Exploring the Effects by Violating Quotas

In this lesson, we will violate some Quotas and analyze the consequences.

WE'LL COVER THE FOLLOWING

Exploring the Effects

Now let's create the objects and explore the effects as we defined the resource quotas in the previous lesson.

Creating the dev Namespace

Let's get started by creating the `dev` Namespace as per our plan.

```
kubectl create \
    -f res/dev.yml \
    --record --save-config
```

We can see from the output that the `namespace "dev"` was `created` as well as the `resourcequota "dev"`. To be on the safe side, we'll describe the newly created `dev` quota.

```
kubectl --namespace dev describe \
    quota dev
```

The **output** is as follows.

```
Name:                dev
Namespace:           dev
Resource             Used  Hard
--------             ----  ----
limits.cpu           0     1
limits.memory        0     1Gi
pods                 0     10
requests.cpu         0     800m
requests.memory      0     500Mi
services.nodeports   0     0
```

We can see that the hard limits are set and that there's currently no usage. That was to be expected since we're not running any objects in the `dev` Namespace.

Creating Resources

Let's spice it up a bit by creating the already too familiar `go-demo-2` objects.

```
kubectl --namespace dev create \
    -f res/go-demo-2.yml \
    --save-config --record

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

We created the objects from the `go-demo-2.yml` file and waited until the `go-demo-2-api` Deployment rolled out.

Looking into the Description

Now we can revisit the values of the `dev` quota.

```
kubectl --namespace dev describe \
    quota dev
```

The **output** is as follows.

```
Name:              dev
Namespace:         dev
Resource           Used  Hard
--------           ----  ----
limits.cpu         400m  1
limits.memory      130Mi 1Gi
pods               4     10
requests.cpu       40m   800m
requests.memory    65Mi  500Mi
services.nodeports 0     0
```

Judging from the `Used` column, we can see that we are, for example, currently running `4` Pods and that we are still below the limit of `10`. One of those Pods was created through the `go-demo-2-db` Deployment, and the other three with the `go-demo-2-api`. If you summarize resources we specified for the containers that form those Pods, you'll see that the values match the used `limits` and `requests`.

Violating the Number of Pods

So far, we did not reach any of the quotas. Let's try to break at least one of them.

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

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

```
...
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: go-demo-2-api
spec:
  replicas: 15
  ...
```

The definition of the `go-demo-2-scaled.yml` is almost the same as the one in `go-demo-2.yml`. The only difference is that the number of replicas of the `go-demo-2-api` Deployment is increased to fifteen. As you already know, that should result in fifteen Pods created through that Deployment.

Applying the Definition

I'm sure you can guess what will happen if we apply the new definition. We'll do it anyway.

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

We applied the new definition. We'll give Kubernetes a few moments to do the work before we take a look at the events it'll generate. So, take a deep breath and count from one to the number of processors in your machine.

```
kubectl --namespace dev get events
```

The **output** of a few of the events generated inside the `dev` Namespace is as follows.

```
...
... Error creating: pods "go-demo-2-api-..." is forbidden: exceeded quota: dev, requested: li
13s         13s          1         go-demo-2-api-6bd767ffb6.150f51f4b3a7ed3f         ReplicaS
...
```

We can see that we reached two of the limits imposed by the Namespace quota. We reached the maximum amount of CPU (`1`) and Pods (`10`). As a result, ReplicaSet controller was forbidden from creating new Pods.

Analyzing the Error

We should be able to confirm which hard limits were reached by describing the `dev` Namespace.

```
kubectl describe namespace dev
```

The **output**, limited to the `Resource Quotas` section, is as follows.

```
...
Resource Quotas
 Name:               dev
 Resource            Used   Hard
 --------            ---    ---
 limits.cpu          1      1
```

```
 limits.memory       190Mi  1Gi
 pods                10     10
 requests.cpu        100m   800m
 requests.memory     95Mi   500Mi
 services.nodeports  0      0
...
```

As the events showed us, the values of `limits.cpu` and `pods` resources are the same in both `User` and `Hard` columns. As a result, we won't be able to create any more Pods, nor will we be allowed to increase CPU limits for those that are already running.

Finally, let's take a look at the Pods inside the `dev` Namespace.

```
kubectl get pods --namespace dev
```

```
NAME              READY STATUS  RESTARTS AGE
go-demo-2-api-... 1/1   Running 0        3m
go-demo-2-api-... 1/1   Running 0        3m
go-demo-2-api-... 1/1   Running 0        5m
go-demo-2-api-... 1/1   Running 0        3m
go-demo-2-api-... 1/1   Running 0        5m
go-demo-2-api-... 1/1   Running 0        3m
go-demo-2-api-... 1/1   Running 0        3m
go-demo-2-api-... 1/1   Running 0        3m
go-demo-2-api-... 1/1   Running 0        5m
go-demo-2-db-...  1/1   Running 0        5m
```

The `go-demo-2-api` Deployment managed to create nine Pods. Together with the Pod created through the `go-demo-2-db`, we reached the limit of ten.

Reverting Back to Previous Definition

We confirmed that the limit and the Pod quotas work. We'll revert to the previous definition (the one that does not reach any of the quotas) before we move onto the next verification.

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

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

The **output** of the latter command should indicate that the `deployment "go-demo-2-api"` was `successfully rolled out`.

demo-2-api was successfully rolled out.

Violating the Memory Quota

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

```
cat res/go-demo-2-mem.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:
  replicas: 3
  ...
  template:
    ...
    spec:
      containers:
      - name: api
        ...
        resources:
          limits:
            memory: "200Mi"
            cpu: 0.1
          requests:
            memory: "200Mi"
            cpu: 0.01
...
```

Both memory request and limit of the `api` container of the `go-demo-2-api`

Deployment is set to `200Mi` while the database remains with the memory request of `50Mi`. Knowing that the `requests.memory` quota of the `dev` Namespace is `500Mi`, it's enough to do simple math and come to the conclusion that we won't be able to run all three replicas of the `go-demo-2-api` Deployment.

Applying the Definition

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

Just as before, we should wait for a while before taking a look at the events of the `dev` Namespace.

```
kubectl --namespace dev get events \
    | grep mem
```

The **output**, limited to one of the entries, is as follows.

```
... Error creating: pods "go-demo-2-api-..." is forbidden: exceeded quota: dev, requested: re
```

We reached the quota of the `requests.memory`. As a result, creation of at least one of the Pods is forbidden. We can see that we requested creation of a Pod that requests `200Mi` of memory. Since the current summary of the memory requests is `455Mi`, creating that Pod would exceed the allocated `500Mi`.

Analyzing the Error

Let's take a closer look at the Namespace.

```
kubectl describe namespace dev
```

The **output**, limited to the `Resource Quotas` section, is as follows.

```
...
Resource Quotas
 Name:               dev
 Resource            Used   Hard
 --------            ---    ---
 limits.cpu          400m   1
 limits.memory       510Mi  1Gi
```

```
 pods                  4       10
 requests.cpu          40m     800m
 requests.memory       455Mi   500Mi

 services.nodeports    0       0
...
```

Indeed, the amount of used memory requests is `455Mi`, meaning that we could create additional Pods with up to `45Mi`, not `200Mi`.

Reverting Back to the Previous Definition

We'll revert to the `go-demo-2.yml` one more time before we explore the last quota we defined.

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

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

Violating the Services Quota

The only quota we did not yet verify is `services.nodeports`. We set it to `0` and, as a result, we should not be allowed to expose any node ports. Let's confirm that is indeed true.

```
kubectl expose deployment go-demo-2-api \
    --namespace dev \
    --name go-demo-2-api \
    --port 8080 \
    --type NodePort
```

The **output** is as follows.

```
Error from server (Forbidden): services "go-demo-2-api" is forbidden: exceeded quota: dev, re
```

All our quotas work as expected. But, there are others. We won't have time to explore examples of all the quotas we can use. Instead, we'll list them all for future reference.

In the next lesson, we will explore the types of Quotas we can define.