sysdig

# 6 Considerations for Kubernetes Capacity Planning

**Capacity management has always been an issue for companies, but it becomes especially challenging when operating workloads in a cloud-native environment. Cloud-native deployments have to take into account, and declare, specific resources required to operate effectively. Yet, achieving the right balance of availability and reliability requires DevOps, Security, and other IT teams to identify their needs as they establish the relationship between the application architecture and the underlying cloud infrastructure.**

There are a variety of factors to consider, and different teams sometimes function with competing needs, so it can be challenging to create a comprehensive and universally-adopted capacity management strategy. Performance and cost will always be factors, and so will the myriad of conditions that regularly change as workload needs shift.

Modern organizations that have adopted a Kubernetes-based approach to application development and workload management will want to prioritize capacity planning. This guide offers a framework that will help enterprises make their environments more efficient and be prepared to support the needs of their Kubernetes clusters through effective use and planning of resources.

## Planning for the reality of Kubernetes limitations

Kubernetes delivers so many advantages, but the reality about its inherent resource constraints is that having too little resources can result in CPU activity being throttled and pods receiving an out-of-memory (OOM) error. Conversely, if pods are requesting too much, then the Kubernetes engine won't have enough space to allocate new workloads and resources will be wasted. The right balance is determined by what a team's most pressing needs are, and those may change over time.

It's important to put context to the process of planning and operating accordingly because capacity planning involves both art and science to implement and make it work. It starts with forecasting the resources required to meet the demands of the infrastructure over a period of time with a reasonable expectation of change to your IT and application development needs. Under- or over-estimating needs could create an environment that is mismatched for what you're trying to accomplish. When you underestimate, you'll likely face intermittent poor performance by Kubernetes, or will experience delays. Overestimating will be costly, as you'll be paying for capacity you never use.

Here are some basic tenets regarding Kubernetes capacity that should be considered as you structure your plans:

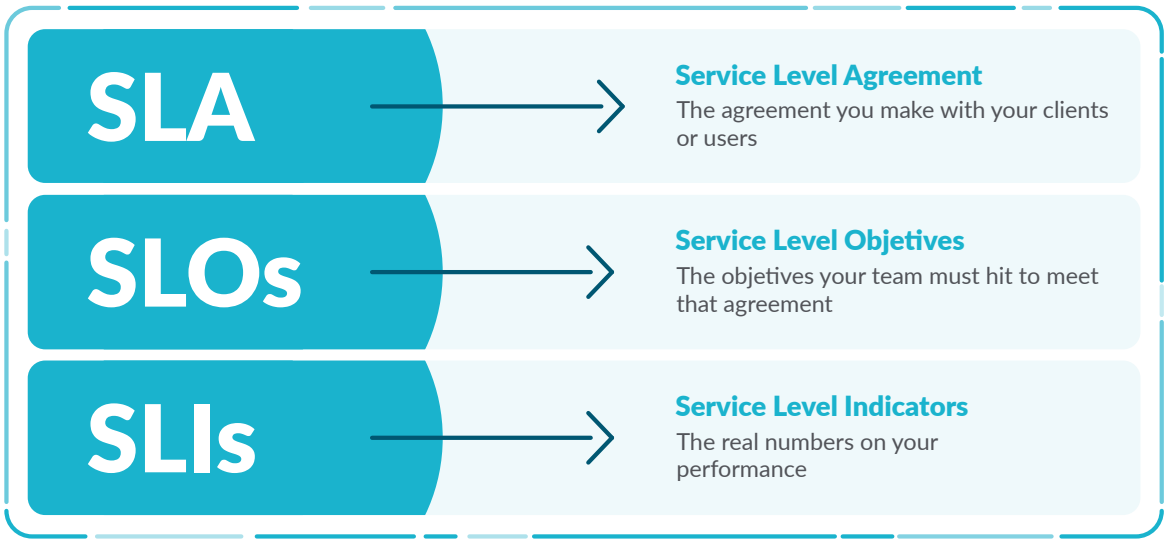
- **Demand changes based on needs and usage:** You will experience periods of low usage, and then there will be critical times when you push the pedal to the metal and need as much capacity as possible. Some of this you can predict, but variance will always be a factor.
- **Even the best laid plans can go awry:** Projects don't go according to schedule and this will impact your capacity needs when you least expect it. So, even if you're careful about anticipating when you need to scale up or down, it's possible that you'll encounter some unforeseen factor that could derail those plans. If your capacity capabilities don't jive with reality, you'll struggle to either make up the difference or consume the available capacity.
- **More isn't always better...either is less:** Every development iteration and change usually incurs some level of additional resources. In terms of delivery, that's great news because your applications will have more functionality and capabilities. But if you haven't made adequate preparations for the additional load, your delivery schedule will fall behind. Should you always plan for more? Should you err on the side of caution and plan for less? Neither is necessarily a good rule of thumb. Rather, take a good, hard look at your needs historically and consider them in future planning. They may not be perfect barometers, but they are likely to provide good guidance about your capacity needs.




# Use service-level indicators and objectives

Consider that capacity planning has traditionally been attached to foundational aspects of an infrastructure, like CPUs, memory, servers, and other traditional types of resources. While it may make sense to start with these factors, what will really help guide your planning is a focus on service-level indicators (SLIs) and service-level objectives (SLOs) and the interplay and dependencies among applications, APIs, and those traditional resources previously mentioned. This provides a more comprehensive and realistic picture of where your needs will exist. That will allow you to more accurately predict for the corresponding capacity needs.

This type of focus offers an intent-based approach that allows for more flexibility with regard to your changing needs. Operating your services and workloads according to intent rather than a static number of resources allows you to more accurately adapt as needs and requirements change. In essence, it allows you to be agile and predictive. Ultimately, a willingness to be brutally realistic in your operational approach will prepare you for inevitable change and the corresponding capacity needs.



2



# Apply resources effectively through autoscaling

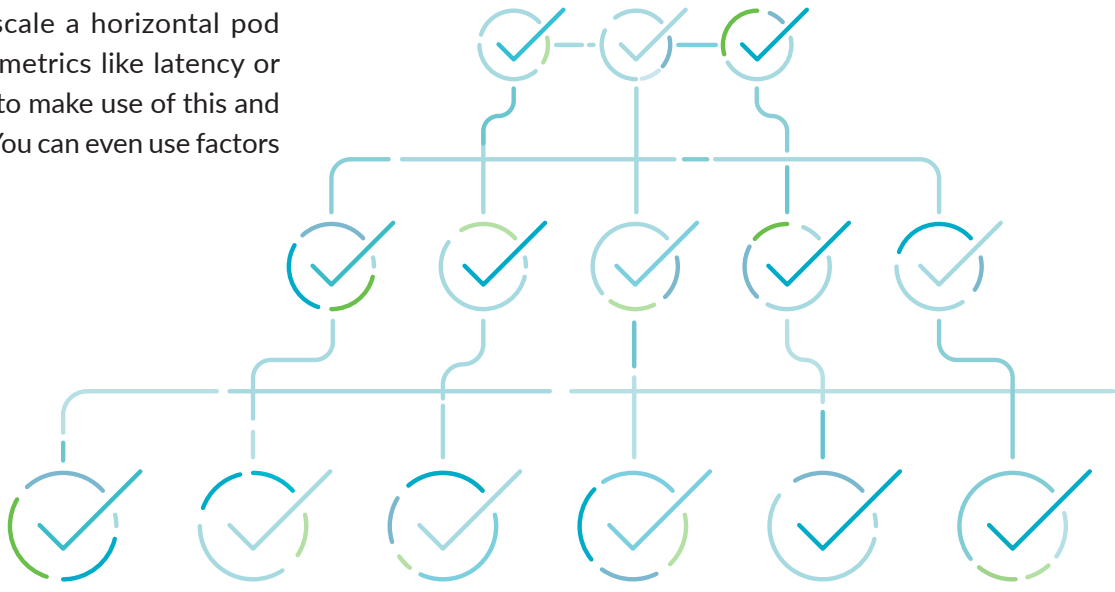
Pods are the way to measure activity and deployment level in Kubernetes, and Kubernetes environments are very attuned to how pods are scheduled - done correctly, they meet the correct needs. If not, then there will be lags and other performance issues.

Kubernetes provides sophisticated control of scheduling of pods to nodes. The end result is that the Kubenetes scheduler will pack all the pods into the cluster nodes based on the necessary requirements and constraints, and this is efficient for most standard needs.

However, in some cases you will want more capacity. This becomes an issue not of number of pods or CPU load, but of request latency. At issue are things like reconciling high-level intent when you're facing undeterminable latency.

Thankfully, Kubernetes autoscaling supports custom metrics and you can typically scale a horizontal pod autoscaler on arbitrary custom metrics like latency or requests per second – you need to make use of this and will be grateful that it's available. You can even use factors not related to Kubernetes objects. For example, if you use an external queue system outside Kubernetes, you can hook up high-level metrics like the number of requests in the queue or an even better time in the queue (average latency) to high-level capacity planning. If you have items in your queue that are older than X seconds, you can simply add another pod.

There's also the ability to listen to cluster events and monitor collections of pods and make very high-level capacity decisions automatically. This happens when a service starts to slow down and requires the need to schedule a few more instances because one of its dependencies is overloaded or the data store being used for storage is out of space.



3



# Rightsize cluster requests

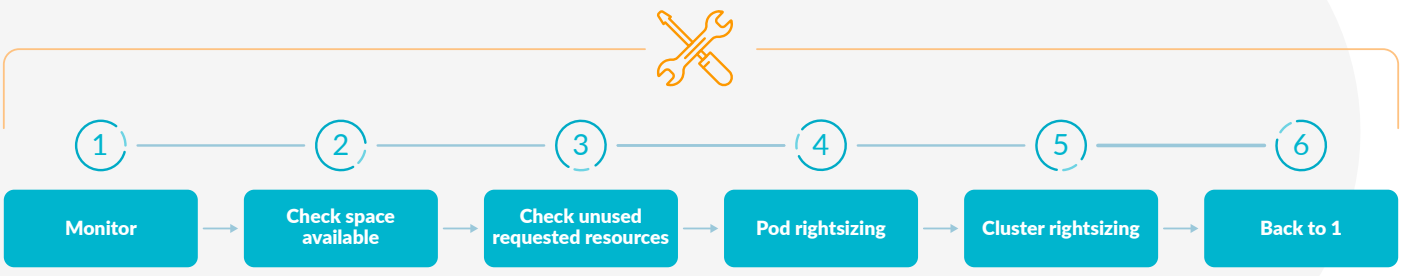
One way to ensure you don't overuse your capacity is simply to conserve it. While that might help you achieve the goal of coming in under your target, you might be reserving far too many resources and therefore limiting your ability to perform needed development and workload management.

Consider that if you find yourself in this situation, you're going to be charged for those resources even if they aren't being used, and it will also make deployments more difficult to schedule. That's why Kubernetes capacity planning is always a balance between the stability and reliability of the cluster, and the correct use of the resources.

There are situations where a container requests more resources than it needs. If it's just one container, it may not have a critical impact on the invoice from your cloud provider. But if this happens in all the containers, you'll have several extra costs in your invoices in a large cluster. Not to mention that if pods are too big, you may spend extra effort debugging scheduling issues. After all, it's harder for Kubernetes to schedule bigger Pods following your priorities.

In Kubernetes capacity planning, to reserve the correct amount of computational resources, you need to analyze the current resource usage of your containers. For that, you could use queries to calculate the average CPU utilization for all the containers belonging to the same workload. After performing some Kubernetes capacity planning operations, you'll need to check the impact of the changes on your infrastructure. For that, you can compare the underutilized CPU cores now against the values from one week before to assess the impact of your optimizations.

### Rightsizing the cluster



4



# Address memory and CPU issues

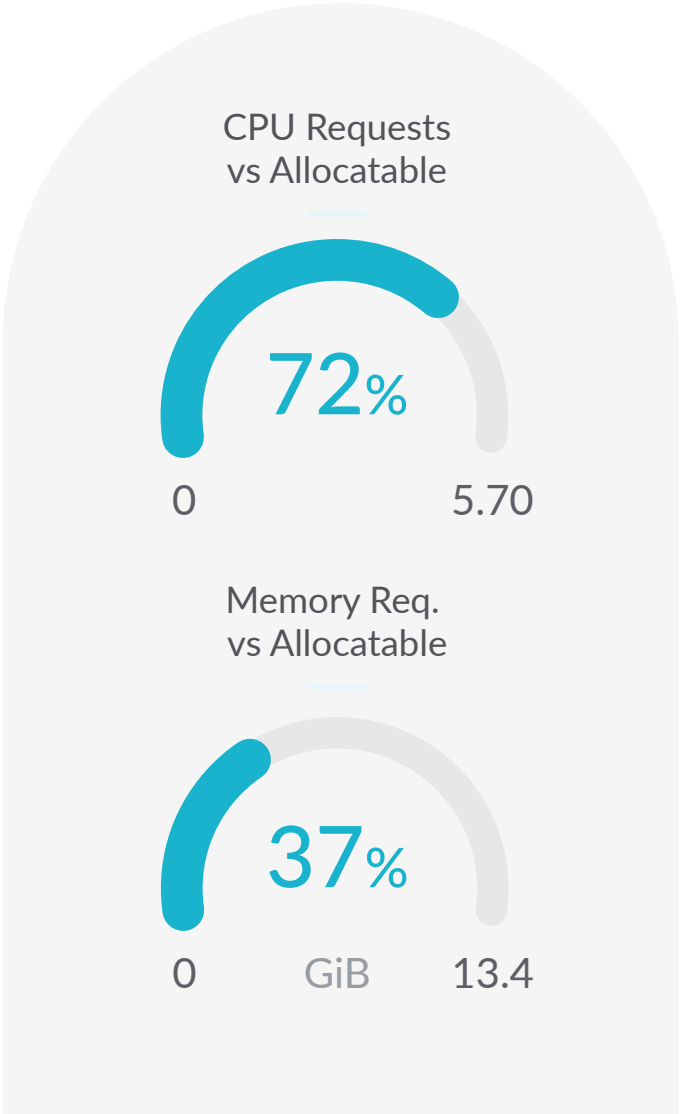
With many types of systems, once memory capacity gets low, an out-of-memory (OOM) safeguard kicks in and kills certain processes based on system-based rules.

The Kubernetes OOM approach initiates an eviction policy when a node is low on memory and typically stops pods by identifying them as "failed." These pods are then scheduled in a different node which frees memory to relieve the memory pressure. This is all fine and well as long as you've identified which nodes should be applied. Kubernetes will not allocate pods that request more memory than is available in a node. But limits can be higher than requests, so the sum of all limits can be higher than node capacity. This is called overcommit, and in practice, if all containers use more memory than requested, it can exhaust the memory in the node. This usually causes the death of some pods in order to free some memory, and it can happen at the worst time possible.

There are many differences on how CPU/memory requests and limits are treated in Kubernetes. A container using more memory than the limit will most likely die, but using CPU doesn't cause Kubernetes to kill a container. CPU management is delegated to the system scheduler, and it uses various mechanisms for the requests and the limits enforcement.

CPU requests are managed using a "shares" system. This means that the resources in the CPU are prioritized depending on the value of shares. Each CPU core is divided into 1,024 shares and the resources with more shares have more CPU time reserved. If a CPU is starved, however, shares won't ensure your app has enough resources, as it can be affected by bottlenecks and general collapse.

Note that a pod without requests and limits is the first of the list to OOM kill. With the CPU, this is not the case. A quality of service (QoS) class is assigned to pods by Kubernetes itself, and a pod without CPU limits is free to use all the CPU resources in the node. The CPU is there to be used, but if you can't control which process is using your resources, you can end up with a lot of problems due to CPU starvation of key processes.



5



# Adapt to limits and requests

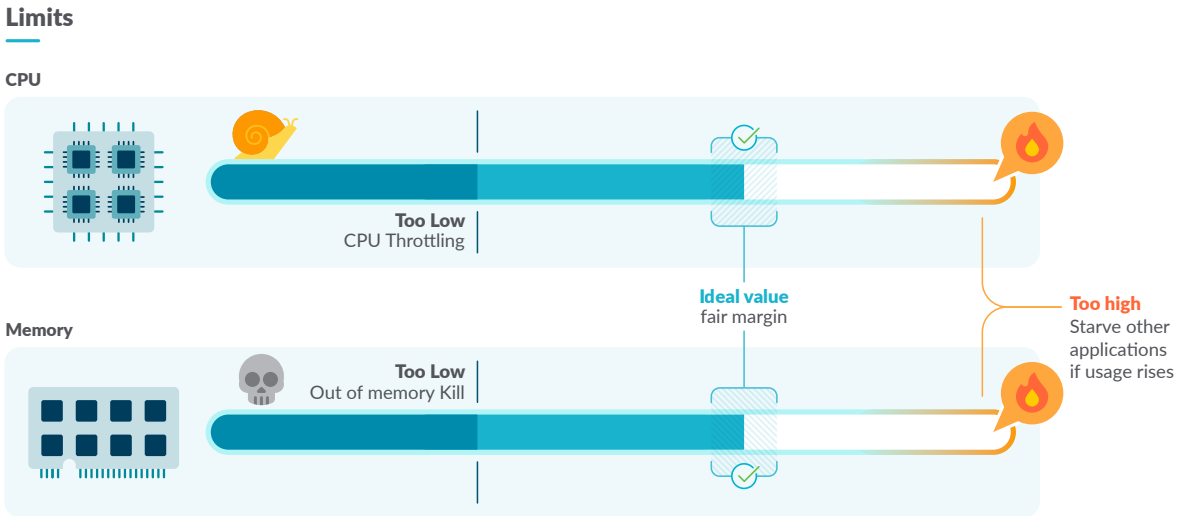
One of the challenges of every distributed system designed to share resources between applications (like Kubernetes), is **how** to properly share the resources. This may seem like a paradox, but applications have historically been designed to run in standalone fashion on bare metal and use all available resources. This creates a new landscape that requires sharing the same space with others, and that makes resource quotas a definitive requirement.

Kubernetes allows administrators to set quotas in namespaces as hard limits for resource usage. The resulting effect is that if you set a CPU request quota in a namespace, then all pods need to set a CPU request in their definition, otherwise they will not be scheduled.

When you have some experience with Kubernetes, you've probably experienced that properly setting requests and limits is of utmost importance for the performance of the applications and cluster.

In an ideal world, your pods should be continuously using exactly the amount of resources you requested, but we know all too well that this isn't likely to happen. Consider a 25% margin up and down the request value as a good barometer. If your usage is much lower than your request, you are wasting money. If it is higher, you are risking performance issues in the node. This is the essence of all the capacity planning we've discussed in this document, but the point is a critical one - expectations don't always match reality, but you can be prepared with a comfortable divergence in either direction.

Regarding limits, achieving a good setting is a matter of try, learn, and try again. There is no optimal value for everyone as it will depend on the nature of the application, the demand model, the tolerance to errors and many other factors.



6



# Use intuitive dashboards to optimize observability

Kubernetes monitoring dashboards enable us to have visibility and insights into activity we couldn't address manually. With the right dashboards, end-users are able to identify issues so they can troubleshoot quickly and optimize clusters without the need of experts.

While these dashboards are critical for troubleshooting, they are also an essential resource for doing capacity planning which helps prevent under- and/or over-provisioning. They can identify where you may have extra costs based on misalignment of platform resources, and can also identify where within your environment you can support new workloads.

An effective dashboard isn't simply a batch of metrics. The data it identifies and renders must be organized in a use-case-oriented fashion, which gives the user a well-structured, informed guide through your Kubernetes cluster. It's critical, therefore, that the user be able to identify not just the general operational details of their environment, but they must also be able to view the behaviors of the individual resources running in it.

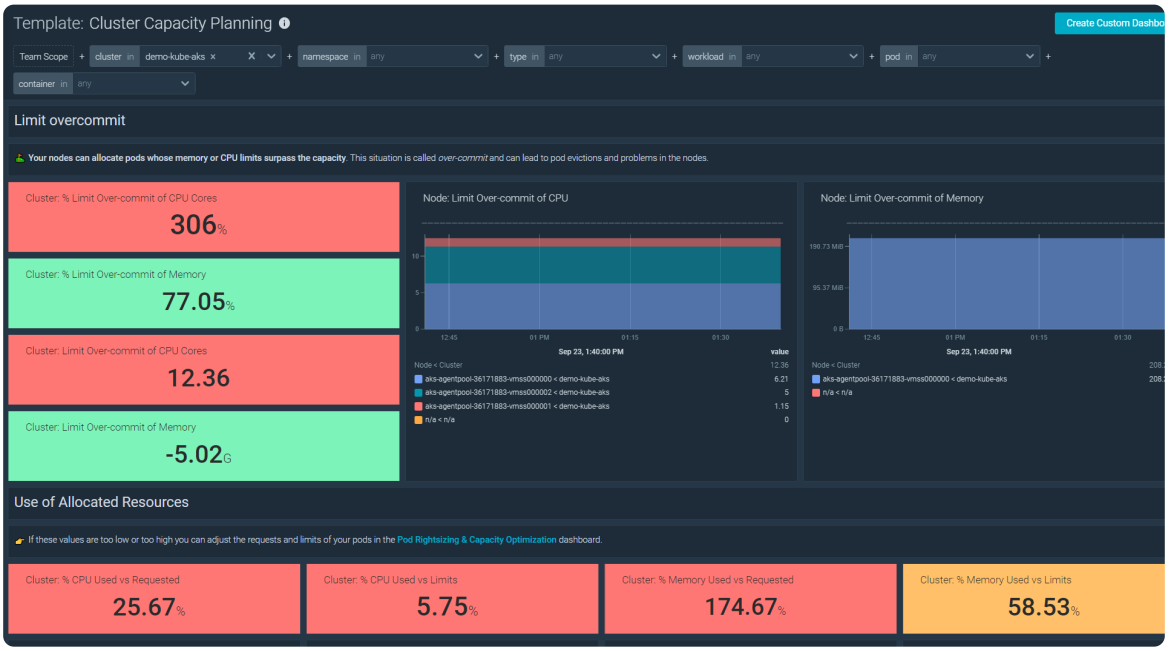
Sysdig's dashboards allow Kubernetes and DevOps admins to identify the operational elements of their Kubernetes clusters with these factors:

- → General overview of cluster behavior
- → Applications that have been deployed into your Kubernetes clusters, where they're integrating with other data sources, and issues that arise within those integrations.
- → Ability to troubleshoot applications directly from the dashboard
- → Create, manage, modify, and remove resources
- → Resource metrics for each Kubernetes object

Comparing the behavior of different containers in different clusters only takes a mouse click. The same can be done to easily compare metrics across cloud providers or regions or narrow down your scope to a particular workload.

This way, Kubernetes dashboards let you check that one app is operating effectively through all of your infrastructure, which enables admins to accurately identify performance status, which then also leads to a better ability to manage costs. Also, the dashboards let you troubleshoot potential issues by drilling down from the clusters to a particular container.

Critical situations aren't the only circumstances when a DevOps engineer needs to check out Kubernetes dashboards. Using the dashboards to optimize cluster configuration is something every SRE should do once in a while.




Dashboards provide all types of data, but it is only actionable intelligence derived from that data that helps organizations get the greatest benefit from their Kubernetes environment. The right dashboard helps you configure the optimal resource requests and limits for CPU and memory by showing the suggested best practices and meaningful tips, allowing you to set the better values for the studied containers.

**Proper resource planning and capacity optimization will allow you to operate Kubernetes with predictability and will improve system resilience. To do this, you will need deep visualize and correlation of all the metrics in your Kubernetes environment. This is where Sysdig can help! Our free trial can help you start improving your Kubernetes capacity planning today.**

**Start Your Free Trial**