Kubernetes Cost Management: A Complete FinOps Guide

snow* | Cloud Cost
powered by anodot

Around 96% of companies now use or are in the process of evaluating Kubernetes[1]. As the maturity and complexity of Kubernetes environments grow, costs quickly spiral out of control when an effective strategy for visibility and optimization is not in place.

In this guide, you will learn to identify key Kubernetes cost drivers, discover strategies for controlling and optimizing spend, and understand best practices for maximizing the ROI of your Kubernetes deployments.

Managing Kubernetes (K8s) Costs is Critical to Realizing Cloud-Driven Revenue Growth

The COVID-19 pandemic accelerated digital transformation, driving businesses to double down on the cloud to **scale up** services and support "never-seen-before" load and demand (e.g., Zoom), and in some cases, efficiently **scale down** applications in response to changing user patterns (e.g., Uber).

As a result, organizations have scrambled to modernize application development processes and re-architect static, on-premises monoliths as agile, microservice-powered cloud apps, fueling the adoption of containers and container orchestration tools like Kubernetes. All major public cloud providers now offer managed K8s services, and according to CNCF's Annual Survey for 2021, 96% of organizations are already using or evaluating Kubernetes.

The promises of Kubernetes are shorter software development and release cycles, easier application upgrades and maintenance, better utilization of cloud resources, on-demand scale and portability between clouds — all potential drivers of corporate revenue growth.

However, in practice, **Kubernetes has introduced potent risks to revenue growth**, primarily due to the complexity it drives:

1. Lack of internal experience and expertise with K8s architecture and management have forced businesses to invest in training, outside services and expensive consultant engagements.
2. High-profile attacks have heightened concerns about security, driving additional budget and investment against vulnerability testing, hardening and policy enforcement.
3. Engineers and architects, who historically did not have to worry about operational costs, are now on the hook for the financial impact of their code's resource utilization, their node selections and pod/container configurations.

This guide is designed to help your cross-functional Kubernetes value realization team — whether you call it cloud FinOps, your Cloud Center of Excellence, or a simple partnering of DevOps and Finance — come together and remove barriers to maximizing the revenue return on your business' investment in Kubernetes.

[1] Annual Survey 2021 - Cloud Native Computing Foundation. https://www.cncf.io/wp-content/uploads/2022/02/CNCF-AR_FINAL-edits-15.2.21.pdf

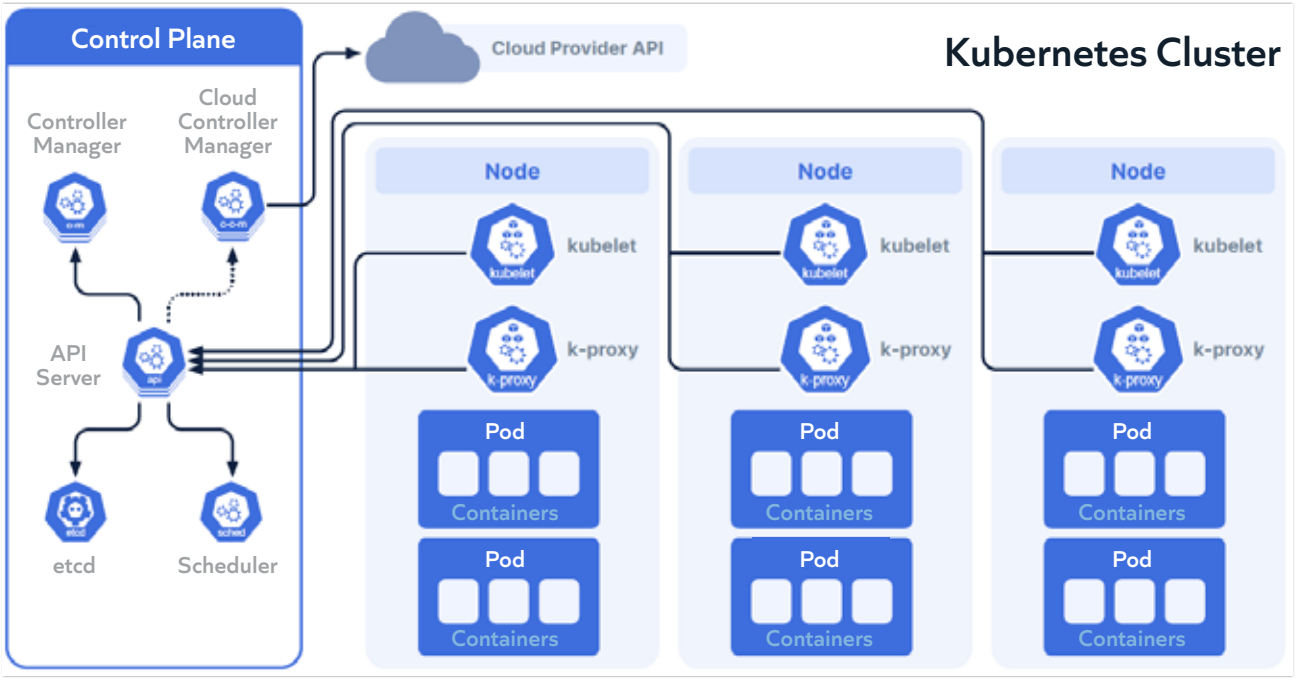


Figure 1. A typical Kubernetes cluster

Understanding Kubernetes Cost Drivers

Optimizing Kubernetes costs isn't an easy task. Kubernetes is as deep a topic as cloud (and even more complex), containing subtopics such as:

- Scheduler and kernel processes
- Resource allocation and monitoring of utilization (at each level of K8s infrastructure architecture)
- Node configuration (vCPU, RAM, and the ratio between those)
- Differences between architectures (like x86 and Arm64)
- Scaling configuration (up and down)
- Associating billable components with business key performance indicators (KPIs)
- and much more

That's a lot for a busy DevOps team to understand and manage. We haven't even mentioned that line-of business stakeholders and Finance team members should have some understanding of each cost driver's function and importance to contribute to a successful FinOps Strategy.

The following sections describe the of the seven major drivers of Kubernetes costs, the importance and function of each, and how each contributes to your cloud bill. These descriptions should be suitable for the consumption of all business stakeholders, and can be used to drive cross-functional understanding of the importance of each cost driver to Kubernetes FinOps.

The Underlying Nodes

Most likely, the cost of the nodes you select will drive a large portion of your Kubernetes costs. A node is the actual server, instance or VM your Kubernetes cluster uses to run your pods and their containers. The resources (compute, memory, etc.) that you make available to each node drive the price you pay when it is running.

For example, in Amazon Web Services (AWS), a set of three c6i.large instances running across three availability zones (AZs) in the U.S. East (Northern Virginia) region can serve as a cluster of nodes. In this case, you will pay $62.05 per node, per month ($0.085 per hour). Selecting larger instance sizes, such as c6i.xlarge, will double your costs to $124.1 per node per month.

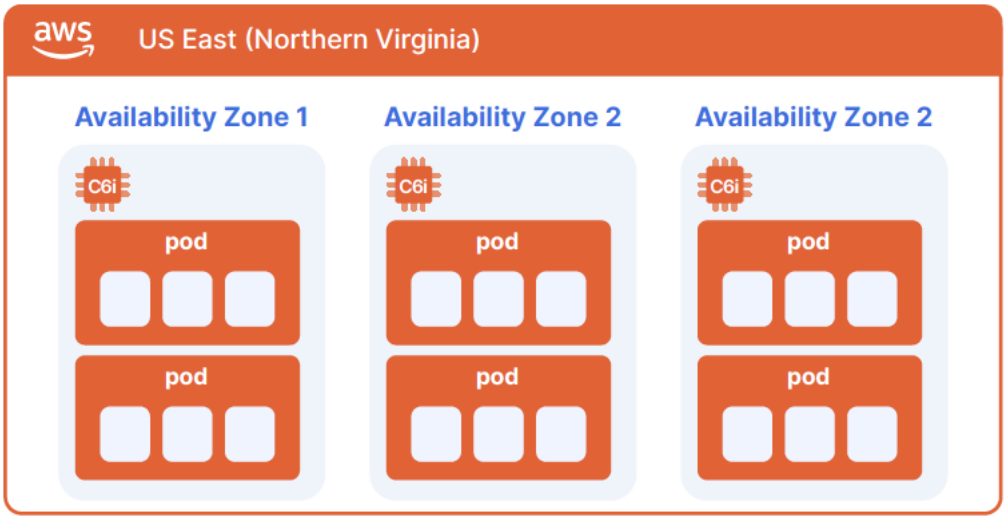


Figure 2. Nodes within a Kubernetes cluster

Parameters that impact a node's price include the operating system (OS), processor vendor (Intel, AMD, or AWS), processor architecture (x86, Arm64), instance generation, CPU and memory capacity and ratio, and the pricing model (On-Demand, Reserved Instances, Savings Plans, or Spot Instances).

You pay for the compute capacity of the node you have purchased even if your pods and their containers do not fully utilize it. Maximizing utilization without negatively impacting workload performance can be quite challenging, and as a result, most organizations find that they are heavily overprovisioned with generally low utilization across their Kubernetes nodes.

Request and Limit Specifications for Pod CPU and Memory Resources

Your pods are not a billable component, but their configurations and resource specifications drive the number of nodes required to run your applications, and the performance of the workloads within.

Assume you are using a c6i.large instance (powered with 2 vCPUs and 4 GiB RAM) as a cluster node, and that 2 GiB of RAM and 0.2 vCPUs are used by the OS, Kubernetes agents, and eviction threshold. In such a case, the remaining 1.8 vCPU and 2 GiB of RAM are available for running your pods. If you request 0.5 GiB of memory per pod, you will be able to run up to four pods on this node. Once a fifth pod is required, a new node will be added to the cluster, adding to your costs. If you request 0.25 GiB of memory per pod, you will be able to run eight pods on each node instance.

Another example of how resource requests impact the number of nodes within a cluster is a case where you specify a container memory limit, but do not specify a memory request. Kubernetes automatically assigns a memory request that matches the limit. Similarly, if you specify a CPU limit, but do not specify a CPU request, Kubernetes will automatically assign a CPU request that matches the limit. As a result, more resources will be assigned to each container than necessarily required, consuming node resources and increasing the number of nodes.

Cluster > us-east-1:619597279328:prod-cluster > ip-172-19-134-19.ec2.internal
> monitoring:prometheus-prometheus-prometheus-oper-prometheus-0

Unblended Usage

Pod Name: monitoring:prometheus-prometheus-prometheus-oper-prometheus-0
Date: 2022-02-06
Instance Type: r5.8xlarge
Resource Id: i-074e7849f0fa22034
Total Pod Cost: $20.620
Total Running Hours: 23
Export as CSV

	Requirements	Usage	Usage Waste	Usage Cost	Usage Waste C...	Total Cost
	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
cpu	0.200 Cores	0.644 Cores	0 Cores	$0.470	$0	$0.470
memory	100.049 GB	207 GB	0 Bytes	$19.280	$0	$19.280
network	0 Bytes	594.656 MB	0 Bytes	$0	$0	$0
datatransfer	0 Bytes	12.620 GB	0 Bytes	$0.130	$0	$0.130
storage	0 Bytes	9.290 GB	0 Bytes	$0.740	$0	$0.740

Figure 3. A detailed view of usage and costs for Kubernetes pods

In practice, many request and limit values are not properly configured, are set to defaults, or are even totally unspecified, resulting in significant costs for organizations.

Persistent Volumes

Kubernetes volumes are directories (possibly containing data), which are accessible to the containers within a pod, providing a mechanism to connect ephemeral containers with persistent external data stores. You can configure volumes as **ephemeral or persistent**. Unlike ephemeral volumes, which are destroyed when a pod ceases to exist, the shutdown of pods does not affect persistent volumes. Both ephemeral and persistent volumes are preserved across individual container restarts.

Volumes are a billable component (similar to nodes). Each volume attached to a pod has costs that are driven by the size (in GB) and the type of the storage volume attached — solid-state drive (SSD) or hard disk drive (HDD). For example, a 200 GB gp3 AWS EBS SSD volume will cost $16 per month.

Affinity and The K8s Scheduler

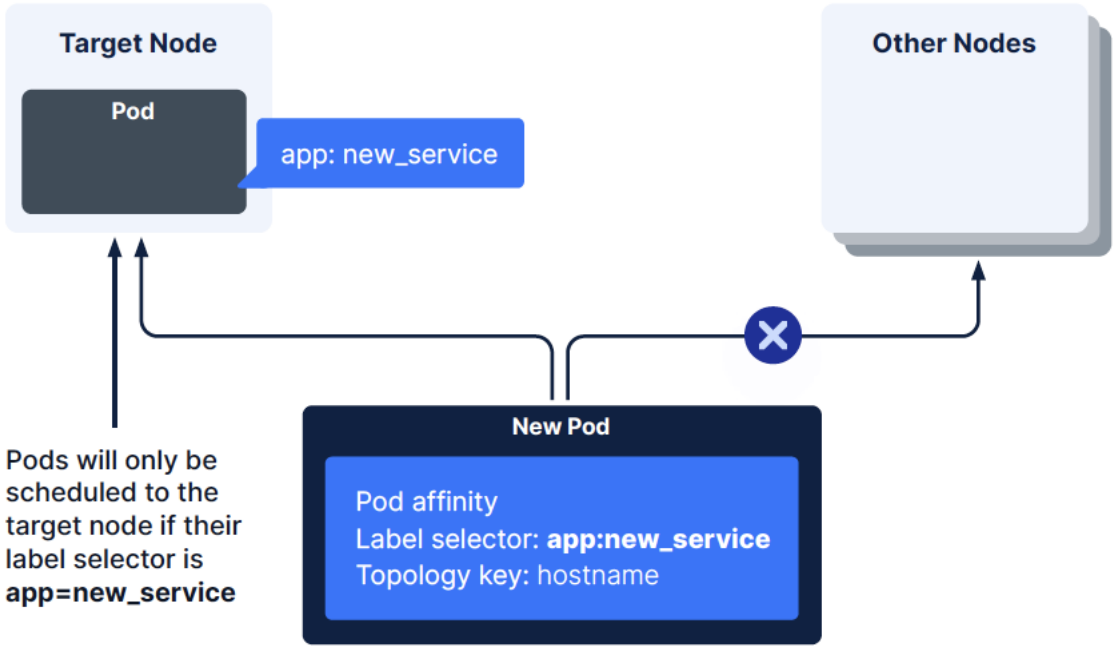
The Kubernetes scheduler is not a billable component, but it is the primary authority for how pods are placed on each node, and as a result, has a great impact on the number of nodes needed to run your pods.

Within Kubernetes, you can define node and pod affinity (and pod anti-affinity), which constrains where pods can be placed. You can define affinities to precisely control pod placement, for use cases such as:

- Dictating the maximum number of pods per node
- Controlling which pods can be placed on nodes within a specific availability zone or on a particular instance type
- Defining which types of pods can be placed together
- Powering countless other scenarios

Such rules impact the number of nodes attached to your cluster, and as a result, impact your Kubernetes costs.

Figure 4. An Example of Kubernetes Pod Scheduling Using Pod Affinity



Note: Pods will only be scheduled to nodes to which they have affinity.

Consider a scenario where an affinity is set to limit pods to one per node and you suddenly need to scale to ten pods. Such a rule would force-increase the number of nodes to ten, even if all ten pods could performantly run within a single node.

Data Transfer Costs

Your Kubernetes clusters are deployed across availability zones (AZs) and regions to strengthen application resiliency for disaster recovery (DR) purposes, however **data transfer costs are incurred anytime pods deployed across availability zones communicate** in the following ways:

- When pods communicate with each other across AZs
- When pods communicate with the control plane
- When pods communicate with load balancers, in addition to regular load balancer charges
- When pods communicate with external services, such as databases
- When data is replicated across regions to support disaster recovery

Network Costs

When running on cloud infrastructure, **the number of IP addresses that can be attached to an instance or a VM is driven by the size of the instance.**

For example, an AWS c6i.large instance can be associated with up to three network interfaces, each with up to ten private IPv4 addresses (for a total of 30). A c6i.xlarge instance can be associated with up to four network interfaces, each with up to 15 private IPv4 addresses (for a total of 60).

Now, imagine using a c6i.large instance as your cluster node when you require over 30 private IPv4 addresses. In such cases, many Kubernetes admins will pick the c6i.xlarge instance to gain the additional IP addresses, but it will cost them double, and the node's CPU and memory resources will likely go underutilized.

Application Architecture

Applications are another example of non-billable drivers that have a major impact on your realized Kubernetes costs. **Often, engineering and DevOps teams will not thoroughly model and tune the resource usage of their applications.** In these cases, developers may specify the amount of resources needed to run each container, but pay less attention to optimizations that can take place at the code and application level to improve performance and reduce resource requirements.

Examples of application-level optimizations include using multithreading versus single-threading or vice versa, upgrading to newer, more efficient versions of Java, selecting the right OS (Windows, which requires licenses, versus Linux), and building containers to take advantage of multiprocessor architectures like x86 and Arm64.

Kubernetes Cost Optimization Strategies

Having examined the main drivers of Kubernetes costs and the challenges surrounding them, here are recommendations and best practices for running cost-optimized Kubernetes workloads on AWS, Microsoft Azure, and Google Cloud (GCP).

Gaining Complete Kubernetes Cost Visibility

Gaining visibility into your container cost and usage data is the first step to controlling and optimizing Kubernetes costs. **Visibility is critical at each level of your Kubernetes deployment:**

Clusters

Nodes

Pods (Namespaces, Labels, and Deployments)

Containers

You will also want visibility within each business transaction. Having deep visibility will help you:

1. Avoid cloud "bill shock" (a common compelling incident where stakeholders find out after-the-fact that they have overspent their cloud budget)
2. Detect anomalies
3. Identify ways to further optimize your Kubernetes costs

For example, when using Kubernetes for development purposes, visibility helps you identify Dev clusters running during off-business hours so you can pause them. In a production environment, visibility helps you identify cost spikes originating from a deployment of a new release, see the overall costs of an application and identify cost per customer or line of business.

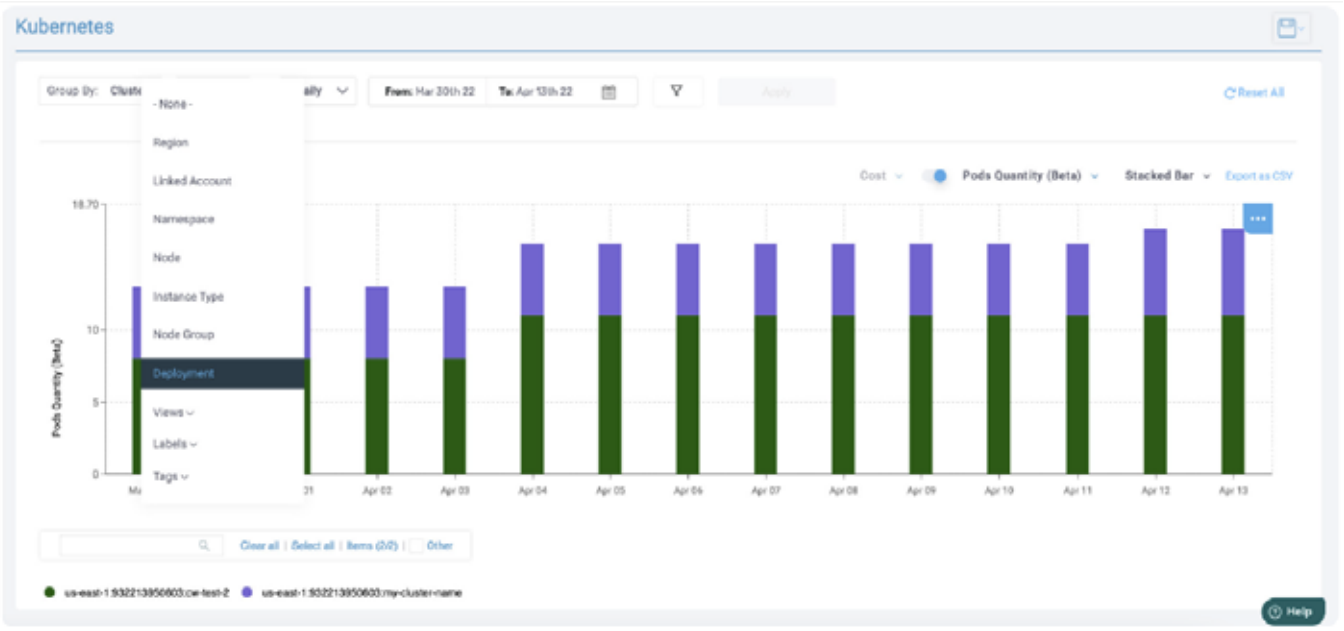


Figure 5. Visibility into a Kubernetes deployment showing a breakdown of usage and cost

Detecting Kubernetes Cost Anomalies

"Bill shock" is too common an occurrence for businesses that have invested in Kubernetes. Anomaly detection intelligence will continuously monitor your usage and cost data and automatically and immediately alert relevant stakeholders on your team so they can take corrective action.

Anomalies can occur due to a wide variety of factors and in many situations.
Common anomaly causes include:

- A new deployment consuming more resources than a previous one
- A new pod being added to your cluster
- Suboptimal scaling rules causing inefficient scale-up
- Miscon d (or not configured) pod resource request specifications (for example, specifying GiB instead of MiB)
- Affinity rules causing unneeded nodes to be added

Save your team the pain of end-of-month invoice shock. Any organization running Kubernetes clusters should **have mechanisms for K8s anomaly detection and anomaly alerting in place** — full stop.

Optimizing Pod Resource Requests

Have organizational policies in place for setting pod CPU and memory requests and limits in your YAML definition files. Once your containers are running, you gain visibility into the utilization and costs of each portion of your cluster: namespaces, labels, nodes, and pods. This is the time to **tune your resource request and limit values based on actual utilization metrics.**

Kubernetes allows you to fine-tune resource requests with granularity up to the MiB (RAM) and a fraction of a CPU, so there is no reason to overprovision and end up with low utilization of the allocated resources.

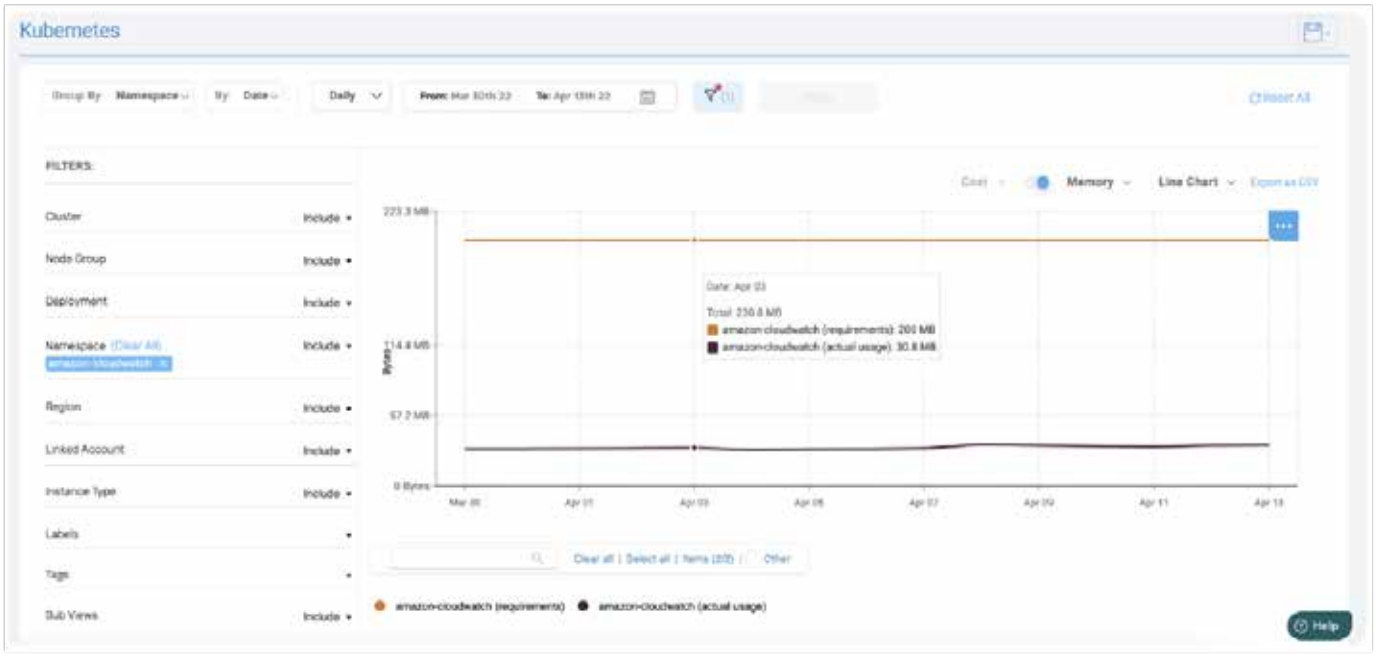


Figure 6. Namespace resource requests versus actual utilization

Node Configuration

Various factors drive node cost. You can address many of them at the configuration level. These include the CPU and memory resources powering each node, OS choice, processor type and vendor, disk space and type, network cards, and more.

When configuring your nodes:

- Use open-source OSes to avoid costly licenses like those required for Windows®, RHEL, and SUSE
- Favor cost-effective processors to benefit from the best price-performance processor option:
    - On AWS, use Graviton-powered instances (Arm64 processor architecture)
    - In GCP, favor Tau instances powered by the latest AMD EPYC processors
- Pick nodes that best fit your pods' needs. This includes picking nodes with the right amount of vCPU and memory resources, and a ratio of the two that best fits your pod's requirements.
    - For example, if your containers require resources with a vCPU to memory ratio of 8, you should favor nodes with such a ratio, like:
        - AWS R instances
        - Azure® Edv5 VMs
        - GCP n2d-highmem-2 machine types
    - In such a case, you will have specific node options per pod with the vCPU and memory ratio needed.

Processor Selection

For many years, all three leading cloud vendors offered only Intel-powered compute resources. Recently, though, **all three cloud providers have enabled various levels of processor choice, each with meaningful cost impacts.** We have benefited from the entry of AMD-powered (AWS, Azure, and GCP) and Arm® architecture Graviton-powered instances (AWS).

These new processors introduce ways to gain better performance while reducing costs. In the AWS case, AMD-powered instances cost 10% less than Intel-powered instances, and Graviton instances cost 20% less than Intel-powered instances. To run on Graviton instances, you should build multi-architecture containers that comply with running on Intel, AMD, and Graviton instance types. You will be able to take advantage of reduced instance prices while also empowering your application with better performance.

Purchasing Options

Take advantage of cloud provider purchasing options. All three leading cloud providers (AWS, GCP, Azure) offer multiple purchasing strategies, such as:

- On-Demand: Basic, list pricing
- Commitment-Based: Savings plans (SPs), reserved instances (RIs), and commitment use discounts (CUDs), which deliver discounts for pre-purchasing capacity
- Spot: Spare cloud service provider (CSP) capacity (when it is available) that offers up to a 90% discount over On-Demand pricing

Define your purchasing strategy choice per node, and prioritize using Spot instances when possible to leverage the steep discount this purchasing option provides. If for any reason Spot isn't a fit for your workload — for example, in the case that your container runs a database — purchase the steady availability of a node that comes with commitment-based pricing. In any case, you should strive to minimize the use of On-Demand resources that aren't covered by commitments.

Autoscaling Rules

Set up scaling rules using a combination of horizontal pod autoscaling (HPA), vertical pod autoscaling (VPA), the cluster autoscaler (CA), and cloud provider tools such as the Cluster Autoscaler on AWS or Karpenter to meet changes in demand for applications.

Scaling rules can be set per metric, and you should regularly fine-tune these rules to ensure they fit your application's real-life scaling needs and patterns.

Kubernetes Scheduler (Kube-Scheduler) Configuration

Use scheduler rules wisely to achieve high utilization of node resources and avoid node overprovisioning. As described earlier, these rules impact how pods are deployed.

In cases such as where affinity rules are set, the number of nodes may scale up quickly (e.g., setting a rule for having one pod per node).

Overprovisioning can also occur when you forget to specify the requested resources (CPU or memory) and instead, only specify the limits. In such a case, the scheduler will seek nodes with resource availability to fit the pod's limits. Once the pod is deployed, it will gain access to resources up to the limit, causing node resources to be fully allocated quickly, and causing additional, unneeded nodes to be spun up.

Managing Unattached Persistent Storage

Persistent storage volumes have an independent lifecycle from your pods, and will remain running even if the pods and containers to which they are attached cease to exist. **Set a mechanism to identify unattached EBS volumes and delete them** after a specific period has elapsed.

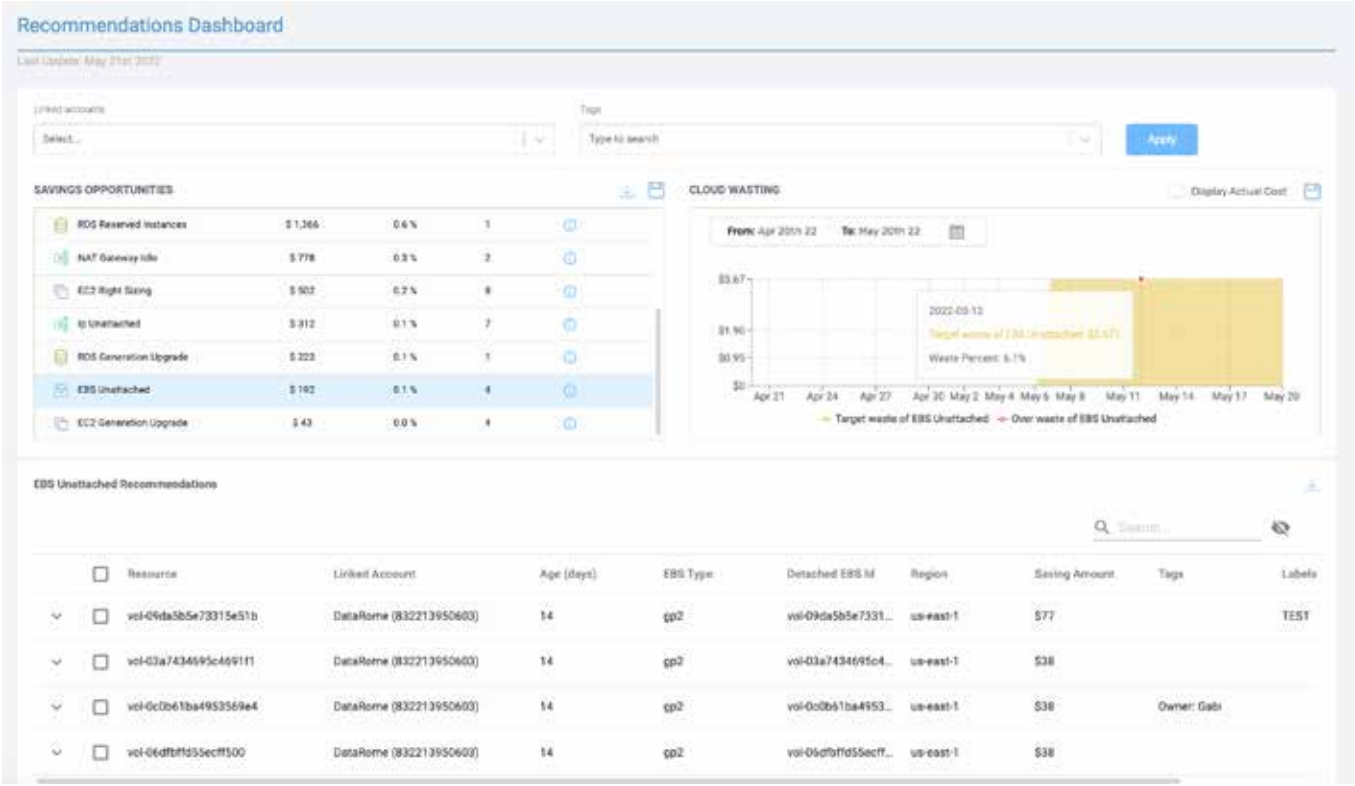


Figure 7. Unattached persistent storage

Optimizing Network Usage to Minimize Data Transfer Charges

Consider designing your network topology so that it will account for the communication needs of pods across AZs and can avoid added data transfer fees. Data transfer charges may also happen when pods communicate across AZs with each other, the control plan, load balancers and other services.

Another approach for minimizing data transfer costs is to deploy namespaces per availability zone (one per AZ), to get a set of single AZ namespace deployments. With such an architecture, pod communication remains within each availability zone, preventing data transfer costs while allowing you to maintain application resiliency with a cross-AZ, high-availability setup.

Minimizing Cluster Counts

When running Kubernetes clusters on public cloud infrastructure such as AWS, Azure, or GCP, you should be aware that **you are charged per cluster**.

In AWS, you are charged $73 per month per cluster you run with Amazon Elastic Kubernetes Service (EKS). Consider minimizing the number of discreet clusters in your deployment to eliminate this additional cost.

Best Practices for Maximizing Your Kubernetes ROI

INFORM: Empower Kubernetes stakeholders with visibility relevant to their role

Stakeholders in managing your Kubernetes deployment costs extend far beyond your end users. Typical K8s cost stakeholder parties include:

- Application end users
- Business unit leaders
- App users within each line of business
- Your application engineering team
- Your DevOps team and practitioners
- Kubernetes admins, engineers and architects
- Your Finance or IT Finance team
- Any formalized FinOps organization with your business or Cloud Center of Excellence

Delivering transparency and a single-source-of-truth system for Kubernetes usage data is table stakes for each of these personas, and is required to align business, operations, and DevOps teams. Dashboard, reports, and alerts are all common methodologies of providing visibility, and leading tools will enable customization of views per persona so that each user sees only the data that impacts their role.

Specific visibility requirements will vary per persona and per team. Typical requirements include varying levels of granular visibility (from your clusters to their containers) and analytics across all your public clouds, including non-container resources and workloads. From a reporting and dashboards perspective, users demand instant data on current K8s cost trends and forecasted costs.

Sophisticated multicloud cost management platforms like Snow Cloud Cost powered by Anodot enable the per-role visibility business stakeholders need by:

- Visualizing and tracking Kubernetes spending and usage across clusters, namespaces, nodes, and pods
- Correlating cloud spending with business KPIs
- Enabling the deepest visibility, analysis, and breakdowns for the costs of non-K8s and Kubernetes cloud components as individual and shared costs, by cost center, and by other levels of categorization and virtual tagging
- Enabling you to unify Kubernetes label keys and traditional resource tag keys to build a combined allocation model

OPTIMIZE: Leverage intelligent recommendations to continuously optimize Kubernetes costs and usage

After enabling appropriate visibility across all your stakeholders, you and your FinOps team can finally **take on the task of optimizing and reducing Kubernetes spending.** With comprehensive K8s visibility, you can fine-tune Kubernetes resource allocation — allocating the exact amount of resources required per cluster, namespace/label, node, pod and container.

Monitoring and configuring your Kubernetes deployments properly will improve infrastructure utilization, reduce instances of overprovisioning, and reduce application infrastructure costs.

Actually implementing continuous optimization procedures proves challenging for many organizations, even with enough visibility. Prioritizing optimizations is a challenge, and in many organizations, getting the engineering buy-in and cycles to actually implement the infrastructure changes that have been identified as cost-saving measures is difficult (as evidenced by multiple FinOps Foundation studies that have identified "Getting Engineers to Take Action" as the recurring primary priority of FinOps teams).

Snow Cloud Cost powered by Anodot provides a shared source of cost visibility and cost optimization recommendations, making continuous improvement a scalable task for multi-stakeholder teams by:

- Making next-step actions to implement optimizations blatantly evident (with explicit management console instructions or CLI commands)
- Specifically outlining the cost impact of each optimization change
- Helping your team to identify anomalies and underutilization at the node and pod level in an ongoing way

OPERATE: Formalize accountability and allocation for Kubernetes costs

As a FinOps strategy leader, **you must gain consensus and instill proper financial control structures for Kubernetes within your organization.** FinOps strategies without accountability and alignment are doomed to failure. Financial governance controls further reduce the risk of overspending and improve predictability.

This operating phase is where the rubber meets the road as far as the results you will gain from your Kubernetes FinOps efforts. If you have put the right controls in place and have an effective formalized cost management process, your team will be enabled to:

- Effectively and fully transition from the slow, on-premises CapEx model to the elastic, real-time OpEx model enabled by the cloud
- Move from the old-world paradigm of Engineering as requestors/Finance as approvers to Engineering and Finance acting as one
- Fully replace predictable, static hardware spend (with long procurement processes) with predictable budgets for on-demand (instant procurement) container resources

All of which helps your organization transition from the antiquated physical infrastructure world with high cost of failure to a paradigm that enables affordable "fast failing" and agile experimentation.

But, how do you ensure formalized accountability practices and procedures are in place? We have established that cost efficiency is a shared responsibility, with the FinOps team in charge of standards. Your FinOps stakeholders must stand up the proper guidelines, cost monitoring, alerting, and optimization processes. Within these constructs, Engineering is tasked with making sure their investments are cost-minded and efficient.

There are additional specific actions you can take to enforce and enhance accountability and cost allocation practices, through:

- Organizing resources by application and, when possible, using dedicated clusters for each app
- Flexibly and strategically defining and assigning namespaces and labels to align usage with cost centers (application, team, or business unit), and unify this approach with traditional resource tagging so you can allocate costs, analyze by cost centers, and perform full allocation across K8s and non-Kubernetes workloads
- Making sure that the teams that are driving costs (in DevOps/Engineering) have cost and usage information at hand, in addition to providing these same details to your product, project, and system owners and managers
- Delivering visibility into which committed-use strategies are in place; this can help incentivize Engineers to leverage Savings-Plan-ready instances over incumbent choices
- Regularly hosting review sessions with stakeholders to review high-level dashboards and socialize the cost impact of optimizations

Have a solid and comprehensive Kubernetes **showback** model in place, and leverage the aforementioned visibility and reporting capabilities (like those enabled by Snow Cloud Cost powered by Anodot) to help your teams understand how they are doing in terms of costs.

Chargeback approaches (where stakeholders are directly invoiced for their cloud spend impact) are appropriate for teams that have required visibility and education, but avoid creating a culture of Kubernetes cost **shameback** — which emphasizes inefficiencies and weaknesses rather than building communication, mentorship, and shared education efforts that enable cross-organizational wins.

Above all, create a fluid flow of communication about what efforts are being made, and what savings results are being achieved. Loudly champion any and all wins and successes.

Cloud and Kubernetes cost management tools like Snow Cloud Cost powered by Anodot help automate and centralize much of this work:

- Automated alerting and reporting can appear within the tools and interfaces your teams already use to show them usage and savings impact without forcing them to regularly open and consult another solution
- Calculate Kubernetes unit costs and answer the question, "for each dollar spent in K8s, how many dollars of revenue did we generate?"
- Help Engineers to take ownership of the cost impact of their choices by showing the results of cost-conscious resource provisioning and utilization

Building Your Strategy for Operationally Maximizing K8s ROI

A successful financial management strategy for Kubernetes infrastructures in the public cloud — whether on AWS, Azure, or GCP — requires educating and uniting stakeholders from parties as diverse as Finance and DevOps around shared goals and processes.

STEP 1: Understand Kubernetes cost drivers

First, **stakeholders from each line of business that consumes Kubernetes services** and the FinOps governing team must develop at least a basic awareness and understanding of each K8s cost driver's function and importance (both direct and indirect).

STEP 2: Evaluate and select strategies for controlling and optimizing costs

Next, these same stakeholders can evaluate different strategies for controlling and optimizing costs against each cost driver and identify those that make sense in accordance with the business' specific focus and goals and objectives.

At this time, it also makes sense to evaluate the Snow Cloud Cost powered by Anodot management tool that provides comprehensive, cross-cloud (multicloud) and cross-technology (AWS, Azure, GCP + Kubernetes) visibility, optimization, and forecasting capabilities. **Snow Cloud Cost powered by Anodot is often selected at this stage by organizations that are focused specifically on financial management of cloud and Kubernetes**, and who prefer to have a single, focused tool that drives cloud and K8s ROI.

STEP 3: Implement a continous Kubernetes optimization practice

Finally, a FinOps plan for operationalizing the selected strategies in an ongoing manner can be created by leveraging the Inform > Optimize > Operate cyclical framework.

Result: Incremental K8s ROI Improvement

The results will be cross-organizational visibility and communication, the continuous improvement of your Kubernetes deployments and the shared success of all stakeholders that manage and benefit from K8s.

Looking for ways to enhance operational efficiency?

Request a Demo

Contact Snow

www.snowsoftware.com

info@snowsoftware.com

Follow Snow

About Snow Software

Snow Software is changing the way organizations understand and manage their technology consumption. Our Technology Intelligence Platform provides comprehensive visibility and contextual insight across software, SaaS, hardware and cloud. With Snow, IT leaders can effectively optimize resources, enhance performance and enable operational agility in a hybrid world.

(Published October, 20th 2022 | Snow_Kubernetes_Co7t_Management_FinOps_Guide_v7)

snow* | Cloud Cost powered by anodot