Deploying New Releases

In this lesson, we will go through the Kubernetes deployment definition and will create a deployment.

WE'LL COVER THE FOLLOWING

- Looking into the Definition
- Creating the Deployment
- Describing the Deployment

Just as we are not supposed to create Pods directly but using other controllers like ReplicaSet, we are not supposed to create ReplicaSets either. Kubernetes Deployments will create them for us. If you're wondering why is this so? You'll have to wait a little while longer to find out.

First, we'll create a few Deployments and, once we are familiar with the process and the outcomes, it'll become obvious why they are better at managing ReplicaSets than we are.

Looking into the Definition

Let's take a look at a Deployment specification for the database ReplicaSet we've been using thus far.

```
cat deploy/go-demo-2-db.yml
```

The **output** is as follows.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-demo-2-db
spec:
  selector:
    matchLabels:
      type: db
```

```
      service: go-demo-2
  template:
    metadata:
      labels:
        type: db
        service: go-demo-2
        vendor: MongoLabs
    spec:
      containers:
      - name: db
        image: mongo:3.3
        ports:
        - containerPort: 28017
```

If you compare this Deployment with the ReplicaSet we created in the previous chapter, you'll probably have a hard time finding a difference. Apart from the `kind` field, they are the same.

Since, in this case, both the Deployment and the ReplicaSet are the same, you might be wondering what the advantage of using one over the other is.

> ℹ We will regularly add `--record` to the `kubectl create` commands. This allows us to track each change to our resources such as a Deployments.

Creating the Deployment

Let's create the Deployment and explore what it offers.

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

The **output** of the latter command is as follows.

```
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
go-demo-2-db  0/1     1            0           4s
```

Describing the Deployment

The Deployment was created. However, `get` does not provide us much info, so let's `describe` it.

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

The **output**, limited to the last few lines, is as follows.

```
...
Events:
  Type    Reason             Age  From                   Message
  ----    ------             ---- ----                   -------
  Normal  ScalingReplicaSet  2m   deployment-controller  Scaled up replica set go-demo-2-db-75fb
```

From the `Events` section, we can observe that the Deployment created a ReplicaSet. Or, to be more precise, that it scaled it. That is interesting.

It shows that Deployments control ReplicaSets. The Deployment created the ReplicaSet which, in turn, created Pods.

Let's confirm that by retrieving the list of all the objects.

```
kubectl get all
```

The **output** is as follows.

```
NAME                                 READY   STATUS    RESTARTS   AGE
pod/go-demo-2-db-694bfb44cb-n6rxl    1/1     Running   0          7m49s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   9m40s

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/go-demo-2-db    1/1     1            1           7m49s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/go-demo-2-db-694bfb44cb    1         1         1       7m49s
```

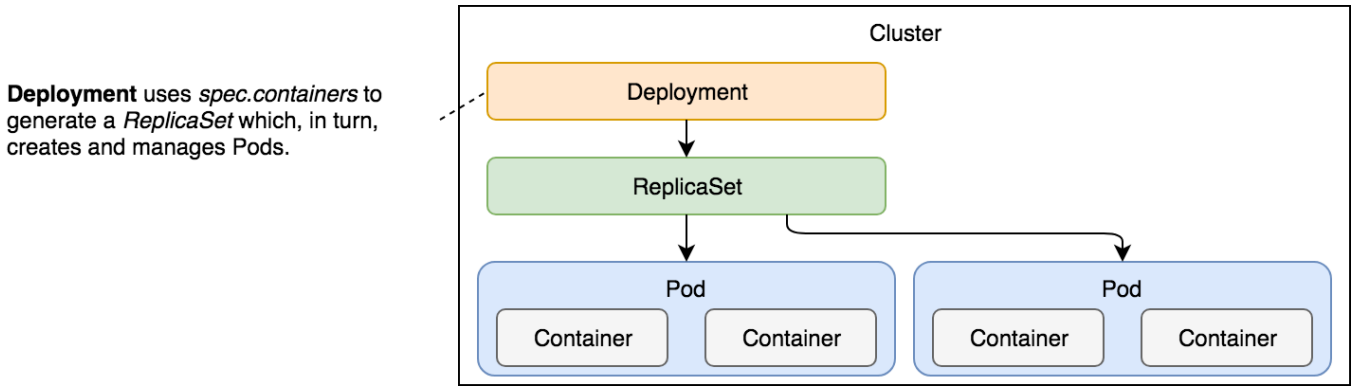
All three objects were created, and you might be wondering why we created the Deployment at all. You might think that we'd have the same result if we created a ReplicaSet directly. You'd be right.

So far, from the functional point of view, there is no difference between a ReplicaSet created directly or using a Deployment.

> The real advantage of Deployments becomes evident if we try to change some of its aspects. For example, we might choose to upgrade MongoDB

to version 3.4.

The following figure summarizes the cascading effect of deployments resulting in the creation of pods, containers, and replicaSets.



Deployment and its cascading effect that creates a ReplicaSet and, through it, Pods

---

In the next lesson, we will go through the sequential breakdown of the process of creating deployment.