





Kubernetes Memory and CPU Resources (request / limit)

Introduction [🔗](#)


This document provides guidance around specifying resource requests and limits for a Kubernetes application.

As we gradually move towards containerized applications, it is important that we all understand the importance of following common best practices with regards to compute resource sharing. Likewise, it's important to understand the consequences of running misconfigured applications, both for itself and other applications running on the host.

 None the guidance presented in this document *is not* exact science. Adjusting your application configuration is an **iterative process**. Likewise, this document will be updated to better reflect the latest recommendations.

 This document is maintained by the **Cloud Platform team**.
Comments and feedback are always appreciated 😊

Rules of thumb [🔗](#)

1. Always start with **high CPU and memory margins**, observe how your application behaves over time (over several hours, days, weeks maybe), with only 3 replicas running, then adjust the settings based on metrics.
2. The more resource **reservation** aligns with **consumption**, the more efficient and elastic the platform will become.
3. If you're unsure about these settings, reach out to the Cloud Platform team in the  [Platform](#) channel.

Overview [🔗](#)

CPU [🔗](#)

Guidance [🔗](#)

1. CPU *request* must be configured.
2. CPU *request* value must reflect your application needs.
3. CPU *limit* must be removed.

Rational [🔗](#)

CPU request reflects your application's actual CPU need under nominal load, plus some margin.

CPU is a **compressible** resource: Reaching maximum CPU capacity on the host will result in severe performance degradation but the application(s) won't be terminated (unless the [QOS profile](#) allows for it).

The CPU *request* is **guaranteed CPU time** for your application.

Even if the host is under CPU pressure (i.e. all CPU cores maxed out), your application will have guaranteed CPU time based on the CPU *request*.

Providing guaranteed CPU time also means there is no longer a need to enforce any CPU limit. For that reason, any container on the host is allowed to (sporadically) consume as much CPU as it needs without inflicting any performance degradation on the other applications.

Finally, metrics have shown that CPU throttling peaks during the first few seconds of the container startup due to CPU limit.



Due to how the [CFS scheduler](#) allocates CPU time, CPU throttling might still appear even if the CPU limit is significantly higher than actual usage.

🔧 Upper and lower bounds [↗](#)

Lower: 5m

Upper: 500m

Half a vCPU worth of time should be enough to crunch most of the work for a single replica. Since the number of cores is limited, and your applications won't have any limit, try to keep the CPU request reasonably low to allow for more containers to be scheduled on the host.

Memory [↗](#)

✅ Guidance [↗](#)

1. Memory *request* must be set.
2. Memory *limit* must be set.
3. Memory *request* and memory *limit* must be identical.
4. Memory value should be set to 150% of the nominal memory consumption

🔧 Rational [↗](#)

Memory request/limit reflects your application's actual memory need under nominal load for a single replicas, plus a significant safety margin to account for the spikes.

Memory is a **non-compressible** resource: Reaching maximum memory capacity in the container (or on the host) will result in service degradation and possible loss of data and/or data corruption due to the immediate termination of process by the [oom-killer](#).

Unlike CPU request, which can be lowered to the point where it closely matches the average CPU usage, memory reservation must always include a **safety margin (150%)**.

❌ Unless your application is extremely stable over time, reducing the **memory margin** is **risky** and **discouraged**.

Finally, setting memory request identical to memory limit prevents **memory overcommitting** on the host, and thus guarantees that the node will *never* be under memory pressure.

Bear in mind that memory request/limit must be set based on the *maximum* memory usage over time. i.e. the value *must* account for possible spikes in memory consumption.

🔧 Upper and lower bounds [↗](#)

Lower: 16Mi

Upper: 512Mi

Ideally, it would best to scale out the replicas to avoid having to allocate more than 409MiB (i.e. 80% of 512MiB) per container. If your application needs more to operate normally, please let us know.

HPA [↗](#)

✅ Guidance [↗](#)

- Memory: 80% of memory value.
- CPU: 150-200% of CPU request.

Rational [↗](#)


In production, the number of replicas is defined with the following in mind:

- On the **legacy** platform, the *minimum* number of replicas is set to **two (2)** to ensure high-availability.
- On **FLO Cloud**, the *minimum* number of replicas is set to **two (2)** to ensure high-availability, given that the application is always deployed across two regions.
- The *maximum* number of replicas is set **ten (10)** so that the load is spread across all nodes of the agent pool.

Developers need to find the right balance between the number of replicas (and account for the overhead of each pod) and the resources that each pod consumes.

In other words, we want to *avoid* the following:

- Having **too few** replicas that each consume tons of resources.
 - If one of them goes away, the service would lose a significant percentage of the overall capacity as well as the in-flight data on that pod.
 - This situation will result in a service degradation and well as additional stress on the two (2) remaining pods until a third one is scheduled. That stress might push misconfigured pods to crash/get killed if the HPA doesn't have time to react.
 - Having too few pods means we're not spreading the load enough across nodes.
- Having **too many** replicas, which will result in a significant CPU/memory footprint on the cluster, thus decreasing the compute efficiency of cluster and leading to additional costs.

 That value is based on current applications and cluster size. It might increase in the future as we gather more performance metrics.

Given that CPU request is set based on actual CPU usage (+ small margin), the CPU threshold must be **above 100%**.

A CPU threshold of 150%-200% provides enough headroom for the application to work comfortably, while avoiding the HPA to trigger too frequently.

We recommend not to go beyond 300% of CPU request.

Given that memory *limit* should **never** be reached, the memory threshold is set so that the HPA will trigger when the average memory usage gets too close to it.

Bear in mind that:

- HPA thresholds are a percentage value based on the container's *request*.
- The HPA will trigger based on the *average of all pods*; if a single pod balloons for no reason, the HPA won't do anything about it.
- The HPA doesn't trigger immediately; these thresholds are defined to give enough time for the HPA to trigger without degrading the application performance.

Beyond these recommendations, application owners should think about what criteria/conditions would make the most sense for scaling their application.

 The HPA configuration is all about striking the right balance, and it's not trivial.

Assuming the memory limit is set high enough, you may start with focusing on adjusting the **CPU threshold first**, and then configure a memory threshold later down the road.

Upper and lower bounds [↗](#)

- CPU:
 - Lower: 150%
 - Upper: 300%


- Memory:
 - Lower: 60%
 - Upper: 90%
- Number of replicas:
 - Non-production
 - Min: 1
 - Max: 3
 - Production
 - Min: 2
 - Max: 10

If you think your application might require using greater values, please reach out to the Cloud Platform team.

How to determine the “right” value ? [🔗](#)

Follow these steps to get a sense of how close your application is to the aforementioned recommendations:

1. Go to [NewRelic → Azure Kubernetes Service - Resource Efficiency](#).
2. The search for your application:
 - a. Set the namespace
 - b. Set the container name
 - c. If your application is a cronjob, select its name from the 3rd menu
 - d. set the Kubernetes cluster
3. Set the duration:
 - a. Over the **last hour**: to get a detailed data on how the application behaves and possibly identify short-lived fluctuations (e.g. spikes).
 - b. Over the **last 24 hours**: to identify how the application behaves over a day/night cycle.
 - c. Over the **last 7 days**: to identify how the application behaves globally.

 Once your application is properly calibrated, overall resource consumption should not differ from one (production) cluster to another. The differences in the amount of traffic per network should reflect on the number of replicas up and running.

In other words, a busy cluster should have a higher number of replicas running than a idle cluster.

CPU [🔗](#)

Select “CPU Efficiency” tab.

Memory [🔗](#)

Select “Memory Efficiency” tab.

Expected results [🔗](#)

CPU [🔗](#)

Tile “CPU usage (cores) [max]“ [🔗](#)

Ideally, all containers of a given deployment should follow the same CPU consumption pattern and no container should consume significantly more than the others.

If some pod(s) consumes unreasonably more resources than the others, the owner of the application should investigate the issue.

Tile “% CPU Request vs. Utilized, with usage ratio < 100%” [🔗](#)

Ideally, the overall *average* CPU usage should float around **75%-85% of the CPU request**.

The presence of a large delta between the lowest and the highest value might be caused by the number of replicas of your application running on each cluster. The number of replicas running (i.e. your deployment and/or HPA configuration) should be adjusted to better adapt to the CPU load.

For example, if your application runs 10 replicas with very low overall CPU usage (compared to the CPU request), it probably means two things:


- The number of replicas running is too high and should be reduced → adjust the HPA threshold(s).

and/or

- The CPU request is too high and should be reduced.

Tiles “CPU throttling (%)” and “CPU throttling (s)” [🔗](#)

Your application should always be at zero (no data) since CPU limit is not set.

 If you observe CPU throttling despite your application not having a CPU limit, please reach out to the Cloud Platform team for further investigation.

Memory [🔗](#)

Tile “Memory usage (Working Set)” [🔗](#)

Ideally, all containers should follow the same memory consumption pattern (as any other container of the same deployment) and no container should consume significantly more than the others.

Tiles “% Memory: Utilized vs. Request” and “% Memory: Utilized vs. Limit” [🔗](#)

Both tiles should be identical, since memory request is equal to memory limit.

Tile “% Memory: Utilized (Working Set) vs. Request, with usage ratio < 100%” [🔗](#)

Memory consumption must always remain *below* 100%.

In the absence of fine tuning, we can expect memory usage at around 50% of memory request.

However, if your application has a *flat* and *stable* memory consumption pattern, then the margin *might* be lowered (from 200% to 150% for example) and the HPA memory threshold *might* be increased a bit (from 75% to 80%-85% for example).

Deployment vs. HPA [🔗](#)

When possible, it's best to align the number of replicas in your **Deployment** with the *minimum* number of replicas in your **HPA** configuration (ideally three (3)). If they are misaligned, the HPA will adjust the number of running replicas soon after the deployment is complete.

Example [🔗](#)

```
1 ---
2 kind: Pod
3 apiVersion: v1
4 metadata:
5   name: my-app
6 spec:
7   containers:
```

```

8   - name: my-app
9     image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
10    resources:
11      requests:
12        cpu: 20m
13        memory: 128Mi
14      limits:
15        memory: 128Mi
16    ---
17    apiVersion: autoscaling/v2
18    kind: HorizontalPodAutoscaler
19    metadata:
20      annotations:
21      labels:
22        app: my-app
23        name: my-app-hpa
24    spec:
25      minReplicas: 3
26      maxReplicas: 10
27      metrics:
28        - resource:
29            name: memory
30            target:
31              averageUtilization: 80
32              type: Utilization
33            type: Resource
34        - resource:
35            name: cpu
36            target:
37              averageUtilization: 200
38              type: Utilization
39            type: Resource
40      scaleTargetRef:
41        apiVersion: apps/v1
42        kind: Deployment
43        name: my-app-deployment

```

Dapr Sidecar [🔗](#)

For the Dapr sidecar, we recommend to set the following values:













- `memory-request: "100Mi"`
- `cpu-request: "5m"`

Media content [🔗](#)

- https://addenergietech.sharepoint.com/:p:/r/sites/TeamLogiciel/Documents%20partages/General/Demos/20230512_RD_De mo.pptx?d=w00bebf9060044533b61eaab16ad14c75&csf=1&web=1&e=Sc3cOs [Connect your OneDrive account](#)
- https://addenergietech-my.sharepoint.com/:v:/g/personal/ysaiari_addenergie_com/ETD91Nb5qydEh0vT4ijZsd8BZy-B05jOk Zp6xmE6vQNT-A?e=hTa1by [Connect your OneDrive account](#)

References [🔗](#)

- [Resource management best practices for Azure Kubernetes Service \(AKS\) - Azure Kubernetes Service](#)
- [Scaling options for applications in Azure Kubernetes Service \(AKS\) - Azure Kubernetes Service](#)

-  [Troubleshoot memory saturation in AKS clusters - Azure](#)
-  [Resource Management for Pods and Containers](#)
-  [Node-pressure Eviction](#)
-  [HorizontalPodAutoscaler Walkthrough](#)
-  [For the Love of God, Stop Using CPU Limits on Kubernetes \(Updated\) | Robusta](#)
-  [What Everyone Should Know About Kubernetes Memory Limits, OOMKilled Pods, and Pizza Parties | Robusta](#)
-  [Chapter 2: Horizontal Autoscaling - Kubernetes Guides - Apptio](#)
-  [Blog - DevOps - Apptio](#)
-  [Kubernetes requests vs limits: Why adding them to your Pods and Namespaces matters | Google Cloud Blog](#)
-  [Resource utilization in HPA is request based](#)
-  [Horizontal Pod Autoscaler should use resource limit as its base for calculation · Issue #72811 · kubernetes/kubernetes](#)
-  [Tim Hockin \(thockin.yaml\) on Twitter / X](#)