Updating Multiple Objects

In this lesson, we will learn how to update multiple deployments at a time.

WE'LL COVER THE FOLLOWING

The Use of Selector Labels

Even though most of the time we send requests to specific objects, almost everything is happening using selector labels. When we updated the Deployments, they looked for matching selectors to choose which ReplicaSets to create and scale. They, in turn, created or terminated Pods also using the matching selectors.

> Almost everything in Kubernetes is operated using label selectors. It's just that sometimes that is obscured from us.

We do not have to update an object only by specifying its name or the YAML file where its definition resides. We can also use labels to decide which object should be updated. That opens some interesting possibilities since the selectors might match multiple objects.

Defining the Deployment

Imagine that we are running several Deployments with Mongo databases and that the time has come to update them all to a newer release. Before we explore how we could do that, we'll create another Deployment so that we have at least two with the database Pods.

Let us first take a look at the definition.

```
cat deploy/different-app-db.yml
```

The **output** is as follows.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: different-app-db
  labels:
    type: db
    service: different-app
    vendor: MongoLabs
spec:
  selector:
    matchLabels:
      type: db
      service: different-app
  template:
    metadata:
      labels:
        type: db
        service: different-app
        vendor: MongoLabs
    spec:
      containers:
      - name: db
        image: mongo:3.3
        ports:
        - containerPort: 27017
```

When compared with the `go-demo-2-db` Deployment, the only difference is in the `service` label. Both have the `type` set to `db`.

Creating the Deployment

Let's create the deployment.

```
kubectl create \
    -f deploy/different-app-db.yml
```

Now that we have two deployments with the `mongo:3.3` Pods, we can try to update them both at the same time.

The trick is to find a label (or a set of labels) that uniquely identifies all the

Deployments we want to update.

Let’s take a look at the list of Deployments with their labels.

```
kubectl get deployments --show-labels
```

The **output** is as follows.

```
NAME              DESIRED  UP-TO-DATE AVAILABLE AGE LABELS
different-app-db 1         1          1         1h  service=different-app,type=db,vendor=Mongo
go-demo-2-api    3         3          3         1h  language=go,service=go-demo-2,type=api
go-demo-2-db     1         1          1         1h  service=go-demo-2,type=db,vendor=MongoLabs
```

We want to update `mongo` Pods created using `different-app-db` and `go-demo-2-db` Deployments. Both are uniquely identified with the labels `type=db` and `vendor=MongoLabs`. Let’s test that.

```
kubectl get deployments \
    -l type=db,vendor=MongoLabs
```

The **output** is as follows.

```
NAME              DESIRED UP-TO-DATE AVAILABLE AGE
different-app-db 1        1          1         1h
go-demo-2-db     1        1          1         1h
```

Updating the Deployments

We can see that filtering with those two labels worked. We retrieved only the Deployments we want to update, so let’s proceed and roll out the new release.

```
kubectl set image deployments \
    -l type=db,vendor=MongoLabs \
    db=mongo:3.4 --record
```

The **output** is as follows.

```
deployment.extensions/different-app-db image updated
deployment.extensions/go-demo-2-db image updated
```

Finally, before we move into the next subject, we should validate that the image indeed changed to `mongo:3.4`.

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

The **output**, limited to the relevant parts, is as follows.

```
...
  Containers:
   db:
    Image:        mongo:3.4
...
```

As we can see, the update was indeed successful, at least with that Deployment. Feel free to describe the Deployment defined in `deploy/different-app-db.yml`. You should see that its image was also updated to the newer version.

In the next lesson, we will go through scaling the deployments we created thus far.