-2-200:~$ cd dep

ubuntu@ip-10-0-2-200:~/dep$

ubuntu@ip-10-0-2-200:~/dep$ vi mydep.yaml

ubuntu@ip-10-0-2-200:~/dep$ 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
ubuntu@ip-10-0-2-200:~/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/v1.13/#deployment-v1-apps

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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl create -f mydep.yaml

deployment.apps/website created
```

```
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get deploy,rs,pods

NAME                            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/website   3         3         3            3           13s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.extensions/website-6dc99878b   3         3         3       13s

NAME                          READY   STATUS    RESTARTS   AGE
pod/website-6dc99878b-4w9tv   1/1     Running   0          13s
pod/website-6dc99878b-8wh9l   1/1     Running   0          13s
pod/website-6dc99878b-q5jqx   1/1     Running   0          13s
ubuntu@ip-10-0-2-200:~/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:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get events --sort-by='{.lastTimestamp}' |
grep website

55s         Normal   ScalingReplicaSet   Deployment   Scaled up replica set
website-6dc99878b to 3
55s         Normal   SuccessfulCreate    ReplicaSet   Created pod: website-
6dc99878b-q5jqx
54s         Normal   SuccessfulCreate    ReplicaSet   Created pod: website-
6dc99878b-8wh9l
54s         Normal   SuccessfulCreate    ReplicaSet   Created pod: website-
6dc99878b-4w9tv
54s         Normal   Scheduled           Pod          Successfully assigned
default/website-6dc99878b-8wh9l to ip-10-0-2-200
54s         Normal   Scheduled           Pod          Successfully assigned
default/website-6dc99878b-q5jqx to ip-10-0-2-200
54s         Normal   Scheduled           Pod          Successfully assigned
default/website-6dc99878b-4w9tv to ip-10-0-2-200
ubuntu@ip-10-0-2-200:~/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`).

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl delete pod website-6dc99878b-q5jqx

pod "website-6dc99878b-q5jqx" deleted
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods

NAME                         READY   STATUS    RESTARTS   AGE
website-6dc99878b-4w9tv      1/1     Running   0          33s
website-6dc99878b-8wh9l      1/1     Running   0          33s
website-6dc99878b-l9rmz      1/1     Running   0          22s
```

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 given pod. Perhaps it has problems and you want to generate a new replacement. Perhaps the current node has problems and you want Kubernetes to reschedule this particular pod somewhere else. 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:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods

NAME                         READY   STATUS    RESTARTS   AGE
website-6dc99878b-4w9tv      1/1     Running   0          85s
website-6dc99878b-8wh9l      1/1     Running   0          85s
website-6dc99878b-l9rmz      1/1     Running   0          74s
ubuntu@ip-10-0-2-200:~/dep$
```

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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods -l appname

NAME                         READY   STATUS    RESTARTS   AGE
website-6dc99878b-4w9tv      1/1     Running   0          97s
website-6dc99878b-8wh9l      1/1     Running   0          97s
website-6dc99878b-l9rmz      1/1     Running   0          86s
ubuntu@ip-10-0-2-200:~/dep$
```

You can also filter by key and value:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods -l appname=webserver

NAME                         READY   STATUS    RESTARTS   AGE
website-6dc99878b-4w9tv      1/1     Running   0          113s
website-6dc99878b-8wh9l      1/1     Running   0          113s
website-6dc99878b-l9rmz      1/1     Running   0          102s
ubuntu@ip-10-0-2-200:~/dep$
```

You can filter by pod name:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get $(kubectl get pods -o name | head -1)

NAME                        READY   STATUS    RESTARTS   AGE
website-6dc99878b-4w9tv     1/1     Running   0          2m6s
ubuntu@ip-10-0-2-200:~/dep$
```

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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl describe $(kubectl get pods -o name | head
-1) | grep -A2 -i label

Labels:         appname=webserver
                pod-template-hash=6dc99878b
                targetenv=demo
ubuntu@ip-10-0-2-200:~/dep$
```

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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get $(kubectl get pods -o name | head -1) -o
json | jq .metadata.labels
```

```
{
  "appname": "webserver",
  "pod-template-hash": "6dc99878b",
  "targetenv": "demo"
}
```

```
ubuntu@ip-10-0-2-200:~/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.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get deploy

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
website   3         3         3            3           2m
ubuntu@ip-10-0-2-200:~/dep$
```

Now we take an slightly more application-centric view.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl rollout status deploy/website

deployment "website" successfully rolled out
ubuntu@ip-10-0-2-200:~/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.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl set image deploy/website podweb=nginx:1.9.1 --record

deployment.extensions/website image updated
ubuntu@ip-10-0-2-200:~/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):

```
ubuntu@ip-10-0-2-200:~/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
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get deploy/website

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
website   3         3         3            3           5m
ubuntu@ip-10-0-2-200:~/dep$
```

Look at the Replica Sets & Pods

```
ubuntu@ip-10-0-2-200:~/dep$  kubectl get rs,pod

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.extensions/website-654d96bc8d   3         3         3       51s
replicaset.extensions/website-6dc99878b    0         0         0       5m13s

NAME                           READY   STATUS    RESTARTS   AGE
pod/website-654d96bc8d-2r225   1/1     Running   0          43s
pod/website-654d96bc8d-ckqlg   1/1     Running   0          45s
pod/website-654d96bc8d-l8jr8   1/1     Running   0          51s
ubuntu@ip-10-0-2-200:~/dep$
```

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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl describe deploy/website | grep -A 10 Events

Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  5m32s  deployment-controller  Scaled up replica set
website-6dc99878b to 3
  Normal  ScalingReplicaSet  70s    deployment-controller  Scaled up replica set
website-654d96bc8d to 1
  Normal  ScalingReplicaSet  64s    deployment-controller  Scaled down replica
set website-6dc99878b to 2
  Normal  ScalingReplicaSet  64s    deployment-controller  Scaled up replica set
website-654d96bc8d to 2
  Normal  ScalingReplicaSet  62s    deployment-controller  Scaled down replica
set website-6dc99878b to 1
  Normal  ScalingReplicaSet  62s    deployment-controller  Scaled up replica set
website-654d96bc8d to 3
  Normal  ScalingReplicaSet  60s    deployment-controller  Scaled down replica
set website-6dc99878b to 0
ubuntu@ip-10-0-2-200:~/dep$
```

Note that the rollout was a smooth transition from one set of Pods controlled by our original ReplicaSet `website-6dc99878b` to our second set of Pods controlled by the RS `website-6dc99878b`.

4. Manually rolling back a deployment

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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl set image deploy/website podweb=nginx:1.7.9 -
-record

deployment.extensions/website image updated
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/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
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY   AGE
website-654d96bc8d   0         0         0       2m49s
website-6dc99878b    3         3         3       7m11s
ubuntu@ip-10-0-2-200:~/dep$
```

Notice which deployment (NAME) is being used.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods

NAME                      READY   STATUS    RESTARTS   AGE
website-6dc99878b-98wqm   1/1     Running   0          59s
website-6dc99878b-cfd5j   1/1     Running   0          55s
website-6dc99878b-mr684   1/1     Running   0          57s
ubuntu@ip-10-0-2-200:~/dep$
```

Confirm your observations once again in the event log.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl describe deploy/website | grep -A 15 Events

Events:
  Type    Reason             Age                    From                   Message
  ----    ------             ----                   ----                   -------
  Normal  ScalingReplicaSet  3m26s                  deployment-controller  Scaled
up replica set website-654d96bc8d to 1
  Normal  ScalingReplicaSet  3m20s                  deployment-controller  Scaled
down replica set website-6dc99878b to 2
  Normal  ScalingReplicaSet  3m20s                  deployment-controller  Scaled
up replica set website-654d96bc8d to 2
  Normal  ScalingReplicaSet  3m18s                  deployment-controller  Scaled
down replica set website-6dc99878b to 1
```

```
  Normal  ScalingReplicaSet  3m18s                deployment-controller  Scaled
up replica set website-654d96bc8d to 3
  Normal  ScalingReplicaSet  3m16s                deployment-controller  Scaled
down replica set website-6dc99878b to 0
  Normal  ScalingReplicaSet  72s                  deployment-controller  Scaled
up replica set website-6dc99878b to 1
  Normal  ScalingReplicaSet  70s                  deployment-controller  Scaled
down replica set website-654d96bc8d to 2
  Normal  ScalingReplicaSet  68s (x2 over 7m48s)  deployment-controller  Scaled
up replica set website-6dc99878b to 3
  Normal  ScalingReplicaSet  66s (x3 over 70s)    deployment-controller
(combined from similar events): Scaled down replica set website-654d96bc8d to 0
ubuntu@ip-10-0-2-200:~/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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl rollout history deploy/website

deployment.extensions/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
ubuntu@ip-10-0-2-200:~/dep$
```

Take a detailed look at a previous deployment version.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl rollout history deploy/website --revision=2

deployments "website" with revision #2
Pod Template:
  Labels:       appname=webserver
        pod-template-hash=654d96bc8d
        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>

ubuntu@ip-10-0-2-200:~/dep$
```

6. Rolling back to a previous Deployment

Confirm the current version of a container is 1.7.9.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods -o json | jq
.items[0].spec.containers[0].image -r

nginx:1.7.9
ubuntu@ip-10-0-2-200:~/dep$
```

Revert to previous version/revision.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl rollout undo deploy/website

deployment.extensions/website
ubuntu@ip-10-0-2-200:~/dep$
```

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

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get deploy/website

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
website   3         3         3            3           8m
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl describe deploy/website | grep -A 15 Events

Events:
  Type    Reason              Age                   From                    
Message
  ----    ------              ----                  ----                    -----
--
  Normal  ScalingReplicaSet   4m56s                 deployment-controller  
Scaled down replica set website-6dc99878b to 2
  Normal  ScalingReplicaSet   4m54s                 deployment-controller  
Scaled down replica set website-6dc99878b to 1
  Normal  ScalingReplicaSet   4m54s                 deployment-controller  
Scaled up replica set website-654d96bc8d to 3
  Normal  ScalingReplicaSet   2m48s                 deployment-controller  
Scaled up replica set website-6dc99878b to 1
  Normal  ScalingReplicaSet   2m46s                 deployment-controller  
Scaled down replica set website-654d96bc8d to 2
  Normal  ScalingReplicaSet   2m44s (x2 over 9m24s)  deployment-controller  
Scaled up replica set website-6dc99878b to 3
  Normal  ScalingReplicaSet   30s (x2 over 5m2s)    deployment-controller  
Scaled up replica set website-654d96bc8d to 1
  Normal  DeploymentRollback  30s                   deployment-controller  
Rolled back deployment "website" to revision 2
  Normal  ScalingReplicaSet   29s (x2 over 4m56s)   deployment-controller  
Scaled up replica set website-654d96bc8d to 2
  Normal  ScalingReplicaSet   27s (x6 over 2m46s)   deployment-controller  
(combined from similar events): Scaled up replica set website-654d96bc8d to 3
```

```
  Normal  ScalingReplicaSet   25s (x2 over 4m52s)    deployment-controller
Scaled down replica set website-6dc99878b to 0
ubuntu@ip-10-0-2-200:~/dep$
```

Note the unique event in the log for the rollback: `DeploymentRollback`

Confirm the container image version has been reverted to 1.9.1:

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get pods -o json | jq
.items[0].spec.containers[0].image -r

nginx:1.9.1
ubuntu@ip-10-0-2-200:~/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.

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl set image deploy/website podweb=nginx:1.7.9;
kubectl rollout pause deploy/website

deployment.extensions/website image updated
deployment.extensions/website paused
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get rs

NAME                 DESIRED   CURRENT   READY   AGE
website-654d96bc8d   3         3         3       6m22s
website-6dc99878b    1         1         1       10m
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl rollout status deploy/website

Waiting for deployment "website" rollout to finish: 1 out of 3 new replicas have
been updated...
^C
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl rollout resume deploy/website

deployment.extensions/website resumed
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl rollout status deploy/website

deployment "website" successfully rolled out
ubuntu@ip-10-0-2-200:~/dep$
```

```
ubuntu@ip-10-0-2-200:~/dep$ kubectl get rs

NAME                     DESIRED   CURRENT   READY   AGE
website-654d96bc8d   0         0         0       7m28s
website-6dc99878b    3         3         3       11m
ubuntu@ip-10-0-2-200:~/dep$
```

Delete your deployment.

8. Health Checks

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

```
ubuntu@ip-10-0-2-200:~/dep$ cd ~

ubuntu@ip-10-0-2-200:~$ mkdir hc

ubuntu@ip-10-0-2-200:~$ cd hc

ubuntu@ip-10-0-2-200:~/hc$ vi hc.yaml

ubuntu@ip-10-0-2-200:~/hc$ 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
ubuntu@ip-10-0-2-200:~/hc$
```

Now run the deployment:

```
ubuntu@ip-10-0-2-200:~/hc$ kubectl create -f hc.yaml

deployment.apps/nginx created
ubuntu@ip-10-0-2-200:~/hc$
```

View your deployment:

```
ubuntu@ip-10-0-2-200:~/hc$ kubectl get deploy,rs,pods

NAME                          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/nginx   3         3         3            3           14s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.extensions/nginx-5fb57bb9c9   3         3         3       14s

NAME                         READY   STATUS    RESTARTS   AGE
pod/nginx-5fb57bb9c9-b6vh2   1/1     Running   0          14s
pod/nginx-5fb57bb9c9-mf599   1/1     Running   0          14s
pod/nginx-5fb57bb9c9-nxnz2   1/1     Running   0          14s
ubuntu@ip-10-0-2-200:~/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:

```
ubuntu@ip-10-0-2-200:~/hc$ kubectl get events --sort-by='{.lastTimestamp}' \
| grep $(kubectl get pods -o name | head -1 | awk -F '/' '{print $2}')

34s         Normal   Scheduled           Pod          Successfully assigned
default/nginx-5fb57bb9c9-b6vh2 to ip-10-0-2-200
34s         Normal   SuccessfulCreate    ReplicaSet   Created pod: nginx-
5fb57bb9c9-b6vh2
ubuntu@ip-10-0-2-200:~/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.

```
ubuntu@ip-10-0-2-200:~/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"

2018/03/30 00:06:59 [notice] 15#15: signal process started
ubuntu@ip-10-0-2-200:~/hc$
```

Now redisplay the events for the pod:

```
ubuntu@ip-10-0-2-200:~/hc$ kubectl get events --sort-by='{.lastTimestamp}' \
| grep $(kubectl get pods -o name | head -1 | awk -F '/' '{print $2}')

65s         Normal    Scheduled            Pod          Successfully assigned
default/nginx-5fb57bb9c9-b6vh2 to ip-10-0-2-200
65s         Normal    SuccessfulCreate     ReplicaSet   Created pod: nginx-
5fb57bb9c9-45x6s
ubuntu@ip-10-0-2-200:~/hc$
```

What happened?

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

```
ubuntu@ip-10-0-2-200:~/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  4m23s                  default-scheduler        Successfully
assigned default/nginx-5fb57bb9c9-b6vh2 to ip-10-0-2-200
  Warning  Unhealthy  2m56s (x3 over 3m16s)  kubelet, ip-10-0-2-200  Liveness
probe failed: Get http://10.32.0.5:80/: dial tcp 10.32.0.5:80: connect:
connection refused
  Normal   Pulling    2m55s (x2 over 4m22s)  kubelet, ip-10-0-2-200  pulling
image "nginx:latest"
  Normal   Pulled     2m55s (x2 over 4m21s)  kubelet, ip-10-0-2-200  Successfully
pulled image "nginx:latest"
  Normal   Created    2m55s (x2 over 4m20s)  kubelet, ip-10-0-2-200  Created
container
  Normal   Started    2m55s (x2 over 4m20s)  kubelet, ip-10-0-2-200  Started
container
  Normal   Killing    2m55s                  kubelet, ip-10-0-2-200  Killing
container with id docker://nginx:Container failed liveness probe.. Container will
be killed and recreated.
ubuntu@ip-10-0-2-200:~/hc$
```

As you can see the Liveness probe is now failing. The nginx container in the pod was created, started, found unhealthy, killed, created and started again.

Remove the related resources.

```
ubuntu@ip-10-0-2-200:~/jobs$ kubectl delete deploy/nginx

deployment.extensions "nginx" deleted
ubuntu@ip-10-0-2-200:~/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:

```
ubuntu@ip-10-0-2-200:~/hc$ cd ~

ubuntu@ip-10-0-2-200:~$ mkdir jobs

ubuntu@ip-10-0-2-200:~$ cd jobs/

ubuntu@ip-10-0-2-200:~/jobs$ vim myjob.yaml

ubuntu@ip-10-0-2-200:~/jobs$ 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
ubuntu@ip-10-0-2-200:~/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:

```
ubuntu@ip-10-0-2-200:~/jobs$ kubectl create -f myjob.yaml

job.batch/pi created
ubuntu@ip-10-0-2-200:~/jobs$
```

Examine the job:

```
ubuntu@ip-10-0-2-200:~/jobs$ kubectl get deploy,rs,pods,job

NAME           READY   STATUS              RESTARTS   AGE
pod/pi-xzx2h   0/1     ContainerCreating   0          20s

NAME           COMPLETIONS   DURATION   AGE
job.batch/pi   0/1           20s        20s
ubuntu@ip-10-0-2-200:~/jobs$


ubuntu@ip-10-0-2-200:~/jobs$ kubectl get deploy,rs,pods,job

NAME           READY   STATUS      RESTARTS   AGE
pod/pi-xzx2h   0/1     Completed   0          30s

NAME           COMPLETIONS   DURATION   AGE
job.batch/pi   1/1           27s        30s
ubuntu@ip-10-0-2-200:~/jobs
```

```
ubuntu@ip-10-0-2-200:~/jobs$ kubectl describe job/pi

Name:           pi
Namespace:      default
Selector:       controller-uid=59c75687-f9cb-11e8-8781-02d9a858fbbc
Labels:         controller-uid=59c75687-f9cb-11e8-8781-02d9a858fbbc
                job-name=pi
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Fri, 07 Dec 2018 02:54:01 +0000
Completed At:   Fri, 07 Dec 2018 02:54:28 +0000
Duration:       27s
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=59c75687-f9cb-11e8-8781-02d9a858fbbc
           job-name=pi
  Containers:
   pi:
    Image:  perl
    Port:   <none>
    Command:
      perl
      -Mbignum=bpi
      -wle
      print bpi(2000)
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  52s   job-controller  Created pod: pi-xzx2h
ubuntu@ip-10-0-2-200:~/jobs$
```

The `kubectl create` 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:

```
ubuntu@ip-10-0-2-200:~/jobs$ kubectl logs $(kubectl get jobs -o name)

3.141592653589793238462643383279502884197169399375105820974944592307816406286208
998628034825342117067982148086513282306647093844609550582231725359408128481117450
284102701938521105559644622948954930381964428810975665933446128475648233786783165
271201909145648566923460348610454326648213393607260249141273724587006606315588174
881520920962829254091715364367892590360011330530548820466521384146951941511609433
057270365759591953092186117381932611793105118548074462379962749567351885752724891
227938183011949129833673362440656643086021394946395224737190702179860943702770539
217176293176752384674818467669405132000568127145263560827785771342757789609173637
178721468440901224953430146549585371050792279689258923542019956112129021960864034
418159813629774771309960518707211349999998372978049951059731732816096318595024459
455346908302642522308253344685035261931188171010003137838752886587533208381420617
177669147303598253490428755468731159562863882353787593751957781857780532171226806
613001927876611195909216420198938095257201065485863278865936153381827968230301952
035301852968995773622599413891249721775283479131515574857242454150695950829533116
861727855889075098381754637464939319255060400927701671139009848824012858361603563
707660104710181942955596198946767837449448255379774726847104047534646208046684259
069491293313677028989152104752162056966024058038150193511253382430035587640247496
473263914199272604269922796782354781636009341721641219924586315030286182974555706
749838505494588586926995690927210797509302955321165344987202755960236480665499119
881834797753566369807426542527862551818417574672890977772793800081647060016145249
192173217214772350141441973568548161361157352552133475741849468438523323907394143
334547762416862518983569485562099219222184272550254256887671790494601653466804988
627232791786085784383827967976681454100953883786360950680064225125205117392984896
084128488626945604241965285022210661186306744278622039194945047123713786960956364
371917287467764657573962413890865832645995813390478027589
ubuntu@ip-10-0-2-200:~/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.)

When you are finished exploring remove the Job:

```
ubuntu@ip-10-0-2-200:~/jobs$ kubectl delete job pi

job.batch "pi" deleted
ubuntu@ip-10-0-2-200:~/jobs$
```

```
ubuntu@ip-10-0-2-200:~/jobs$ kubectl get deploy,rs,pods,job

No resources found.
ubuntu@ip-10-0-2-200:~/jobs$
```

Congratulations you have completed the lab!