18Oct2022

Day**11**

Kubernetes Resource Manager II

Policies-Resource Quotas

Policies-Limit Ranges

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

Monitoring resources through Metrix-Server

https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/#metrics-server

https://github.com/kubernetes-sigs/metrics-server

Metrics Server is a scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines.

```
# kubectl top nodes
error: Metrics API not available
# kubectl top pod
error: Metrics API not available
```

Installation

https://github.com/kubernetes-sigs/metrics-server

```
#  kubectl   apply   -f  https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
# kubectl get deployments.apps -A
# kubectl get deployments.apps --namespace kube-system
# kubectl top nodes
Error from server (ServiceUnavailable): the server is currently unable to handle the request (get nodes.metrics.k8s.io)
# kubectl get deployments.apps --namespace kube-system
# kubectl get pods --namespace kube-system
# kubectl logs pods/metrics-server-678f4bf65b-jnm2f --namespace kube-system
```

to disable tls certification request,

```
# kubectl edit deployments.apps --namespace kube-system metrics-server
42        - --kubelet-preferred-address-types=InternalIP                                   ->modify to ‘InternalIP’ remove others
43        - --kubelet-insecure-tls                                                          ->append this line
```

Note: If metric server doesn’t start after above steps then also do this below line

Line below dnsPolicy: ClusterFirst

90 hostNetwork: true

:wq!

```
# kubectl top nodes
Error from server (ServiceUnavailable): the server is currently unable to handle the request (get nodes.metrics.k8s.io)
# kubectl rollout restart deployment --namespace kube-system metrics-server
# kubectl top nodes
NAME                 CPU(cores)  CPU%   MEMORY(bytes)   MEMORY%
master1.example.com  152m         7%     1684Mi          43%
node1.example.com     60m         6%     543Mi           28%
```

Resource Management for Pods and Containers

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

-When you specify a Pod, you can optionally specify how much of each resource a container need.

-The most common resources to specify are:

1- CPU
2- memory (RAM)

Resource types

CPU and memory are each a resource type

A resource type has a base unit.

CPU represents compute processing and is specified in units of Kubernetes CPUs.

CPU measurement is in millicores. with this measure format each **1CPU** is splinted into **1000units(millicores)**

Memory is specified in units of **bytes**.

Memory can express as plain integer in B, K, M, G, T, P

Requests and Limits

Requests ->minimum requirement to start

when you specify the resource request for containers in a Pod, the kube-scheduler uses this information to decide which node to place the Pod on.

Limits ->maximum access to resources

when you specify a resource limit for a container, the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set.

ex

```
# kubectl run nginx --image nginx -o yaml --dry-run=client >nginxpod.yaml
# vim nginxpod.yaml
apiVersion: v1
kind: Pod
metadata:
name: nginx
spec:
containers:
  - image: nginx
    name:  nginx
    resources:
     requests:                                        ->to start, minimum
       cpu: "4m"
       memory: "2Mi"
     limits:                                          ->to make limit, maximum
       cpu: "8m"
       memory: "4Mi"
:wq!
```

```
# kubectl top pods
NAME                     CPU(cores)  MEMORY(bytes)
rediscj-27764091-tfq6k   2m          2Mi
```

Policies

https://kubernetes.io/docs/concepts/policy/

Resource Quotas

https://kubernetes.io/docs/concepts/policy/resource-quotas/

-When several users or teams share a cluster with a fixed number of nodes, there is a concern that one team could use more than its fair share of resources.

-Resource quotas are a tool for administrators to address this concern.

-A resource quota, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per **namespace**.

Implement Resource Quotas

1- create namespace
2- assign namespace to specific node/nodes
3- Enable Resource Quotas and define it on Namespace

1- create namespace

```
# kubectl create namespace testspace -o yaml --dry-run=client >testspace.yaml
# vim testspace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: testspace
:wq!
# kubectl apply -f testspace.yaml
# kubectl get ns
testspace        Active  6s
```

2- assign namespace to specific node/nodes

2-1-enable PodNodeSelector

```
# vim /etc/kubernetes/manifests/kube-apiserver.yaml
20    - --enable-admission-plugins=NodeRestriction,PodNodeSelector
:wq!
# watch kubectl get nodes
```

2-2-lable target node/nodes and namespace

```
# kubectl label nodes node2.example.com lbl=testspace
# kubectl get nodes node2.example.com --show-labels
# cat testspace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: testspace
  annotations:
    scheduler.alpha.kubernetes.io/node-selector: lbl=testspace
:wq!
# kubectl apply -f testspace.yaml
```

verify

```
# kubectl run nginx --image nginx --namespace testspace
# kubectl get pods --namespace testspace -o wide
nginx   0/1    ContainerCreating   0        16s  <none>   node2.example.com
# kubectl delete pod --namespace testspace nginx --force --grace-period=0
```

3- define Resource Quotas on Namespace

```
# kubectl api-resources | grep -i "resource"
resourcequotas              quota       v1                   true        ResourceQuota
#  vim  /etc/kubernetes/manifests/kube-apiserver.yaml
20    - --enable-admission-plugins=NodeRestriction,PodNodeSelector,ResourceQuota
:wq!
# watch kubectl get nodes
# cat testspacerq.yaml
kind: ResourceQuota
apiVersion: v1
metadata:
  name: testspacerq
namespace: testspace
spec:
  hard:
    pods: 5 requests.cpu:
    "10m"
    requests.memory: "8Mi"
    limits.cpu: "18m"
    limits.memory: "12Mi"
:wq!
# kubectl apply -f testspacerq.yaml --dry-run=client
# kubectl apply -f testspacerq.yaml
# kubectl get resourcequotas --namespace testspace
NAME            AGE   REQUEST                                                   LIMIT
testspacerq   11s   pods: 0/5, requests.cpu: 0/10m, requests.memory: 0/8Mi   limits.cpu: 0/18m, limits.memory: 0/12Mi
# kubectl describe  resourcequotas --namespace testspace
```

Verify

```
# kubectl create deployment dpl1 --image redis --replicas 6 --namespace testspace
# kubectl run redis --image redis --namespace testspace -o yaml --dry-run=client >redispod.yaml
# vim nginxpod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginxpod
  namespace: testspace
spec:
  containers:
  - image: nginx
    name: nginxcnt
    resources:
      requests:
        cpu: "11m"
        memory: "6Mi"
      limits:
        cpu: "19m"
        memory: "8Mi"
:wq!
# kubectl create -f redispod.yaml
Error from server (Forbidden): error when creating "redispod.yaml": pods "redispod" is forbidden: exceeded quota: testspacerq, requested: limits.cpu=19m,requests.cpu=11m, used:
limits.cpu=0,requests.cpu=0, limited: limits.cpu=18m,requests.cpu=10m
# cat nginxpod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginxpod
  namespace: testspace
spec:
  containers:
  - image: nginx
    name: nginxcnt
    resources:
      requests:
        cpu: "3m"
        memory: "4Mi"
      limits:
        cpu: "6m"
        memory: "12Mi"
:wq!
# kubectl create -f redispod.yaml
# kubectl get pods --namespace testspace nginxpod
# kubectl describe  pods --namespace testspace nginxpod
# kubectl describe resourcequotas --namespace testspace testspacerq
Name:                   testspacerq
Namespace:              testspace
Resource                Used  Hard
limits.cpu              6m    18m
limits.memory           12Mi  12Mi
pods                    1     5
requests.cpu            3m    10m
requests.memory         4Mi   8Mi
```

Limit Ranges

https://kubernetes.io/docs/concepts/policy/limit-range/

-By default, containers run with unbounded compute resources on a Kubernetes cluster.

-Using Kubernetes resource quotas, administrators can restrict consumption and creation of cluster resources (such as CPU, memory, and persistent storage) within a specified namespace.

issue

-Within a namespace, a Pod can consume as much CPU and memory as is allowed by the ResourceQuotas that apply to that namespace. ->issue is here

-As a cluster operator, or as a namespace-level administrator, you might also be concerned about making sure that a single object cannot monopolize all available resources within a namespace.

solution

A LimitRange is a policy to constrain the resource allocations (limits and requests) that you can specify for each applicable object kind (such as Pod or PersistentVolumeClaim) in a namespace.

Enable LimitRange

by default, is Enabled

Implementing Limit Ranges

1- define Metrics Server
2- assign namespace to specific node/nodes
3- implement Resource Quotas
4- implement LimitRanges

```
# kubectl top nodes
# cat testspace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: testspace
  annotations:
    scheduler.alpha.kubernetes.io/node-selector: lbl=testspace
# kubectl get nodes node2.example.com --show-labels
node2.example.com Ready  <none> 82d  v1.24.3  beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=node2.example.com,kubernetes.io/os=linux,lbl=testspace
# kubectl get  resourcequotas --namespace testspace testspacerq
NAME         AGE   REQUEST                                   LIMIT
testspacerq  41m   pods: 0/5, requests.cpu: 0/10m, requests.memory: 0/8Mi   limits.cpu: 0/18m, limits.memory: 0/12Mi
# kubectl api-resources | grep -i "limit"
limitranges                 limits       v1                          true        LimitRange
# vim testspacelr.yaml
kind: LimitRange
apiVersion: v1
metadata:
  name: testspacelr
namespace: testspace
spec:
  limits:
   - type: Container          ->Object Name
     min:
       cpu: "3m"              ->min amount of CPU that single container can request it if defines it inside Pod.
     memory: "5Mi"          ->min amount of MEMORY that single container can request it if defines it inside Pod.
     max:
       cpu: "5m"              ->max amount of CPU that single container can achieve it if defines it inside Pod.
       memory: "14Mi"         ->max amount of MEMORY that single container can achieve it if defines it inside Pod.
     defaultRequest:                                                                                   ->requests
       cpu: “4m”              ->min amount of CPU that single container can request it if doesn’t define it inside Pod.          shouldn’t less than min-cpu
       memory: “6Mi”          ->min amount of MEMORY that single container can request it if doesn’t define it inside Pod.       shouldn’t less than min-memory
     default:                                                                                          ->limits
       cpu: “4m”              ->max amount of CPU that single container can achieve it if doesn’t define it inside Pod.          shouldn’t greater than min-cpu
       memory: “12Mi”         ->max amount of MEMORY that single container can achieve it if doesn’t define it inside Pod.       shouldn’t greater than min-memory
:wq!
# kubectl apply -f testspacelr.yaml --dry-run=client
# kubectl apply -f testspacelr.yaml
# kubectl describe limitranges --namespace testspace
# kubectl describe resourcequotas --namespace testspace
```

verify

```
# cat nginxpod2.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginxpod
  namespace: testspace
spec:
  containers:
  - image: nginx
    name: nginxcnt
```

```
# cat nginxpod1.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginxpod
  namespace: testspace
spec:
  containers:
  - image: nginx
    name: nginxcnt
    resources:
    requests:
        cpu: "3m"
      memory: "5Mi"
      limits:
        cpu: "5m"
        memory: "12Mi"
```