Lab 11 – Configure Quotas and Limits

Quotas and Limits

Namespaces let different users or teams to share a cluster with a fixed number of nodes. It can be a concern that one team could use more than its fair share of resources. Resource quotas are the tool to address this concern.

A resource quota provides constraints that limit aggregate resource consumption per namespace. It can limit the quantity of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed in that project.

Users create resources in the namespace, and the quota system tracks usage to ensure it does not exceed hard resource limits defined in the resource quota. If creating or updating a resource violates a quota constraint, the request will fail. When quota is enabled in a namespace for compute resources like cpu and memory, users must specify resources consumption, otherwise the quota system rejects pod creation.

Applying Resource Quotas and Limits

This lab demonstrates a typical setup to control for resource usage in a namespace.

The cluster-admin is operating a cluster on behalf of a user population and the cluster-admin wants to control the amount of resources that can be consumed in a particular namespace to promote fair sharing of the cluster and control cost.

The cluster-admin has the following goals:

- Limit the amount of compute resource for running pods
- Limit the number of persistent volume claims to control access to storage
- Limit the number of load balancers to control cost
- Prevent the use of node ports to preserve scarce resources
- Provide default compute resource requests to enable better scheduling decisions

Create a namespace

Let’s create a new namespace called quota-example:

Copy

```
kubectl config set-context quota-example
```

Output:

```
Context "quota-example" created.
```

Copy

```
kubectl config use-context quota-example
```

Output:

```
Switched to context "quota-example"
```

Copy

```
kubectl config get-contexts
```

Output:

```
CURRENT   NAME                          CLUSTER      AUTHINFO
NAMESPACE

          kubernetes-admin@kubernetes   kubernetes   kubernetes-admin

          multi-app



          –help
```

Copy

```
cat > quota.yaml <<EOF

apiVersion: v1

kind: ResourceQuota
```

```
metadata:
  name: quota
spec:
  hard:
    cpu: "20"
    memory: 1Gi
    persistentvolumeclaims: "10"
    pods: "10"
    replicationcontrollers: "20"
    resourcequotas: "1"
    secrets: "10"
    services: "5"
EOF
```

Copy

```
kubectl create -f quota.yaml
```

Output:

```
resourcequota "quota" created
```

Copy

```
kubectl describe quota quota
```

Output:

```
Name:          project-quota
Namespace:     default
Resource       Used  Hard
--------       ----  ----
limits.cpu     0     1
limits.memory  0     1Gi
pods           0     10


Name:                   quota
Namespace:              default
Resource                Used  Hard
--------                ----  ----
cpu                     0     20
memory                  0     1Gi
persistentvolumeclaims  1     10
pods                    0     10
replicationcontrollers  0     20
resourcequotas          2     1
secrets                 4     10
services                1     5
```

Applying default resource requests and limits

Pod authors rarely specify resource requests and limits for their pods.

Since we applied a quota to our project, let’s see what happens when an end-user creates a pod that has unbounded cpu and memory by creating an nginx container.

Copy

```
kubectl run nginx1 --image=nginx --replicas=1
```

Output:

```
deployment.apps "nginx1" created
```

Now let’s look at the pods that were created.

Copy

```
kubectl get pods
```

Output:

```
NAME                     READY     STATUS    RESTARTS   AGE

nginx1-65899c769f-kbgj7   1/1       Running   0          59s
```

Copy

```
kubectl describe deployment nginx1
```

Output:

```
Name:                   nginx

Namespace:              quota-example

CreationTimestamp:      Fri, 13 Apr 2018 14:44:30 +0000

Labels:                 run=nginx
```

```
Annotations:            deployment.kubernetes.io/revision=1

Selector:               run=nginx

Replicas:               1 desired | 1 updated | 1 total | 1 available
| 0 unavailable

StrategyType:           RollingUpdate

MinReadySeconds:        0

RollingUpdateStrategy:  1 max unavailable, 1 max surge

Pod Template:

  Labels:  run=nginx

  Containers:

   nginx:

    Image:        nginx

    Port:

    Host Port:

    Environment:

    Mounts:

  Volumes:

Conditions:

  Type           Status  Reason

  ----           ------  ------

  Available      True    MinimumReplicasAvailable

  Progressing    True    NewReplicaSetAvailable
```

```
OldReplicaSets:

NewReplicaSet:   nginx-65899c769f (1/1 replicas created)

Events:

  Type    Reason             Age   From                   Message

  ----    ------             ----  ----                   -------

  Normal  ScalingReplicaSet  1m    deployment-controller  Scaled up re
plica set nginx-65899c769f to 1
```

Copy

```
kubectl describe rs nginx1
```

Output:

```
Name:           nginx-65899c769f

Namespace:      quota-example

Selector:       pod-template-hash=2145573259,run=nginx

Labels:         pod-template-hash=2145573259

                run=nginx

Annotations:    deployment.kubernetes.io/desired-replicas=1

                deployment.kubernetes.io/max-replicas=2

                deployment.kubernetes.io/revision=1

Controlled By:  Deployment/nginx

Replicas:       1 current / 1 desired

Pods Status:    1 Running / 0 Waiting / 0 Succeeded / 0 Failed
```

```
Pod Template:

  Labels:  pod-template-hash=2145573259

           run=nginx

  Containers:

   nginx:

    Image:        nginx

    Port:

    Host Port:

    Environment:

    Mounts:

  Volumes:

Events:

  Type    Reason            Age   From                   Message

  ----    ------            ----  ----                   -------

  Normal  SuccessfulCreate  1m    replicaset-controller  Created pod:
nginx-65899c769f-kbgj7
```

Copy

```
cat > limits.yaml <<EOF

apiVersion: v1

kind: LimitRange

metadata:

  name: limits
```

```
spec:

  limits:

  - default:

      cpu: 200m

      memory: 512Mi

    defaultRequest:

      cpu: 100m

      memory: 256Mi

    type: Container

EOF
```

Copy

```
kubectl create -f limits.yaml
```

Output:

```
limitrange "limits" created
```

Copy

```
kubectl describe limits limits
```

Output:

```
Name:       limits

Namespace:  quota-example
```

```
Type        Resource  Min  Max  Default Request  Default Limit  Max Li
mit/Request Ratio

----        --------  ---  ---  ---------------  -------------  ------
-----------------

Container   cpu       -    -    100m             200m           -

Container   memory    -    -    256Mi            512Mi          -
```

Copy

```
kubectl run nginx2 --image=nginx --replicas=1 --requests=cpu=100m,memo
ry=256Mi --limits=cpu=200m,memory=512Mi
```

Copy

```
kubectl get pods
```

Output:

```
NAME                      READY     STATUS    RESTARTS   AGE

nginx1-65899c769f-8hgf7    1/1       Running    0          9m

nginx2-7f4cff6589-gr95w   1/1       Running   0          5m
```

Copy

```
kubectl describe pod nginx2
```