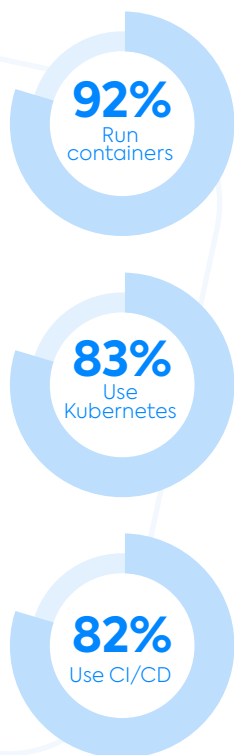


White Paper

---

# Making Kubernetes Day 2 operations scalable, effortless & cost-effective



## Ecosystem evolution

Since its first release in 2014, Kubernetes has quickly evolved and matured to be the defacto way to run applications in the cloud. Today, more and more companies are using Kubernetes in production, and applying the container orchestrator to many different use cases.

## Introduction

Kubernetes makes it easier to run and deploy applications in the cloud and brings many benefits like speed, agility, automation, and cost efficiencies. However, in order for companies to unlock the strategic advantages of using Kubernetes, they must first solve for the operational hurdles of managing, maintaining and scaling these applications and the environments they are built, tested and deployed in.

There has been significant maturity and evolution in the ecosystem since Kubernetes was first introduced. For example, the [CNCF survey 2020](#) shows that 92% of respondents run containers in production, 83% use Kubernetes in production, and 82% use CI/CD pipelines in production.

Much of the focus of these early years is just getting Kubernetes clusters up and running. Now, as more companies deploy applications into production with Kubernetes, it is critical to think about the long-term operational impact of maintaining and scaling Kubernetes clusters. The CNCF survey also shows that 30% of respondents use serverless in production, and 27% leverage service meshes. Adoption of these newer solutions is growing fast.

Well-oiled clusters take more than just standing up Kubernetes to move fast. For sustained success, and to reap the benefits of the cloud in the long-term, users need to find ways to effectively and efficiently manage Day 2 operations.

# The challenges of Day 2 Kubernetes operations



Day 0  
**Design**

Day 1  
**Deploy**

Day 2  
**Scale**

Once you install and set up Kubernetes and start running workloads, the next milestone is to keep it highly available, performant, secure, and observable. These are key Day 2 challenges that DevOps needs to solve for, and much of it comes down to infrastructure management and operations.

While Kubernetes handles the application scaling and deployment, it doesn't handle the underlying cloud infrastructure, nor does Kubernetes and cloud infrastructure speak the same language. This gap is a core challenge to building properly functioning Kubernetes environments that can scale well beyond Day 2. Ultimately, it's up to the user to figure out the answers to these questions, among many others:

- How do I make sure to choose the right infrastructure size for my applications?
- How do I ensure application deployment isn't delayed because of infrastructure?
- How will I know when something goes wrong, and how will I fix it?
- How do I ensure reliability of my systems but keep it cost-efficient?

**Here are key concerns that organizations are looking to address with Kubernetes:**

- **High availability:** The ability to service SLOs, and ensure minimal to no downtime during upgrades. This also involves fast recovery during inevitable failures.
- **Developer enablement:** This is the task of the DevOps (platform) team to provision environments for Dev, QA, and Prod in a quick, self-service manner. Also, configuring a CI/CD pipeline that is flexible and automated.
- **Resource utilization:** DevOps teams need to ensure there is enough capacity for peak times, and only the required amount of instances are being utilized at any given time. While workloads cannot be compromised, the way they are run should be cost effective. This involves choosing cheaper instances, and handling termination of these instances gracefully.



## What's the difference between instance types?

### On-demand instances

Scales up and down for variable demand, and has expensive per-unit costs.

### Reserved capacity

Committed future use that is more affordable per unit but less flexible.

### Spot instances

Discounted spare capacity that can be taken away at any moment.

All these are hallmarks of managing traditional data centers. However, with Kubernetes there are many more moving pieces involved, many of which are new and changing by the day (Eg., service mesh tooling). All this makes Day 2 operations a real challenge for any organization adopting Kubernetes.

## Day 2 management of infrastructure & workloads

Workloads describe themselves in ways that are completely different from how infrastructure describes itself. This creates a mismatch where two major parts of the stack are speaking two different languages and is part of the challenge that needs to be solved when building a properly functioning, well-oiled Kubernetes environment that can scale beyond Day 2 and in the future.




When we think about Day 2 operations, one thing that is certain is that needs change on a day-to-day basis. This is a lot of change happening from an infrastructure perspective. Infrastructure is most commonly described in terms of size, families, and types of pricing—there are small, medium, large, and extra large sized instances. They are available on demand, as reserved instances, or as excess capacity. For example, on day one you may run small workloads, the next you may need medium-sized workloads, and the following day you may have workloads that require on demand nodes.

Then there are specific requirements around how workloads express themselves. In Kubernetes, workloads speak in terms of CPU/Memory/GPU resource requests and limits, Labels, Taints, Tolerations, network & storage requirements, and affinities. In one case, a workload could have a zone affinity where it must only be run in a specific availability zone. In another case, there may be pod affinity where a workload must be run alongside another pod, or it may have an anti-affinity and cannot be run if there is another specific pod running on the same node.

### Compute capacity

-  Instance size
-  Instance family
-  Instance type

### Container requests and consumption

-  vCPU
-  Memory
-  GPU

“

Autoscaling cloud infrastructure is a black box for us.



To bridge the gap between infrastructure and workloads organizations need to find ways to abstract infrastructure. They need to spend less time in infrastructure plumbing, and more time running workloads. However, this is easier said than done. Many take the DIY route to manage their infrastructure, which is the harder way to do this.

## The DIY Kubernetes data plane management stack

Many companies have taken a DIY approach using open source tools to manage container infrastructure. This forces DevOps teams to learn and operate various tools and processes necessary for building healthy Kubernetes environments. Instead of focusing on applications, DevOps teams are figuring out how to:

- Scale infrastructure up and down automatically
- Manage node pools
- Maintain enough capacity for scaling bursts
- Use excess capacity reliably for cost savings
- Handle interruptions

### Cluster-autoscaler – Scaling nodes up or down automatically

As mentioned earlier, Kubernetes does not handle infrastructure, it merely knows the number of healthy nodes that are registered to it at any given time. Kubernetes uses the registered and healthy nodes in the cluster and allocates workloads to them. Once all available nodes are fully allocated, Kubernetes will wait for additional infrastructure and will keep the additional workloads pending until the new infrastructure is present in the cluster. This makes it the user's responsibility to ensure that Kubernetes has enough infrastructure (by scaling the cluster in or out) to accommodate all of the relevant workloads.

**Cluster Autoscaler automatically adjusts the size of a Kubernetes cluster when...<sup>1</sup>**

- Pods are pending due to insufficient resources
- A node is underutilized and its pods can be placed on an existing node

<sup>1</sup>) <https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler>



## Notes about node pools

- Same node pool, same size nodes
- Changes in one node pool impact other regions and zones in the same cluster
- Independently managed by the user

This scaling up and down of nodes can be done using cluster-autoscaler. In order to manage cluster-autoscaler, software delivery teams need to understand its various operational aspects, handle failures, and manage overprovisioning.

[Cluster-autoscaler has its limitations as well](#). If users want to utilize different kinds of compute, they'll need to manage multiple node pools independently. Using excess capacity for cost savings with cluster-autoscaler introduces performance and availability risks that have to be managed as well. All this responsibility falls on the shoulders of the software delivery team as an exercise to do as none of these tasks are automated.

## Node pools – Dealing with differences across multi cluster environments

As cloud applications and services grow, with clusters of many microservices running across regions and availability zones, it is essential to manage the nodes in clusters as groups, and not individually. This way operations can be scaled without adding complexity, and it puts more control in the hands of the software delivery team. This requires an understanding of how node pools work.

Node pools are a group of nodes in a cluster that all have the same configuration. The cluster-autoscaler can manage numerous node pools. Different node pools serve different purposes. Each node pool would have different instance types, sizes, and families to cater to changing needs after a cluster is created. Node pools on AWS, Azure, and GCP behave differently from each other because of the unique characteristics of each cloud. Apart from this each application has different needs, and each team has specific needs such as air-gapped environments for stronger isolation.

There are limitations with node pools that need to be considered. Within a node pool all nodes need to be of the same size. Additionally, if a cluster spans multiple availability zones or regions, changes in one node pool are replicated across all zones and regions. This can consume a lot of resources. Similarly, when a node pool is deleted, it is deleted across all regions and zones. This can be restrictive.



Using  
spot instances  
offer up to  
90% cost savings  
**but...**

- No SLA from cloud providers
- Termination with no more than a 30-second warning
- Capacity availability varies based on region, size and time of day

## Node termination handler – Anticipating & planning for interruptions to workloads

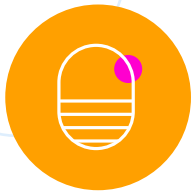
As applications grow and consume more resources, cloud cost savings and optimization becomes a strategic organizational goal. The way to handle this in the cloud is with the use of excess capacity (such as AWS' Spot instances) to gain access to resources at a much cheaper rate. However, the challenge with this is to balance low cost with the sudden termination of these excess capacity instances.

Excess capacity is a powerful way for organizations to scale large workloads cost effectively. Excess capacity instances are unused instances that AWS (or a similar cloud vendor) offers, which can be used for a huge cost savings of up to 90%. The caveat is that these instances can be reclaimed by the cloud provider at any time with a short notice (up to 2 minutes in AWS, and as little as 30 seconds in Azure and GCP). When this happens, the workloads on the excess capacity instance needs to be drained, or moved to another instance for uninterrupted processing.

These types of interruptions are inevitable with excess capacity instances, but when they happen the software delivery team needs to be aware of the interruption, and have a plan for the workloads running on a particular node. This is the job of the node termination handler. It listens for interruptions, and based on interruptions, it transitions workloads from existing nodes to new nodes so that the workloads can continue execution and serve their purpose.

The software delivery team has the responsibility to install node termination handler, manage updates to it, deal with bugs as they arise, scale, upgrade, or downgrade as needed. This added overhead is a drain on productivity and makes routine maintenance inefficient





## Headroom

A buffer of capacity can reduce the risk of pending pods when traffic spikes, ensuring that workloads have enough resources to run without waiting for new instances to spin up.

## Cluster overprovisioner – Schedule low priority pods in a simplistic way

While Kubernetes allows for autoscaling of pods and resources, there are times when a cluster has maxed out all the underlying instances provisioned for it. When this happens a new instance needs to be provisioned, which can take 2 minutes or more. During this time, the excess pods go into a pending state until the new instances are ready to host them. These few minutes can significantly impact time-sensitive workloads and even cause end users to experience service disruption. To avoid this pending state, the cluster overprovisioner can allocate a buffer of free instances that can be utilized when a cluster reaches its limit.

Cluster overprovisioner comes with its own set of challenges though. It allows scheduling of low priority pods, but it has no intelligence about how to deploy workloads. It can merely provision infrastructure resources without actually knowing what workloads need. For example, it doesn't understand which workloads are constantly scaling up or down, and it has no context on how to allocate resources. In this case, the software delivery team needs to build in logic for cluster overprovisioner to make decisions on how to better allocate workloads on the available buffer instances. In other words, deployments need to be clearly defined and the necessary overprovisioned resources need to be available.

## Vertical Pod Autoscaler (VPA) – Rightsizing resources across pods

[Vertical Pod Autoscaler](#) (VPA) is an open source tool to manage resource requests and limits for pods within a node. As the number of tasks running inside a cluster scale, it is important to set limits on the amount of resources each task can consume. This prevents a single resource-intensive task from consuming resources at the expense of neighboring tasks. Vertical Pod Autoscaler helps with this. For example, it can restrict the number of CPU/Memory/GPU requests coming from any pod that is over-requesting resources. It can also allocate additional CPU/Memory/GPU resources to a pod that is under-requesting and is in need of more resources. This process is called rightsizing, and is necessary for optimal use of resources.

The challenges with VPA are when updating the available resources for a pod. VPA restarts all containers in the pod during an update, and may even reprovision these containers on a different node. At times there can be conflicting policies for a single pod, and these conflicts will need to be manually resolved.





## Cluster roll – Updating applications on the fly

Cluster upgrades are a routine yet essential part of day 2 Kubernetes operations. They are required to keep applications secure, and frequent upgrades are the norm in a fast-moving ecosystem like Kubernetes. Additionally, as businesses look to innovate with new features, more frequent releases are the order of the day. However, there are numerous challenges with upgrades that need to be addressed for seamless day 2 operations.

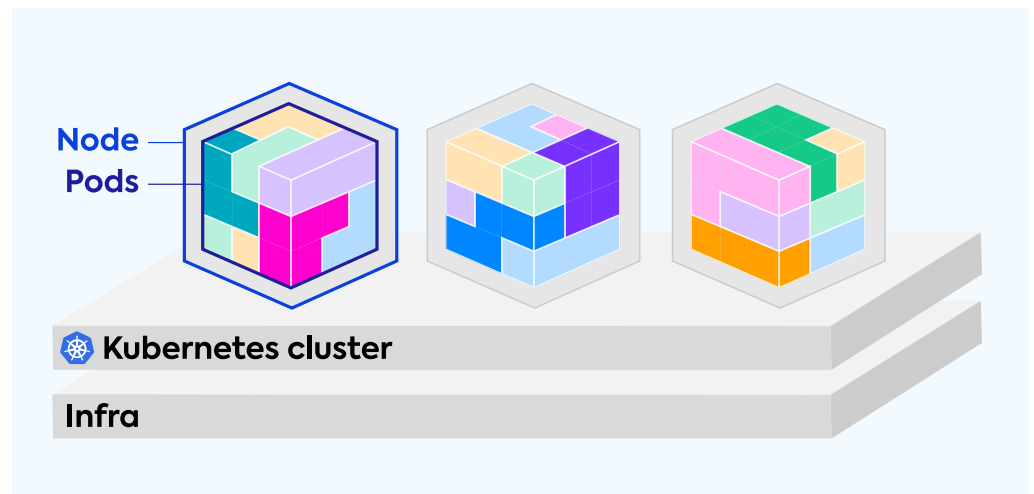
As applications are deployed and updated more frequently than before, it can greatly impact performance if the part that is being updated needs to stop running during the update. The opposite of this scenario is a rolling update.

A rolling update is when an application is updated or upgraded without any noticeable downtime. With Kubernetes, you may not be able to avoid downtime when updating an instance, a node group or the entire cluster. Yet, organizations need this capability as they push for more frequent deployments without impact to the running applications. Some organizations may provision duplicate resources and temporarily channel traffic to the duplicate until the upgrade is complete. Once complete, they would redirect traffic back to the original updated cluster or node group.

## Bin-packing – Matching workloads with the right instances

Part of Day 2 operations is to improve resources allocation and drive down costs without impacting performance. Often instances are left running with underallocated capacity. The ability to reschedule pods on nodes for better resource allocation is called bin packing. Cluster-autoscaler comes with some bin packing capabilities, but this feature only kicks in when an instance is 50% underallocated. This still leaves a lot of unused capacity.

What is needed is a way to handle bin packing so that pods are actively rescheduled on nodes with any free capacity to host them. This would drastically reduce the number of instances required, and save on costs. Operationally, it would simplify things as there are fewer instances to manage and a smaller system footprint. However, setting this up as a DIY project is out of the question for the average software delivery team, which is typically a few engineers supporting dozens or hundreds of developers.



## Cost analysis – Bringing cost accountability to every team

Knowing where cloud resources are being used and what is being spent helps organizations to understand the ROI of their cloud investments, and build sustainable, long-term cloud operations. However, organizations today struggle to understand how much of compute or storage resources are allocated to any particular workload, or business unit. This lack of information makes it hard to project costs, track and monitor expenses, identify wasted resources and set budgets and internal chargeback. Without these insights, strategic decision-making is impaired and teams are less effective.

# The risk of a DIY Kubernetes data plane management stack

In a typical organization, the ratio of DevOps engineer to application developer is imbalanced. A software delivery team of ten people typically supports hundreds or even a thousand application developers. DevOps engineers are stretched thin, and have an ever-growing list of developer requirements and Jira tickets to be responded to. Even if a highly productive software delivery team has initial success and a few quick wins, the ever-changing demands of the business will become overwhelming. Add to this the shortage of high quality DevOps engineers, keeping the team continually well-staffed is a challenge. For all these reasons, a DIY Kubernetes management stack may just about do the job, but is far from ideal as operations scale.

The solution is to opt for a serverless container infrastructure platform like Ocean from Spot by NetApp, which handles the infrastructure plumbing, and allows software delivery teams to enable better developer experiences.

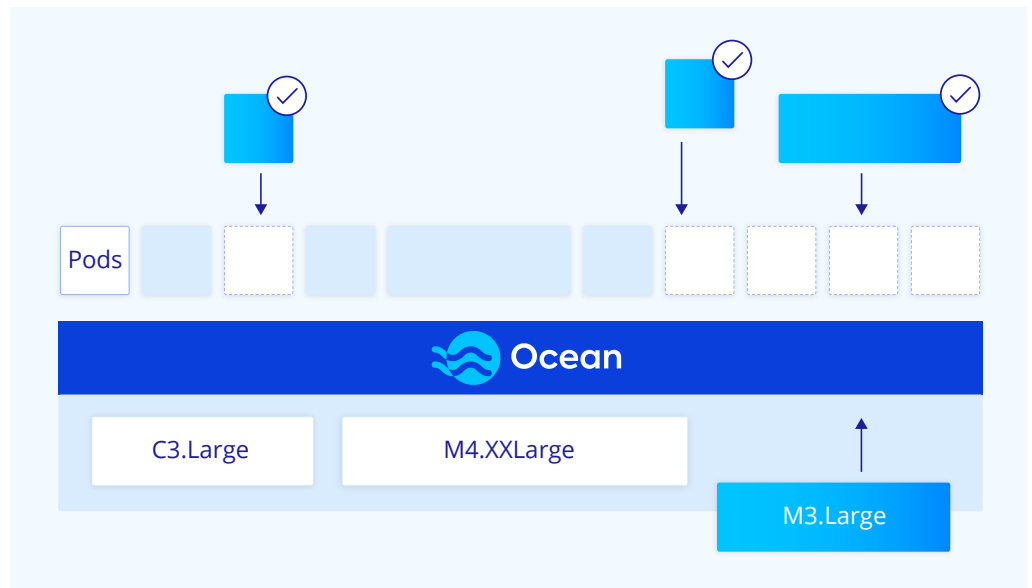


## Serverless infrastructure engine for containers

**Ocean** frees up software delivery teams from manual maintenance of Kubernetes clusters, and enables them to focus on improving workflows, and providing a better developer experience for teams that rely on them. It continually analyzes container infrastructure to drive better resource allocation, and save cloud costs in the process. Ocean can integrate with all major managed Kubernetes services, and lets you get the most out of each of them.

### Here are the key components of Ocean and the tasks they perform:

- **Ocean autoscaler:** This is how Ocean automatically scales the number of nodes up or down making sure there are enough resources to run workloads.



Autoscaler takes a container-driven approach where the infrastructure is scaled according to the Pods' requirements and constraints.

- **Virtual Node Group:** VNGs allow users to configure multiple types of node pools on the same cluster. They provide a layer of abstraction for different types of workloads. Ocean does this using multiple machine types, sizes, availability zones, and life cycles within the same cluster. Additionally, Ocean also leverages excess capacity instances seamlessly managing their interruptions by draining resources to new instances immediately.
- **Ocean-controller:** This handles interruptions to workloads, and ensures that underlying infrastructure unavailability or failure doesn't impact running workloads. It acts as the agent that is located in the cluster and sends relevant data to Ocean. This data is used to perform actions such as draining excess capacity instances that are about to be terminated and moving its workloads to a new instance.

- **Headroom:** Ocean maintains spare capacity, or headroom, to ensure that compute capacity is instantly available to handle scaling bursts. This feature allows the user to set a buffer amount of resources, or set up automatic headroom to allow Ocean to dynamically adjust infrastructure to the demands of applications. By understanding the history, metrics and demands of an application overtime, Ocean maintains the performance and stability of your applications during traffic spikes while limiting over provisioning
- **Cluster roll:** It allows performing updates to the entire cluster or just part of it in rolling batches. The benefit is uninterrupted workloads.
- **Right-sizing:** Ocean analyzes the usage of CPU and memory of container instances continually and provides recommendations on what should be the requested resources on a container level. It does this by comparing the current requests against the actual consumption of resources in each container. This results in cost savings and better resource utilization.
- **Cost analysis:** Ocean tracks cloud spend and enables cost showback to each team and application so you can have better visibility into cloud expenses, and optimize them as needed.

As you scale your usage of Kubernetes beyond Day 2, you need not have your software delivery team spend its best time managing Kubernetes plumbing tasks. Instead, leverage a serverless container infrastructure platform like Ocean and free your software delivery teams to better empower your application developers.

## Key takeaways

1. All tasks that follow the initial setup and configuration of Kubernetes fall under Day 2 operations. There are many aspects such as high availability, security, observability, storage, and developer enablement that come into play in Day 2 operations.
2. Infrastructure and workloads describe themselves differently in Kubernetes. Workloads describe themselves in the form of Labels, Taints, Tolerations, and Network & Storage requirements, whereas infrastructure describes itself in terms of instance size, memory, and pricing model. This is a big challenge with Day 2 Kubernetes operations.
3. Some organizations opt for the DIY route in managing Kubernetes. This puts the onus of business continuity squarely on the shoulders of the software delivery team.
4. To manage Day 2 Kubernetes operations the software delivery team needs to consider aspects such as cluster autoscaling, node pools, node termination handling, cluster overprovisioning and more. This weighs heavily on an already overworked software delivery teams.
5. End-to-end CI/CD pipeline automation is a myth. In reality, the software delivery team ends up manually moving the process along at each step.
6. [Ocean](#) is a serverless infrastructure platform that provides a hands-free experience of managing Day 2 Kubernetes operations. It includes the necessities like autoscaling, node pools, and termination handling. On top of this, Ocean has unique features like headroom, right-sizing, cluster roll, and cost analysis that enable software delivery teams to get the most out of Kubernetes from Day 2 and beyond. Leverage Ocean for an effortless, yet highly optimized Kubernetes management experience.