Allocating Insufficient Resource than the Actual Usage

In this lesson, we will explore what happens when we allocate insufficient resource than the actual usage of an application.

WE'LL COVER THE FOLLOWING

Allocating Insufficient Memory

Let's take a look at a slightly modified version of the `go-demo-2` definition.

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

When compared with the previous definition, the difference is only in `resources` of the `db` container in the `go-demo-2-db` Deployment.

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: 20Mi
            cpu: 0.5
          requests:
            memory: 10Mi
```

```
            cpu: 0.3
```

The memory limit is set to `20Mi` and the request to `10Mi`. Since we already know from Metrics Server's data that MongoDB requires around `35Mi`, memory resources are this time, much lower than the actual usage.

Applying the Definition

Let's see what will happen when we apply the new configuration.

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

kubectl get pods
```

We applied the new configuration and retrieved the Pods. The **output** is as follows.

```
NAME              READY STATUS    RESTARTS AGE
go-demo-2-api-... 1/1   Running   0        1m
go-demo-2-api-... 1/1   Running   0        1m
go-demo-2-api-... 1/1   Running   0        1m
go-demo-2-db-...  0/1   OOMKilled 2        17s
```

In your case, the status might not be `OOMKilled`. If so, wait for a while longer and retrieve the Pods again. The status should eventually change to `CrashLoopBackOff`.

As you can see, the status of the `go-demo-2-db` Pod is `OOMKilled` (Out Of Memory Killed). Kubernetes detected that the actual usage is way above the limit and it declared the Pod as a candidate for termination.

The container was terminated shortly afterward. Kubernetes will recreate the terminated container a while later only to discover that the memory usage is still above the limit. And so on, and so forth. The loop will continue.

> **i** A container can exceed its memory request if the node has enough available memory. On the other hand, a container is not allowed to use more memory than the limit. When that happens, it becomes a candidate for termination.

Looking into the Deployment's Description

Let's describe the Deployment and see the status of the `db` container.

```
kubectl describe pod go-demo-2-db
```

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

```
...
Containers:
  db:
    ...
    Last State:     Terminated
      Reason:       OOMKilled
      Exit Code:    137
      ...
Events:
  Type    Reason  Age              From               Message
  ----    ------  ----             ----               -------
  ...
  Warning BackOff 3s (x8 over 1m) kubelet, minikube Back-off restarting failed container
```

We can see that the last state of the db container is `OOMKilled`. When we explore the events, we can see that, so far, the container was restarted eight times with the reason `BackOff`.

In the next lesson, we will explore what happens when an application gets excessive resources allocated for itself than its actual usage.