Auto-Scale Pods Based on Resource Utilization

In this lesson, we will see Auto Scaling of Pods based on Resource Utilization in action.

WE'LL COVER THE FOLLOWING

Auto-scale based on resource usage

So far, the `HPA` has not yet performed auto-scaling based on resource usage. Let's do that now. First, we'll try to create another `HorizontalPodAutoscaler` but, this time, we'll target the StatefulSet that runs our MongoDB. So, let's take a look at yet another YAML definition.

Create `HPA`

```
cat scaling/go-demo-5-db-hpa.yml
```

The **output** is as follows.

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: db
  namespace: go-demo-5
spec:
```

```
  scaleTargetRef:
    apiVersion: apps/v1
    kind: StatefulSet
    name: db
  minReplicas: 3
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 80
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 80
```

That definition is almost the same as the one we used before. The only difference is that this time we're targeting `StatefulSet` called `db` and that the minimum number of replicas should be `3`.

Let's apply it.

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

Let's take another look at the `HorizontalPodAutoscaler` resources.

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

The **output** is as follows.

```
NAME REFERENCE      TARGETS                         MINPODS MAXPODS REPLICAS
AGE
api  Deployment/api 41%/80%, 0%/80%                 2       5       2
5m
db   StatefulSet/db <unknown>/80%, <unknown>/80% 3       5       0
20s
```

We can see that the second `HPA` was created and that the current utilization is `unknown`. That must be a similar situation as before. Should we give it some time for data to start flowing in? Wait for a few moments and retrieve `HPAs` again. Are the targets still `unknown`?

again. Are the targets still `unknown`?

Resource utilization not getting shown

There might be something wrong since the resource utilization continued being unknown. Let's describe the newly created `HPA` and see whether we'll be able to find the cause behind the issue.

```
kubectl -n go-demo-5 describe hpa db
```

The **output**, limited to the event messages, is as follows.

```
...
Events:
... Message
... -------
... New size: 3; reason: Current number of replicas below Spec.MinReplicas
... missing request for memory on container db-sidecar in pod go-demo-5/db
-0
... failed to get memory utilization: missing request for memory on contai
ner db-sidecar in pod go-demo-5/db-0
```

Please note that your **output** could have only one event or even none of those. If that's the case, please wait for a few minutes and repeat the previous command.

If we focus on the first message, we can see that it started well. `HPA` detected that the current number of replicas is below the limit and increased them to three. That is the expected behavior, so let's move to the other two messages.

`HPA` could not calculate the percentage because we did not specify how much memory we are requesting for the `db-sidecar` container. Without `requests`, `HPA` cannot calculate the percentage of the actual memory usage. In other words, we missed specifying resources for the `db-sidecar` container and `HPA` could not do its work. We'll fix that by applying `go-demo-5-no-hpa.yml`.

Create `HPA` with new definition

Let's take a quick look at the new definition.

```
cat scaling/go-demo-5-no-hpa.yml
```

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

```
...
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: db
  namespace: go-demo-5
spec:
  ...
  template:
    ...
    spec:
      ...
      - name: db-sidecar
        ...
        resources:
          limits:
            memory: "100Mi"
            cpu: 0.2
          requests:
            memory: "50Mi"
            cpu: 0.1
...
```

The only noticeable difference, when compared with the initial definition, is that this time we defined the resources for the `db-sidecar` container. Let's apply it.

```
kubectl apply \
    -f scaling/go-demo-5-no-hpa.yml \
    --record
```

Next, we'll wait for a few moments for the changes to take effect, before we retrieve the `HPAs` again.

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

This time, the **output** is more promising.

```
NAME REFERENCE      TARGETS          MINPODS MAXPODS REPLICAS AGE
api  Deployment/api 66%/80%, 10%/80% 2       5       2        16m
db   StatefulSet/db 60%/80%, 4%/80%  3       5       3        10m
```

Resource utilization getting shown

Both `HPAs` are showing the current and target resource usage. Neither reached the target values, so `HPA` is maintaining the minimum number of replicas. We can confirm that by listing all the Pods in the `go-demo-5` Namespace.

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

The **output** is as follows.

```
NAME    READY STATUS  RESTARTS AGE
api-... 1/1   Running 0        42m
api-... 1/1   Running 0        46m
db-0    2/2   Running 0        33m
db-1    2/2   Running 0        33m
db-2    2/2   Running 0        33m
```

We can see that there are two Pods for the `api` Deployment and three replicas of the `db` StatefulSet. Those numbers are equivalent to the `spec.minReplicas` entries in the `HPA` definitions.

Q

Without `requests`, `HPA` can calculate the percentage of the actual memory usage.

COMPLETED 0%

1 of 1

Actual memory usage above the target value

Let's see what happens when the actual memory usage is above the target value.

We'll modify the definition of one of the `HPAs` by lowering one of the targets as a way to reproduce the situation in which our Pods are consuming more resources than desired.

Let's take a look at a modified `HPA` definition.

```
cat scaling/go-demo-5-api-hpa-low-mem.yml
```

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

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: api
  namespace: go-demo-5
spec:
  ...
  metrics:
  ...
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 10
```

We decreased `targetAverageUtilization` to `10`. That will surely be below the current memory utilization, and we'll be able to witness `HPA` in action. Let's apply the new definition.

```
kubectl apply \
    -f scaling/go-demo-5-api-hpa-low-mem.yml \
    --record
```

Please wait a few moments for the next iteration of data gathering to occur, and retrieve the `HPAs`.

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

The **output** is as follows.

```
NAME REFERENCE      TARGETS          MINPODS MAXPODS REPLICAS AGE
```

```
api   Deployment/api 49%/10%, 10%/80% 2        5        2         44m
db    StatefulSet/db 64%/80%, 5%/80%  3        5        3         39m
```

We can see that the actual memory of the `api` `HPA` (`49%`) is way above the threshold (`10%`). However, the number of replicas is still the same (`2`). We'll have to wait for a few more minutes before we retrieve `HPAs` again.

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

This time, the **output** is slightly different.

```
NAME REFERENCE       TARGETS          MINPODS MAXPODS REPLICAS AGE
api  Deployment/api 49%/10%, 10%/80% 2       5       4        44m
db   StatefulSet/db 64%/80%, 5%/80%  3       5       3        39m
```

We can see that the number of replicas increased to `4`. `HPA` changed the Deployment, and that produced the cascading effect that resulted in the increased number of Pods.

Let's describe the api `HPA`.

```
kubectl -n go-demo-5 describe hpa api
```

The **output**, limited to the messages of the events, is as follows.

```
...
Events:
... Message
... -------
... New size: 2; reason: Current number of replicas below Spec.MinReplicas
... New size: 4; reason: memory resource utilization (percentage of reques
t) above target
```

We can see that the `HPA` changed the size to `4` because `memory resource utilization (percentage of request)` was `above target`.

`HPA` continue to scale up the Deployment

Since, in this case, increasing the number of replicas did not reduce memory consumption below the `HPA` target, we should expect that the `HPA` will continue scaling up the Deployment until it reaches the limit of `5`. We'll confirm that assumption by waiting for a few minutes and describing the `HPA`

one more time.

```
kubectl -n go-demo-5 describe hpa api
```

The **output**, limited to the messages of the events, is as follows.

```
...
Events:
... Message
... -------
... New size: 2; reason: Current number of replicas below Spec.MinReplicas
... New size: 4; reason: memory resource utilization (percentage of reques
t) above target
... New size: 5; reason: memory resource utilization (percentage of reques
t) above target
```

We got the message stating that the new size is now `5`, thus proving that the `HPA` will continue scaling up until the resources are below the target or, as in our case, it reaches the maximum number of replicas.

We can confirm that scaling indeed worked by listing all the Pods in the `go-demo-5` Namespace.

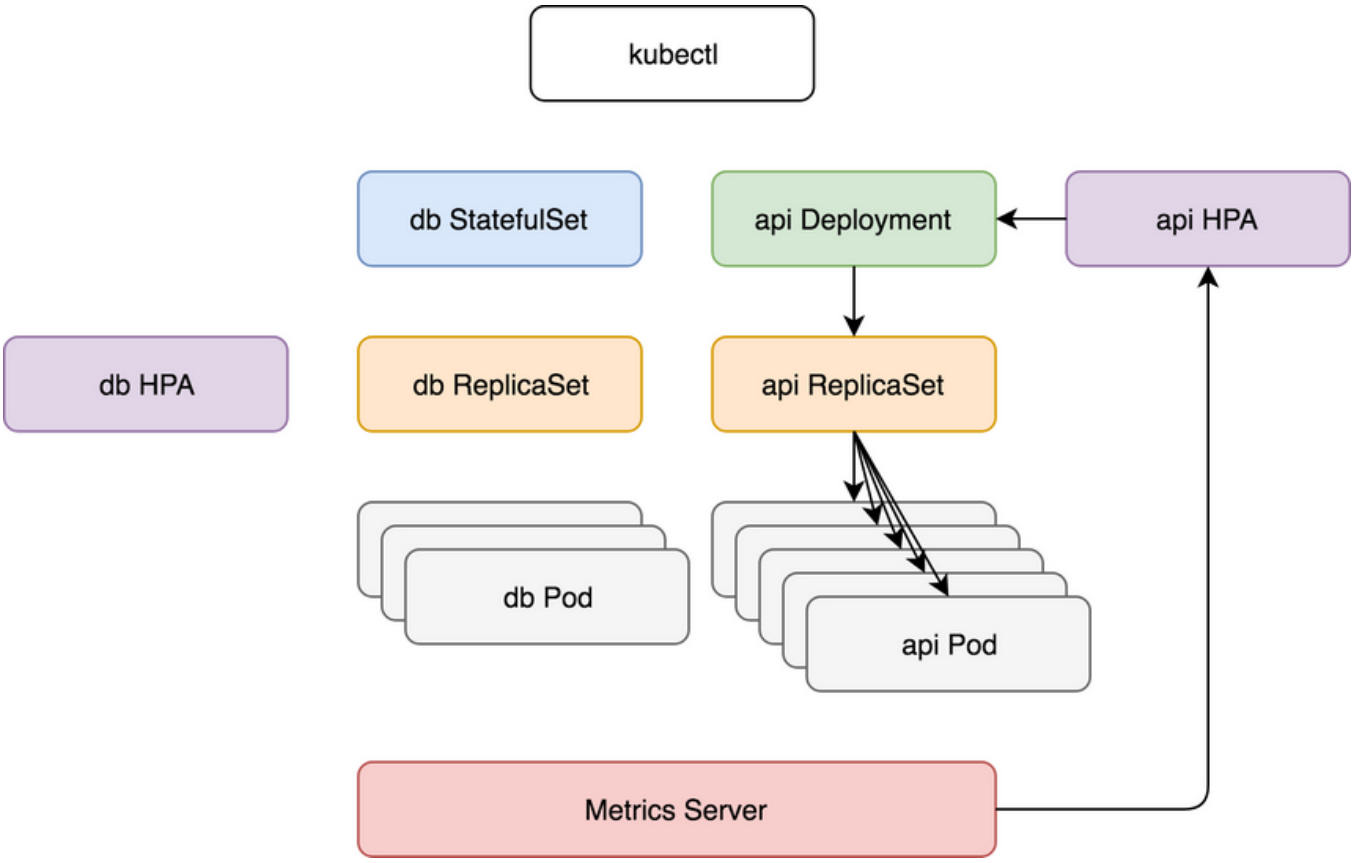
```
kubectl -n go-demo-5 get pods
```

The **output** is as follows.

```
NAME    READY STATUS  RESTARTS AGE
api-... 1/1   Running 0        47m
api-... 1/1   Running 0        51m
api-... 1/1   Running 0        4m
api-... 1/1   Running 0        4m
api-... 1/1   Running 0        24s
db-0    2/2   Running 0        38m
db-1    2/2   Running 0        38m
db-2    2/2   Running 0        38m
```

As we can see, there are indeed five replicas of the `api` Deployment.

`HPA` retrieved data from the `Metrics Server`, concluded that the actual resource usage is higher than the threshold, and manipulated the Deployment with the new number of replicas.



HPA scaling through manipulation of the Deployment.

Auto descale based on resource usage

Next, we'll validate that descaling works as well. We'll do that by re-applying the initial definition that has both the memory and the CPU set to eighty percent. Since the actual memory usage is below that, the `HPA` should start scaling down until it reaches the minimum number of replicas.

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

Just as before, we'll wait for a few minutes before we describe the `HPA`.

```
kubectl -n go-demo-5 describe hpa api
```

The **output**, limited to the events messages, is as follows.

```
...
Events:
... Message
    -------
```

```
...
... New size: 2; reason: Current number of replicas below Spec.MinReplicas
... New size: 4; reason: memory resource utilization (percentage of reques
t) above target
... New size: 5; reason: memory resource utilization (percentage of reques
t) above target
... New size: 3; reason: All metrics below target
```

As we can see, it changed the size to `3` since all the `metrics` are `below target`.

A while later, it will descale again to two replicas and stop since that's the limit we set in the `HPA` definition.

In the next lesson, we will see whether to define `HPA` in Deployments or StatefulSets.