Specify Replicas in Deployments or Statefulsets?

In this lesson, we will explore different strategies regarding where to define the replicas, in our Deployments or StatefulSets.

WE'LL COVER THE FOLLOWING

- `HPA` modifies the Deployment
- Remove the number of replicas in Deployment
- Not specify the number of replicas in Deployment
- Test the new release of Deployment

Knowing that `HorizontalPodAutoscaler (HPA)` manages auto-scaling of our applications, the question might arise regarding `replicas`. Should we define them in our Deployments and StatefulSets, or should we rely solely on `HPA` to manage them? Instead of answering that question directly, we'll explore different combinations and, based on results, define the strategy.

`HPA` modifies the Deployment

First, let's see how many Pods we have in our cluster right now.

```
kubectl -n go-demo-5 get pods
```

The **output** is as follows.

```
NAME    READY STATUS  RESTARTS AGE
api-... 1/1   Running 0        27m
api-... 1/1   Running 2        31m
db-0    2/2   Running 0        20m
db-1    2/2   Running 0        20m
db-2    2/2   Running 0        21m
```

We can see that there are two replicas of the `api` Deployment, and three

replicas of the `db` StatefulSets.

Let's say that we want to roll out a new release of our `go-demo-5` application. The definition we'll use is as follows.

```
cat scaling/go-demo-5-replicas-10.yml
```

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

```
...
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api
  namespace: go-demo-5
spec:
  replicas: 10
...

apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: api
  namespace: go-demo-5
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 80
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 80
```

The important thing to note is that our `api` Deployment has `10` replicas and that we have the `HPA`. Everything else is the same as it was before.

What will happen if we apply that definition?

```
kubectl apply \
  -f scaling/go-demo-5-replicas-10.yml

kubectl -n go-demo-5 get pods
```

We applied the new definition and retrieved all the Pods from the `go-demo-5` Namespace. The **output** of the latter command is as follows.

```
NAME    READY STATUS            RESTARTS AGE
api-... 1/1   Running           0        9s
api-... 0/1   ContainerCreating 0        9s
api-... 0/1   ContainerCreating 0        9s
api-... 1/1   Running           2        41m
api-... 1/1   Running           0        22s
api-... 0/1   ContainerCreating 0        9s
api-... 0/1   ContainerCreating 0        9s
api-... 1/1   Running           0        9s
api-... 1/1   Running           0        9s
api-... 1/1   Running           0        9s
db-0    2/2   Running           0        31m
db-1    2/2   Running           0        31m
db-2    2/2   Running           0        31m
```

Kubernetes complied with our desire to have ten replicas of the `api` and created eight Pods (we had two before). At the first look, it seems that `HPA` does not have any effect. Let's retrieve the Pods one more time.

```
kubectl -n go-demo-5 get pods
```

The **output** is as follows.

```
NAME    READY STATUS  RESTARTS AGE
api-... 1/1   Running 0        30s
api-... 1/1   Running 2        42m
api-... 1/1   Running 0        43s
api-... 1/1   Running 0        30s
api-... 1/1   Running 0        30s
db-0    2/2   Running 0        31m
db-1    2/2   Running 0        32m
db-2    2/2   Running 0        32m
```

Our Deployment de-scaled from ten to five replicas. `HPA` detected that there are more replicas than the maximum threshold and acted accordingly. But what did it do? Did it simply remove five replicas? That could not be the case since that would only have a temporary effect. If `HPA` removes or adds Pods, Deployment would also remove or add Pods, and the two would be fighting with each other. The number of Pods would be fluctuating indefinitely. Instead, `HPA` modified the Deployment.

Let's describe the `api`.

```
kubectl -n go-demo-5 \
  describe deployment api
```

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

```
...
Replicas: 5 desired | 5 updated | 5 total | 5 available | 0 unavailable
...
Events:
... Message
... -------
...
... Scaled up replica set api-5bbfd85577 to 10
... Scaled down replica set api-5bbfd85577 to 5
```

The number of replicas is set to `5` desired. `HPA` modified our Deployment. We can observe that better through the event messages. The second to last states that the number of replicas was scaled up to `10`, while the last message indicates that it scaled down to `5`. The former is the result of us executing a rolling update by applying the new Deployment, while the latter was produced by `HPA` modifying the Deployment by changing its number of replicas.

So far, we observed that `HPA` modifies our Deployments. No matter how many replicas we defined in a Deployment (or a StatefulSets), `HPA` will change it to fit its own thresholds and calculations. In other words, when we update a Deployment, the number of replicas will be temporarily changed to whatever we have defined, only to be modified again by `HPA` a few moments later. That behavior is unacceptable.

If `HPA` changed the number of replicas, there is usually a good reason for that.

Resetting that number to whatever is set in a Deployment (or a StatefulSet) can produce serious side-effects.

Remove the number of replicas in Deployment

Let's say that we have three replicas defined in a Deployment and that `HPA` scaled it to thirty because there is an increased load on that application. If we `apply` the Deployment because we want to roll out a new release, for a brief period, there will be three replicas, instead of thirty. As a result, our users would experience slow response times from our application, or some other effect caused by too few replicas serving too much traffic. We must try to avoid such a situation. The number of replicas should be controlled by `HPA` at all times. That means we'll need to change our strategy.

If specifying the number of replicas in a Deployment does not produce the effect we want, we might just as well remove them altogether. Let's see what happens in that case.

We'll use `go-demo-5.yml` definition, so let's see how it differs from `go-demo-5-replicas-10.yml` that we used previously.

```
diff \
  scaling/go-demo-5-replicas-10.yml \
  scaling/go-demo-5.yml
```

The **output** shows that the only difference is that, this time, we are not specifying the number of replicas.

Let's apply the change and see what happens.

```
kubectl apply \
  -f scaling/go-demo-5.yml

kubectl -n go-demo-5 \
  describe deployment api
```

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

```
...
Replicas: 1 desired | 5 updated | 5 total | 5 available | 0 unavailable
```

```
Events:
... Message
... -------
...
... Scaled down replica set api-5bbfd85577 to 5
... Scaled down replica set api-5bbfd85577 to 1
```

Applying the Deployment without `replicas` resulted in `1 desired`. Sure, `HPA` will scale it up to `2` (its minimum) soon enough, but we still failed in our mission to maintain the number of replicas defined by `HPA` at all times.

What else can we do? No matter whether we define our Deployment with or without `replicas`, the result is the same. Applying the Deployment always cancels the effect of the `HPA`, even when we do **NOT** specify `replicas`. Actually, that statement is incorrect. We can accomplish the desired behavior without `replicas` if we know how the whole process works.

Q Select which one is true.

COMPLETED 0%

1 of 1

Not specify the number of replicas in Deployment

If `replicas` is defined for a Deployment, it will be used every time we `apply` a definition. If we change the definition by removing `replicas`, the Deployment

will think that we want to have one, instead of the number of replicas we had before. But, if we never specify the number of `replicas`, they will be entirely controlled by `HPA`.

Let's test it out.

```
kubectl delete -f scaling/go-demo-5.yml
```

We deleted everything related to the `go-demo-5` application. Now, let's test how the Deployment behaves if `replicas` is not defined from the start.

```
kubectl apply \
  -f scaling/go-demo-5.yml

kubectl -n go-demo-5 \
  describe deployment api
```

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

```
...
Replicas: 1 desired | 1 updated | 1 total | 0 available | 1 unavailable
...
```

Seems that we failed. The Deployment did set the number of replicas to `1`. But, what you cannot see, is that replicas are not defined internally.

Nevertheless, a few moments later, our Deployment will be scaled up by `HPA` to two replicas. That is the expected behavior, but we'll confirm it anyway.

```
kubectl -n go-demo-5 \
  describe deployment api
```

You should see from the output that the number of replicas was changed (by `HPA`) to `2`.

Test the new release of Deployment

Now comes the final test. If we make a new release of the Deployment, will it scale down to `1` replica, or will it stay on `2`?

We'll apply a new definition. The only difference, when compared with the one currently running, is in the tag of the image. That way we'll guarantee that the Deployment will be indeed updated.

```
kubectl apply \
  -f scaling/go-demo-5-2-5.yml

kubectl -n go-demo-5 \
  describe deployment api
```

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

```
...
Replicas: 2 desired | 1 updated | 3 total | 2 available | 1 unavailable
...
Events:
... Message
... -------
... Scaled up replica set api-5bbfd85577 to 1
... Scaled up replica set api-5bbfd85577 to 2
... Scaled up replica set api-745bc9fc6d to 1
```

We can see that the number of replicas, set by the `HPA`, is preserved.

Don't be alarmed if you see in the `events` that the number of replicas was scaled to `1`. That's the second ReplicaSet spin up by the Deployment. You can see that by observing the name of the ReplicaSet. The Deployment is doing rolling updates by juggling two ReplicaSets in the attempt to roll out the new release without downtime. That is unrelated to auto-scaling, and I assume that you already know how rolling updates work. If you don't, you know where to learn it.

Now comes the critical question. **How should we define replicas in Deployments and StatefulSets?**

> 📌 If you plan to use `HPA` with a Deployment or a StatefulSet, do **NOT** declare `replicas`. If you do, each rolling update will cancel the effect of the `HPA` for a while. Define `replicas` only for the resources that are **NOT** used in conjunction with `HPA`.

In the next lesson, we will revise and test the concepts we have learned so far through a short quiz.