Scaling Deployments

In this lesson, we will learn to scale deployments using a YAML file and will discuss automated scaling briefly.

WE'LL COVER THE FOLLOWING

Scaling Using YAML Files

There are quite a few different ways we can scale Deployments. Everything we do in this section is not unique to Deployments and can be applied to any Controller, like ReplicaSet, and those we did not yet explore.

If we decide that the number of replicas changes with relatively low frequency or that Deployments are performed manually, the best way to scale is to write a new YAML file or, even better, modify the existing one. Assuming that we store YAML files in a code repository, by updating existing files we have a documented and reproducible definition of the objects running inside a cluster.

We already performed scaling when we applied the definition from the `go-demo-2-scaled.yml`. We'll do something similar, but with Deployments.

Looking into the File

Let's take a look at `deploy/go-demo-2-scaled.yml`.

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

We won't display the contents of the whole file since it is almost identical `deploy/go-demo-2.yml`. The only difference is the number of replicas of the `go-demo-2-api` Deployment.

```
...
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-demo-2-api
spec:
  replicas: 5
...
```

Applying the Definition

At the moment, we're running three replicas. Once we apply the new definition, it should increase to five.

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

Please note that, even though the file is different, the names of the resources are the same so `kubectl apply` did not create new objects. Instead, it updated those that changed. In particular, it changed the number of replicas of the `go-demo-2-api` Deployment.

Verification

Let's confirm that there are indeed five replicas of the Pods controlled through the Deployment.

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

The **output**, limited to the `deploy/go-demo-2-api`, is as follows.

```
...
NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/go-demo-2-api   5/5     5            5           3h26m
...
```

The result should come as no surprise. After all, we executed the same process before, when we explored ReplicaSets.

Automated Scaling

While scaling Deployments using YAML files (or other Controllers) is an excellent way to keep documentation accurate, it rarely fits the dynamic nature of the clusters. We should aim for a system that will scale (and de-scale) services automatically.

When scaling is frequent and, hopefully, automated, we cannot expect to update YAML definitions and push them to Git. That would be too inefficient and would probably cause quite a few unwanted executions of delivery pipelines if they are triggered through repository WebHooks. After all, do we really want to push updated YAML files multiple times a day?

> The number of `replicas` should not be part of the design. Instead, they are a fluctuating number that changes continuously (or at least often), depending on the traffic, memory and CPU utilization, and so on.

Depending on release frequency, the same can be said for `image`. If we are practicing continuous delivery or deployment, we might be releasing once a week, once a day, or even more often. In such cases, new images would be deployed often, and there is no strong argument for the need to change YAML files every time we make a new release. That is especially true if we are deploying through an automated process (as we should).

We'll explore automation later on. For now, we'll limit ourselves to a command similar to `kubectl set image`. We used it to change the `image` used by Pods with each release.

Scaling the Deployment

Similarly, we'll use `kubectl scale` to change the number of replicas. Consider this an introduction to automation that is coming later on.

```
kubectl scale deployment \
kubectl scale deployment \
    go-demo-2-api --replicas 8 --record
```

We scaled the number of replicas associated with the Deployment `go-demo-2-api`. Please note that, this time, we did not use `-f` to reference a file. Since we have two Deployments specified in the same YAML, that would result in scaling of both. Since we wanted to limit it to a particular Deployment, we used its name instead.

Verification

Let's confirm that scaling indeed worked as expected.

```
kubectl get -f deploy/go-demo-2.yml
```

The **output**, limited to Deployments, is as follows.

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/go-demo-2-db   1/1     1            1           4h40m

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/go-demo-2-api   8/8     8            8           3h28m
```

As we mentioned earlier, we'll dedicate quite a lot of time to automation, and you won't have to scale your applications manually. However, it is useful to know that the `kubectl scale` command exists. For now, you know how to scale Deployments (and other Controllers).

In the next lesson, we will test your understanding related to this chapter with the help of a short quiz.