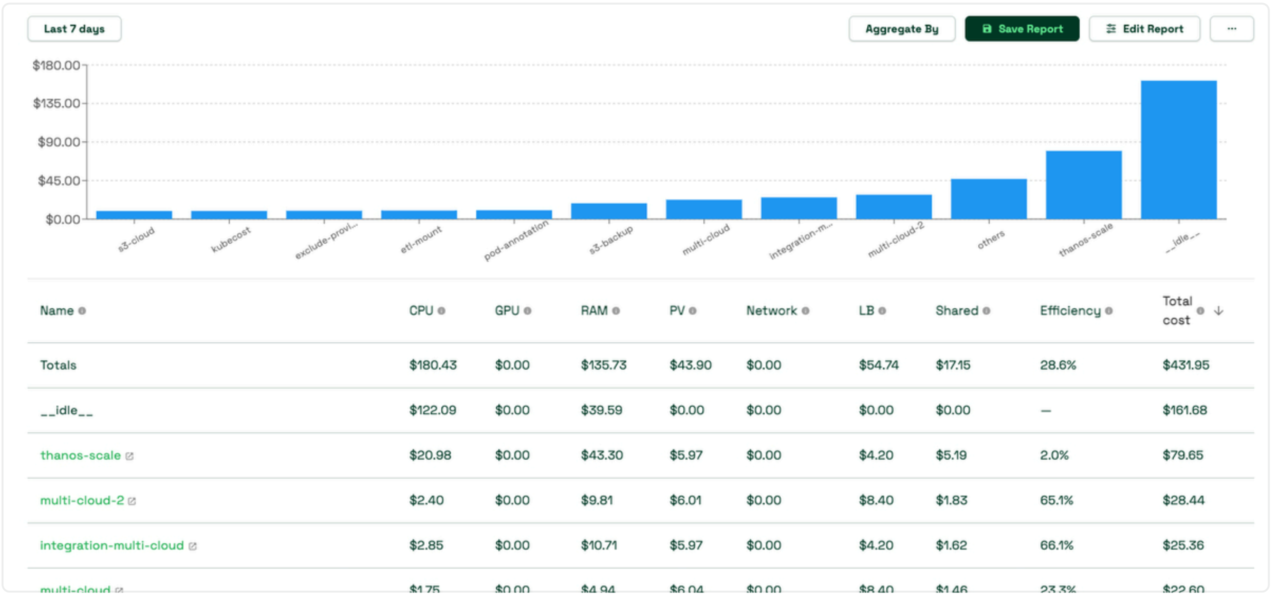
Efficiency and Idle

For teams interested in reducing their Kubernetes costs, it's beneficial to first understand how provisioned resources have been used. There are two major concepts to start with: pod resource efficiency and cluster idle costs.



Name	CPU	GPU	RAM	PV	Network	LB	Shared	Efficiency	Total cost ↓
Totals	$180.43	$0.00	$135.73	$43.90	$0.00	$54.74	$17.15	28.6%	$431.95
__idle__	$122.09	$0.00	$39.59	$0.00	$0.00	$0.00	$0.00	–	$161.68
thanos-scale	$20.98	$0.00	$43.30	$5.97	$0.00	$4.20	$5.19	2.0%	$79.65
multi-cloud-2	$2.40	$0.00	$9.81	$6.01	$0.00	$8.40	$1.83	65.1%	$28.44
integration-multi-cloud	$2.85	$0.00	$10.71	$5.97	$0.00	$4.20	$1.62	66.1%	$25.36
multi-cloud	$1.75	$0.00	$4.94	$6.04	$0.00	$8.40	$1.46	23.3%	$22.60

The Allocations view aggregated by namespace, which shows efficiency & idle

Efficiency

Pod resource efficiency is defined as the resource utilization versus the resource request over a given time window. It is cost-weighted and can be expressed as follows:

> *(((CPU Usage / CPU Requested) * CPU Cost) + ((RAM Usage / RAM Requested) * RAM Cost)) / (RAM Cost + CPU Cost)*
> where
> *CPU Usage = rate(container_cpu_usage_seconds_total) over the time window*
> *RAM Usage = avg(container_memory_working_set_bytes) over the time window*

For example, if a pod is requesting 2CPU and 1GB, using 500mCPU and 500MB, CPU on the node costs $10/CPU, and RAM on the node costs $1/GB, we have ((0.5/2) * 20 + (0.5/1) * 1) / (20 + 1) = 5.5 / 21 = 26%

Idle

Cluster idle cost is defined as the difference between the cost of allocated resources and the cost of the hardware they run on. Allocation is defined as the max of usage and requests. It can also be expressed as follows:

> *idle_cost = sum(cluster_cost) - (cpu_allocation_cost + ram_allocation_cost + gpu_allocation_cost)*
> where
> *allocation = max(request, usage)*

Node idle cost can be expressed as:

> *idle_cost = sum(node_cost) - (cpu_allocation_cost + ram_allocation_cost + gpu_allocation_cost)*
> where
> *allocation = max(request, usage)*

So, idle costs can also be thought of as the cost of the space that the Kubernetes scheduler could schedule pods, without disrupting any existing workloads, but it is not currently.

Sharing idle

Idle can be charged back to pods on a cost-weighted basis or viewed as a separate line item. As an example, consider the following representations:

- [...] = cluster
- (...) = node
- wN = workload
- -- = idle capacity

Then, a cluster might look like:

[(w1, w2, w3, w4, --, --), (w5, --, --, --, --, --)]

In total, there are 12 units of resources, and idle can be shared as follows:

- **Separate**: In this single cluster across two nodes, there are 7 total idles.
- **Share By Node**: The first node has 4 resources used and 2 idle. The second node has 1 resource used and 5 idle. If you share idle by node, then w1-4 will share 2 idles, and w5 will get 5 idles.
- **Share By Cluster**: The single cluster has 5 resources used and 7 idle. If you share idle by cluster, then w1-5 will share the 7 idles.

Distributing idle when aggregating

If for example you are aggregating by namespace, idle costs will be distributed to each namespace proportional to how much that namespace costs. Specifically:

> *namespace_cpu_idle_cost = (namespace_cpu_cost / (total_cpu_cost - idle_cpu_cost)) * idle_cpu_cost*

This same principle applies for ram, and also applies to any aggregation that is used (e.g. Deployment, Label, Service, Team).

Target values for efficiency and idle

The most common pattern for cost reduction is to ensure service owners tune the efficiency of their pods, and ensure cluster owners scale resources to appropriately minimize idle.

Efficiency targets can depend on the SLAs of the application. See our Request Right-Sizing API doc for more details.

It's recommended to target idle in the following ranges:

- CPU: 50%-65%
- Memory: 45%-60%
- Storage: 65%-80%

Target figures are highly dependent on the predictability and distribution of your resource usage (e.g. P99 vs median), the impact of high utilization on your core product/business metrics, and more. While too low resource utilization is wasteful, too high utilization can lead to latency increases, reliability issues, and other negative behavior.