When working with containers in Kubernetes, it’s important to know what are the resources involved and how they are needed. Some processes will require more CPU or memory than others. Some are critical and should never be starved.

Knowing that, we should configure our containers and Pods properly in order to get the best of both.

In this article, we will see:

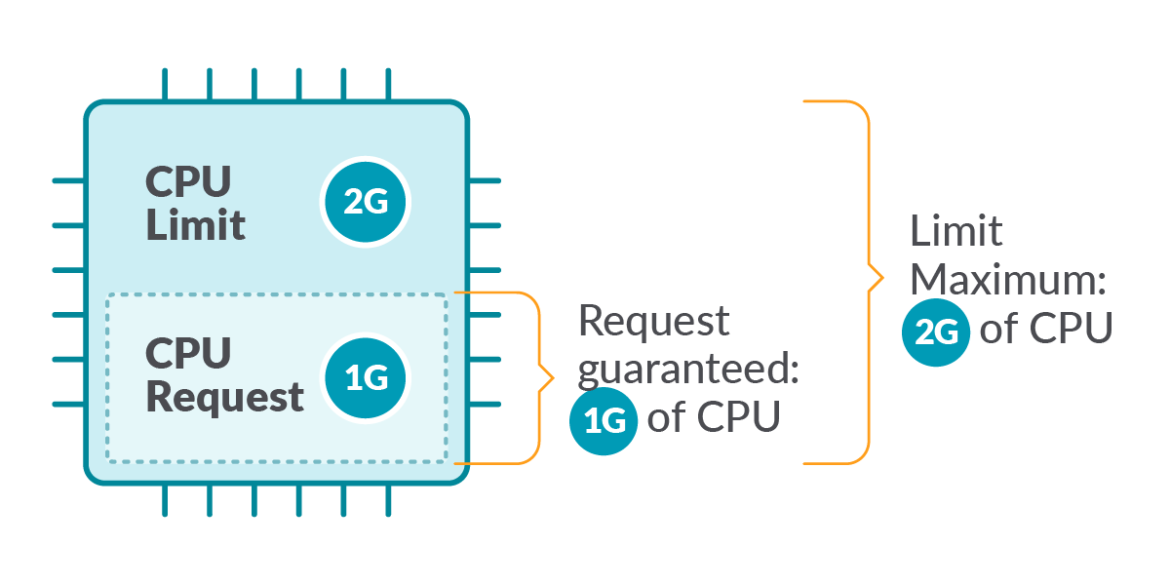
* [Introduction to Kubernetes Limits and Requests](https://sysdig.com/blog/kubernetes-limits-requests/#introduction)
* [Hands-on example](https://sysdig.com/blog/kubernetes-limits-requests/#handson)
* [Kubernetes Requests](https://sysdig.com/blog/kubernetes-limits-requests/#kubernetesrequests)
* [Kubernetes Limits](https://sysdig.com/blog/kubernetes-limits-requests/#kuberneteslimits)
* [CPU particularities](https://sysdig.com/blog/kubernetes-limits-requests/#cpuparticularities)
* [Memory particularities](https://sysdig.com/blog/kubernetes-limits-requests/#memoryparticularities)
* [Namespace ResourceQuota](https://sysdig.com/blog/kubernetes-limits-requests/#namespaceresourcequota)
* [Namespace LimitRange](https://sysdig.com/blog/kubernetes-limits-requests/#namespacelimitrange)
* [Conclusion](https://sysdig.com/blog/kubernetes-limits-requests/#conclusion)

**Introduction to Kubernetes Limits and Requests**

Limits and Requests are important settings when working with Kubernetes. This article will focus on the two most important ones: CPU and memory.

**Kubernetes defines Limits as the** **maximum amount of a resource** to be used by a container. This means that the container can never consume more than the memory amount or CPU amount indicated.

**Requests, on the other hand, are the minimum guaranteed amount of a resource** that is reserved for a container.



**Hands-on example**

Let’s have a look at this deployment, where we are setting up limits and requests for two different containers on both CPU and memory.

kind: Deployment

apiVersion: extensions/v1beta1

…

template:

  spec:

    containers:

      - name: redis

        image: redis:5.0.3-alpine

        resources:

limits:

            memory: 600Mi

            cpu: 1

requests:

            memory: 300Mi

            cpu: 500m

      - name: busybox

        image: busybox:1.28

        resources:

limits:

            memory: 200Mi

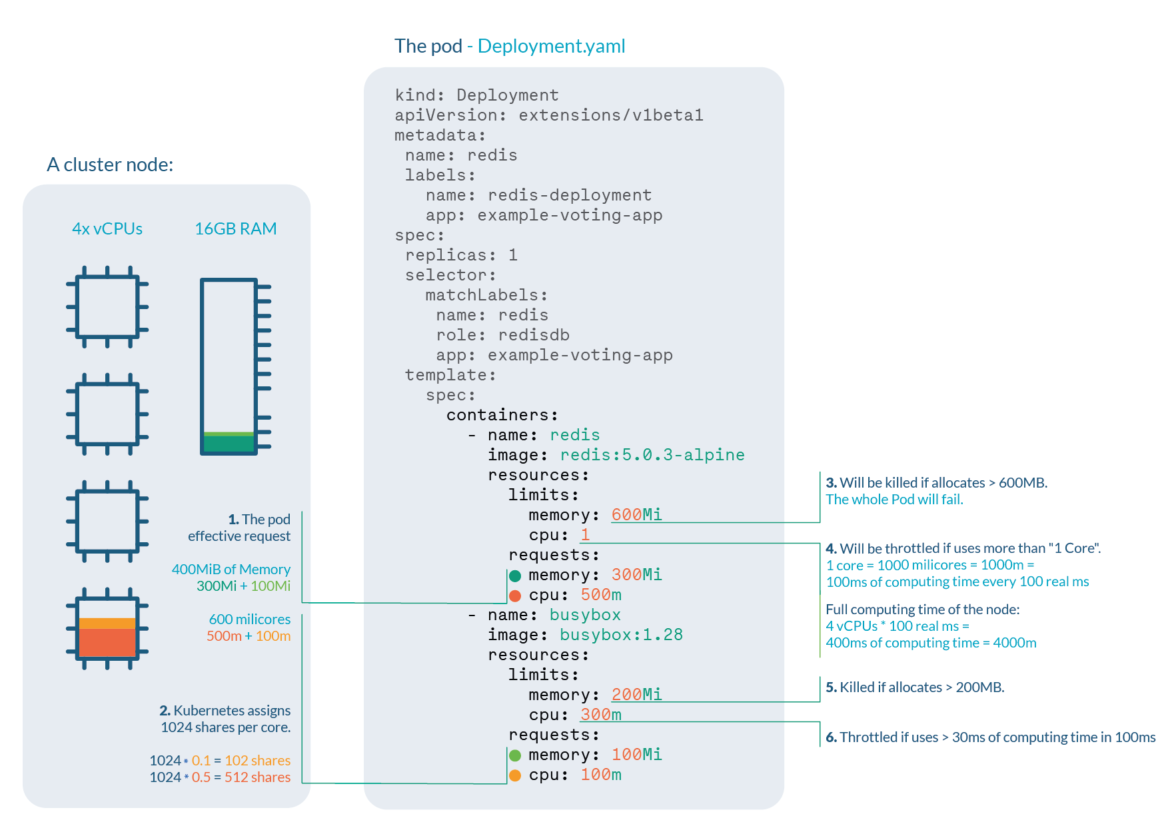
            cpu: 300m

requests:

            memory: 100Mi

            cpu: 100mCode language: JavaScript (javascript)

Let’s say we are running a cluster with, for example, 4 cores and 16GB RAM nodes. We can extract a lot of information:

[](https://sysdig.com/wp-content/uploads/Kubernetes-Limits-and-Request-05.png)

1. **Pod effective request** is 400 MiB of memory and 600 millicores of CPU. You need a node with enough free allocatable space to schedule the pod.
2. **CPU shares** for the redis container will be 512, and 102 for the busybox container. Kubernetes always assign 1024 shares to every core, so redis: 1024 \* 0.5 cores ≅ 512 and busybox: 1024 \* 0.1cores ≅ 102
3. Redis container will be **OOM killed** if it tries to allocate more than 600MB of RAM, most likely making the pod fail.
4. Redis will suffer **CPU throttle** if it tries to use more than 100ms of CPU in every 100ms, (since we have 4 cores, available time would be 400ms every 100ms) causing performance degradation.
5. Busybox container will be **OOM killed** if it tries to allocate more than 200MB of RAM, resulting in a failed pod.
6. Busybox will suffer **CPU throttle** if it tries to use more than 30ms of CPU every 100ms, causing performance degradation.

**Kubernetes Requests**

Kubernetes defines requests as a **guaranteed minimum amount of a resource** to be used by a container.

Basically, it will set the minimum amount of the resource for the container to consume.

When a Pod is scheduled, kube-scheduler will check the Kubernetes requests in order to allocate it to a particular Node that can satisfy at least that amount for all containers in the Pod. If the requested amount is higher than the available resource, the Pod will not be scheduled and remain in Pending status.

For more information about Pending status, check [Understanding Kubernetes Pod pending problems](https://sysdig.com/blog/kubernetes-pod-pending-problems/).

In this example, in the container definition we set a request for 100m cores of CPU and 4Mi of memory:

resources:

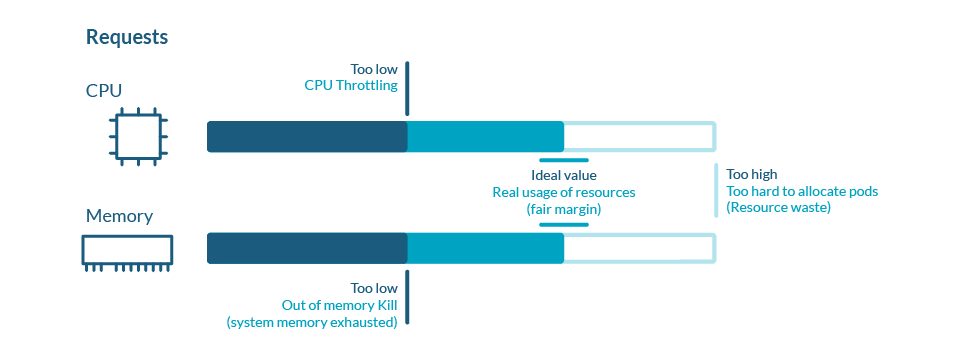
requests:

        cpu: 0.1

        memory: 4MiCode language: JavaScript (javascript)

Requests are used:

* When allocating Pods to a Node, so the indicated requests by the containers in the Pod are satisfied.
* At runtime, the indicated amount of requests will be guaranteed as a minimum for the containers in that Pod.



**Kubernetes Limits**

Kubernetes defines **limits** as a **maximum amount of a resource** to be used by a container.

This means that the container can never consume more than the memory amount or CPU amount indicated.

resources:

limits:

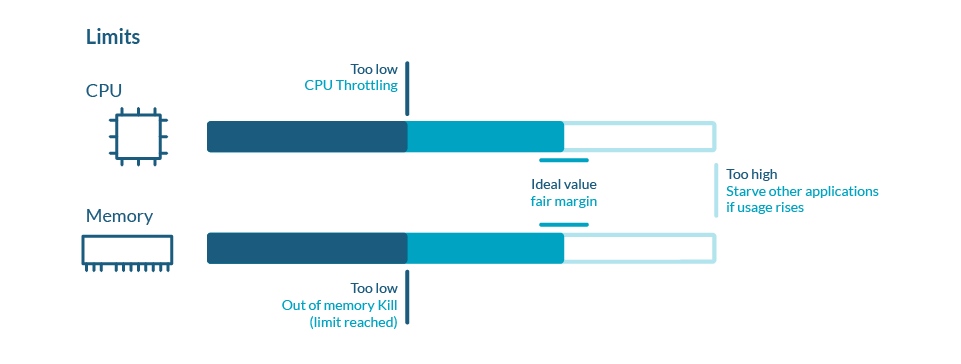
cpu: 0.5

memory: 100Mi

Code language: JavaScript (javascript)

Limits are used:

* When allocating Pods to a Node. If no requests are set, by default, Kubernetes will assign requests = limits.
* At runtime, Kubernetes will check that the containers in the Pod are not consuming a higher amount of resources than indicated in the limit.

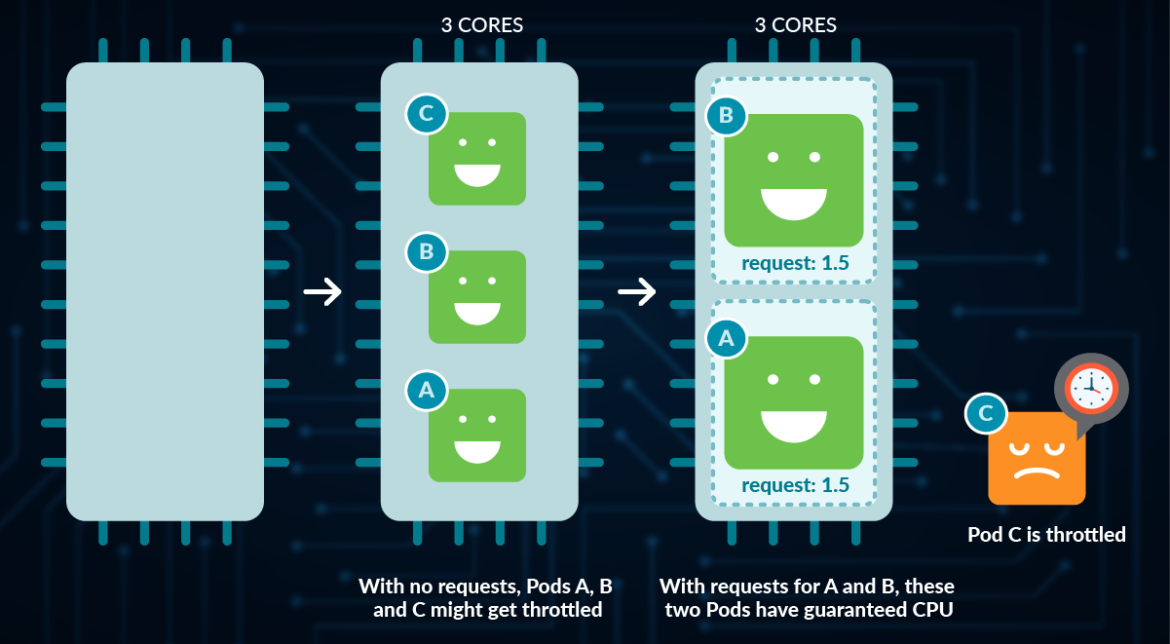


**CPU particularities**

CPU is a **compressible resource**, meaning that it can be stretched in order to satisfy all the demand. In case that the processes request too much CPU, some of them will be throttled.

**CPU** represents **computing processing time**, measured in cores.

* You can use millicores (m) to represent smaller amounts than a core (e.g., 500m would be half a core)
* The minimum amount is 1m
* A Node might have more than one core available, so requesting CPU > 1 is possible





**Webinar: Prometheus Got Out of Hand, Discover What Bloomreach Did Next!** [**REGISTER NOW**](https://sysdig.com/resources/webinars/prometheus-got-out-of-hand-discover-what-bloomreach-did-next/)

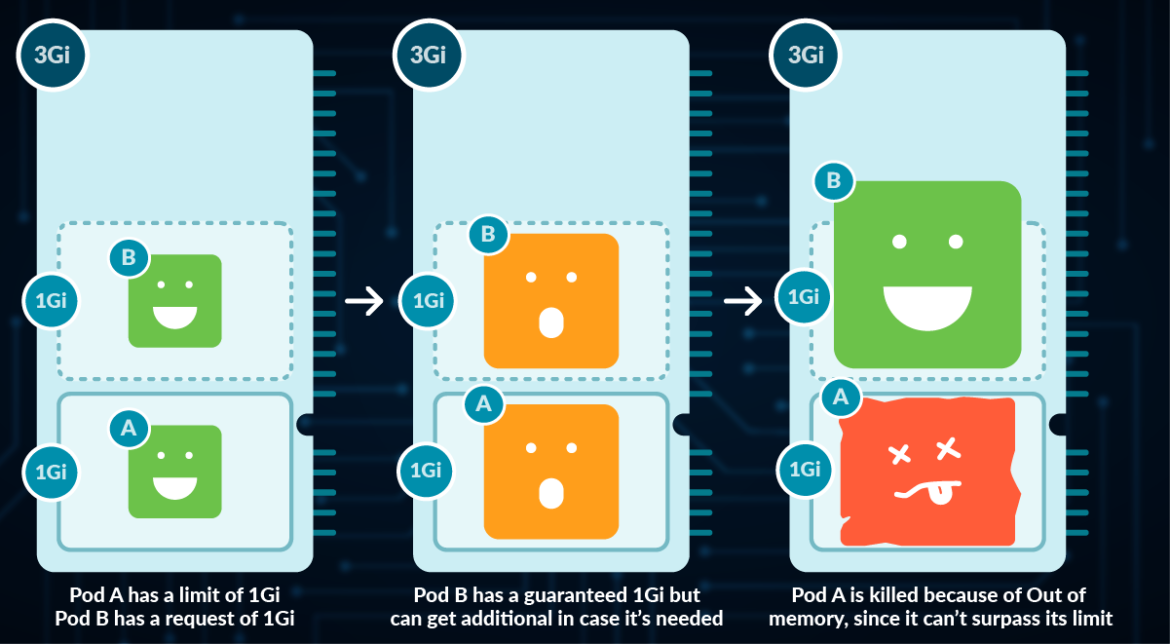
**Memory particularities**

**Memory** is a **non-compressible** resource, meaning that it can’t be stretched in the same manner as CPU. If a process doesn’t get enough memory to work, the process is killed.

Memory is measured in Kubernetes in **bytes**.

* You can use, E, P, T, G, M, k to represent Exabyte, Petabyte, Terabyte, Gigabyte, Megabyte and kilobyte, although only the last four are commonly used. (e.g., 500M, 4G)
* Warning: don’t use lowercase m for memory (this represents Millibytes, which is ridiculously low)
* You can define Mebibytes using Mi, as well as the rest as Ei, Pi, Ti (e.g., 500Mi)

*A Mebibyte (and their analogues Kibibyte, Gibibyte,…) is 2 to the power of 20 bytes. It was created to avoid the confusion with the Kilo, Mega definitions of the metric system. You should be using this notation, as it’s the canonical definition for bytes, while Kilo and Mega are multiples of 1000*



**Best practices**

In very few cases should you be using limits to control your resources usage in Kubernetes. This is because if you want to avoid starvation (ensure that every important process gets its share), you should be using requests in the first place.

By setting up limits, you are only preventing a process from retrieving additional resources in exceptional cases, causing an OOM kill in the event of memory, and Throttling in the event of CPU (process will need to wait until the CPU can be used again).

For more information, check the [article about OOM and Throttling](https://sysdig.com/blog/troubleshoot-kubernetes-oom/).

If you’re setting a request value equal to the limit in all containers of a Pod, that Pod will get the Guaranteed Quality of Service.

Note as well, that Pods that have a resource usage higher than the requests are more likely to be evicted, so setting up very low requests cause more harm than good.For more information, check the article about [Pod eviction and Quality of Service](https://sysdig.com/blog/kubernetes-pod-evicted/).