Updating Deployments

In this lesson, we will learn to update the Kubernetes Deployments.

WE'LL COVER THE FOLLOWING

Updating the db Image

Let's see what happens when we `set` a new image to the `db` Pod.

```
kubectl set image \
    -f deploy/go-demo-2-db.yml \
    db=mongo:3.4 \
    --record
```

It'll take a while until the new image is pulled.

Describing the Deployment

Once it's done, we can `describe` the Deployment by checking the events it created.

```
kubectl describe \
    -f deploy/go-demo-2-db.yml
```

The last few lines of the **output** are as follows.

```
...
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  19m   deployment-controller  Scaled up replica set go-demo-2-db-
  Normal  ScalingReplicaSet  5m    deployment-controller  Scaled up replica set go-demo-2-db-
  Normal  ScalingReplicaSet  0s    deployment-controller  Scaled down replica set go-demo-2-d
```

We can see that it created a new ReplicaSet and that it scaled the old ReplicaSet to `0`. If, in your case, the last line did not appear, you'll need to wait until the new version of the `mongo` image is pulled.

Instead of operating directly on the level of Pods, the Deployment created a new ReplicaSet which, in turn, produced Pods based on the new image. Once they became fully operational, it scaled the old ReplicaSet to `0`.

Since we are running a ReplicaSet with only one replica, it might not be clear why it used that strategy. When we create a Deployment for the API, things will become more evident.

Looking into the Cluster

To be on the safe side, we might want to retrieve all the objects from the cluster.

```
kubectl get all
```

The **output** is as follows.

```
NAME                                 READY   STATUS              RESTARTS   AGE
pod/go-demo-2-db-694bfb44cb-n6rxl    1/1     Running             0          17m
pod/go-demo-2-db-6b97cd9dfc-kbzp4    0/1     ContainerCreating   0          47s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   19m

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/go-demo-2-db    1/1     1            1           17m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/go-demo-2-db-694bfb44cb    1         1         1       17m
replicaset.apps/go-demo-2-db-6b97cd9dfc    1         1         0       48s
```

As you can see, both ReplicaSets are there. However, one is inactive (scaled to `0`).

You'll notice that contained within the name of the Pod is a hash which matches the hash in the name of the new ReplicaSet, namely `f8d4b86ff`. Even though it might look like it is a random value, it is not.

If you destroy the Deployment and create it again, you'll notice that the hash in the Pod name and ReplicaSet name remain consistent. This value is generated by hashing the PodTemplate of the ReplicaSet. As long as the PodTemplate is the same, the hash value will be the same as well. That way a Deployment can know whether anything related to the Pods has changed and, if it does, will create a new ReplicaSet.

Exploring Ways to Update Deployment

Updating Using Commands

The `kubectl set image` command is not the only way to update a Deployment. We could also have used `kubectl edit` as well.

The command would be as follows.

```
kubectl edit -f deploy/go-demo-2-db.yml
```

> ⚠ **Please do NOT execute it.** If you do, you'll need to type `:q` followed by the enter key to exit.

The above `edit` command is not a good way to update the definition. It is unpractical and undocumented. The `kubectl set image` is more useful if we'd like to integrate Deployment updates with one of the CI/CD tools.

Since we'll have a chapter dedicated to continuous deployment, we'll continue using `kubectl set image` for now.

Updating the YAML File

Another alternative would be to update the YAML file and execute the `kubectl apply` command. While that is a good idea for applications that do not update frequently, it does not fit well with those that change weekly, daily, or even

hourly.

MongoDB is one of those that might get updated with a new release only a couple of times a year so having an always up-to-date YAML file in your source code repository is an excellent practice.

Finishing off by Adding a Service

We used `kubectl set image` just as a way to introduce you to what's coming next when we explore frequent deployments without downtime.

A simple update of Pod images is far from what Deployment offers. To see its real power, we should deploy the API. Since it can be scaled to multiple Pods, it'll provide us with a much better playground.

Before we move on, let's finish with the database by adding a Service and, therefore, enabling internal cluster communication to it.

```
kubectl create \
    -f deploy/go-demo-2-db-svc.yml \
    --record
```

Now that we have added the service, in the next lesson, we will learn how to deploy with zero down-time.