Deployments

A deployment is a supervisor for pods and replica sets, giving you fine-grained control over how and when a new pod version is rolled out as well as rolled back to a previous state.

Let's create a deployment called `sise-deploy` that supervises two replicas of a pod as well as a replica set:

```
cat << EOF > d09.yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: sise-deploy
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: sise
    spec:
      containers:
      - name: sise
        image: mhausenblas/simpleservice:0.5.0
        ports:
        - containerPort: 9876
        env:
        - name: SIMPLE_SERVICE_VERSION
          value: "0.9"
EOF

kubectl create -f d09.yaml
```

```
deployment.apps "sise-deploy" created
```

You can see the deployment, the replica set and the pods it looks after like so:

```
kubectl get deploy
```

```
NAME                          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
sise-deploy                   2         2         2            0           26s
```

```
kubectl get rs
```

```
NAME                     DESIRED   CURRENT   READY     AGE
sise-deploy-cb88d6cf6    2         2         2         2m
```

```
kubectl get pods
```

```
NAME                           READY     STATUS    RESTARTS   AGE
sise-deploy-cb88d6cf6-p56lh    1/1       Running   0          2m
sise-deploy-cb88d6cf6-th7zr    1/1       Running   0          2m
```

Note the naming of the pods and replica set, derived from the deployment name.

At this point in time the sise containers running in the pods are configured to return the version `0.9`. Let's verify that from within the cluster (using `kubectl exec` into one of the pods):

```
kubectl exec -it sise-deploy-cb88d6cf6-p56lh bash
```

```
root@sise-deploy-cb88d6cf6-p56lh:/usr/src/app#
```

```
curl localhost:9876/info
```

```
{ "host": "localhost:9876", "version": "0.9", "from": "127.0.0.1" }
```

Let's now see what happens if we change that version to `1.0` in an updated deployment:

```
cat << EOF > d10.yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: sise-deploy
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: sise
    spec:
      containers:
      - name: sise
        image: mhausenblas/simpleservice:0.5.0
        ports:
        - containerPort: 9876
        env:
        - name: SIMPLE_SERVICE_VERSION
          value: "1.0"
EOF
```

```
kubectl apply -f d10.yaml
```

```
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment.apps "sise-deploy" configured
```

Note that you could have used `kubectl edit deploy/sise-deploy` alternatively to achieve the same by manually editing the deployment.

What we now see is the rollout of two new pods with the updated version `1.0` as well as the two old pods with version `0.9` being terminated:

```
kubectl get pods
```

```
NAME                           READY     STATUS        RESTARTS   AGE
sise-deploy-554b64c5c6-9xcsc   1/1       Running       0          1m
sise-deploy-554b64c5c6-r4gtl   1/1       Running       0          30s
sise-deploy-cb88d6cf6-p56lh    1/1       Terminating   0          11m
sise-deploy-cb88d6cf6-th7zr    1/1       Terminating   0          11m
```

Also, a new replica set has been created by the deployment:

```
kubectl get rs
```

```
NAME                     DESIRED   CURRENT   READY     AGE
sise-deploy-554b64c5c6   2         2         2         1m
sise-deploy-cb88d6cf6    0         0         0         11m
```

Note that during the deployment you can check the progress using `kubectl rollout status deploy/sise-deploy`.

```
kubectl rollout status deploy/sise-deploy
```

```
deployment "sise-deploy" successfully rolled out
```

To verify that if the new `1.0` version is really available, we execute from within a pod:

```
kubectl exec -it sise-deploy-554b64c5c6-9xcsc bash
```

```
root@sise-deploy-554b64c5c6-9xcsc:/usr/src/app#
```

```
curl localhost:9876/info
```

```
{ "host": "localhost:9876", "version": "1.0", "from": "127.0.0.1" }
```

A history of all deployments is available via:

```
kubectl rollout history deploy/sise-deploy
```

```
deployments "sise-deploy"
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

If there are problems in the deployment Kubernetes will automatically roll back to the previous version, however you can also explicitly roll back to a specific revision, as in our case to revision 1 (the original pod version):

```
kubectl rollout undo deploy/sise-deploy --to-revision=1
```

```
deployment.apps "sise-deploy"
```

```
kubectl rollout history deploy/sise-deploy
```

```
deployments "sise-deploy"
REVISION  CHANGE-CAUSE
2         <none>
3         <none>
```

```
kubectl get pods
```

```
NAME                           READY     STATUS        RESTARTS   AGE
sise-deploy-554b64c5c6-9xcsc   1/1       Terminating   0          6m
sise-deploy-554b64c5c6-r4gtl   1/1       Terminating   0          6m
sise-deploy-cb88d6cf6-2xmtp    1/1       Running       0          10s
sise-deploy-cb88d6cf6-66p4s    1/1       Running       0          12s
```

```
kubectl get rs
```

```
NAME                     DESIRED   CURRENT   READY     AGE
sise-deploy-554b64c5c6   0         0         0         6m
sise-deploy-cb88d6cf6    2         2         2         17m
```

```
kubectl exec -it sise-deploy-cb88d6cf6-2xmtp bash
```

```
curl localhost:9876/info
```

```
{ "host": "localhost:9876", "version": "0.9", "from": "127.0.0.1" }
```

At this point in time we're back at where we started, with two new pods serving again version `0.9`.

Finally, to clean up, we remove the deployment and with it the replica sets and pods it supervises:

```
kubectl delete deploy sise-deploy
```

```
deployment.extensions "sise-deploy" deleted
```