RX-M Cloud Native Consulting

Kubernetes

Lab 5 – 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

user@ubuntu:~$ cd dep

user@ubuntu:~/dep$

user@ubuntu:~/dep$ vim mydep.yaml

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

apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: website
  labels:
    bu: sales
spec:
  replicas: 3
  template:
    metadata:
      labels:
        appname: webserver
        targetenv: demo
    spec:
      containers:
      - name: podweb
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

The first thing you might notice is that Deployments are a beta resource type. Deployments 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 beta specification for Deployments can be found here:

https://kubernetes.io/docs/api-reference/v1.7/#deployment-v1beta1-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:

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

deployment "website" created

user@ubuntu:~/dep$
```

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

NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/website   3         3         3            3           27s

NAME                    DESIRED   CURRENT   READY     AGE
rs/website-205870431    3         3         3         27s

NAME                          READY     STATUS    RESTARTS   AGE
po/website-205870431-7cd9k    1/1       Running   0          27s
po/website-205870431-h530h    1/1       Running   0          27s
po/website-205870431-l7czq    1/1       Running   0          27s

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 | grep website

1m         1m          1         website-205870431-7cd9k   Pod                                    Normal
Scheduled           default-scheduler       Successfully assigned website-205870431-7cd9k to ubuntu
1m         1m          1         website-205870431-7cd9k   Pod          spec.containers{podweb}   Normal
Pulling             kubelet, ubuntu         pulling image "nginx:1.7.9"
1m         1m          1         website-205870431-7cd9k   Pod          spec.containers{podweb}   Normal    Pulled
kubelet, ubuntu         Successfully pulled image "nginx:1.7.9"
1m         1m          1         website-205870431-7cd9k   Pod          spec.containers{podweb}   Normal
Created             kubelet, ubuntu         Created container with id
15a14aa6f5faa02c20c65406572a07073ff454ff30a6a65ad8e3baa46be2b514
1m         1m          1         website-205870431-7cd9k   Pod          spec.containers{podweb}   Normal
Started             kubelet, ubuntu         Started container with id
15a14aa6f5faa02c20c65406572a07073ff454ff30a6a65ad8e3baa46be2b514
1m         1m          1         website-205870431-h530h   Pod                                    Normal
Scheduled           default-scheduler       Successfully assigned website-205870431-h530h to ubuntu
1m         1m          1         website-205870431-h530h   Pod          spec.containers{podweb}   Normal
Pulling             kubelet, ubuntu         pulling image "nginx:1.7.9"
1m         1m          1         website-205870431-h530h   Pod          spec.containers{podweb}   Normal    Pulled
kubelet, ubuntu         Successfully pulled image "nginx:1.7.9"
1m         1m          1         website-205870431-h530h   Pod          spec.containers{podweb}   Normal
Created             kubelet, ubuntu         Created container with id
771547aa87b75b6c9debb86a43c1676a09737afefe1c1d13fe8f52d2b6769c1e
1m         1m          1         website-205870431-h530h   Pod          spec.containers{podweb}   Normal
Started             kubelet, ubuntu         Started container with id
771547aa87b75b6c9debb86a43c1676a09737afefe1c1d13fe8f52d2b6769c1e
1m         1m          1         website-205870431-l7czq   Pod                                    Normal
Scheduled           default-scheduler       Successfully assigned website-205870431-l7czq to ubuntu
1m         1m          1         website-205870431-l7czq   Pod          spec.containers{podweb}   Normal
Pulling             kubelet, ubuntu         pulling image "nginx:1.7.9"
1m         1m          1         website-205870431-l7czq   Pod          spec.containers{podweb}   Normal    Pulled
kubelet, ubuntu         Successfully pulled image "nginx:1.7.9"
1m         1m          1         website-205870431-l7czq   Pod          spec.containers{podweb}   Normal
Created             kubelet, ubuntu         Created container with id
294f3a303d660a4771e0f912a21d9758ee62f6b74bcf042445dc75838644d65f
1m         1m          1         website-205870431-l7czq   Pod          spec.containers{podweb}   Normal
Started             kubelet, ubuntu         Started container with id
294f3a303d660a4771e0f912a21d9758ee62f6b74bcf042445dc75838644d65f
1m         1m          1         website-205870431         ReplicaSet                             Normal
```

```
SuccessfulCreate    replicaset-controller   Created pod: website-205870431-h530h
1m         1m          1         website-205870431         ReplicaSet                             Normal
SuccessfulCreate    replicaset-controller   Created pod: website-205870431-7cd9k
1m         1m          1         website-205870431         ReplicaSet                             Normal
SuccessfulCreate    replicaset-controller   Created pod: website-205870431-l7czq
1m         1m          1         website                   Deployment                             Normal
ScalingReplicaSet   deployment-controller   Scaled up replica set website-205870431 to 3

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.

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:

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

NAME                       READY     STATUS    RESTARTS   AGE
website-205870431-7cd9k    1/1       Running   0          2m
website-205870431-h530h    1/1       Running   0          2m
website-205870431-l7czq    1/1       Running   0          2m

user@ubuntu:~/dep$
```

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

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

NAME                       READY     STATUS    RESTARTS   AGE
website-205870431-7cd9k    1/1       Running   0          2m
website-205870431-h530h    1/1       Running   0          2m
website-205870431-l7czq    1/1       Running   0          2m

user@ubuntu:~/dep$
```

You can also filter by key and value:

```
user@ubuntu:~/dep$ kubectl get pods -l appname=webserver

NAME                        READY     STATUS    RESTARTS   AGE
website-205870431-7cd9k     1/1       Running   0          2m
website-205870431-h530h     1/1       Running   0          2m
website-205870431-l7czq     1/1       Running   0          2m

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-205870431-7cd9k     1/1       Running   0          3m

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=2870940145
                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": "2870940145",
  "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      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
website   3         3         3            3           2m

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 "website" image updated

user@ubuntu:~/dep$
```

Alternative is to use `kubectl edit deployment/website`

Check the status of the rollout:

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

deployment "website" successfully rolled out

user@ubuntu:~/dep$
```

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

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
website   3         3         3            3           5m

user@ubuntu:~/dep$
```

Look at the Replica Sets & Pods

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

NAME                    DESIRED   CURRENT   READY     AGE
rs/website-205870431    0         0         0         6m
```

```
rs/website-4061902189   3         3         3         55s

NAME                            READY     STATUS    RESTARTS   AGE
po/website-4061902189-7kz2j   1/1       Running   0          44s
po/website-4061902189-kt956   1/1       Running   0          55s
po/website-4061902189-v11lw   1/1       Running   0          43s

user@ubuntu:~/dep$
```

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

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

Name:                   website
Namespace:              default
CreationTimestamp:      Tue, 12 Sep 2017 22:12:17 -0700
Labels:                 bu=sales
Annotations:            deployment.kubernetes.io/revision=2
Selector:               appname=webserver,targetenv=demo
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:       appname=webserver
                targetenv=demo
  Containers:
   podweb:
    Image:              nginx:1.9.1
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
  Progressing   True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  website-4061902189 (3/3 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason
Message
  ---------     --------        -----   ----                    -------------   --------        ------
-------
  7m            7m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-205870431 to 3
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 1
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 2
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 2
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 1
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 3
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 0

user@ubuntu:~/dep$
```

Note that the rollout was a smooth transition from one set of Pods controlled by our original ReplicaSet `website-205870431` to our second set of Pods controlled by the RS `website-4061902189`.

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 "website" image updated

user@ubuntu:~/dep$
```

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

deployment "website" successfully rolled out
```

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

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

NAME                 DESIRED   CURRENT   READY     AGE
website-205870431    3         3         3         9m
website-4061902189   0         0         0         4m

user@ubuntu:~/dep$
```

Notice which deployment (NAME) is being used.

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

NAME                       READY     STATUS    RESTARTS   AGE
website-205870431-4bwfs   1/1       Running   0          35s
website-205870431-62fv5   1/1       Running   0          38s
website-205870431-x65g5   1/1       Running   0          36s

user@ubuntu:~/dep$
```

Confirm your observations once again in the event log.

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

Name:                   website
Namespace:              default
CreationTimestamp:      Tue, 12 Sep 2017 22:12:17 -0700
Labels:                 bu=sales
Annotations:            deployment.kubernetes.io/revision=3
Selector:               appname=webserver,targetenv=demo
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:       appname=webserver
                targetenv=demo
  Containers:
   podweb:
    Image:              nginx:1.7.9
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
  Progressing   True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  website-205870431 (3/3 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason
Message
  ---------     --------        -----   ----                    -------------   --------        ------
-------
  10m           10m             1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-205870431 to 3
  4m            4m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 1
  4m            4m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 2
  4m            4m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 2
  4m            4m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 1
  4m            4m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 3
  4m            4m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 0
  56s           56s             1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-205870431 to 1
  54s           54s             1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-4061902189 to 2
  54s           49s             4       deployment-controller                   Normal          ScalingReplicaSet
(combined from similar events): Scaled down replica set website-4061902189 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

deployments "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

deployments "website" with revision #2
Pod Template:
  Labels:       appname=webserver
        pod-template-hash=4061902189
        targetenv=demo
  Containers:
   podweb:
    Image:      nginx:1.9.1
    Port:       80/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.

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

deployment "website" rolled back

user@ubuntu:~/dep$
```

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

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

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
website   3         3         3            3           12m

user@ubuntu:~/dep$
```

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

Name:                   website
Namespace:              default
CreationTimestamp:      Tue, 12 Sep 2017 22:12:17 -0700
Labels:                 bu=sales
Annotations:            deployment.kubernetes.io/revision=4
Selector:               appname=webserver,targetenv=demo
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
```

```
Pod Template:
  Labels:       appname=webserver
                targetenv=demo
  Containers:
   podweb:
    Image:              nginx:1.9.1
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
  Progressing   True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  website-4061902189 (3/3 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason
Message
  ---------     --------        -----   ----                    -------------   --------        ------
-------
  12m           12m             1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-205870431 to 3
  7m            7m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 1
  7m            7m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 2
  7m            7m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 2
  3m            3m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-205870431 to 1
  3m            3m              1       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-4061902189 to 2
  24s           24s             1       deployment-controller                   Normal          DeploymentRollback
Rolled back deployment "website" to revision 2
  3m            23s             7       deployment-controller                   Normal          ScalingReplicaSet
(combined from similar events): Scaled up replica set website-4061902189 to 2
  7m            22s             2       deployment-controller                   Normal          ScalingReplicaSet
Scaled up replica set website-4061902189 to 3
  7m            22s             2       deployment-controller                   Normal          ScalingReplicaSet
Scaled down replica set website-205870431 to 1
  6m            20s             2       deployment-controller                   Normal          ScalingReplicaSet
```

```
Scaled down replica set website-205870431 to 0

user@ubuntu:~/dep$
```

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

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 "website" image updated
deployment "website" paused

user@ubuntu:~/dep$
```

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

NAME                 DESIRED   CURRENT   READY     AGE
website-205870431    1         1         1         13m
website-4061902189   3         3         3         8m

user@ubuntu:~/dep$
```

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

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

^C

user@ubuntu:~/dep$
```

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

deployment "website" resumed

user@ubuntu:~/dep$
```

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

deployment "website" successfully rolled out

user@ubuntu:~/dep$
```

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

NAME                 DESIRED   CURRENT   READY     AGE
website-205870431    3         3         3         14m
website-4061902189   0         0         0         9m

user@ubuntu:~/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*):

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

user@ubuntu:~$ mkdir hc

user@ubuntu:~$ cd hc

user@ubuntu:~/hc$ vi hc.yaml

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

apiVersion: apps/v1beta1
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 create -f hc.yaml

deployment "nginx" created

user@ubuntu:~/hc$
```

View your deployment:

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

NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/nginx   3         3         3            3           14s

NAME                 DESIRED   CURRENT   READY     AGE
rs/nginx-101291927   3         3         3         14s

NAME                       READY     STATUS    RESTARTS   AGE
po/nginx-101291927-n78jd   1/1       Running   0          14s
po/nginx-101291927-n7w09   1/1       Running   0          14s
po/nginx-101291927-x38cd   1/1       Running   0          14s

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 | grep $(kubectl get pods -o name | head -1 | awk -F '/' '{print $2}')

8m        8m        1         nginx-101291927-n78jd      Pod                                    Normal
Scheduled            default-scheduler       Successfully assigned nginx-101291927-n78jd to ubuntu
8m        8m        1         nginx-101291927-n78jd      Pod          spec.containers{nginx}    Normal    Pulling
kubelet, ubuntu         pulling image "nginx:latest"
8m        8m        1         nginx-101291927-n78jd      Pod          spec.containers{nginx}    Normal    Pulled
kubelet, ubuntu         Successfully pulled image "nginx:latest"
8m        8m        1         nginx-101291927-n78jd      Pod          spec.containers{nginx}    Normal    Created
```

```
kubelet, ubuntu         Created container with id f1c3413b14d4ad1389d698846f3dd5259cefadce5a88bb6ab424f40e42309b5c
8m        8m        1         nginx-101291927-n78jd      Pod          spec.containers{nginx}    Normal    Started
kubelet, ubuntu         Started container with id f1c3413b14d4ad1389d698846f3dd5259cefadce5a88bb6ab424f40e42309b5c
8m        8m        1         nginx-101291927            ReplicaSet                             Normal
SuccessfulCreate     replicaset-controller   Created pod: nginx-101291927-n78jd

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"

2017/07/14 19:09:28 [notice] 18#18: signal process started

user@ubuntu:~/hc$
```

Now redisplay the events for the pod:

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

15m        15m         1         nginx-2181692066-9cfxt    Pod                                      Normal
Scheduled               default-scheduler       Successfully assigned nginx-2181692066-9cfxt to ubuntu
15m        15m         1         nginx-2181692066-9cfxt    Pod                                      Normal
SuccessfulMountVolume   kubelet, ubuntu         MountVolume.SetUp succeeded for volume "default-token-gl2cc"
14m        15m         2         nginx-2181692066-9cfxt    Pod          spec.containers{nginx}      Normal
Pulling                 kubelet, ubuntu         pulling image "nginx:latest"
14m        15m         2         nginx-2181692066-9cfxt    Pod          spec.containers{nginx}      Normal
Pulled                  kubelet, ubuntu         Successfully pulled image "nginx:latest"
14m        15m         2         nginx-2181692066-9cfxt    Pod          spec.containers{nginx}      Normal
Created                 kubelet, ubuntu         Created container
14m        15m         2         nginx-2181692066-9cfxt    Pod          spec.containers{nginx}      Normal
Started                 kubelet, ubuntu         Started container
14m        14m         3         nginx-2181692066-9cfxt    Pod          spec.containers{nginx}      Warning
Unhealthy               kubelet, ubuntu         Liveness probe failed: Get http://10.32.0.6:80/: dial tcp
10.32.0.6:80: getsockopt: connection refused
14m        14m         1         nginx-2181692066-9cfxt    Pod          spec.containers{nginx}      Normal
```

```
Killing                 kubelet, ubuntu         Killing container with id docker://nginx:pod "nginx-2181692066-
9cfxt_default(0657e7d2-6bae-11e7-bbf7-000c29f5877a)" container "nginx" is unhealthy, it will be killed and re-
created.
15m        15m         1         nginx-2181692066          ReplicaSet                              Normal
SuccessfulCreate        replicaset-controller   Created pod: nginx-2181692066-9cfxt

user@ubuntu:~/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.

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

deployment "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

user@ubuntu:~$ cd jobs/

user@ubuntu:~/jobs$ vim myjob.yaml

user@ubuntu:~/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

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 spec for our Job includes a selector which will match anything with the label “app=pi”.

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 create -f myjob.yaml

job "pi" created

user@ubuntu:~/jobs$
```

Examine the job:

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

NAME      DESIRED   SUCCESSFUL   AGE
jobs/pi   1         1            1m
```

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

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

Name:           pi
Namespace:      default
Selector:       controller-uid=7bd22273-38da-11e7-b8ef-000c2949d6f4
Labels:         controller-uid=7bd22273-38da-11e7-b8ef-000c2949d6f4
                job-name=pi
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Sun, 14 May 2017 12:21:08 -0700
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
Pod Template:
  Labels:       controller-uid=7bd22273-38da-11e7-b8ef-000c2949d6f4
                job-name=pi
  Containers:
   pi:
    Image:      perl
    Port:
    Command:
      perl
      -Mbignum=bpi
      -wle
      print bpi(2000)
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Events:
  FirstSeen     LastSeen        Count   From            SubObjectPath   Type            Reason
Message
  ---------     --------        -----   ----            -------------   --------        ------                  --
-----
  1m            1m              1       job-controller                  Normal          SuccessfulCreate
Created pod: pi-fhftr
user@ubuntu:~/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:

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

3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381964428810975665933446128475648233786783165271201909145648566923460348610454326648213393607260249141273724587006606315588174881520920962829254091715364367892590360011330530548820466521384146951941511609433057270365759591953092186117381932611793105118548074462379962749567351885752724891227938183011949129833673362440656643086021394946395224737190702179860943702770539217176293176752384674818467669405132000568127145263560827785771342757789609173637178721468440901224953430146549585371050792279689258923542019956112129021960864034418159813629774771309960518707211349999998372978049951059731732816096318595024459455346908302642522308253344685035261931188171010003137838752886587533208381420617177669147303598253490428755468731159562863882353787593751957781857780532171226806613001927876611195909216420198938095257201065485863278865936153381827968230301952035301852968995773622599413891249721775283479131515574857242454150695950829533116861727855889075098381754637464939319255060400927701671139009848824012858361603563707660104710181942955596198946767837449448255379774726847104047534646208046684259069491293313677028989152104752162056966024058038150193511253382430035587640247496473263914199272604269922796782354781636009341721641219924586315030286182974555706749838505494588586926995690927210797509302955321165344987202755960236480665499119881834797753566369807426542527862551818417574672890977772793800081647060016145249192173217214772350141441973568548161361157352552133475741849468438523323907394143334547762416862518983569485562099219222184272550254256887671790494601653466804988627232791786085784383827967976681454100953883786360950680064225125205117392984896084128488626945604241965285022210661186306744278622039194945047123713786960956364371917287467764657573962413890865832645995813390478027589

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.)

When you are finished exploring remove the Job:

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

job "pi" deleted
```

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

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

No resources found.

user@ubuntu:~/jobs$
```

Congratulations you have completed the Kubernetes Deployments and Replica Sets lab!