# Autoscaling strategies in Kubernetes clusters

In Kubernetes, several things are referred to as "autoscaling", including:

* [Horizontal Pod Autoscaler](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/" \t "https://learnk8s.io/_blank).
* [Vertical Pod Autoscaler](https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler" \t "https://learnk8s.io/_blank).
* [Cluster Autoscaler](https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler" \t "https://learnk8s.io/_blank).

Those autoscalers belong to different categories because they address other concerns.

**Horizontal Pod Autoscaler (HPA)** is designed to increase the replicas in your deployments.

As your application receives more traffic, you could have the autoscaler adjusting the number of replicas to handle more requests.

The Horizontal Pod Autoscaler (HPA) inspects metrics such as memory and CPU at a regular interval. If the metrics pass a user-defined threshold, the autoscaler creates more Pods.

**Vertical Pod Autoscaler (VPA)** is useful when you can't create more copies of your Pods, but you still need to handle more traffic.

As an example, you can't scale a database (easily) only by adding more Pods.

A database might require sharding or configuring read-only replicas.

But you can make a database handle more connections by increasing the memory and CPU available to it.

That's precisely the purpose of the vertical autoscaler — increasing the size of the Pod.

**Cluster Autoscaler (CA)**.

When your cluster runs low on resources, the Cluster Autoscaler provision a new compute unit and adds it to the cluster.

If there are too many empty nodes, the cluster autoscaler will remove them to reduce costs.

While these components all "autoscale" something, they are entirely unrelated to each other.

They all address very different use cases and use other concepts and mechanisms.

And they are developed in separate projects and can be used independently from each other.

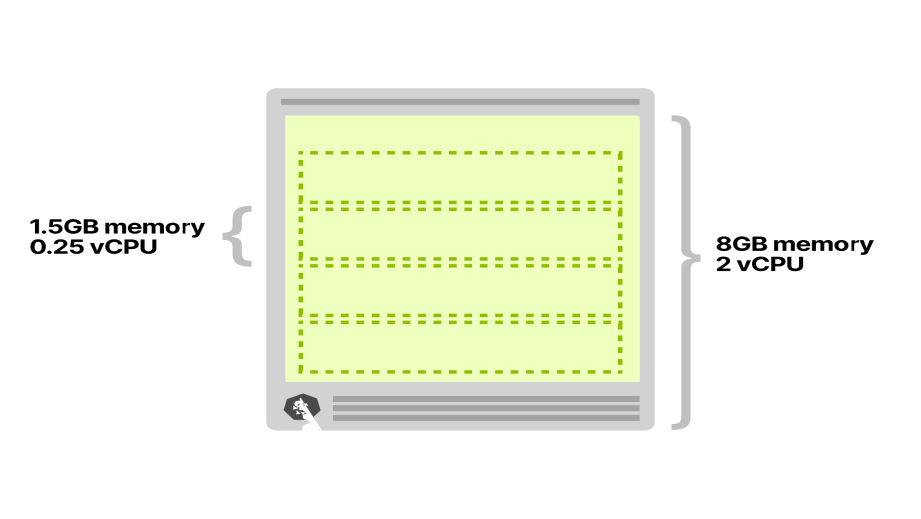
**However, scaling your cluster requires fine-tuning the setting of the autoscalers so that they work in concert.**

Let's have a look at an example.

## When autoscaling pods goes wrong

Imagine having an application that requires and uses 1.5GB of memory and 0.25 vCPU at all times.

You provisioned a cluster with a single node of 8GB and 2 vCPU — it should be able to fit four pods perfectly (and have a little bit of extra space left).



You deploy a single Pod and set up:

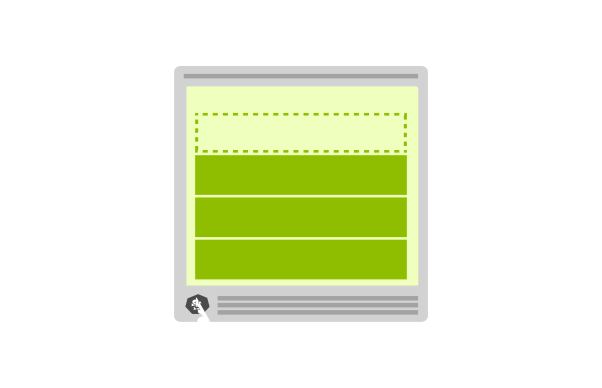
1. An **Horizontal Pod Autoscaler** adds a replica every 10 incoming requests (i.e. if you have 40 concurrent requests, it should scale to 4 replicas).
2. A **Cluster Autoscaler** to create more nodes when resources are low.

The Horizontal Pod Autoscaler can scale the replicas in your deployment using Custom Metrics such as the queries per second (QPS) from an Ingress controller.

You start driving traffic 30 concurrent requests to your cluster and observe the following:

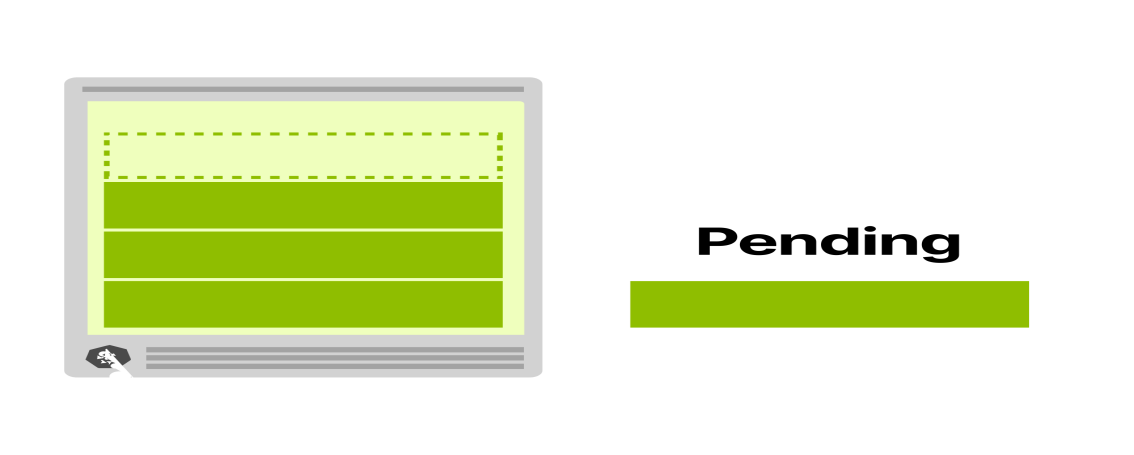
1. The **Horizontal Pod Autoscaler** starts scaling the Pods.
2. Two more Pods are created.
3. The **Cluster Autoscaler** doesn't trigger — no new node is created in the cluster.

It makes sense since there's enough space for one more Pod in that node.

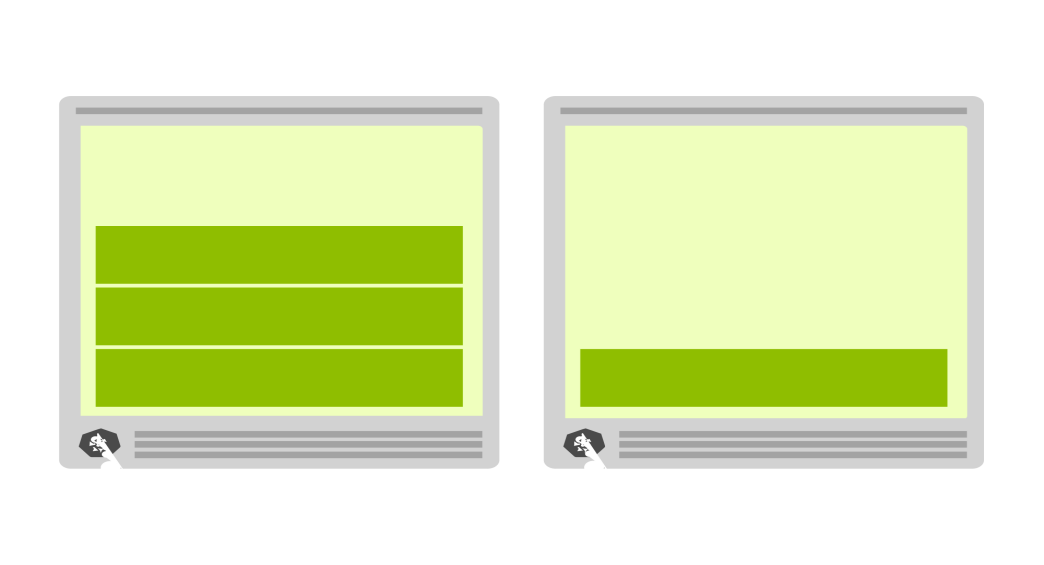


You further increase the traffic to 40 concurrent requests and observe again:

1. The **Horizontal Pod Autoscaler** creates one more Pod.
2. The Pod is pending and cannot be deployed.
3. The **Cluster Autoscaler** triggers creating a new node.
4. The node is provisioned in 4 minutes. After that, the pending Pod is deployed.



When you scale to four replicas, the fourth replicas isn't deployed in the first node. Instead, it stays “Pending".



The autoscaler creates a new node, and the pod is finally deployed.

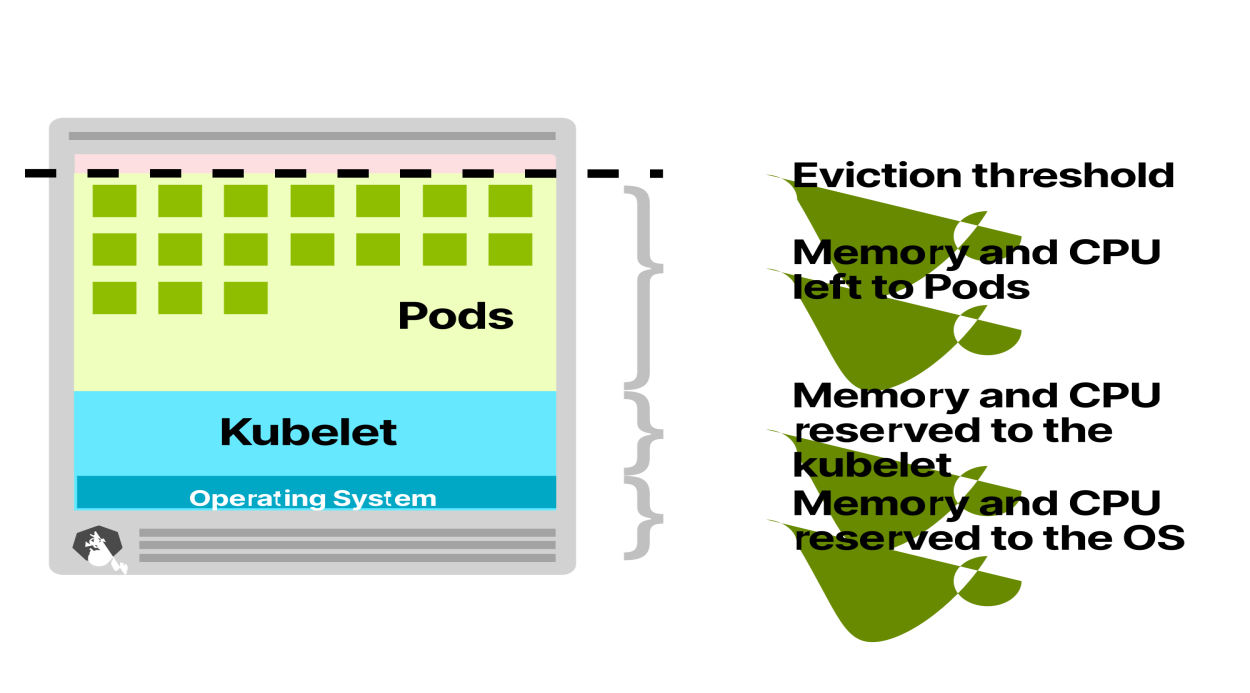
Why is the fourth Pod not deployed in the first node?

Pods deployed in your Kubernetes cluster consume resources such as memory, CPU and storage.

However, on the same node, **the operating system and the kubelet require memory and CPU too.**

In a Kubernetes worker node's memory and CPU are divided into:

1. **Resources needed to run the operating system and system daemons** such as SSH, systemd, etc.
2. **Resources necessary to run Kubernetes agents** such as the Kubelet, the container runtime, [node problem detector](https://github.com/kubernetes/node-problem-detector" \t "https://learnk8s.io/_blank), etc.
3. **Resources available to Pods.**
4. **Resources reserved to the[eviction threshold](https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/" \l "eviction-thresholds" \t "https://learnk8s.io/_blank)**.



As you can guess, [all of those quotas are customisable](https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/" \l "eviction-thresholds" \t "https://learnk8s.io/_blank), but you need to account for them.

In an 8GB and 2 vCPU virtual machine, you can expect:

* 100MB of memory and 0.1 vCPU to be reserved for the operating system.
* 1.8GB of memory and 0.07 vCPU to be reserved for the Kubelet.
* 100MB of memory for the eviction threshold.

**The remaining ~6GB of memory and 1.83 vCPU are usable by the Pods.**

If your cluster runs a DeamonSet such as kube-proxy, you should further reduce the available memory and CPU.

Considering kube-proxy has requests of 128MB and 0.1 vCPU, only ~5.9GB of memory and 1.73 vCPU are available to run Pods.

Running a CNI like Flannel and a log collector such as Fluentd will further reduce your resource footprint.

After accounting for all the extra resources, you have space left for only three pods.

OS 100MB, 0.1 vCPU +

Kubelet 1.8GB, 0.07 vCPU +

Eviction threshold 100MB, 0 vCPU +

Daemonsets 128MB, 0.1 vCPU +

======================================

Used 2.1GB, 0.27 vCPU

======================================

Available to Pods 5.9GB, 1.73 vCPU

Pod requests 1.5GB, 0.25 vCPU

======================================

Total (4 Pods) 6GB, 1vCPU

The fourth stays "Pending" unless it can be deployed on another node.

Since the Cluster Autoscaler knows that there's no space for a fourth Pod, why doesn't it provision a new node?

Why does it wait for the Pod to be pending before it triggers creating a node?

## How the Cluster Autoscaler works in Kubernetes

**The Cluster Autoscaler doesn't look at memory or CPU available when it triggers the autoscaling.**

Instead, the Cluster Autoscaler reacts to events and checks for any unschedulable Pods every 10 seconds.

A pod is unschedulable when the scheduler is unable to find a node that can accommodate it.

For example, when a Pod requests 1 vCPU but the cluster has only 0.5 vCPU available, the scheduler marks the Pod as unschedulable.

**That's when the Cluster Autoscaler initiates creating a new node.**

The Cluster Autoscaler scans the current cluster and [checks if any of the unschedulable pods would fit on in a new node.](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md" \l "what-are-expanders" \t "https://learnk8s.io/_blank)

If you have a cluster with several node types (often also referred to as node groups or node pools), the Cluster Autoscaler will pick one of them using the following strategies:

* **Random** — picks a node type at random. This is the default strategy.
* **Most pods** — selects the node group that would schedule the most pods.
* **Least waste** — selects the node group with the least idle CPU after scale-up.
* **Price** — select the node group that will cost the least (only works for GCP at the moment).
* **Priority** — selects the node group with the highest priority (and you manually assign priorities).

Once the node type is identified, the Cluster Autoscaler will call the relevant API to provision a new compute resource.

If you're using AWS, the Cluster Autoscaler will provision a new EC2 instance.

On Azure, it will create a new Virtual Machine and on GCP, a new Compute Engine.

It may take some time before the created nodes appear in Kubernetes.

Once the compute resource is ready, the [node is initialised](https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-tls-bootstrapping/" \t "https://learnk8s.io/_blank) and added to the cluster where unscheduled Pods can be deployed.

**Unfortunately, provisioning new nodes is usually slow.**

It might take several minutes to provision a new compute unit.