June 02 2025

6. Resource Requirements and Limits

* Each and every node will have "CPU" & "Memory"
* And the pod will also require "Certain amount of CPU" & "memory" to place this pod on a node.
* If the node doesnot have enough memory or CPU resources then the scheduler won't schedule the pods on those nodes & instead places the pod on the one whose sufficient resources are available.
* If there is no sufficient resources available on any of the nodes, the scheduler holds back scheduling the pods & you'll see pod in 'Pending state".

```
kubectl describe pod <pod-name>
```

```
NAME    READY   STATUS    RESTARTS   AGE
Nginx   0/1     Pending   0          7m
```

```
Events:
Reason                  Message
------                  -------
Failed Scheduling       No nodes are available that match all of the
                        following predicates:: Insufficient CPU
```

Resource Request:-

We can specify the CPU & memory required for a pod when creating one.

For eg: It could be one CPU & one gigabyte of memory

This is known as "resource request"

⇒ So the minimum amount of CPU & memory requested by the containers have to be specified in the "pod-definition file", when the scheduler tries to place the pod on the node, it uses these numbers to identify a node which has sufficient amount of resources available

pod-definition.yaml

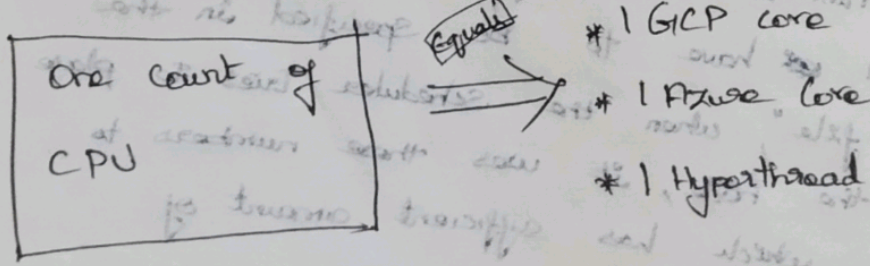
```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080
    resources:
      requests:
        memory: "4Gi"
        CPU: 2
```

when the pod gets placed on the node, the pod gets a guaranteed amount of resources available for it.

Resource - CPU

⇒ 0.1 CPU can also expressed as 100m ..[m = milli]

⇒ you can go as low as "1m" but not lower than that



Resource - Memory

You could specify 256 mebibyte using the Mi suffix

(or) Specify the same value in memory like this 268435456

(or) Specify the same value in memory like this 268M

256Mi = 268435456 (or) 268M

⇒ Use Suffix G for gigabyte

1G (Gigabyte) = 1,000,000,000 bytes

1M (Megabyte) = 1,000,000 bytes

1K (Kilobyte) = 1,000 bytes

1 Gi (Gibibyte) = 1,073,741,824 bytes

1 Mi (Mebibyte) = 1,048,576 bytes

1 Ki (Kibibyte) = 1,024 bytes

Limits

If a container is running on a node by default, a container has no limit to the resources it can consume on a node, so say a container that is part of a pod starts with one CPU on a node, it can go up & consume as much resources as it requires that suffocates the native processes on the node or other containers of resources.

However, we can set the limit for the resource usage on these pods

For eg:

If we set a limit to one VCPU to the Container, a Container will be limited to consume only one VCPU

from that node.

The same goes with memory,

For eg:- you can set a limit of 512 mebibyte on containers

* we can specify the limits under limits section.

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080

    resources:
      requests:
        memory: "1Gi"
        CPU: 1
      limits:
        memory: "2Gi"
        CPU: 2
```

⇒ When the pod is created, K8s sets "new limits" for the container

(*). The limits & requests are set for each container within a pod, so if there are multiple containers then each container can have a requests or limit set for its own

Exceeds limit

What happens when a pod tries to exceed resources beyond its specified limit?

Incase of CPU:-

⇒ The system throttles the CPU, so that it doesnot go beyond the specified limit.

⇒ A Container cannot use more CPU resources than its limit.

Incase of memory:-

⇒ This is not the case with memory,

⇒ A container can use more memory resources than its limit. So if a pod tries to consume more memory than its limit constantly. Then, the pod will be terminated & you'll see that the pod terminated with an OOM [Out of Memory] error in the logs or in the output of the described command when you run it.

Default Behaviour:

→ By default, K8S does not have a "CPU" or "memory request" or "limit set", so, this means that any pod can consume as much resources as required on any node & suffocate other pods or processes that are running on the node of resources. this is very important to note.

Behaviour - CPU

Lets see how "CPU requests" & "Limits" works

⇒ Lets say there are 2 pods requesting for "CPU resources" on the cluster

NO REQUESTS & NO LIMITS

⇒ Without a resource (or) limit set one pod can consume all the cpu resources on the node & prevent the 2nd pod of required resources so, this is not ideal.

NO REQUESTS but limits specified

requests = limits

In this case, K8S automatically sets requests to the same as limits

Eg:- "Requests" and "Limits" are assumed to be 3, in this case & each pod is guaranteed 3 VCPUs & no more than that as limits are also set to the same

REQUESTS AND LIMITS are set

⇒ In this case, whose 'requests' & 'limits' are set,

⇒ In this case, each pod gets a guaranteed # of CPU requests, which is one vCPU & can go upto the limits that are defined which is one vCPU but not more.

⇒ This might look to be the most ideal scenario. However the issue is that if pod1 needs more CPU cycles for some reason & pod2 isn't really consuming that many CPU cycles, then we don't want to limit pod1 of CPU

⇒ we would like to allow pod1 to use the available CPU cycles as long as pod2 doesn't really need it.

⇒ If there're sufficient CPU cycles available on the system then why not let the pods use them So, we dont want to unnecessarily limit resources of CPU cycles. So, this is not really ideal scenario

(*) Setting REQUESTS but NO LIMITS

In this case, because "request" is set each pod is guaranteed "one vCPU". However, because limits're not set when available, any pod can consume as many CPU cycles as available. But at any point in time. If "pod2" requires additional CPU cycles or whatever it has requested, then it will be guaranteed its requested CPU cycle.

So this is the most ideal setup

Behaviour - Memory

If 2 pods competing for "memory resources" on the Cluster

NO REQUESTS and NO LIMITS

⇒ Without resource (or) limit set

* One pod can consume all memory resource on the node & prevent the 2nd pod of required resource so this is not ideal

NO REQUESTS specified but LIMITS

⇒ we have no requests specified, but we do have limits specified,

⇒ In this case, K8S automatically sets requests to the same as limits

⇒ For eg:- Requests & limits are assumed to be 3 gibibytes in this case, & each pod is guaranteed 3 gibibytes & no more as limits is also the same

Requests = limits

REQUESTS & LIMITS are set

Here, "Requests" & "limits" are set

In this case, each pod gets a guaranteed amount of memory which is one gibibyte & can go up to the limits defined which is 3 gibibytes, but not more

setting "REQUESTS" but "no limits"

⇒ setting requests but no limits

⇒ Because requests are set, each pod is guaranteed one gibibyte, However because limits are not set when available any pod can consume as much memory as available and if pod 2 requests more memory to free up pod one, the only option available is to kill it, unlike CPU, we cannot throttle memory. Once memory is assigned to a pod, the only way to kind of relieve it is to kill the pod & free up all the memory that's used by it.

12:20 By default, we know that kgs does not have "resource requests" or "limits" configured for pods. But then how do we ensure that every pod created has some default set, This is possible with "limit ranges"

Limit Ranges :-

⇒ Limit Ranges can help you define default values to be set for containers in pods that're created without a "request" or "limit" specified in "pod definition file"

⇒ Thus is applicable at the "name space" level

⇒ "Limit Ranges" is "object", So you create a "definition file"

Explanation of the fields

Field	Meaning
default	If no limit is specified in the pod, this will be applied
defaultRequest	If no request is specified, this is used.
max	Prevents users from setting limits higher than this
min	Prevents users from setting limits lower than this

type: Container Applies to individual containers inside a pod

limit-range-cpu.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-resource-constraint
spec:
  limits:
  - default:
      cpu: 500m
    defaultRequest:
      cpu: 500m
    max:
      cpu: "1"
    min:
      cpu: 100m
    type: Container
```

- default: } If no limit is specified then this is the limit that will be applied.
- defaultRequest: } This is the defaultRequest that will be applied, if no request is specified.
- max: cpu: "1" } we can request maximum of this value.
- min: cpu: 100m } we can request minimum of this value & not below than this

⇒ This is the same goes for "Memory" use "memory" instead of "CPU"

NOTE:-
These limits are enforced when a pod is created so, if you create or change a limit range it doesnot affect existing pods.

It'll only affect newer pods that're created after the limit range is created or updated

Is there any way to restrict the total amount of resources that can be consumed by applications deployed in K8s cluster

Eg:- If we've to say that all pods together that shouldn't consume more than this much of

CPU or memory, we can create "Quota" at "namespace" level

=> Resource Quota is "namespace object" that can be created to set hard limits for "requests" & "limits"

resource-quota.yaml

```
apiVersion: v1
Kind: ResourceQuota
metadata:
  name: my-resource-quota
spec:
  hard:
    requests.cpu: 4
    requests.memory: 4Gi
    limits.cpu: 10
    limits.memory: 10Gi
```

This resource quota limits the total requested CPU & memory to 4 gibibyte & it defines a maximum limit of CPU consumed by all the pods together to be 10 & maximum limit of memory consumed by all the pods together to be 10 Gibibyte

A quick note on editing pods and Deployments

Edit a POD.

Remember, you CANNOT edit specifications of an existing POD other than the below

* spec.containers[*].image
* spec.initContainers[*].image
* spec.activeDeadlineSeconds
* spec.tolerations

For eg:-

You cannot edit the

* Environment variables
* service accounts
* resource limits (all of which we'll discuss later) of a running pod.

But if you really have to, then we have 2 options.

1) Run the `kubectl edit pod <podname>`

⇒ This command will open the pod specification in an editor (vi editor). Then edit the required property. When you try to save it, you will be denied.

⇒ This is because we're attempting to edit a field on the pod that is not editable.

```
# * spec: Forbidden: pod updates may not change fields other
than 'spec.containers[*].image',
#
#
#
apiVersion: v1
"/tmp/kubectl-edit-ccvrq.yaml" 125L, 4010C
```

A copy of the file with your changes is saved in a temporary location

You can delete the existing pod by running the command.

```
Kubectl delete pod webapp
```

Then create a new pod with your changes using the temporary file.

```
Kubectl Create -f /tmp/kubectl-edit-ccrvq.yaml
```

2) The second option is to extract the pod definition with YAML format to a file using the command.

```
kubectl get pod webapp -o yaml > my-new-pod.yaml
```

Then make the changes to the exported file using an editor (vi editor)

```
vi my-new-pod.yaml
```

Then delete the existing pod

```
Kubectl delete pod webapp
```

Then create a new pod with the edited file

```
Kubectl Create -F my-new-pod.yaml
```

Edit Deployments

With Deployments you can easily edit any field / property of the POD template.

Since the pod template is a child of the deployment specification, with every change the deployment will automatically delete & create a new pod with new changes

So if you are asked to edit a property of a POD part of a deployment you may do that simply by running the command.

```
kubectl edit deployment my-deployment
```