Resources management limits and quotas

Containers Resources Management

- 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.
- 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.

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: frontend
spec:
 containers:
 - name: app
   image: nginx
   resources:
     requests:
       memory: "64Mi"
       cpu: "250m"
     limits:
       memory: "128Mi"
       cpu: "500m"
```

Containers Resources Management

What are requests and limits?

Requests and limits are the mechanisms Kubernetes uses to control containers resources such as CPU, memory, and ephemeral storage.

1. **Requests**: resources that container is guaranteed to get by Kubernetes. It's the minimum amount of resources that are needed to work.
2. **Limits**: resources that container should not pass. The container is only allowed to go up to that threshold, otherwise Kubernetes will restrict it.

Containers Resource Types

There are 3 core resources that could be configured via requests and limits: (those resources used from underneath nodes)

- **CPU**
  Measured in 1 vCPU/Core; thus, half of CPU core represented as "0.5" which also equivalent to "500m".
- **Memory**
  Measured in bytes and can expressed as a plain integer or using suffixes, the following are same value: "128974848", "129M", "123Mi".
- **Local ephemeral storage** (e.g. 'emptyDir' volume)
  Measured in bytes and can expressed as a plain integer or using suffixes, the following are same value: "128974848", "129M", "123Mi".

Containers Resource Types

Binary vs. decimal data measurements

BINARY SYSTEM		
NAME	FACTOR	VALUE IN BYTES
kibibyte (KiB)	2^{10}	1,024
mebibyte (MiB)	2^{20}	1,048,576
gibibyte (GiB)	2^{30}	1,073,741,824
tebibyte (TiB)	2^{40}	1,099,511,627,776
pebibyte (PiB)	2^{50}	1,125,899,906,842,624
exbibyte (EiB)	2^{60}	1,152,921,504,606,846,976
zebibyte (ZiB)	2^{70}	1,180,591,620,717,411,303,424
yobibyte (YiB)	2^{80}	1,208,925,819,614,629,174,706,176

DECIMAL SYSTEM		
NAME	FACTOR	VALUE IN BYTES
kilobyte (KB)	10^{3}	1,000
megabyte (MB)	10^{6}	1,000,000
gigabyte (GB)	10^{9}	1,000,000,000
terabyte (TB)	10^{12}	1,000,000,000,000
petabyte (PB)	10^{15}	1,000,000,000,000,000
exabyte (EB)	10^{18}	1,000,000,000,000,000,000
zettabyte (ZB)	10^{21}	1,000,000,000,000,000,000,000
yottabyte (YB)	10^{24}	1,000,000,000,000,000,000,000,000

Containers Resource Types

What happens if a container exceeded the configured limits?

- **CPU**
  Kubernetes will enter “overcommit” state and will just “throttle” (limit) the container’s usage.
- **Memory or ephemeral storage**
  Kubernetes will “evict” (kill) the container’s pod and recreate it.

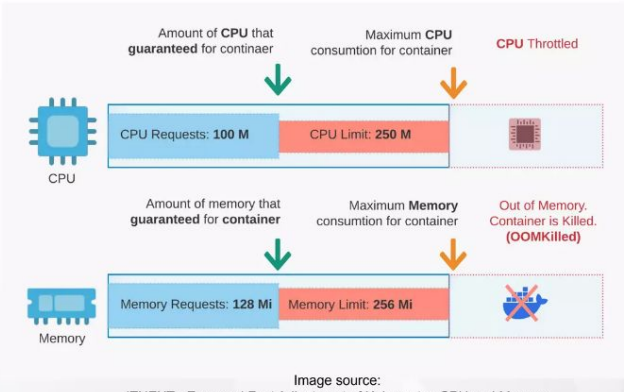


Image source:
ITNEXT - Easy and Fast Adjustment of Kubernetes CPU and Memory

What if you specify a container's limit, but not its request?

kubectl apply -f pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: default-mem-demo-2
spec:
  containers:
  - name: default-mem-demo-2-ctr
    image: nginx
    resources:
      limits:
        memory: "1Gi"
```

requests=limits

kubectl describe pod default-mem-demo-2

```
resources:
  limits:
    memory: 1Gi
  requests:
    memory: 1Gi
```


What if you specify a container's request, but not its limit?

kubectl apply -f pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: default-mem-demo-3
spec:
 containers:
 - name: default-mem-demo-3-ctr
   image: nginx
   resources:
     requests:
       memory: "128Mi"
```

no limits

kubectl describe pod default-mem-demo-3

```
resources:
  requests:
    memory: 128Mi
```

LimitRange Object

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-resource-constraint
spec:
  limits:
  - default:   # this section defines default limits
      cpu: 500m
    defaultRequest: # this section defines default requests
      cpu: 500m
    max: # max and min define the limit range
      cpu: "1"
    min:
      cpu: 100m
    type: Container
```

A **LimitRange** object can:

- Set default request/limit for compute resources in a namespace and automatically inject them to Containers at runtime.
- Enforce minimum and maximum compute resources usage per Pod or Container in a namespace.

ResourceQuota Object

- A resource quota, defined by a **ResourceQuota** object, 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 by resources in that namespace.

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
 name: compute-resources
spec:
 hard:
   requests.cpu: "1"
   requests.memory: 1Gi
   limits.cpu: "2"
   limits.memory: 2Gi
```

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
 name: object-counts
spec:
 hard:
   configmaps: "10"
   persistentvolumeclaims: "4"
   pods: "4"
   secrets: "10"
   services: "10"
   services.loadbalancers: "2"
```

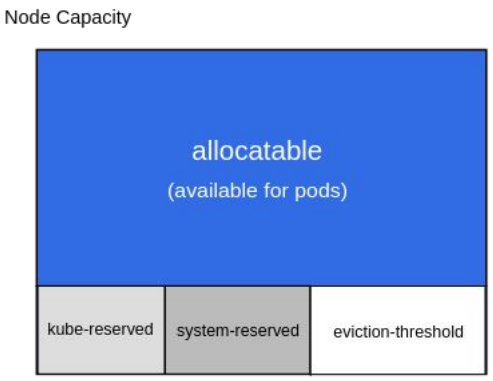
Capacity and Allocatable Resources

The maximum resources available for any container is the maximum resources on a single Kubernetes node.

However, not all Kubernetes node resources is available for the pods.

Part of the node resources are saved for Kubernetes agent essential components, operating system, and eviction threshold.

- **Capacity**: total node resources.
- **Allocatable**: resources available for Pods.



THANKS!

Any questions?