## Selecting the correct memory and CPU requests for your Pods

****The cluster autoscaler makes scaling decisions based on the presence of pending pods.****

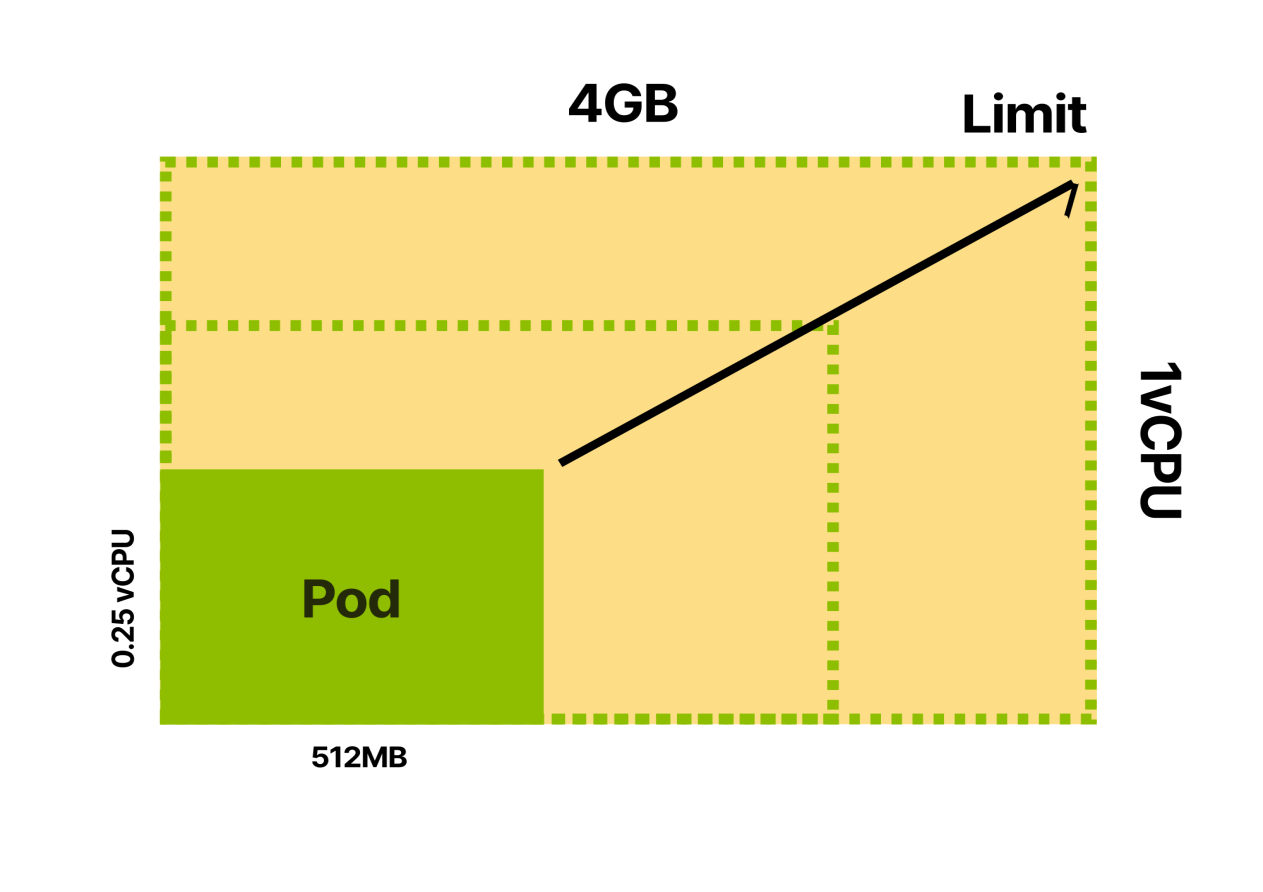
The Kubernetes scheduler assigns (or not) a Pod to a Node based on its memory and CPU requests.

Hence, it's essential to set the correct requests on your workloads, or you might be triggering your autoscaler too late (or too early).

*Let's have a look at an example.*

You decide to profile an application, and you found out that:

* Under average load, the application consumes 512MB of memory and 0.25 vCPU.
* At peak, the application should consume up to 4GB of memory and 1 vCPU.



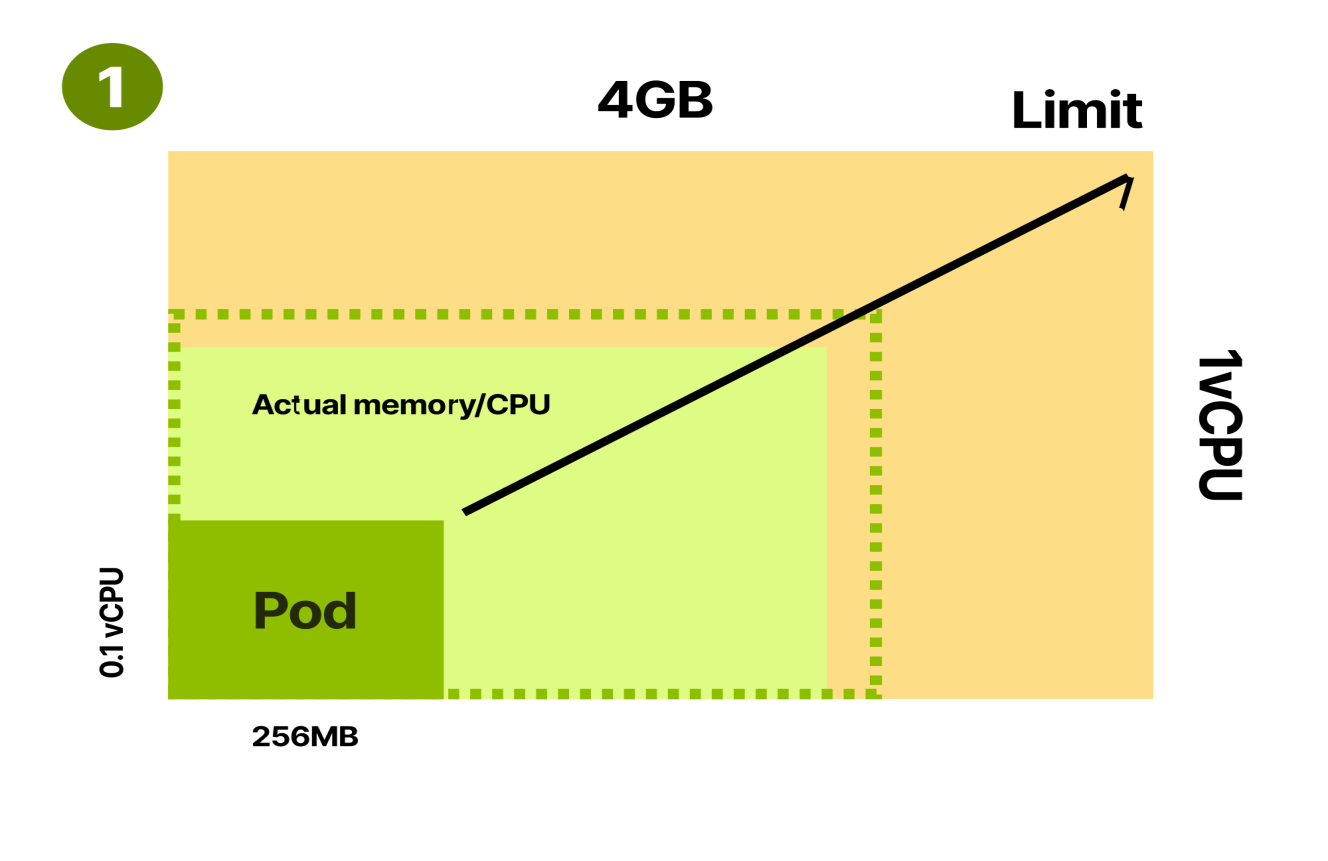
The limit for your container is 4GB of memory and 1 vCPU.

*However, what about the requests?*

The scheduler uses the Pod's memory and CPU requests to select the best node before creating the Pod.

So you could:

1. ****Set requests lower than the actual average usage.****
2. Be conservative and ****assign requests closer to the limit.****
3. ****Set requests to match the actual limits.****



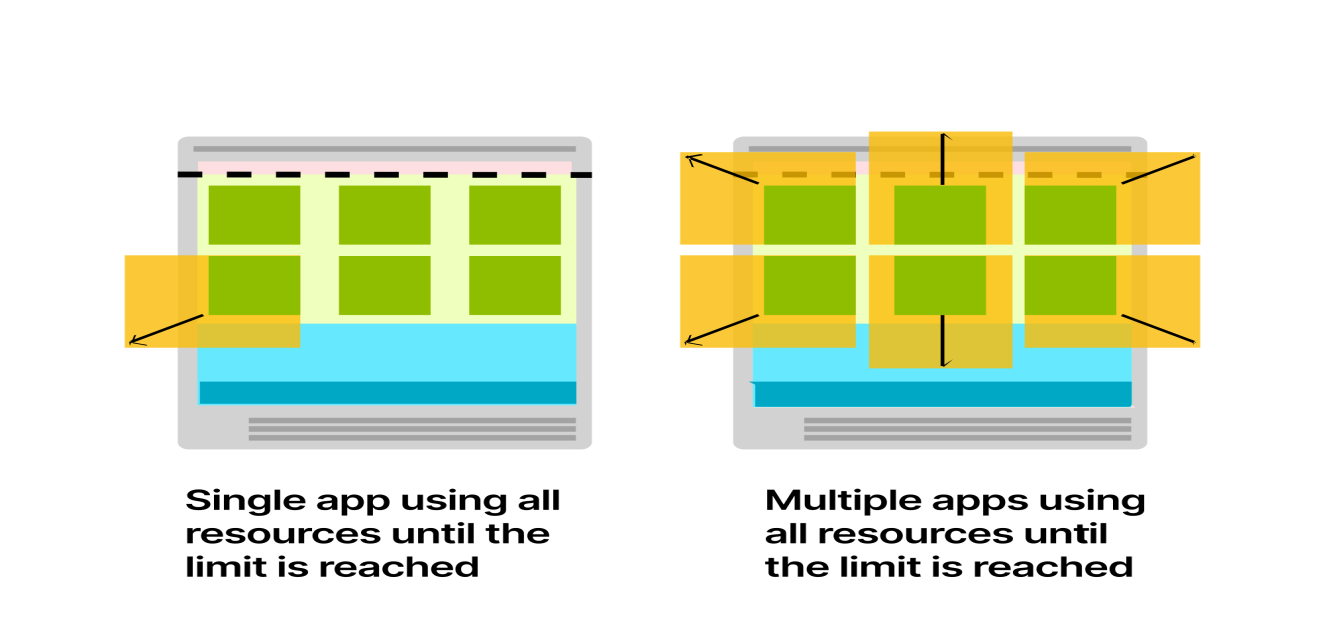
You could assign requests that are ****lower**** than the average app consumption.

****Defining requests lower than the actual usage is problematic since your nodes will be often overcommitted.****

As an example, you can assign 256MB of memory as a memory request.

The Kubernetes scheduler can fit twice as many Pods for each node.

However, Pods use twice as much memory in practice and start competing for resources (CPU) and being evicted (not enough memory on the Node).



****Overcommitting nodes can lead to excessive evictions, more work for the kubelet and a lot of rescheduling.****

*What happens if you set the request to the same value of the limit?*

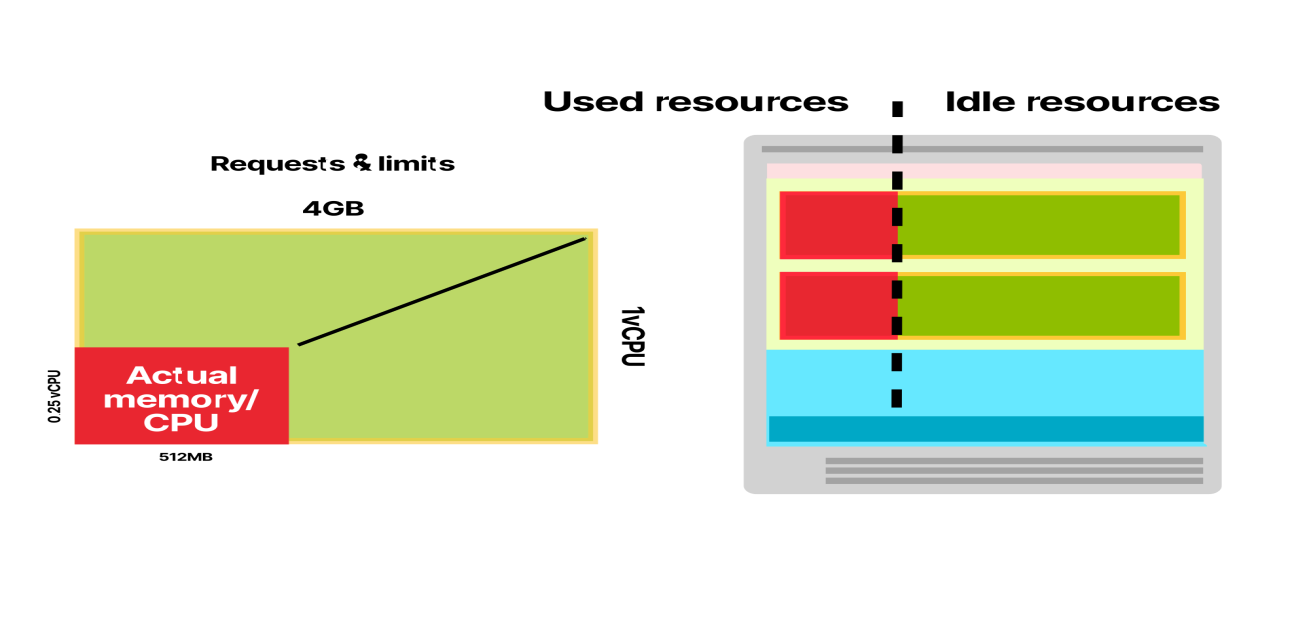
You can set request and limits to the same values.

In Kubernetes, this is often referred to as [Guaranteed Quality of Service class](https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/" \l "qos-classes" \t "https://learnk8s.io/_blank) and refers to the fact that it's improbable that the pod will be terminated and evicted.

The Kubernetes scheduler will reserve the entire CPU and memory for the Pod on the assigned node.

****Pods with Guaranteed Quality of Service are stable but also inefficient.****

If your app uses 512MB of memory on average, but you reserve 4GB for it, you have 3.5GB unused most of the time.



*Is it worth it?*

If you want extra stability, yes.

If you want efficiency, you might want to lower the requests and leave a gap between those and the limit.

This is often referred to as ****Burstable Quality of Service class**** and refers to the fact that the Pod baseline consumption can occasionally burst into using more memory and CPU.

When your requests match the app's actual usage, the scheduler will pack your pods in your nodes efficiently.

****Occasionally, the app might require more memory or CPU.****

1. If there are resources in the Node, the app will use them before returning to the baseline consumption.
2. If the node is low on resources, the pod will compete for resources (CPU), and the kubelet might try to evict the Pod (memory).

*Should you use Guaranteed or Burstable quality of Service?*

*It depends.*

1. ****Use Guaranteed Quality of Service (requests equal to limits) when you want to minimise rescheduling and evictions for the Pod.**** An excellent example is a Pod for a database.
2. ****Use Burstable Quality of Service (requests to match actual average usage) when you want to optimise your cluster and use the resources wisely.**** If you have a web application or a REST API, you might want to use a Burstable Quality of Service.

*How do you select the correct requests and limits values?*

****You should profile the application and measure memory and CPU consumption when idle, under load and at peak.****

A more straightforward strategy consists of deploying the Vertical Pod Autoscaler and wait for it to suggest the correct values.

The Vertical Pod Autoscaler collects the data from the Pod and applies a regression model to extrapolate requests and limits.

## What about downscaling a cluster?

****Every 10 seconds, the Cluster Autoscaler decides to remove a node only when the request utilization falls below 50%.****

In other words, for all the pods on the same node, it sums the CPU and memory requests.

If they are lower than half of the node's capacity, the Cluster Autoscaler will consider the current node for downscaling.

It's worth noting that the Cluster Autoscaler does not consider actual CPU and memory usage or limits and instead only looks at resource requests.

Before the node is removed, the Cluster Autoscaler executes:

* [Pods checks](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md" \l "what-types-of-pods-can-prevent-ca-from-removing-a-node" \t "https://learnk8s.io/_blank) to make sure that the Pods can be moved to other nodes.
* [Nodes checks](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md" \l "i-have-a-couple-of-nodes-with-low-utilization-but-they-are-not-scaled-down-why" \t "https://learnk8s.io/_blank) to prevent nodes from being destroyed prematurely.

If the checks pass, the Cluster Autoscaler will remove the node from the cluster.

## Why not autoscaling based on memory or CPU?

****CPU or memory-based cluster autoscalers don't care about pods when scaling up and down.****

Imagine having a cluster with a single node and setting up the autoscaler to add a new node with the CPU reaches 80% of the total capacity.

You decide to create a Deployment with 3 replicas.

The combined resource usage for the three pods reaches 85% of the CPU.

A new node is provisioned.

*What if you don't need any more pods?*

You have a full node idling — not great.

****Usage of these type of autoscalers with Kubernetes is discouraged.****

## Summary

Defining and implementing a successful scaling strategy in Kubernetes requires you to master several subjects:

* Allocatable resources in Kubernetes nodes.
* Fine-tuning refresh intervals for Metrics Server, Horizontal Pod Autoscaler and Cluster Autoscalers.
* Architecting cluster and node instance sizes.
* Container image caching.
* Application benchmarking and profiling.

But with the proper monitoring tool, you can iteratively test your scaling strategy and tune the speed and costs of your cluster.