



State of Kubernetes Cost Optimization

Authored by: Billy Lamberti • Fernando Rubbo • Kent Hua • Li Pan • Melissa Kendall

June 2023

Table of contents

Executive summary	3	Final thoughts	19
Key Findings	4	Acknowledgements	20
Kubernetes cost optimization golden signals	6	Authors	21
Why do Elite performers establish the practices to be followed?	8	Methodology	23
The At Risk segment	10	The Resources group	24
Balancing reliability and cost efficiency	13	Workload rightsizing	24
What do High and Elite performers teach us about how to improve cost efficiency?	17	Demand based downscaling	25
		Cluster bin packing	25
		The Discount group	26
		Other Metrics	27
		Research population	27
		Method	27
		Steps for analysis	27



Executive summary

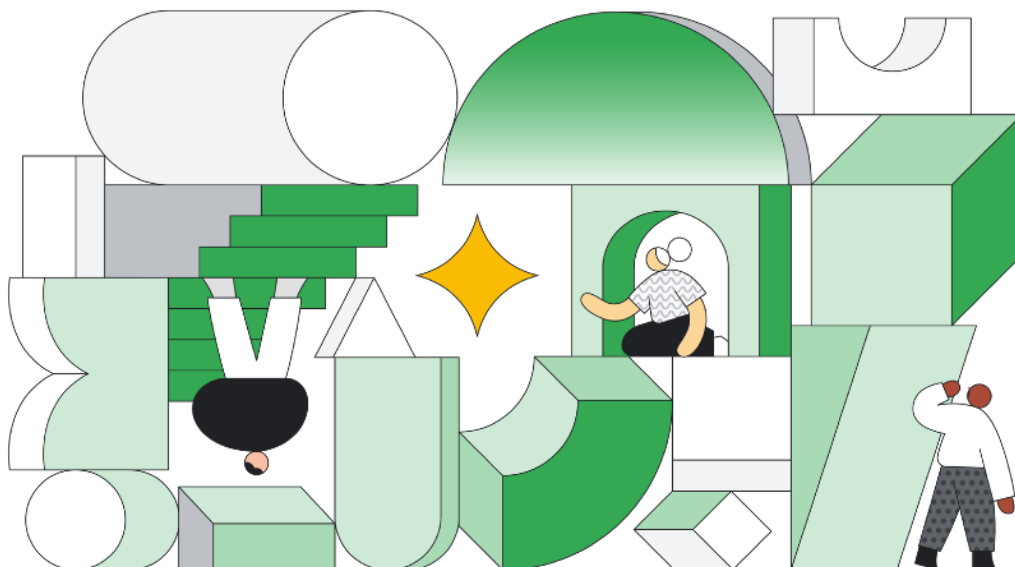
Given the economic headwinds that started with the COVID-19 and the macroeconomic challenges that followed (such as high inflation, supply chain disruptions, and geopolitical tensions), IT teams have been pursuing cloud cost optimization to achieve better resilience and re-invest saved resources in providing a better and more innovative experience to their customers. While [FinOps](#) is known to be a key practice for driving both resiliency and cloud transformation, cost optimization has proven to be challenging to execute especially when cloud native platforms, such as Kubernetes, are taken into account.

Kubernetes provides a rich feature set and scheduling capabilities. However, optimizing the use of Kubernetes without impacting the end user experience and overall reliability of the related applications requires great understanding of the capabilities provided by the platform.

It is with great pleasure that Google presents the *State of Kubernetes Cost Optimization* report. This report is a quantitative analysis of real-world large-scale anonymized data. It provides insights

and best practices to the community (e.g. platform administrators, application operators, and developers) on how to manage public cloud Kubernetes clusters in a cost efficient manner, without compromising the performance and reliability of workloads. The findings of this report are critical for organizations currently running clusters in the public cloud and organizations ramping up on Kubernetes, so that they can learn while they are migrating to the cloud.

Our research focuses on examining how capabilities and practices predict the outcome for cost optimization performance of Kubernetes clusters running on any cloud provider. Because cloud cost optimization is a continuous process that requires the engagement of different teams across an organization, we propose that teams should measure their performance using what we call the [golden signals](#): workload rightsizing, demand based downscaling, cluster bin packing, and discount coverage. As described in the [Methodology section](#), these signals are used by this report to segment clusters into different levels of cost optimization performance.



Key Findings

1. Kubernetes cost optimization starts with understanding the importance of setting appropriate resource requests

In addition to Kubernetes using CPU and memory requests for bin packing, scheduling, cluster autoscaling, and horizontal workload autoscaling, Kubernetes also uses resource requests to classify a [Pod's Quality of Service \(QoS\)](#) class. Kubernetes relies on this classification to make decisions about which Pods should be immediately killed when a given node's utilization approaches its capacity. Not setting requests indirectly assigns the BestEffort class to Pods.

Whenever [Kubernetes needs to reclaim resources at node-pressure](#), without any warning or graceful termination, BestEffort Pods are the first to be killed, potentially causing disruption to the overall application. A similar problem happens with Burstable workloads that constantly use more memory than they have requested. Because cost optimization is a discipline to drive cost reduction while maximizing business value, indiscriminately deploying these kinds of workloads impacts the behavior of Kubernetes bin packing, scheduling, and autoscaling algorithms. It can also negatively affect end user experience and, consequently, your business.



2. Workload rightsizing is the most important golden signal

The research has found that workload resource requests are, on average, substantially over-provisioned. Even *Elite* performers that efficiently set memory requests have room for improvement when it comes to setting CPU requests. The research also found that workload rightsizing has the biggest opportunity to reduce resource waste. Most importantly, cluster owners that focus on addressing discount coverage or cluster bin packing without addressing workload rightsizing may find that they have to re-do their efforts when their workloads become properly sized. To reduce over-provisioning, companies should first create awareness with technical teams about the importance of setting requests, and then focus on rightsizing their workloads.

3. Some clusters struggle to balance reliability and cost efficiency

There are some fairly large clusters (2.1x bigger than *Low* performers) that demonstrate, through a set of enabled features, a bigger than average intent to optimize costs. However, due to an unintentional over use of both BestEffort and memory under-provisioned Burstable Pods, cluster nodes are often overloaded. This situation increases the risk of intermittent and hard to debug performance and reliability issues that were discussed in finding #1. In our exploratory analysis, cluster owners shared that they either didn't know they were running large amounts of BestEffort and memory under-provisioned Burstable Pods, or they didn't understand the consequences of deploying them.

4. End user experience can be compromised when cost optimization efforts don't consider reliability

Due to how the Kubernetes scheduler works, clusters running large quantities of BestEffort Pods or memory under-provisioned Burstable Pods tend to have low cluster bin packing (slightly higher than *Low* performers). Platform admins could interpret this signal as a false opportunity for cost savings through scaling down the cluster or rightsizing cluster nodes. If platform admins implement these changes, it can result in application disruption due to the increased chances of Pods being terminated without warning or eviction time.

5. Demand-based downscaling relies heavily on workload autoscaling

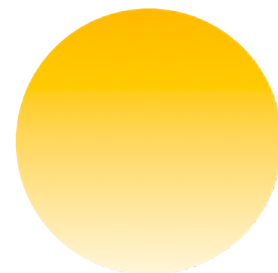
The research has found that *Elite* performers can scale down 4x more than *Low* performers. This happens because *Elite* performers take advantage of existing autoscaling capabilities more than any other segment. In this context, autoscaling capabilities refers to Cluster Autoscaler (CA), Horizontal Pod Autoscaler (HPA), and Vertical Pod Autoscaler (VPA). The data shows that *Elite* performers enable CA 1.4x, HPA 2.3x, and VPA 18x more than *Low* performers. In other words, enabling CA is not enough to make a cluster scale down during off-peak hours. To autoscale a cluster, it is necessary to configure workload autoscaling (e.g. HPA and VPA) properly. The more workloads you manage to scale down during off-peak hours, the more efficiently CA can remove nodes.

6. *Elite* and *High* performers take advantage of cloud discounts

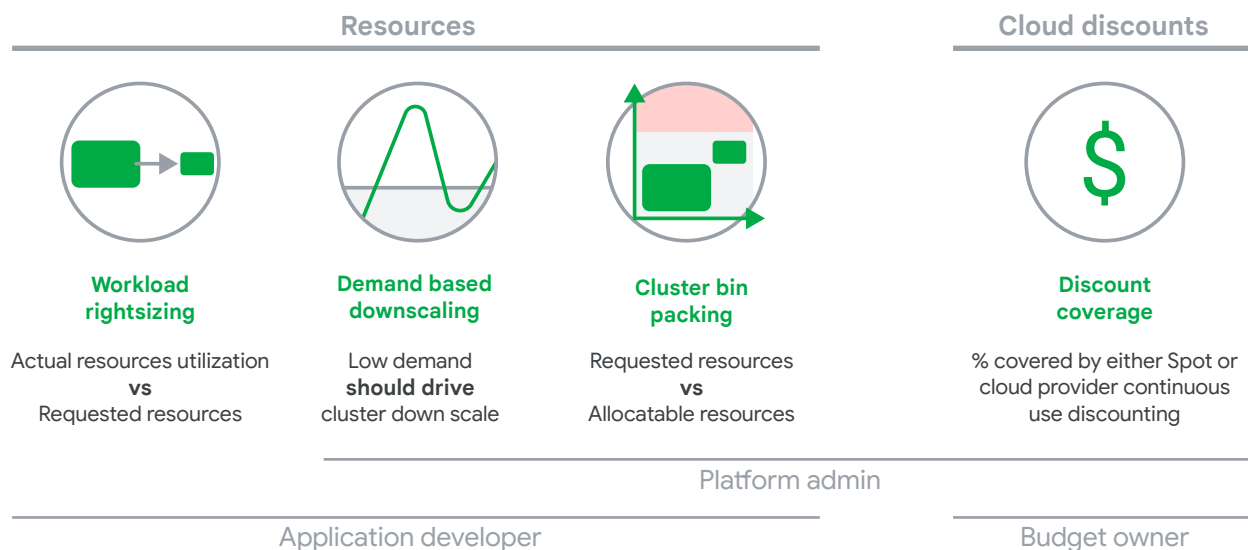
Elite and *High* performers adopt cloud discounts 16.2x and 5.5x more than *Low* performers, respectively. Because these segments run the largest clusters (6.2x and 2.3x bigger than *Low* performers, respectively), they tend to have better in-house Kubernetes expertise and dedicated teams focusing on cost optimization activities. This allows them to incorporate the use of significantly discounted Spot VMs and forecast long term commitments.

7. Chargeback can be compromised if Pods don't set requests properly

Most Kubernetes cost allocation tooling leverages resource requests to compute costs. When Pods do not set requests properly, such as when using BestEffort Pods and under-provisioned Burstable Pods, it can limit [showback](#) and [chargeback](#) accuracy. For example, even in a scenario where a given BestEffort Pod consumes a large amount of CPU and memory from a cluster, no cost is attributed to the Pod because it requests no resources.



Kubernetes cost optimization golden signals



Both *the increase in container costs* and *the need to reduce waste* are highlighted as top challenges in the [2023 State of FinOps report](#). The [Flexera's 2023 State of Cloud Report](#) also noted *optimizing existing use of cloud* as the top initiative for the seventh year in a row. Despite these items being broader to many public cloud products, they are predominant and especially challenging to Kubernetes clusters. Kubernetes is a complex distributed system with many features that, when not used correctly, can lead to increased levels of over-provisioning. To avoid being billed for resources that you don't need, observability is critical; you can't manage what you can't see. After several years of iterating over numerous metrics and feedback through customer engagements, Google has identified four key signals ("the golden signals") that help you to measure the cost optimization performance of your Kubernetes clusters on public clouds.

As you can see in the image above, the golden signals are broken into two distinct groups. The *Resources group* focuses on the capacity of using the CPU and Memory that you are paying for, while the *Cloud discounts group* focuses on the ability to take advantage of cloud provider discounts. Although it is recommended that companies measure some of these signals, such as workload rightsizing and demand based downscaling, at both the cluster and workload levels, this research segments and compares clusters, not workloads. Furthermore, all data in the report is measured at the cluster level. Along with the signals, we also highlight the role responsibilities that are often shared between different teams.

Note: We use the term application developer, or simply developer, interchangeable with any role that is responsible for managing Kubernetes configuration files, usually developers, DevOps, application operators, etc.

Google Cloud

The Resources group

Workload rightsizing measures the capacity of developers to use the CPU and memory they have requested for their applications.

Demand based downscaling measures the capacity of developers and platform admins to make their cluster scale down during off-peak hours.

Cluster bin packing measures the capacity of developers and platform admins to fully allocate the CPU and memory of each node through Pod placement.

The Cloud Discounts group

Discount coverage measures the capacity of platform admins to leverage machines that offer significant discounts, such as Spot VMs, as well as the capacity of budget owners to take advantage of long-term continuous use discounts offered by cloud providers.



Why do *Elite* performers establish the practices to be followed?

As discussed previously, this report uses [Kubernetes cost optimization golden signals](#) to group clusters into different segments. Using this segmentation we analyzed the distinguishing features of each segment. The following table summarizes how *Medium*, *High*, and *Elite* performing cluster metrics compare to *Low* performers, in number of times, for each individual golden metric signal. For example, 2.8x represents 2.8 times better than *Low* performing clusters.

As you can see, the *Elite* segment performs better on all golden signal metrics. Therefore, to meet the demand of running cost-efficient Kubernetes clusters without compromising the reliability and the performance of applications, the *Elite* performers establish the practices to be followed. The main reasons are summarized as follows.

Comparison of each segment to *Low* performers

Performers	CPU Workload rightsizing	Memory Workload rightsizing	Demand based downscaling	CPU Cluster bin packing	Memory Cluster bin packing	Discount coverage
Medium	1.3x	1.3x	2.0x	1.6x	1.8x	1.7x
High	2.2x	2.1x	3.2x	1.9x	2.4x	5.4x
Elite	2.8x	2.7x	4.0x	2.1x	2.9x	16.2x

Elite performers take advantage of cloud discounts 16.2x more than *Low* performers. They also consistently consume the compute resources they pay for better than the *Low* performers (5.1x more for CPU and 2.7x more for memory). Because *Elite* performers surpass the other segments on both the golden signals and the adoption of cost optimization related features (e.g. Cluster Autoscaler, Horizontal Pod Autoscaler, cost allocation tools, etc), we can assume their teams have a better understanding of the Kubernetes platform and are better prepared to forecast long term commitments. These capabilities make it possible for them to run considerably lower priced workloads compared to other segments.

Elite performers take advantage of cloud discounts **16.2x** more than *Low* performers.

Compared to *Low* performers, *Elite* performers consistently consume more of the resources they pay for - **5.1x** more CPU, and **2.7x** more memory.

Google Cloud

When it comes to correctly sizing their workloads, *Elite* performers achieve 2.8x better CPU and 2.7x better memory than *Low* performers. These numbers show that developers deploying to *Elite* performer clusters have a better understanding of the resources required by their applications in each environment (eg. testing, staging, and production environments).

For the *demand based downscaling* signal, *Elite* performers demonstrate 4x more capacity to scale down their clusters during off-peak hours than *Low* performers. It is important to remember that scaling down a cluster is a shared responsibility between developers and platform teams. If developers don't scale down their workloads, platform teams' ability to scale down a cluster is limited. On the other hand, if developers manage to scale down their workloads during off-peak hours, platform admins can fine tune Cluster Autoscaler to satisfy demand according to business needs.

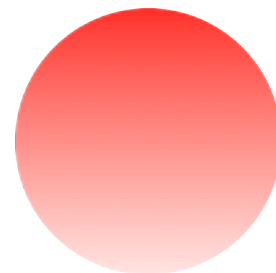
For *cluster bin packing*, we have found that *Elite* performers can better pack their Pods into their cluster nodes for both CPU and memory compared to *Low* performers (2.1x and 2.9x, respectively). *Cluster bin packing* is another shared responsibility between developers and platform teams, where both should collaborate to find the appropriate machine shape to better fit the workloads.

In the [next section](#) we discuss the *At Risk* segment, which has the highest probability across all segments to negatively impact end user experience. [Balance reliability and cost efficiency](#) discusses best practices for avoiding the pitfall of having your workloads killed without any warning or graceful termination. [What do High and Elite performers teach us about how to improve cost efficiency?](#) presents best practices, and [Final thoughts](#) presents concluding remarks.

Compared to *Low* performers, *Elite* performers rightsize their workloads **2.8x** better on CPU, and **2.7x** better on memory.

Elite performers demonstrate **4x** more capacity to scale down their clusters during off-peak hours than *Low* performers.

Compared to *Low* performers, *Elite* performers better pack their Pods into their cluster nodes - **2.1x** better on CPU and **2.9x** better on memory.



The *At Risk* segment

In addition to segmenting *Elite* from *High*, *Medium*, and *Low* performers, we created another segment, which we call the *At Risk* segment, with clusters where the sum of actual resource utilization is generally higher than the sum of their workloads' requested resources. We decided to separate these clusters into a different segment because after several years of customer engagements, Google has identified that the usage pattern of these cluster's results in a higher risk of intermittent and hard to debug reliability issues caused by the way [Kubernetes reclaims resources at node-pressure](#).

In summary, whenever a cluster's node resource utilization approaches its capacity, [kubelet](#) enters into a "self-defense mode" by terminating Pods immediately, meaning without any warning or graceful termination, to reclaim the starved resources. This situation is caused by Pods that use more resources than they have requested, such as BestEffort Pods and memory under-provisioned Burstable Pods. These Pods are also

the first ones to be killed by kubelet. Even though the default Kubernetes behavior prefers to schedule incoming Pods on nodes with low bin packing, the bin packing algorithm doesn't take into account actual resource utilization, it only considers resources requested. However, the Kubernetes scheduler continues to schedule incoming BestEffort and under-provisioned Burstable Pods on a few low bin packed nodes, causing these nodes to have higher than requested utilization. This situation can trigger the kubelet "self-defense mode".

To avoid negatively impacting end user experience and to avoid spending time debugging intermittent and hard to predict application errors, both BestEffort Pods and memory under-provisioned Burstable Pods must be used with caution. Because they can be killed by kubelet whenever a node is under pressure, application developers and operators must fully understand the consequences of running them.



Google Cloud

The research has found that the clusters in the *At Risk* segment use more resources than they have requested because they deploy significantly more BestEffort Pods (1.6x more than *Low* performers and 2.7x more than *Elite* performers).

Both BestEffort Pods and memory-underprovisioned Burstable Pods remain useful for utilizing the temporary idle capacity from a cluster, but developers should adopt these kinds of Pods only for **best effort workloads that can be killed at any moment, without any eviction time**.

The *At Risk* segment is composed of medium size clusters (2.1x bigger than *Low* performers and 3x smaller than *Elite* performers, in number of nodes). These clusters demonstrate, through a set of enabled features, bigger than the average intent to optimize cost (3.9x more adoption of cloud discounts, 1.5x more adoption of cost allocation tools, and 3.1x more VPA deployments using recommendation mode than *Low* performers). Despite intent to optimize, their chargeback or showback solutions are more likely to attribute inaccurate costs to teams or divisions. This happens because most Kubernetes cost allocation tools leverage resource requests to compute costs, and the large adoption of BestEffort Pods causes this segment to constantly use more CPU and memory than they have requested. For example, the research shows that clusters in this segment use close to 60% more memory than they have requested. Because most Kubernetes cost allocation tools don't examine actual utilization, any usage that exceeds what was requested is not accounted from a cost perspective. For example, if a cluster's Pods request 25 mCPU but use 75 mCPU, cost allocation tools consider the additional 50 mCPU as unused.

At Risk segment deploys more BestEffort Pods - **1.6x** more than *Low* performers, and **2.7x** more than *Elite* performers.

Both BestEffort Pods and memory-underprovisioned Burstable Pods remain useful for utilizing the temporary idle capacity from a cluster, but developers should adopt these kinds of Pods only for **best effort workloads that can be killed at any moment, without any eviction time**.



Google Cloud

We also found that the *At Risk* segment has relatively low cluster bin packing (1.3x worse CPU bin packing and 1.7x worse memory bin packing than *Medium* performers). Some platform teams may view this as a false opportunity for cost optimizing their clusters. If cluster operators act on this false optimization opportunity, they can cause disruption due to the increased chance of BestEffort Pods and memory under-provisioned Burstable Pods being terminated without any warning or eviction time.

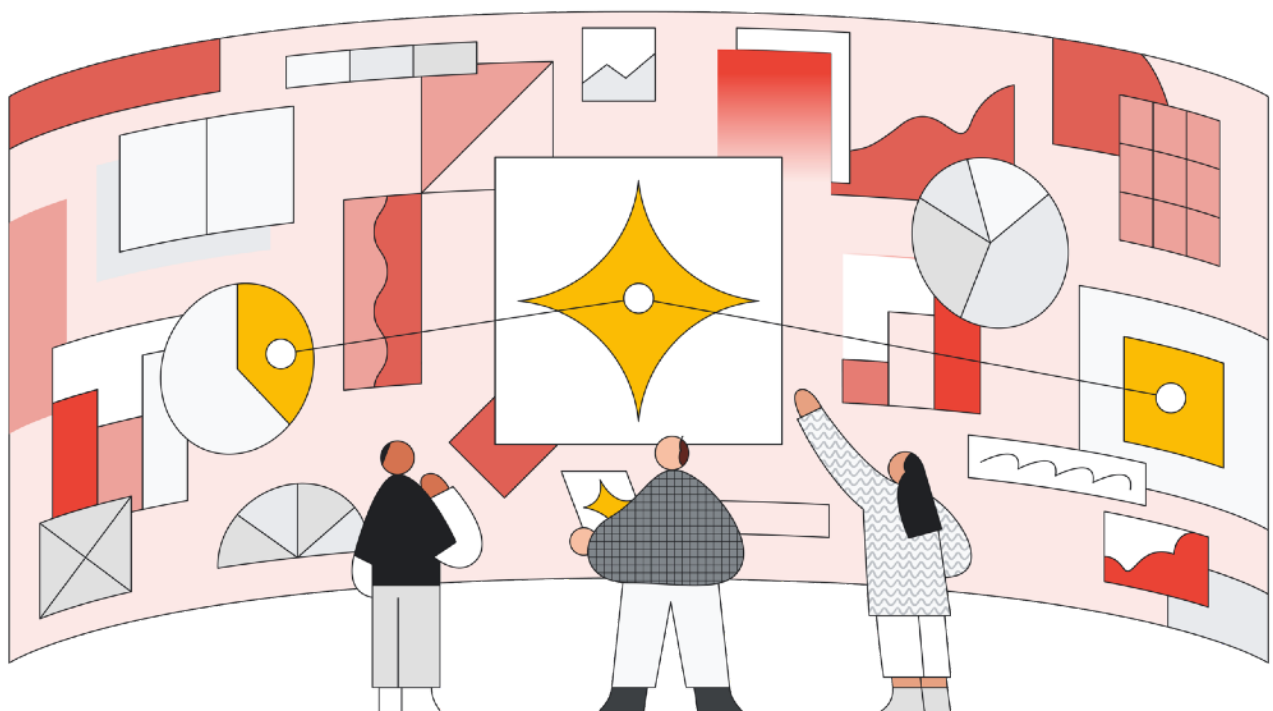
Based on low *cluster bin packing* and minimal adoption, between all segments, of cost optimization features (1.3x smaller adoption of Cluster Autoscaler, 2.2x less fine tuning of Cluster Autoscaler downscaling, and 1.7x smaller adoption of HPA than *Low* performers), we believe that the *At Risk* segment is attempting to mitigate reliability issues discussed in this section by keeping clusters artificially over-provisioned.

The [next section](#) discusses strategies and best practices you can follow to avoid the pitfalls faced by clusters in the *At Risk* segment.

At Risk segment adopts **1.3x** less Cluster Autoscaler than *Low* performers.

At Risk segment fine tune **2.2x** less Cluster Autoscaler than *Low* performers.

At Risk segment adopts **1.7x** less Horizontal Pod Autoscaler than *Low* performers.






Balancing reliability and cost efficiency

As discussed previously, beyond making it harder to horizontally scale your applications and impacting cost attribution solutions, Kubernetes bin packing, scheduling, and cluster autoscaler algorithms, the unintentional over use of BestEffort Pods and memory under-provisioned Burstable Pods can also negatively affect the end user experience. Although the *At Risk* segment has a higher risk of facing such problems, the research has found that all segments have the opportunity to reduce the use of BestEffort Pods and memory under-provisioned Burstable Pods, and consequently reduce, or even eliminate, debugging intermittent errors caused by kubelet killing Pods without warning.

To accomplish this task, companies must invest in providing technical teams with training, visibility, and guardrails.

For training, companies should ensure developers and operators understand the [Kubernetes Quality of Service \(QoS\) model](#) and how their configuration choices can impact the reliability of their applications when cluster nodes are under resource pressure:

BestEffort	Burstable	Guaranteed
<pre>apiVersion: v1 ... spec: containers: - name: qos-example image: nginx</pre>	<pre>apiVersion: v1 ... spec: containers: - name: qos-example image: nginx resources: requests: cpu: 500m memory: 100Mi limits: cpu: 1 memory: 200Mi</pre>	<pre>apiVersion: v1 ... spec: containers: - name: qos-example image: nginx resources: requests: cpu: 3 memory: 200Mi limits: cpu: 3 memory: 200Mi</pre>
 Pod can be killed and marked as Fail at any time	 Pod can be killed and marked as Fail while using more memory than requested	 As the name says, Pod is guaranteed to run

- **BestEffort Pods** are Pods that don't have any requests and limits set for their containers. Kubernetes kills these Pods first when a node is running out of memory. As the name suggests, these Pods are meant exclusively for running best effort workloads. In other words, it is not a problem if the workload doesn't run right away, if it takes longer to finish, or if it is inadvertently restarted or killed.




Google Cloud

- **Burstable Pods** are Pods with containers that have resource requests with upper, or unbounded (not set), limits. Pods using more memory than they have requested while the node is under resource pressure can also be killed because the Pod's memory can't be compressed. Burstable Pods are not meant to be constantly running above what was requested. Instead, they are meant for workloads that occasionally require additional resources, such as speeding up the Pod's startup time or bursting while HPA is creating a new replica.
- **Guaranteed Pods** are Pods where containers have either an equal amount of request and limit resources or set limits only (Kubernetes automatically copies limits to requests). These Pods are meant for running workloads with strict resource needs. As they cannot burst, these Pods have higher priority and are guaranteed to not be killed before BestEffort and Burstable Pods.

Because deploying BestEffort Pods and memory under-provisioned Burstable Pods can cause disruption to workloads, we recommend the following best practices:

- Avoid using BestEffort Pods for workloads that require a minimum level of reliability.
- Set memory requests equal to memory limits for all containers in all Burstable Pods. You can set upper limits for CPU because Kubernetes can throttle CPU requests whenever needed, though this can impact application performance. Setting upper limits for CPU allows your applications to use idle CPU from nodes without worrying about abrupt workload termination.

Best practice for Burstable workloads

<pre>Burstable apiVersion: v1 ... spec: containers: - name: qos-example image: nginx resources: requests: cpu: 500m memory: 100Mi limits: cpu: 1 memory: 200Mi</pre> <p> Pod can be killed and marked as Fail while using more memory than requested</p>		<pre>Burstable apiVersion: v1 ... spec: containers: - name: qos-example image: nginx resources: requests: cpu: 500m memory: 200Mi limits: cpu: 1 memory: 200Mi</pre> <p> Pod can have CPU throttled to request</p>
---	---	---

Google Cloud

For visibility, both DevOps and platform admins should provide application owners with rightsizing recommendations, while highlighting and tracking the use of workloads that are at reliability risk, such as all BestEffort Pods and memory under-provisioned Burstable Pods. It is important to adopt dashboards, warnings, alerts, and actionable strategies such as automatic issue ticketing or automatic pull requests with actual recommendations. These strategies are even more effective when they are integrated into the developers workstreams, such as into developer IDEs, developer portals, CI/CD pipelines, etc. Such alternatives have been shown to be useful, not only for improving reliability of the overall applications, but also for building a more cost conscious culture.

For guardrails, both DevOps and platform teams can build solutions that enforce the best practices discussed in this section. Enforcement can be done using standard Kubernetes constructs, such as [validation and mutation webhooks](#), or by using policy controller frameworks, such as [Open Policy Agent \(OPA\) Gatekeeper](#). It is also important to have an in-place break glass process to allow teams that understand Kubernetes QoS and its consequences to take advantage of idle cluster resources. For example, when a developer creates a merge request without setting resources, a validation pipeline could either demand a specialized peer review or annotate the merge request with a warning. If the code is merged into the main branch and run in production, the team could enforce an annotation that states the team understands the consequences and they want to bypass organization policy validations.

In the situation where a team has prepared workloads for abrupt termination and they fully understand the consequences of running BestEffort Pods, there is an opportunity to utilize their idle cluster resources and increase savings by running best effort workloads on Spot VMs. Spot VMs are often offered at a discounted price in exchange for cloud providers being allowed to terminate and reclaim resources on short notice. Platform teams can implement this strategy using a mutation webhook to append a [node affinity preference](#) to Spot VMs on all Pods not setting requests. This leaves room on standard nodes for workloads that require greater reliability. Once this strategy is incorporated into an organization's policy or automation pipeline, if no Spot VMs are provisioned, standard VMs are used.



Google Cloud

If the platform team doesn't want to allow best practices to be bypassed, the recommendation is to validate and reject non-compliant workloads. If that is not a possibility, for workloads that are tolerant to graceful restarts, an alternative is to either recommend or enforce the adoption of [Vertical Pod Autoscaler using the Auto mode](#).

Note: When this report was written, VPA required Pods to be restarted to update Pod's resources. However, the [KEP #1287: In-Place Update of Pod Resources](#) became an alpha feature in Kubernetes 1.27. This feature will allow future versions of VPA to, in the majority of the cases, update resource values without restarting a Pod.

Finally, as shown below, you can also set defaults for container resources using the [Kubernetes LimitRange API](#). Because defaults can result in your workload becoming either under- or over-provisioned, this should not replace the recommendation of adopting VPA for rightsizing Pods. The benefit of using defaults for resources is that resources can be applied when the workload is first deployed, while VPA is still in [LowConfidence](#) mode. In this mode, VPA does not update Pod resources due to not having enough data to make a confident decision.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: my-container-resources-defaults
  namespace: my-namespace
spec:
  limits:
  - default: # defines default resources for limits
    memory: 500Mi
  defaultRequest: # defines default resources for requests
    cpu: 250m
    memory: 500Mi
  type: Container
```



What do *High* and *Elite* performers teach us about how to improve cost efficiency?

Cluster owners that adopt a continuous practice to measure and improve [Kubernetes cost optimization golden signals](#) can earn significant cost savings by running lower priced workloads in the public cloud. Therefore, to meet the demand of cost-efficient Kubernetes clusters without compromising reliability and performance, it is important to understand and follow the practices of *High* and *Elite* performers.

Our research shows that both *High* and *Elite* performers run the largest clusters across all segments (*High* performers run clusters 2.3x larger and *Elite* performers run clusters 6.2x larger, in number of nodes, than *Low* performers). Larger clusters allow resources to be better utilized. For example, while a few workloads are idle, other workloads can be bursting. Large clusters also tend to be [multi-tenant clusters](#). The higher the number of tenants, the more operational challenges need to be managed. Such a pattern demands specialized platform teams that need to put in place company cost optimization policies, best practices, and guardrails.

The platform teams from *High* and *Elite* performers also tend to enable more cluster level cost optimization features than *Low* performers, such as Cluster Autoscaling (1.3x and 1.4x, respectively) and cost allocation tooling (3x and 4.6x, respectively). While Cluster Autoscaler allows clusters to automatically add and remove nodes as needed, cost allocation tools enable companies to implement [showback or](#)

[chargeback](#) processes to allocate or even bill the costs associated with each department's or division's usage from multi-tenant Kubernetes clusters.

However, to enable clusters to scale down during off-peak hours as much as *High* and *Elite* performers do (3.2x and 4x more than *Low* performers, respectively), it is also necessary to scale workloads according to demand. The research shows that *High* and *Elite* performers use two main approaches. Firstly, the adoption of workload autoscaling APIs (e.g. HPA and VPA in Auto/Init mode), in which *High* and *Elite* performers demonstrate 1.5x and 2.3x higher adoption than *Low* performers, respectively. Secondly, *High* and *Elite* clusters run 3.6x and 4x more jobs to completion than *Low* performers, respectively. Unfortunately, it was not possible to measure which strategy has the biggest impact, as the segments contain clusters that use a variety of such approaches.

The research has also found that the adoption of workload autoscaling and jobs to completion is much more important for downscaling a cluster than optimizing Cluster Autoscaler for a faster and more aggressive scale down. In other words, if developers don't properly configure their workloads to scale according to demand, platform teams are limited in their ability to scale down their cluster. The research has also found that all segments, including *Elite* performers, could improve demand based autoscaling efficiency by increasing the adoption of HPA and VPA.

Google Cloud

High and *Elite* segments also demonstrate better adherence to workload best practices. For example, developers from *High* and *Elite* performers rightsize their workloads up to 2.2x and 2.8x better than *Low* performers, and deploy 50% and 70% less BestEffort Pods than clusters in the *At Risk* segment, respectively. These excellent results reflect developers' better knowledge of their applications and demonstrate the value of *High* and *Elite* segments specialized platform teams that can build tools and custom solutions. As a result, the *High* and *Elite* segments can continuously enforce policies, provide golden paths, optimize rightsizing (e.g. VPA at recommendation mode deployed 12x and 16.1x more by these segments, respectively, than *Low* performers) and provide best practices recommendations to developer teams.

Lastly, despite [the challenges](#) of building and operating data-centric applications on Kubernetes, the community has seen a rapid increase in deployments of databases and other stateful workloads. *High* and *Elite* performers, who run 1.8x and 2x more StatefulSets than *Low* performers, are the main drivers of this increase.

The [Data on Kubernetes 2022 research](#) shows the benefits of running data-centric applications include:

- ease of scalability
- ability to standardize management of workloads
- consistent environments from development to production
- ease of maintenance
- improved security
- improved use of resources
- co-location of latency sensitive workloads
- ease of deployment



Final thoughts

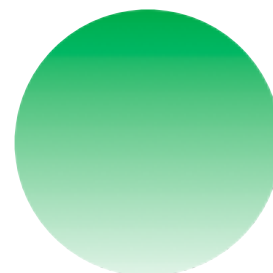
As discussed throughout this report, many Kubernetes features and external tools rely on Pods setting resource requests properly so that organizations can achieve the best reliability, performance, and cost efficiency from the platform. The unintentional over use of Pods that don't set requests or constantly utilize more memory than requested can cause unexpected behavior in clusters and applications, including negatively impacting end user experience. Even though the *At Risk* segment faces these issues at higher frequency, the research has found that there are opportunities for all segments, including *Elite* performers, to improve both reliability and cost efficiency.

It is important to highlight that addressing the opportunities presented by the [Kubernetes cost optimization signals](#) should be continuously prioritized. Moreover, both cluster and application owners should prioritize correctly sizing applications and ensuring applications can automatically scale based on demand, instead of primarily focusing on cluster bin packing and seeking additional discount coverage. This is because significant changes in workload rightsizing and autoscaling may result in re-work required for bin packing and long-term discount strategies.

Even though this research did not analyze clusters across segments within a company, Google has identified a pattern through cost optimization engagements with customers, where larger multi-tenant clusters tend to gather the majority of organizational Kubernetes expertise. This has resulted in many long tail clusters being managed by teams whose main goal is to deliver a service, not manage a platform. This pattern

can also be seen in this report where the largest clusters, while being the hardest to operate, have demonstrated the best cost optimization performance. To overcome the long tail challenges, organizations should invest in defining and enforcing company-wide policies, patterns, and golden pre-approved paths, as well as finding the right balance between control, extensibility, and flexibility.

Finally, platform teams should be aware that measuring cluster bin packing alone doesn't tell the entire story. Looking at the *At Risk* segment, we can see low values for cluster bin packing and high values (sometimes above 100%) for workload rightsizing. This is a reflection of the over use of BestEffort Pods and under-provisioned Burstable Pods, in which the Pods use more cluster resources than they have requested, consequently skewing both signals. If platform teams decide to save money by packing better clusters in such a situation, the risk of running into reliability issues increases accordingly.



Authors



Billy Lamberti

Billy is a Data Scientist for the Cloud Product Analytics Team at Google. He works on projects related to identifying opportunities to make products at Google Cloud more efficient and effective. Before his time at Google, he worked on projects related to satellite imagery, medical and health sciences, image shape analysis, and explainable artificial intelligence. He received his PhD in Computational Sciences and Informatics with a specialization in Data Science.



Fernando Rubbo

Fernando Rubbo is a Cloud Solutions Architect at Google, where he builds global solutions and advises customers on best practices for modernizing and optimizing applications running on Google Cloud. Throughout his career, he has worn many hats, including customer solutions engineer, team lead, product manager, software and platform architect, software and platform consultant, and software developer. Fernando also holds a MS in Computer Science from *UFRGS* University.



Kent Hua

Kent Hua is a Global Solution Manager at Google Cloud, advocating solutions that help organizations modernize applications and accelerate their adoption of cloud technologies on Google Cloud. He is a co-author of *Cloud Native Automation with Google Cloud Build*, a book to help individuals and organizations automate software delivery. His focus on customer success is paramount in his current and previous roles as an enterprise architect, pre-sales engineer and consultant.

Google Cloud



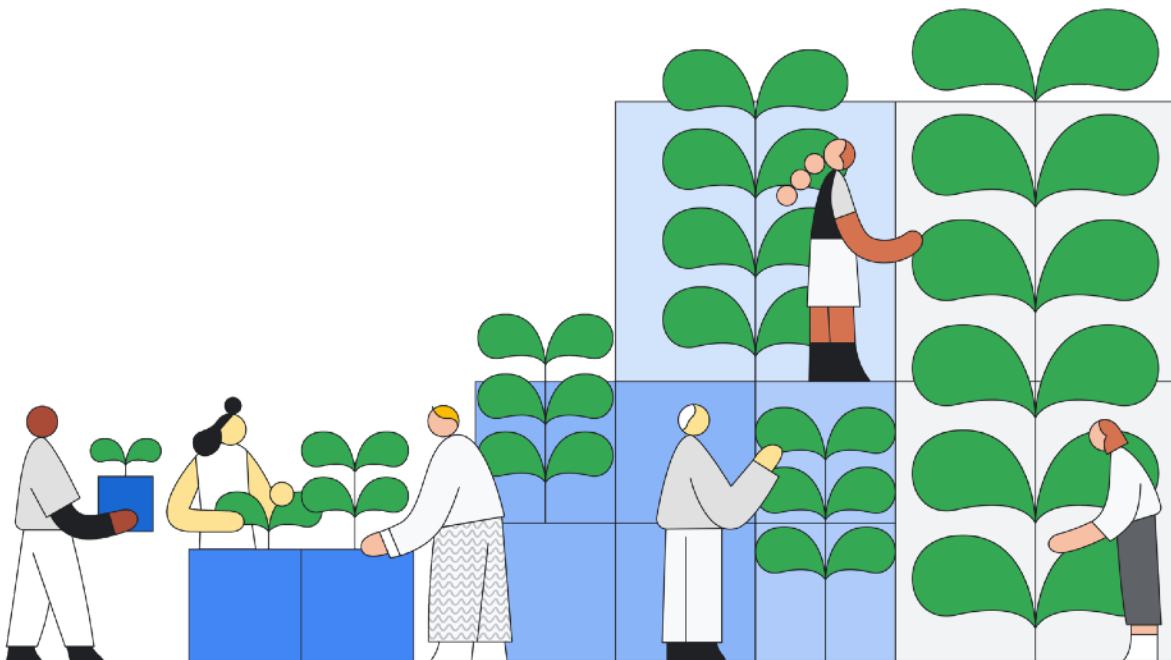
Li Pan

Li Pan is an Operations and Infrastructure Data scientist at Google Cloud, where she leads a team to optimize GCE efficiency, focusing on intelligence to allocate virtual machines into optimal locations, and strategies for new products or new features to existing products to consume unsold capacity. Before Google, she worked in other tech companies as a data scientist. She received her Ph.D. in Mathematics from the University of California, San Diego in 2013.



Melissa Kendall

Melissa Kendall is a technical writer at Google where she focuses on GKE networking, cost optimization, and observability. Outside Google, she is a creative writer, public speaker, and lover of all things Halloween.



Methodology

The State of Kubernetes Cost Optimization report groups anonymized GKE clusters into 5 segments (i.e. *At Risk*, *Low*, *Medium*, *High*, *Elite*) based on [Kubernetes cost optimization golden signals](#). Due to possible privacy concerns, the original dataset and the actual metrics values are not shared. Instead, the report provides the comparison ratio between different segments. For instance, the relative magnitude of *Elite* performers compared to *Low* performers in a given metric. For example, if the *Elite* and *Low* performers have 45 and 10 for a given metric, respectively, the corresponding ratio would be 4.5. This ratio means that the typical *Elite* performers are 4.5 times better than the typical *Low* performers for a given metric.

Each observation represents an anonymized GKE cluster on a given day. Therefore, each unique anonymous GKE cluster can appear more than once. Our notation is as follows:

- Let there K be total unique clusters.
- For a given cluster, k , the number of times the k^{th} cluster occurs (or the number of days the cluster is "alive") is C_k .
- This leads us to, $k \in \{1, 2, \dots, K\}$ and $|k| = C_k$ where $|\cdot|$ is the cardinality operator.

The cardinality operator counts the number of observations that belong to a set. For example, if the number of alive days with a particular cluster from 01/01/2023 to 01/31/2022 was 10, then the cardinality would be 10.

There can be many nodes within a cluster. In this report, we only measured the user nodes, excluding [control plane nodes](#), as described in the following sections.



The Resources group

We assume that some clusters are well utilized, and others that are not. The following three metrics capture such behavior:

- Workload rightsizing
- Demand based downscaling
- Cluster bin packing

Workload rightsizing

The CPU and RAM workload rightsizing is defined for the k^{th} cluster as, respectively:

$$r_{1,k} = \frac{\sum_{k=1}^{C_k} \text{Average Recommended CPU}}{\sum_{k=1}^{C_k} \text{Average CPU Request}},$$

$$r_{2,k} = \frac{\sum_{k=1}^{C_k} \text{Average Recommended RAM}}{\sum_{k=1}^{C_k} \text{Average RAM Request}}.$$

In the previous equations:

- The *Average Recommended CPU/RAM* is the mean of Vertical Pod Autoscaler (VPA) Recommendations over the daily average.
 - Google computes VPA recommendations at scale for every single workload running in any GKE cluster. Instead of using actual Pod utilization, we have chosen to use Pod's VPA recommendations, which is more constant and slightly higher than actual Pod utilization.
- $r_{1,k}$ and $r_{2,k}$ have a support (or a range of possible values) of $[0, \infty)$. However, they are usually between $[0,1)$. There are cases where a user doesn't set a requested CPU or RAM amount for some containers, which may result in a workload rightsizing at the cluster level being greater than 1.

Demand based downscaling

Demand based downscaling is defined for the k^{th} cluster as

$$r_{3,k} = 1 - \frac{\text{Minimum number of nodes}}{\text{Max number of nodes}}.$$

This value has a support of [0,1]. A value close to 1 corresponds to a cluster that is using all of its nodes. A value close to 0 corresponds to a cluster that is not using all of its nodes. We would expect clusters with a "high" capacity of downscaling would have values closer to 1.

Cluster bin packing

CPU and RAM bin packing is defined for the k^{th} cluster as, respectively,

$$r_{4,k} = \frac{\sum_{k=1}^{C_k} \text{Average CPU Request}}{\sum_{k=1}^{C_k} \text{Average CPU Allocatable}},$$

$$r_{5,k} = \frac{\sum_{i=k}^{C_k} \text{Average RAM Request}}{\sum_{k=1}^{C_k} \text{Average RAM Allocatable}}.$$

Note that both "Average Request" and "Average Allocatable" is the daily average. This value has a support of [0,1]. A value close to 0 corresponds to a cluster in which workloads are not requesting the allocated CPUs or RAM, while a value close to 1 corresponds to a cluster in which workloads are requesting all the allocated CPUs or RAM. We would expect clusters with a "high" bin packing to have values closer to 1.

Google Cloud

The Discount group

In the discount group, we only have one metric: discount coverage. This metric defines the percentage of cluster core hours which are covered by cloud discounts (or percentage discounted as a shorthand), is defined for the k^{th} cluster as

$$\begin{aligned} r_{6,k} &= \frac{\text{Spot Core Hours}}{\text{Total Core Hours}} + \frac{\text{CUD Core Hours}}{\text{Total Core Hours}} \\ &= \frac{\text{Spot Core Hours} + \text{CUD Core Hours}}{\text{Total Core Hours}} \\ &= \frac{\text{Discounted Core Hours}}{\text{Total Core Hours}} \end{aligned}$$

Percentage discounted has a support of [0,1]. A value close to 0 or 1 indicates that the cluster is not or is utilizing discounted nodes, respectively. We would expect clusters with a "high" cloud discount to have values closer to 1.

Spot VMs are often offered at a high discounted price in exchange for cloud providers being allowed to terminate and reclaim resources on short notice ([Source](#), [Source 2](#)). Thus, customers should take a balanced approach of using Spot VMs for non-critical workloads that can tolerate disruption.

Continuous use discounts (CUD) correspond to those renewed contracts at a larger discount. CUD are ideal for predictable and steady state usage ([Source](#)).

Thus, $r_{6,i}$ corresponds to the proportion of core hours utilizing Spot or CUD discounts. While Spot and CUD differ in their nature, they are part of all major public cloud providers offered as discount options.



Other Metrics

Other metrics are provided in this report to describe the segments. However, none were used in the creation of the segments. Thus, they are not expanded in great detail.

Research population

To run this research we analyzed all GKE clusters in an anonymized form, except for those with 3 or less nodes. We chose to exclude these clusters as likely testing clusters or clusters with low usage. We chose to make this exclusion because the default GKE cluster creation includes 3 nodes. Data was used from multiple separate months. This was done to confirm that the values were similar across the different months to avoid a seasonality effect. The final reported values utilized throughout the document are from January 2023.

Method

For this research we have used a classification tree technique. Tree models segment the feature space into partitions using rules. If the problem has a continuous outcome, a regression tree is used where the predicted value is the mean value of a given segment. If the output is a series of classes, then a classification tree is used where the prediction is a particular class. For more information about trees, see [An Introduction to Statistical Learning with Applications in R](#).

For this report, we constructed our classes and formulated a classification tree setup. We were able to formulate the problem as a classification tree by using handmade and data driven classes because we have domain expertise. Generally speaking, those observations with many metrics closer to one are considered “Elite Performers” clusters and those with many metrics closer to 0

are considered “Low Performer” clusters. Thus, we made the naive assumption that each metric contributes equally to the consideration of which group it belongs to while also using quantiles to determine the exact cutoff values for each metric.

This process resulted in 5 segments: Low, Medium, High, Elite, and At Risk. The “Low” to “Elite” segments are cohorts who range in their ability to be computationally efficient. For example, the “Elite” segment are able to use their resources in an exceptional manner. As another example, the “Medium” segment might excel in one area, but struggle in another. “At Risk” clusters have the sum of their actual resources utilization is more often than not above the sum of their workloads' resources requested

Steps for analysis

The high-level steps for performing this analysis were as follows:

1. Collect the data.
2. Build the class labels.
 - a. Determine the inputs.
 - b. Determine the number of classes.
3. Build the classification tree.
4. Summarize the segments based on the predictions from the tree.

This process was repeated for each month of interest. For example, these 4 steps were performed for November independent of the 4 steps performed for February.



The logo for Google Cloud, featuring the word "Google" in its signature multi-colored font (blue, red, yellow, green, red) followed by the word "Cloud" in a dark grey, sans-serif font.

Google Cloud