Defining Container Memory and CPU Resources

In this lesson, we will get familiar with the CPU, memory and other resource management terminologies.

WE'LL COVER THE FOLLOWING

Getting Familiar with the Terminologies

So far, we did not specify how much memory and CPU containers should use, nor what their limits should be. If we do that, Kubernetes' scheduler will have a much better idea about the needs of those containers, and it'll make much better decisions on which nodes to place the Pods and what to do if they start "misbehaving".

Looking into the Definition

Let's take a look at a modified `go-demo-2` definition.

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

The specification is almost the same as those we used before. The only new entries are in the `resources` section.

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: 200Mi
            cpu: 0.5
          requests:
            memory: 100Mi
            cpu: 0.3
...
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: 100Mi
            cpu: 200m
          requests:
            memory: 50Mi
            cpu: 100m
...
```

We specified `limits` and `requests` entries in the `resources` section.

Understanding CPU Resources

CPU resources are measured in `cpu` units. The exact meaning of a `cpu` unit depends on where we host our cluster. If servers are virtualized, one `cpu` unit is equivalent to one virtualized processor (vCPU). When running on bare-metal with Hyperthreading, one `cpu` equals one Hyperthread. For the sake of simplification, we'll assume that one `cpu` resource is one CPU processor (even though that is not entirely true).

If one container is set to use `2` CPU, and the other is set to `1` CPU, the later is

guaranteed half as much processing power.

CPU values can be fractioned. In our example, the `db` container has the CPU requests set to `0.5` which is equivalent to half CPU. The same value could be expressed as `500m`, which translates to five hundred millicpu. If you take another look at the CPU specs of the `api` container, you'll see that its CPU limit is set to `200m` and the requests to `100m`. They are equivalent to `0.2` and `0.1` CPUs.

Understanding Memory Resources

Memory resources follow a similar pattern as CPU. The significant difference is in the units. Memory can be expressed as:

- `K` (kilobyte)
- `M` (Megabyte)
- `G` (Gigabyte)
- `T` (Terabyte)
- `P` (Petabyte)
- `E` (Exabyte).

We can also use the power-of-two equivalents `Ki`, `Mi`, `Gi`, `Ti`, `Pi`, and `Ei`.

If we go back to the `go-demo-2-random.yml` definition, we'll see that the `db` container has the limit set to `200Mi` (two hundred megabytes) and the requests to `100Mi` (one hundred megabytes).

Limits and Requests

We have already mentioned `limits` and `requests` quite a few times and yet we have not explained what each of them mean.

Limits

A limit represents the amount of resources that a container should not pass. The assumption is that we define limits as upper boundaries which, when reached, indicate that something went wrong, as well as a way to guard our resources from being overtaken by a single rogue container due to memory leaks or similar problems.

If a container is restartable, Kubernetes will restart a container that exceeds its memory limit. Otherwise, it might terminate it. Bear in mind that a

terminated container will be recreated if it belongs to a Pod (as all Kubernetes-controlled containers do).

Unlike memory, CPU limits never result in termination or restarts. Instead, a container will not be allowed to consume more than the CPU limit for an extended period.

Requests

Requests represent the expected resource utilization. They are used by Kubernetes to decide where to place Pods depending on actual resource utilization of the nodes that form the cluster.

If a container exceeds its memory requests, the Pod it resides in might be evicted if a node runs out of memory. Such eviction usually results in the Pod being scheduled on a different node, as long as there is one with enough available memory.

If a Pod cannot be scheduled to any of the nodes due to lack of available resources, it enters the pending state waiting until resources on one of the nodes are freed, or a new node is added to the cluster.

In the next lesson, we will get practical by applying the go-demo-2-random.yml definition and playing around with it.