# Kubernetes Resources Request and Limits



Document Level Classification

200

> When requests and limits should be set to different versus the same parameters?

Kubernetes requests and limits impact an application’s performance, stability, and cost. When tasked with determining the right values for these parameters it's hard. Each parameter is optional and is specified at the container level.

Requests are what the container is guaranteed to get and used by kube-scheduler for scheduling and resource allocation decisions.

Limits are what the kube-scheduler uses to handle resource contention and is the resource threshold a container never exceeds. It deals with two types of resources: compressible and incompressible. When a pod starts to hit CPU limits Kubernetes will throttle it without terminating or evicting

it. Now when the pod starts using more memory than its limit it will get killed by OOM.

- CPU is considered a "**compressible**" resource
- Memory is considered an "**incompressible**" resource

How the requests and limits parameters are used, defines the QoS class assigned to a pod during the scheduling process.

**Guaranteed** is applied if:

- Every container in the pod must have a memory limit and a memory request.
- For every container in the pod, the memory limit must equal the memory request.
- Every container in the pod must have a CPU limit and a CPU request.
- For every container in the pod, the CPU limit must equal the CPU request.

**Burstable** is applied if:

- At least one Container in the Pod has a memory or CPU request. Limit is greater than the request.

**BestEffort** is applied if:

- No limits or requests are defined for memory or cpu

When node resources are under pressure and the kubelet starts to reclaim node resources, it will start to evict end-user pods. The QoS applied to the pod during the scheduling stage influences the eviction process.

The eviction order is determined by the following parameters:

- Does the pod's resource usage exceed requests?
- The pod's priority
- The pod's resource usage relative to requests

Which results in, the kubelet ranking and evicting pods in the following order:

- **BestEffort** or **Burstable** pods where the usage exceeds requests. These pods are evicted based on their Priority and then by how much their usage level exceeds the request.
- **Guaranteed** pods and **Burstable** pods where the usage is less than requests are evicted last, based on their Priority.
- **Guaranteed** pods are *guaranteed only when requests and limits are specified for all the containers and they are equal*. These pods will never be evicted because of another pod's resource consumption.

By default: since *memory is incompressible*, set **limit = request** to ensure Kubernetes doesn't attempt to overcommit available memory, reducing the possibility of an OOM killer being triggered.

Overall, setting CPU and memory requests and limits to the ***same*** for "critical/sensitive" workloads is "recommended" since the QoS of guaranteed is applied. This ensures resources requested by the application's containers will be available to it when it gets scheduled and the containers will be the "last" to be evicted when the kubelet is attempting to reclaim node resources.

Example that results in guaranteed

- spec.containers[].resources.requests.cpu = 4
- spec.containers[].resources.requests.memory = 16Gi
- spec.containers[].resources.limits.cpu = 4
- spec.containers[].resources.limits.memory = 16Gi

Setting the requests and limits to ***different*** parameters will allow the application to burst between the selected values or if no limits were applied, the application could use all non reserved could use all non reserved resources are available on the node. This is only recommended for CPU since it is compressible. Memory requests and limits should be the same to prevent overcommit of memory resources.

Example that results in burstable

- spec.containers[].resources.requests.cpu = 500m
- spec.containers[].resources.requests.memory = 1Gi
- spec.containers[].resources.limits.cpu = 2
- spec.containers[].resources.limits.memory = 1Gi

Not setting requests or limits will result in the application being killed first when resources are being reclaimed.

Things to be aware of:

- There are memory and CPU reservations for the kublet, kube-proxy, operating system, etc which mean the "user" workloads won't have access to all system resources.
- If the pod has multiple containers and you do not specify any requests or limits for all the containers. This applies to injected containers, such as vault-injector and any others that may get injected at runtime. The QoS will be Burstable.
- Vertical Pod Autoscaler can inform/suggest requests and limits based on runtime metrics. (Recommended)
- Limits greater than requests = overcommit
    - Fine for CPU due to Completely Fair Scheduler (CFS)
    - Fine for Memory **if** demand is less than node capacity
- Ensure you set the limit high enough so the pod isn't OOMKilled if it starts to exceed the requested limit. Using a tool like the vertical pod autoscaler in suggest mode can help define the recommended limit to set based on runtime over a period of time.
- By default the kubelet enforces CPU limits using Completely Fair Scheduler (CFS). However, when using the Kubernetes CPU manager static policy, you can launch a container pinned to a particular set of CPU cores by specifying a whole integer value for the CPU limits and requests.

## Resources

- https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/
- https://kubernetes.io/docs/concepts/scheduling-eviction/pod-overhead/
- https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/
- https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/
- https://kubernetes.io/docs/concepts/scheduling-eviction/scheduler-perf-tuning/#how-the-scheduler-iterates-over-nodes
- https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/
- https://kubernetes.io/docs/concepts/scheduling-eviction/node-pressure-eviction/
- https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits
- https://sysdig.com/blog/kubernetes-limits-reqhttps://blog.turbonomic.com/kubernetes-cpu-throttling-the-silent-killer-of-response-time-and-what-to-do-about-ituests/
- https://k8s.af/ (search for CPU limits)
- https://blog.kubecost.com/blog/requests-and-limits/

---