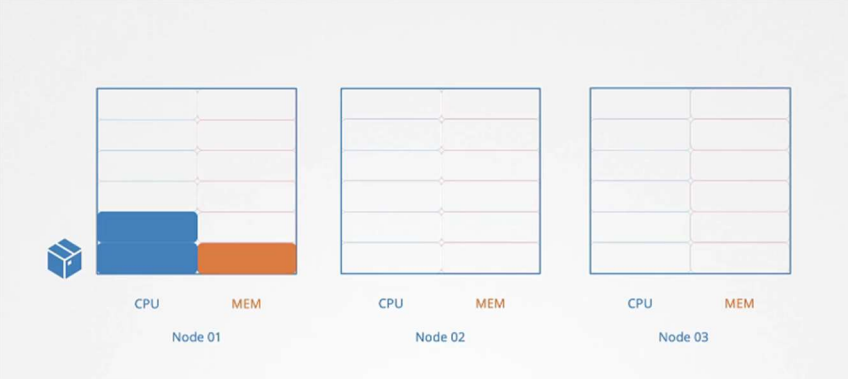
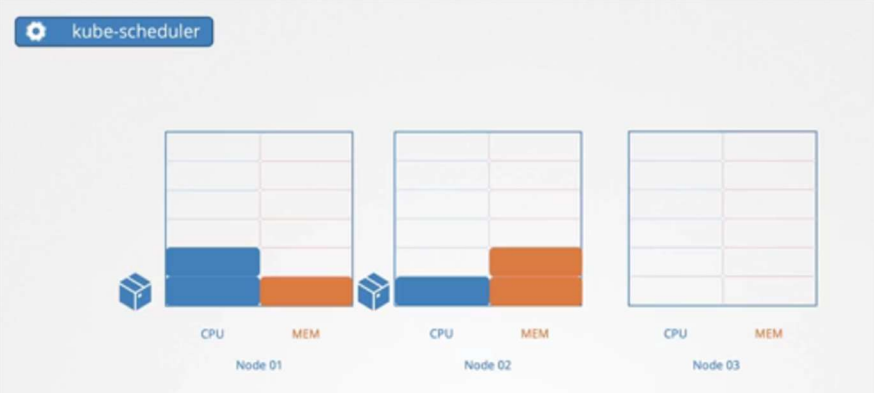
Resource Requirements and Limits

Let's look at resource requirements. Let us start by looking at a three node Kubernetes cluster. Each node has a set of CPU and memory resources available. Now every pod requires a set of resources to run. In this case, for example, this pod requires two CPUs and one memory unit. Now whenever a pod is placed on a node, it consumes the resources available on that node



Now as we have discussed before, it is the Kubernetes scheduler that decides which node a pod goes to. The scheduler takes into consideration the amount of resources required by a pod and those available on the nodes and identifies the best node to place a pod on. In this case, the scheduler schedules a new pod on node two because there are sufficient resources available on that node

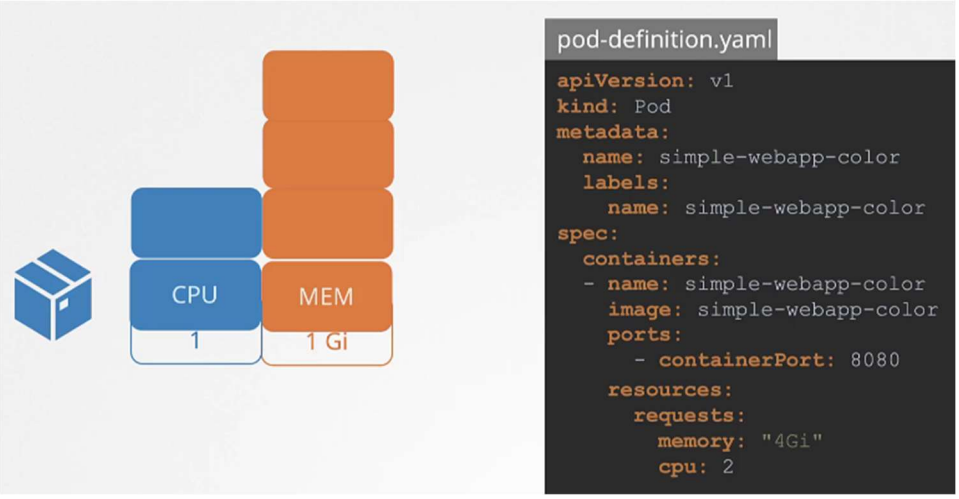


If nodes have no sufficient resources available, the scheduler avoids placing the pod on those nodes and instead places the pod on one where sufficient resources are available. And if there is no sufficient resources available on any of the nodes, then the scheduler holds back scheduling the pod. You will see the pod in a pending state. If you look at the events using the kubectl describe pod command, you will see there is an insufficient CPU.

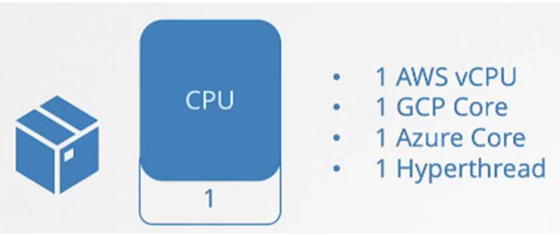
```
NAME        READY     STATUS    RESTARTS   AGE
Nginx       0/1       Pending   0          7m

Events:
  Reason                 Message
  ------                 -------
  FailedScheduling  No nodes are available that match all of the following predicates:: Insufficient cpu (3).
```

Now, let's focus on the resource requirements for each pod. You can specify the amount of CPU and memory required for a pod when creating one. For example, it could be one CPU and one gibibyte of memory, and this is known as the resource request for a container. It represents the minimum amount of CPU or memory requested by the container. When the scheduler tries to place the pod on a node, it uses these numbers to identify a node with a sufficient amount of resources available.

To specify resource requirements in the sample pod-definition file, add a section called "resources." Under this section, add "requests" and specify the values for memory and CPU usage. For example, you can set it to four gibibytes of memory and two counts of CPU. When the scheduler receives a request to place this pod, it looks for a node that has the specified amount of resources available. Once the pod is placed on a node, it gets a guaranteed amount of resources available for its usage



So what does one count of CPU really mean? Now you can specify any value as low as 0.1. So 0.1 CPU can also be expressed as 100m, where m stands for milli. And you can go as low as 1m, but not lower than that. Now one count of CPU is equivalent to one vCPU. So that's one vCPU in AWS. So if you're looking at the AWS cloud or it could be referred to as one core in GCP or Azure or one hyperthread on other systems. And you could request a higher number of CPUs for the container provided your nodes are sufficiently funded.

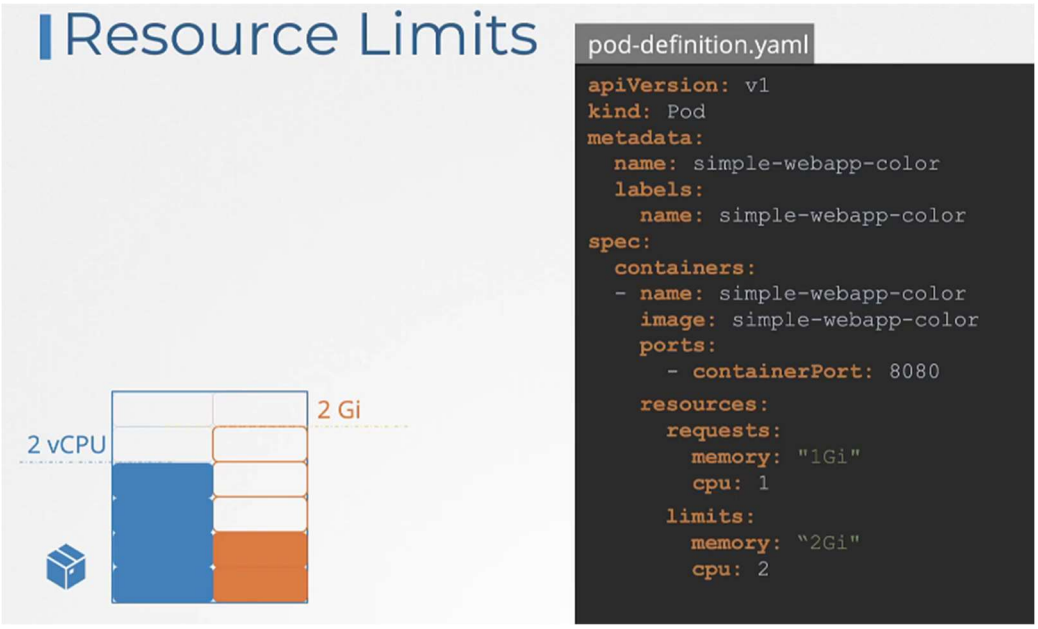


Now similarly with memory, you could specify 256 mebibyte using the Mi suffix **256Mi** or specify the same value in memory like **268435456**. Does the full number, the whole number, and/or specify the same value in memory like **268M,** or use the suffix G for gigabyte. So note the difference between G and Gi. So G is gigabyte and it refers to 1,000 megabytes, whereas Gi refers to gibibyte and that would be equal to 1,024 mebibytes. So the same applies to megabyte and mebibyte, and kilobyte and kibibyte.

Now let's look at a container running on a node. And by default, a container has no limit to the resources it can consume on a node. So say a container that's part of a pod starts with one CPU on a node, it can go up and consume as much resources as it requires and that suffocates the native processes on the node or other containers of resources.

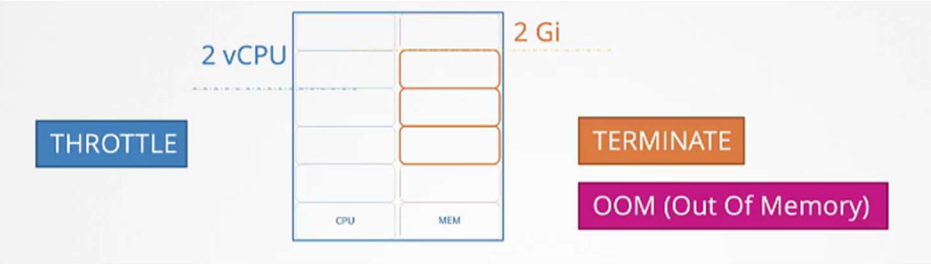
However, you can set a limit for the resource usage on these pods. For example, if you set a limit of one vCPU to the containers, a container will be limited to consume only one vCPU from that node. So the same goes with memory. For example, you can set a limit of 512 mebibyte on containers like this.

Now you can specify the limits under the limits section, under the resources section in your pod-definition file. So here, specify the new limits for memory and CPU like this. Now when the pod is created, Kubernetes sets new limits for the container. And remember that the limits and requests are set for each container within a pod. So if there are multiple containers, then each container can have a request or limit set for its own.



So what happens when a pod tries to exceed resources beyond its specified limit? In case of the CPU, the system throttles the CPU so that it does not go beyond the specified limit. A container cannot use more CPU resources than its limit.

However, 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, the pod will be terminated and you'll see that the pod terminated with an OOM error in the logs or in the output of the described command when you run it. So that's what is OOM refers to out of memory kill.



So now that we have learned what resource requests are and what limits are and how they function and what happens when a particular container or pod hits the limits that were defined, let's see what the default configuration is, right?

So by default, Kubernetes 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 and suffocate other pods or processes that are running on the node of resources. So this is very, very important to note.

So let's just look at how CPU requests and limits work.

Let's say there are two pods competing for CPU resources on the cluster. And when I say pod, I mean a container within a pod, right? So just keep that in mind.

So without a resource or limit set, one pod can consume all the CPU resources on the node and prevent the second part of required resources. So of course this is not ideal.

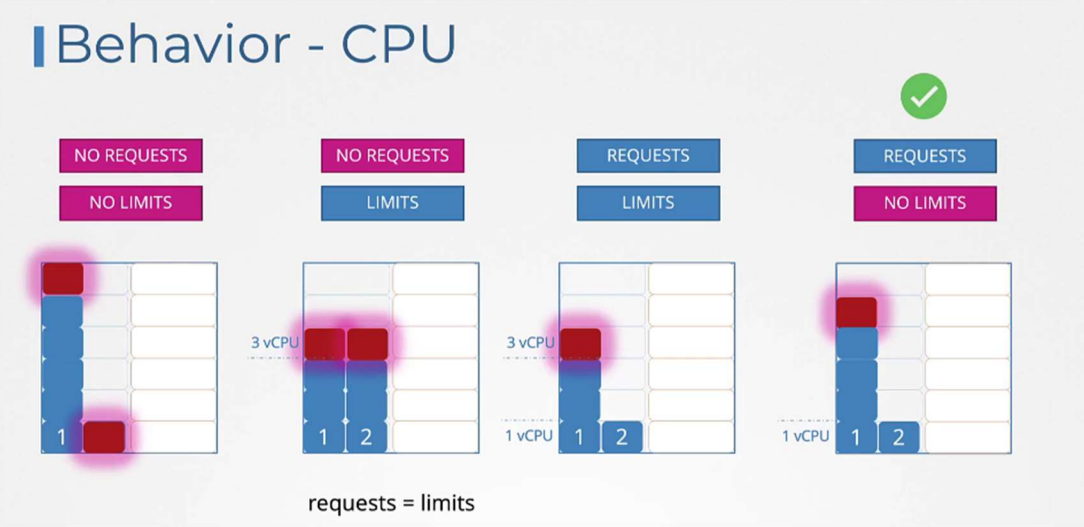
Now let's look at another case where we have no request specified, but we do have limits specified. In this case, Kubernetes automatically sets requests to the same as limits. For example, request and limits are assumed to be three, in this case, and each pod is guaranteed three vCPUs and no more than that as limits are also set to the same.

The next one is where requests and limits are set. In this case, each pod gets a guaranteed number of CPU requests, which is one vCPU and can go up to the limits that are defined which is three vCPU, but not more.

So this might look to be the most ideal scenario. However, the issue is that if pod one needs more CPU cycles for some reason and pod two isn't really consuming that many CPU cycles, then we don't want to limit pod one of CPU, right? So we'd like to allow pod one to use the available CPU cycles as long as pod two doesn't really need it. So if there are sufficient CPU cycles available on the system, then why not let the pods use them, right? So we don't want to unnecessarily limit resources of CPU cycles.

So that is not really the ideal scenario and that's where the last scenario comes in. So setting requests but no limits. In this case, because requests are set, each pod is guaranteed one vCPU. However, because limits are not set when available, any pod can consume as many CPU cycles as available. But at any point in time, if pod two 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. Of course, there are cases where you absolutely may want to limit a pod of resources and in that case you may set limits. For example, a good use case for setting limits is our labs themselves where all the labs that you guys have been going through and accessing as part of this course, they are hosted as containers on a cluster, right? And since it's made accessible to the public and users can run any kind of workload that they want, we set limits to prevent a user from misusing the infrastructure to, let's say, perform Bitcoin mining or other resource-consuming activities.



So that works for us in that case. But in your case, if you don't want to restrict your application to consume additional CPU if needed, then you could consider not setting limits. But remember, if you were to do that, you need to make sure that all the pods have some requests set because that's the only way a pod will have resources guaranteed when there are no limits set for other pods, right?

So if there is any pod that has no request set and there are no limits set for all the other pods, then it's possible that any pod could consume all of the memory, all the CPU that's available on the node and starve the pod that has no request defined. So just make sure that you have set requests for all the nodes.

So a couple of things to note, the requests then limits may be different for each pod, but for the sake of simplicity, we are assuming that it's the same for both pods in these examples that I'm sharing here, right? But you can have absolutely different requests or limits set for each containers within each pod.

So also note that these recommendations are just for CPU.

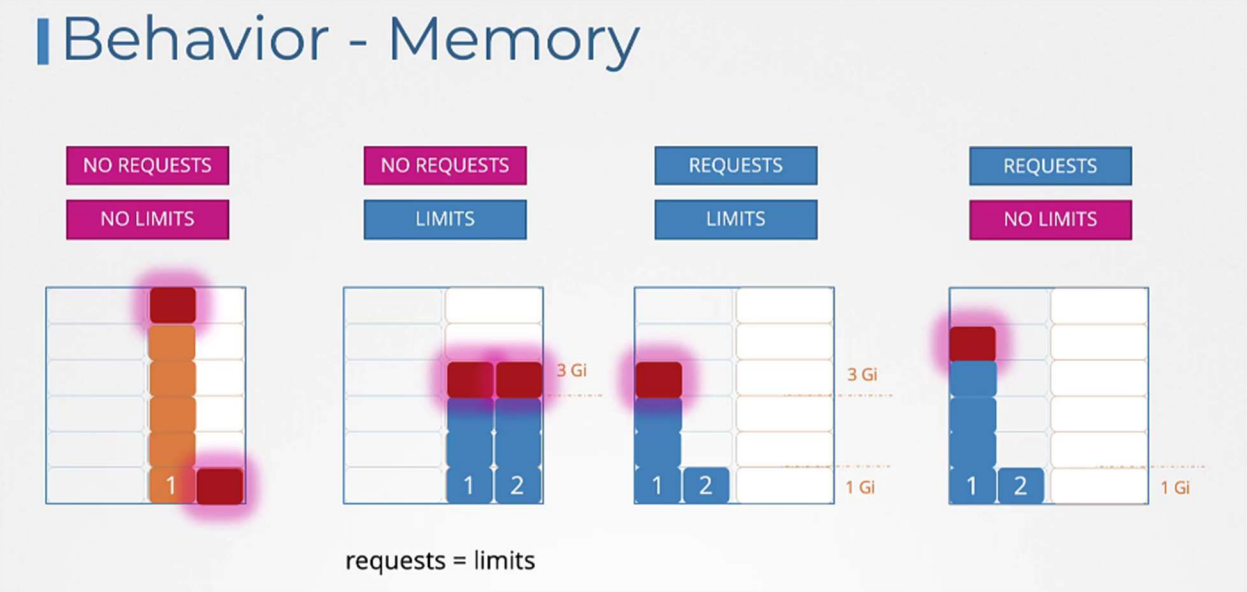
So let's look at how it works for memory next.

So it's kind of similar. So if you look at the memory, let's say there are two, in the first case, there are two pods competing for memory resources on the cluster. And without a resource or limit set, one pod can consume all the memory resources on the node and prevent the second part of required resources. So this is not ideal.

Now let's look at the case where we have no requests specified, but we do have limits specified. And in this case, container is automatically sets request to the same as limits. So for example, our request and limits are assumed to be three gibibytes in this case. And each pod is guaranteed three gibibytes and no more as limits is also the same.

The next one is where requests and limits are set. In this case, each pod gets a guaranteed amount of memory which is one gibibyte and can go up to the limits defined which is three gibibytes, but not more.

And the last one is setting requests but no limits. In this case, 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 two requests more memory to free up pod one, the only option available is to kill it.



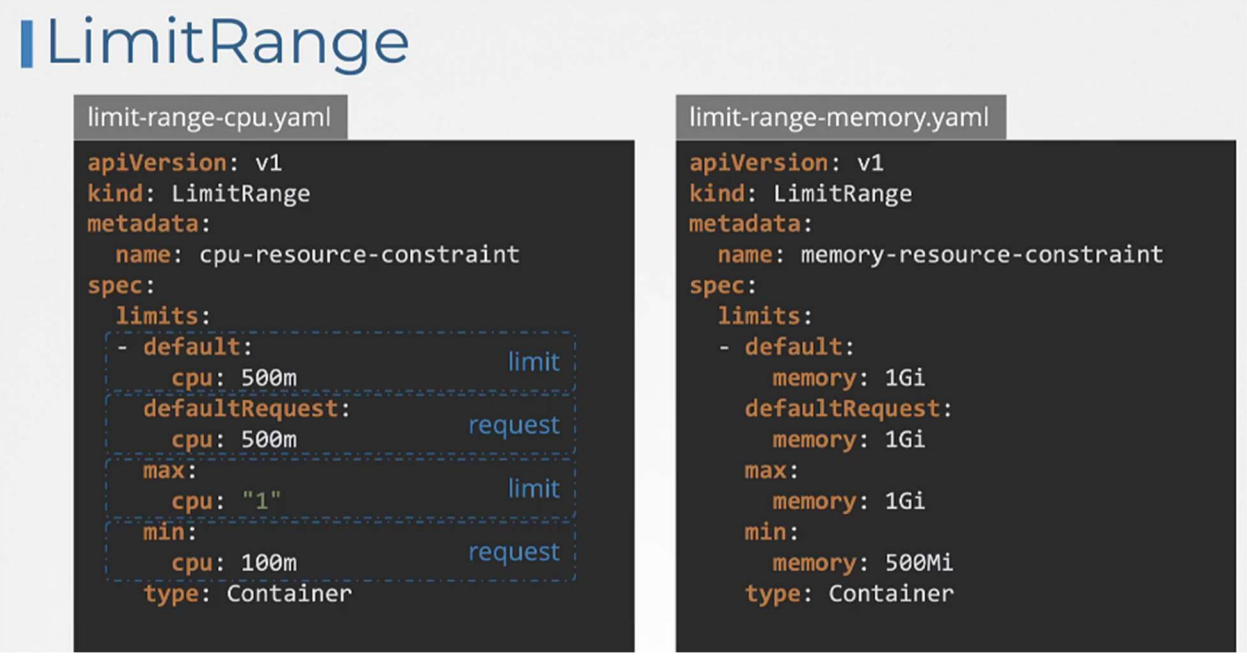
You know, because unlike CPU, we cannot throttle memory. Once memory is assigned to a pod, the only way to kind of retrieve it is to kill the pod and free up all the memory that are used by it.

Okay, so now as we discussed before, by default, Kubernetes does not have resource requests or limits configured for pods. But then how do we ensure that every pod created has some default set? Now this is possible with limit ranges.

So limit ranges can help you define default values to be set for containers in pods that are created without a request or limit specified in the pod-definition files. This is applicable at the namespace level. So remember that. And this is an object.

So you create a definition file with the apiVersion set to v1, kind set to LimitRange, and we'll give it a name cpu-resource-constraint. We then set the default limit to 500m, defaultRequests to the same as well. We will also specify a max cpu as one and a minimum as 100m. So the max refers to the maximum limit that can be set on a container in a pod and minimum refers to the minimum request a container in a pod can make. So these are, of course, some example values, not a recommendation or anything. So you must set whatever is best for your applications.

So the same goes for memory. Use memory instead of CPU and specify the defaults and max and main values in this form. Note that these limits are enforced when a pod is created. So if you create or change a limit range, it does not affect existing pods. It'll only affect newer pods that are created after the limit range is created or updated.



And finally, is there any way to restrict the total amount of resources that can be consumed by applications deployed in a Kubernetes cluster? For example, if we had to say that all the pods together shouldn't consume more than this much of CPU or memory, what we could do is create quotas at a namespace level.

So a resource quota is a namespace level object that can be created to set hard limits for requests and limits. In this example, this resource quota limits the total requested CPU in the current namespace to four and memory to four gibibytes. And it defines a maximum limit of CPU consumed by all the pods together to be 10 and memory to be 10 gibibytes as well, right? So that's another option that can be explored.