### **Lab Exercise 10- Implementing Resource Quota in Kubernetes**

Jahnavi Anand | B1

500106908 | R2142220089

### **Objective:**

In Kubernetes, Resource Quotas are used to control the resource consumption of namespaces. They help in managing and enforcing limits on the usage of resources like CPU, memory, and the number of objects (e.g., Pods, Services) within a namespace. This exercise will guide you through creating and managing Resource Quotas to limit the resources used by applications in a specific namespace.

**Step 1: Understand Resource Quotas**

Resource Quotas allow you to:

* Limit the amount of CPU and memory a namespace can use.
* Control the number of certain types of resources (e.g., Pods, Services, PersistentVolumeClaims) in a namespace.
* Prevent a namespace from consuming more resources than allocated, ensuring fair usage across multiple teams or applications.

**Step 2: Create a Namespace**

First, create a namespace where you will apply the Resource Quota. This helps in isolating and controlling resource usage within that specific namespace.

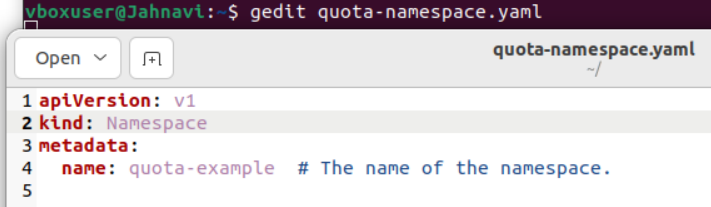
Create a YAML file named ***quota-namespace.yaml*** with the following content:

apiVersion: v1

kind: Namespace

metadata:

name: quota-example # The name of the namespace.



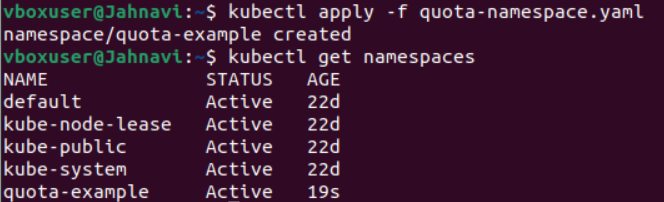
Apply the YAML to create the namespace:

kubectl apply -f quota-namespace.yaml

Verify that the namespace is created:

kubectl get namespaces

You should see quota-example listed in the output.



**Step 3: Define a Resource Quota**

Next, create a Resource Quota YAML file named ***resource-quota.yaml*** with the following content:

apiVersion: v1

kind: ResourceQuota

metadata:

name: example-quota # The name of the Resource Quota.

namespace: quota-example # The namespace to which the Resource Quota will apply.

spec:

hard: # The hard limits imposed by this Resource Quota.

requests.cpu: "2" # The total CPU resource requests allowed in the namespace (2 cores).

requests.memory: "4Gi" # The total memory resource requests allowed in the namespace (4 GiB).

limits.cpu: "4" # The total CPU resource limits allowed in the namespace (4 cores).

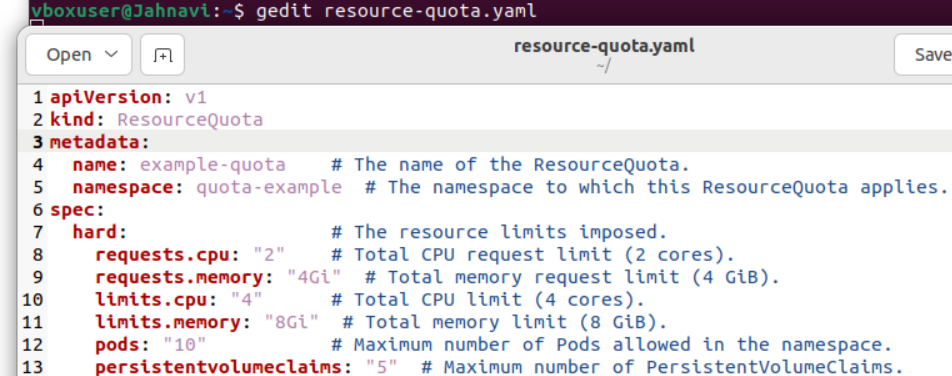
limits.memory: "8Gi" # The total memory resource limits allowed in the namespace (8 GiB).

pods: "10" # The total number of Pods allowed in the namespace.

persistentvolumeclaims: "5" # The total number of PersistentVolumeClaims allowed in the namespace.

configmaps: "10" # The total number of ConfigMaps allowed in the namespace.

services: "5" # The total number of Services allowed in the namespace.

****

**Step 4: Apply the Resource Quota**

Apply the Resource Quota YAML to the namespace:

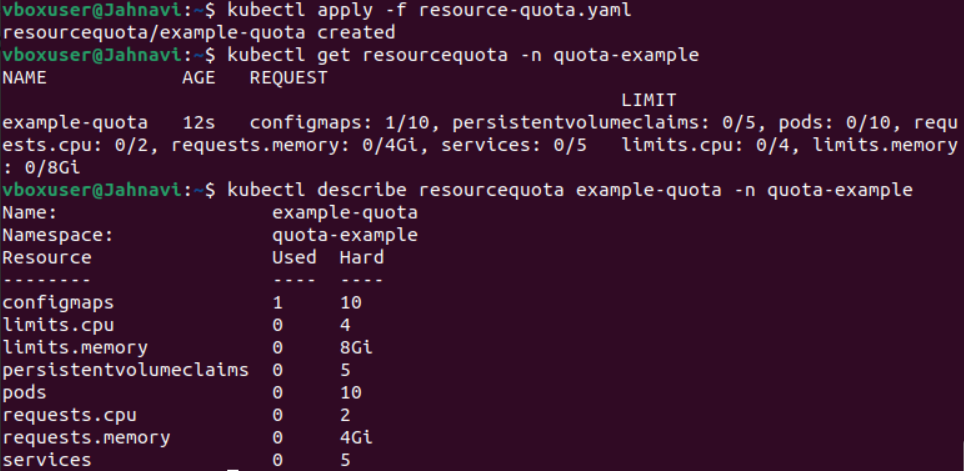
kubectl apply -f resource-quota.yaml

Verify that the Resource Quota is applied:

kubectl get resourcequota -n quota-example

To see the details of the applied Resource Quota:

kubectl describe resourcequota example-quota -n quota-example

****

**Step 5: Test the Resource Quota**

Let's create some resources in the quota-example namespace to see how the Resource Quota affects them.

Deploy a ReplicaSet with Resource Requests and Limits

Create a YAML file named ***nginx-replicaset-quota.yaml*** with the following content:

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: nginx-replicaset

namespace: quota-example

spec:

replicas: 5 # Desired number of Pod replicas.

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

resources: # Define resource requests and limits.

requests:

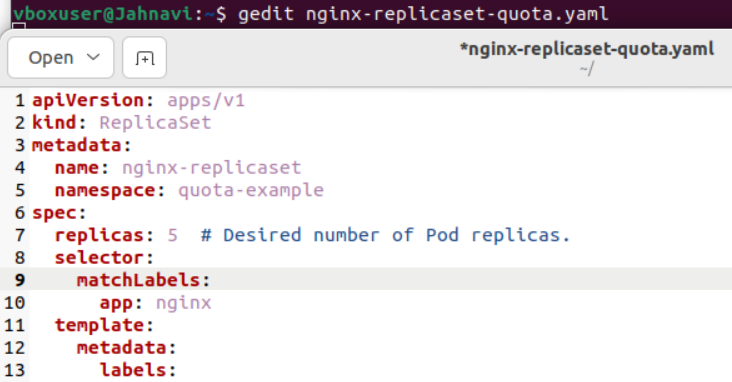
memory: "100Mi"

cpu: "100m"

limits:

memory: "200Mi"

cpu: "200m"

****

**Explanation:**

This ReplicaSet requests a total of 500m CPU and 500Mi memory across 5 replicas.

It also limits each replica to use a maximum of 200m CPU and 200Mi memory.

Apply this YAML to create the ReplicaSet:

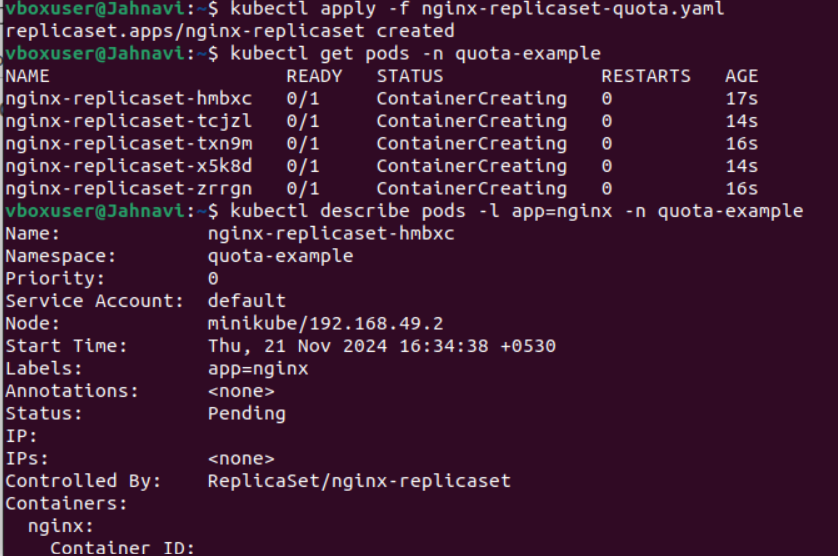
kubectl apply -f nginx-replicaset-quota.yaml

Check the status of the Pods and ensure they are created within the constraints of the Resource Quota:

kubectl get pods -n quota-example

To describe the Pods and see their resource allocations:

kubectl describe pods -l app=nginx -n quota-example



Attempt to Exceed the Resource Quota

Try creating additional resources to see if they are rejected when exceeding the quota. For example, create more Pods or increase the CPU/memory requests to exceed the quota limits.

Create a YAML file named ***nginx-extra-pod.yaml*** with the following content:

apiVersion: v1

kind: Pod

metadata:

name: nginx-extra-pod

namespace: quota-example

spec:

containers:

- name: nginx

image: nginx:latest

resources:

requests:

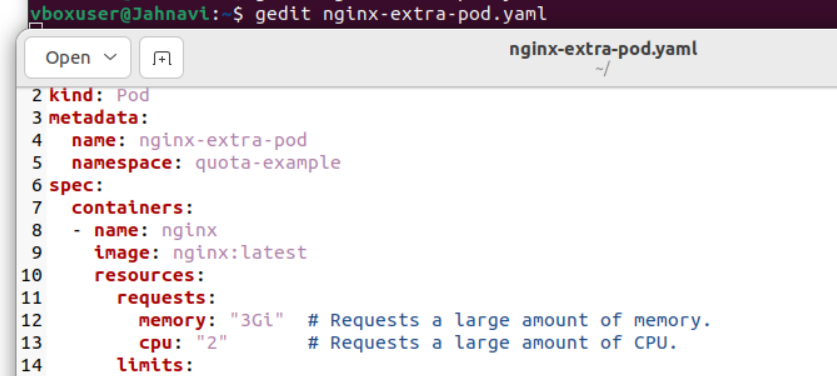
memory: "3Gi" # Requests a large amount of memory.

cpu: "2" # Requests a large amount of CPU.

limits:

memory: "4Gi"

cpu: "2"



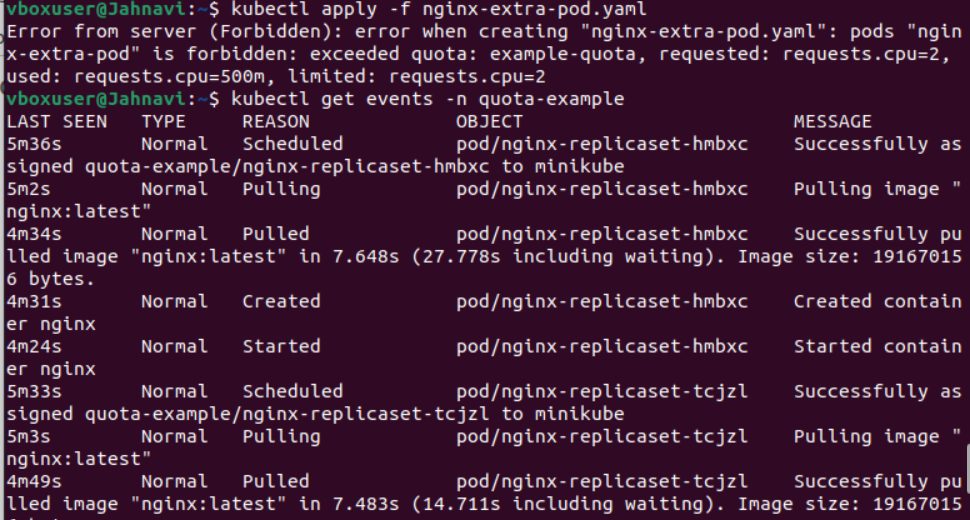
Apply this YAML to create the Pod:

kubectl apply -f nginx-extra-pod.yaml

This should fail due to exceeding the Resource Quota. Check the events to see the failure reason:

kubectl get events -n quota-example

Look for error messages indicating that the Pod creation was denied due to resource constraints.



**Step 6: Clean Up Resources**

To delete the resources you created:

kubectl delete -f nginx-replicaset-quota.yaml

kubectl delete -f nginx-extra-pod.yaml

kubectl delete -f resource-quota.yaml

kubectl delete namespace quota-example

