Adjusting Resources Based on Actual Usage

In this lesson, we will find out the actual resource usage of running containers and adjust the allocation according to it.

WE'LL COVER THE FOLLOWING

Getting the Actual Resource Usage

We saw some of the effects that can be caused by a discrepancy between resource usage and resource specification. It's only natural that we should adjust our specification to reflect the actual memory and CPU usage better.

The DB Container

Let's start with the database.

```
kubectl top pods
```

The **output** is as follows.

```
NAME              CPU(cores) MEMORY(bytes)
go-demo-2-api-... 0m         4Mi
go-demo-2-api-... 0m         4Mi
go-demo-2-api-... 0m         4Mi
go-demo-2-db-...  4m         34Mi
```

As expected, an `api` container uses even less resources than MongoDB. Its memory is somewhere between `3Mi` and `6Mi`. Its CPU usage is so low that

Metrics Server rounded it to `0m`.

> The memory usage resources may differ from the above mentioned limits.

💡 Things to Keep in Mind

Equipped with this knowledge, we can proceed to update our YAML definition. Still, before we do that, we need to clarify a few things.

The metrics we collected are based on applications that do nothing. Once they start getting a real load and start hosting production size data, the metrics would change drastically.

What you need is a way to predict how much resources an application will use in production, not in a simple test environment. You might be inclined to run stress tests that would simulate a production setup. It's significant, but it does not necessarily result in real production-like behavior.

Replicating the production and behavior of real users is tough. Stress tests will get you half-way. For the other half, you'll have to monitor your applications in production and, among other things, adjust resources accordingly.

There are many additional things you should take into account but, for now, we wanted to stress that applications that do nothing are not a good measure of resource usage. Still, we're going to imagine that the applications we're currently running are under production-like load and that the metrics we retrieved represent how the applications would behave in production.

> ℹ Simple test environments do not reflect production usage of resources. Stress tests are a good start, but not a complete solution. Only production provides real metrics.

Adjusting the Resources

Let's take a look at a new definition that better represents resource usage of the applications.

```
cat pod/go-demo-2.yml
```

```
cat res/go-demo-2.yml
```

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-demo-2-db
spec:
  ...
  template:
    ...
    spec:
      containers:
      - name: db
        image: mongo:3.3
        resources:
          limits:
            memory: "100Mi"
            cpu: 0.1
          requests:
            memory: "50Mi"
            cpu: 0.01
...
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-demo-2-api
spec:
  ...
  template:
    ...
    spec:
      containers:
      - name: api
        image: vfarcic/go-demo-2
        ...
        resources:
          limits:
            memory: "10Mi"
            cpu: 0.1
          requests:
            memory: "5Mi"
            cpu: 0.01
```

That is much better. The resource requests are only slightly higher than the current usage. We set the memory limits value to double that of the requests so that the applications have ample resources for occasional (and short-lived) bursts of additional memory consumption. CPU limits are much higher than the requests mostly because it was not good to put anything less than a tenth of a CPU as the limit.

Anyways, the point is that requests are close to the observed usage and limits

are higher so that applications have some space to breathe in case of a temporary spike in resource usage.

Applying the New Definition

All that's left is to apply the new definition.

```
kubectl apply \
    -f res/go-demo-2.yml \
    --record

kubectl rollout status \
    deployment go-demo-2-api
```

The `deployment "go-demo-2-api"` was `successfully rolled out`, and we can move onto the next subject.

In the next lesson, we will explore the Quality of Service (QoS) contracts.