Lesson 03 Demo 04

Managing Container Resources with a Resource Quota

Objective: To create a namespace with a resource quota for managing resources within a Kubernetes container

Tools required: kubeadm, kubectl, kubelet, and containerd

Prerequisites: A Kubernetes cluster (refer to Demo 01 from Lesson 01 for setting up a cluster)

Steps to be followed:

1. Create a namespace with a resource quota
2. Validate the resource quota by creating pods

Step 1: Create a namespace with a resource quota

1.1 Execute the following command to create a new namespace named **quotaz**:
kubectl create ns quotaz

```
labsuser@master:~$ kubectl create ns quotaz
namespace/quotaz created
labsuser@master:~$
```

1.2 Execute the following command to check the created namespace:
kubectl get ns

```
labsuser@master:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   6m10s
kube-node-lease   Active   6m10s
kube-public       Active   6m10s
kube-system       Active   6m10s
quotaz            Active   3m36s
labsuser@master:~$
```

1.3 Execute the following command to check the resources within the **quotaz** namespace:
kubectl get all -n quotaz

```
labsuser@master:~$ kubectl get all -n quotaz
No resources found in quotaz namespace.
labsuser@master:~$
```

You can see that there are no resources within the **quotaz** namespace.

1.4 Execute the following command to create a resource quota YAML file:
vi resourcequota.yaml

```
labsuser@master:~$ vi resourcequota.yaml
```

1.5 Write the following code inside the **resourcequota.yaml** file:

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: mem-cpu-demo
 namespace: quotaz
spec:
 hard:
  requests.cpu: "1"
  requests.memory: 1Gi
  limits.cpu: "2"
  limits.memory: 2Gi
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
  namespace: quotaz
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
~
~
~
```

1.6 Execute the following command to apply the resource quota configuration from the specified YAML file to the Kubernetes cluster:
kubectl apply -f resourcequota.yaml

```
labsuser@master:~$ kubectl apply -f resourcequota.yaml
resourcequota/mem-cpu-demo created
labsuser@master:~$
```

1.7 Run the following command to view the resource quota allotted to the **quotaz** namespace:
kubectl get resourcequota -n quotaz

```
labsuser@master:~$ kubectl get resourcequota -n quotaz
NAME           AGE     REQUEST                                      LIMIT
mem-cpu-demo   2m35s   requests.cpu: 0/1, requests.memory: 0/1Gi   limits.cpu: 0/2, limits.memory: 0/2Gi
labsuser@master:~$
```

1.8 Verify the details of the created resource quota using the following command:
kubectl describe resourcequota -n quotaz

```
labsuser@master:~$ kubectl describe resourcequota -n quotaz
Name:            mem-cpu-demo
Namespace:       quotaz
Resource         Used  Hard
--------         ----  ----
limits.cpu       0     2
limits.memory    0     2Gi
requests.cpu     0     1
requests.memory  0     1Gi
labsuser@master:~$
```

Step 2: Validate the resource quota by creating pods

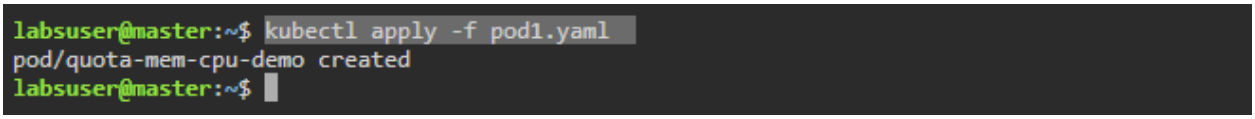
2.1 Run the following command to create a YAML file that will define a pod within resource quota limits:
vi pod1.yaml

```
labsuser@master:~$ vi pod1.yaml
```

2.2 Write the following code inside the **pod1.yaml** file:

```
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo
  namespace: quotaz
spec:
   containers:
- name: quota-mem-cpu-demo-ctr
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "800m"
      requests:
        memory: "600Mi"
        cpu: "400m"
```

2.3 Execute the following command to apply the created pod to the Kubernetes cluster:
kubectl apply -f pod1.yaml



2.4 Execute the following command to check the used resources:
kubectl describe resourcequota -n quotaz

```
pod/quota-mem-cpu-demo created
labsuser@master:~$ kubectl describe resourcequota -n quotaz
Name:            mem-cpu-demo
Namespace:       quotaz
Resource         Used   Hard
--------         ----   ----
limits.cpu       800m   2
limits.memory    800Mi  2Gi
requests.cpu     400m   1
requests.memory  600Mi  1Gi
labsuser@master:~$
```

2.5 Execute the following command to create a YAML file that defines another pod:
vi pod2.yaml

```
labsuser@master:~$ vi pod2.yaml
```

2.6 Write the following code inside the **pod2.yaml** file:

```
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo-2
  namespace: quotaz
spec:
  containers:
  - name: quota-mem-cpu-demo-2-ctr
    image: redis
    resources:
      limits:
        memory: "2Gi"
        cpu: "800m"
      requests:
        memory: "700Mi"
        cpu: "400m"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo-2
  namespace: quotaz
spec:
  containers:
  - name: quota-mem-cpu-demo-2-ctr
    image: redis
    resources:
      limits:
        memory: "2Gi"
        cpu: "800m"
      requests:
        memory: "700Mi"
        cpu: "400m"
```

2.7 Execute the following command to apply the second pod to the Kubernetes cluster:
kubectl apply –f pod2.yaml

```
labsuser@master:~$ kubectl apply -f pod2.yaml
Error from server (Forbidden): error when creating "pod2.yaml": pods "quota-mem-cpu-demo-2" is forbidden: exceeded quota: mem-cpu-demo, requested: limits.memory=2Gi,requests.memory=700Mi, used: limits.memory=800Mi,requests.memory=600Mi, limited: limits.memory=2Gi,requests.memory=1Gi
labsuser@master:~$
```

An error occurs, which proves that resources cannot be created beyond the resource quota set for a specific namespace.

By following these steps, you have successfully created a namespace with a resource quota to manage resources within a Kubernetes container.