StormForge

# 4 Ways Cloud Waste is Driving Inefficiency and Unnecessary Expense

A Guide to Kubernetes Cost Optimization

# What's Inside

CHAPTER 1

# Introduction

The cloud. When the concept first came to light, we weren't exactly sure what it was going to be. Eventually the IT landscape understood the concept and it turned into a boon for consumers who needed online storage sync and sharing. That gave way to the likes of Google Drive and iCloud and made Android and iOS possible.

But then, with the advent of Software as a Service (SaaS) the cloud became something far more important. Mobile devices and even desktops could use web applications and other cloud technologies as if they existed on the native platform. This opened up an entirely new world for developers looking to expand their horizons beyond the standard client/server metaphor.

Cloud-native technology then gave birth to containers, which expanded the field beyond the wildest dreams of developers everywhere. At this point, scalable application and service deployment weren't just possible, they could be automated, self-healing, and could even save businesses considerable money. No longer did businesses have to spend their precious budgetary dollars on costly hardware capable of handling the scale necessary to meet an ever-growing demand.

ACCORDING TO THE CLOUD NATIVE COMPUTING FOUNDATION

**84%** **of businesses were running containers in production in 2019.**

ACCORDING TO A 2020 RED HAT REPORT

**56%** **of organizations expected their use of containers would continue to grow.**

And developers everywhere jumped on the container bandwagon. Back in 2019, **Cloud Native Computing Foundation reported** that 84% of businesses were running containers in production. That number has continued to grow. In fact, in **2020 Red Hat reported** that 56% of organizations expected their use of containers would continue to grow.

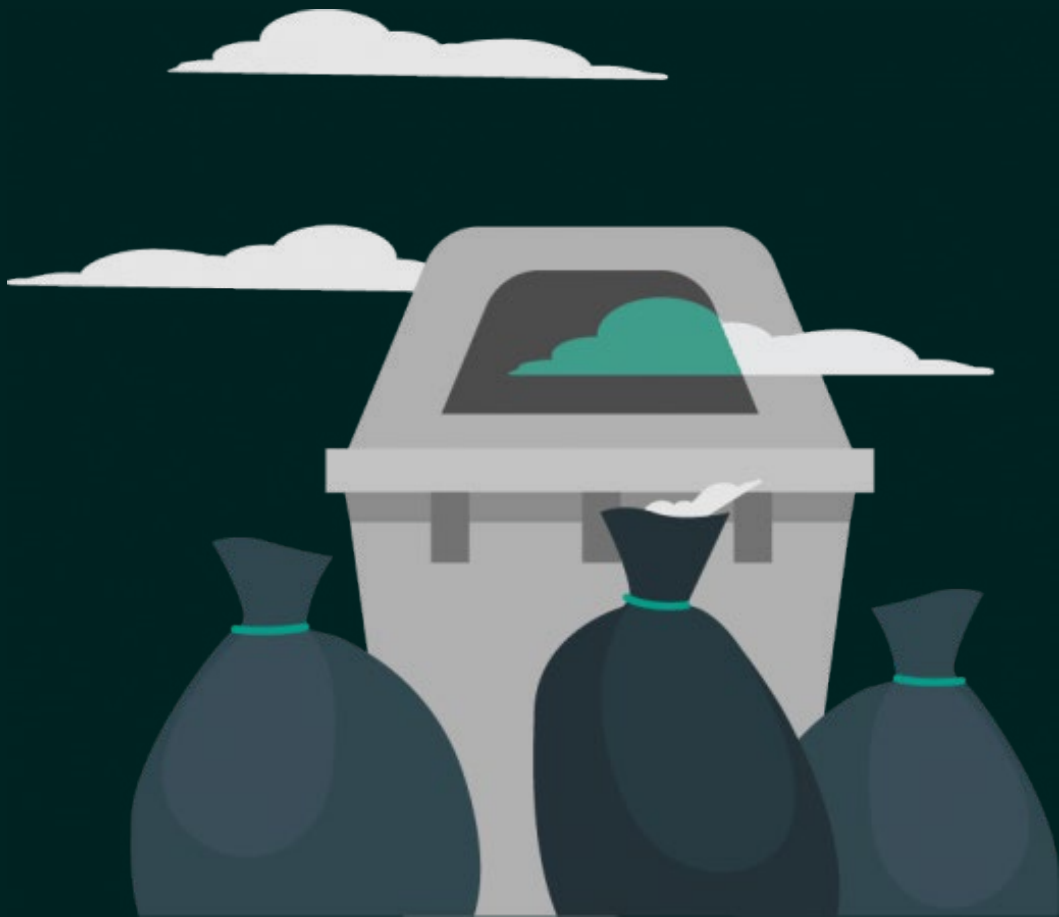
One of the primary reasons for the explosion of container and cloud-native technology is cost-effectiveness. Developers can deploy pods and containers to Kubernetes clusters running in either on-premises data centers or cloud-hosted providers and scale them to meet massive demand and then roll those deployments into Continuous Integration/ Continuous Deployment (CI/CD) pipelines. With automation in play, those containers will enjoy lighting-fast scalability, self-healing, and unmatched performance.

If they are configured properly. And that is the crux of the problems facing cloud-native developers. When containers and pods aren't perfectly configured, there can be considerable waste involved. Cost, performance, time, and effort are precious resources for your business, and you owe it to your company to develop for and use the cloud with a very high level of efficiency, otherwise, you're not just throwing money out the door, your applications and services aren't performing as well as you might think.

But how do you avoid cloud waste, to optimize efficiency in your deployments? To reach a balance, you must remove waste and add automation. That's where the sweet spot exists. Let's see if we can demystify this a bit.

CHAPTER 2

# What is “Cloud Waste”?



First off, what exactly is “cloud waste?” Simply put, cloud waste is when purchased cloud resources go unused or are underused.

Let’s explain that, with regards to cloud-native development.

When your developers painstakingly craft manifests for container and pod deployments, they most likely configure resource requests and limits (e.g. CPU and memory) along with the number of replicas to deploy. For each container or service, that’s at least three resources to configure (each of which could have both a request and a limit). That’s fine, when you have one or two containers or services within a cluster. But when those deployments start growing, even with only a few containers and/or services, you’re looking at the possibility of millions of possible combinations for the configuration of necessary resources. When that’s the case, your developers have to play a guessing game as to what combination of resource settings will result in the best performance at the lowest cost.

For example: A cluster contains containers A-Z and services 1-10. Right out of the gate we have up to 108 different resources to configure, which leads to a seemingly endless number of combinations and configurations. The developers create the necessary YAML files for the deployment, with their best guesses as to much each resource will require. They deploy the pod to the Kubernetes cluster, only to find it underperforms.

Back to the configuration drawing board.

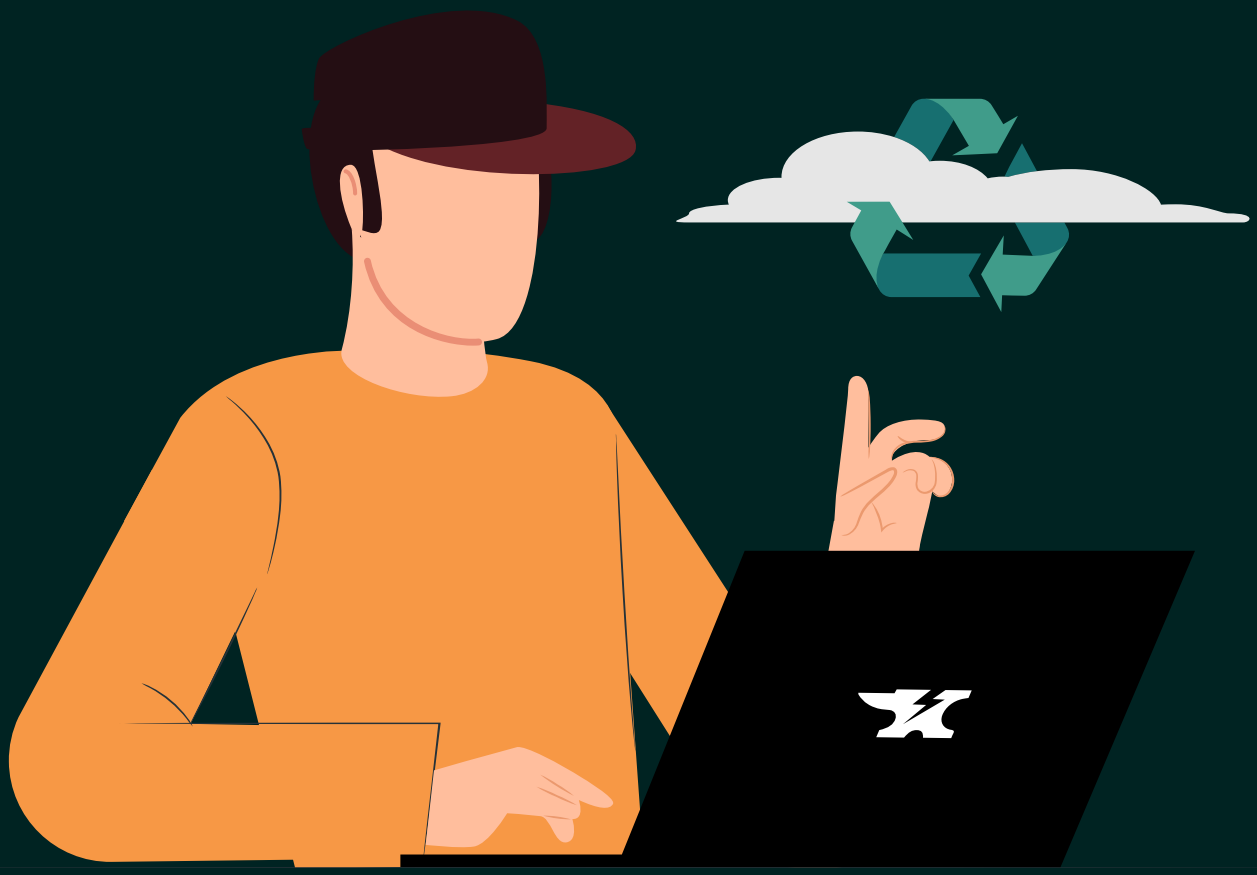
Your developers set out to increase CPU and Memory resources, assuming those to be the reason why the pod isn’t up to demand. Once the pod is updated, it starts performing as needed.

Thing is, did your developers over-commit resources? If that’s the case, you’re now facing cloud waste and will wind up paying for those resources, even if the pod isn’t using them. Why? Because your developers have provisioned those containers as such.

Remember, cloud-native is a pay-for-what-you-use proposition. Even if your pod doesn’t need the resources that have been assigned, if it uses it, you pay for it. That could cost you a considerable portion of your cloud budget—much of which is going to waste.

CHAPTER 3

# The Promise of Cloud and Cloud-native



Cloud-native was supposed to be the savior of budgets everywhere. In fact, this new-world-order made considerable promises for four very important benefits.

# 01.

## Efficiency

Cloud-native makes it possible for your company to deploy apps and services with an efficiency that you've never before experienced. Consider this: By using a container orchestrator (such as Kubernetes), it is possible to automatically schedule deployments and allocate resources based on demand. And as demand increases, your deployments can automatically adjust to meet those demands. To do that manually would require far more time and resources. For a developer to do that as efficiently as an orchestrator is simply impossible. That level of efficiency is a big draw for enterprise businesses.

However, autoscaling with Kubernetes is not as simple as flipping a switch. Autoscalers require configuration and tuning, and if not done effectively you can say goodbye to those efficiency gains.


"During our effort to eliminate waste, we found a number of large services not using horizontal-pod-autoscaler (HPA), and services that were using HPA, but in a largely sub-optimal way such that it never effectively scaled the services (high minReplicas or low maxReplicas). A focused effort around service tuning improved our utilization, and also maximized the impact of the cluster auto-scaling work…"

**READ THE ARTICLE**

# 02.

## Performance

Cloud-native development radically changed how companies develop and deploy applications. But with this new model comes a level of complication that didn't exist with the standard model. When you deploy a traditional application or service on standard hardware, if it didn't perform well, you could always upgrade the hardware—add a more powerful CPU or more RAM. With cloud-native development, performance increases are much simpler, while also being more complex. Instead of increasing hardware, you change resource requests within manifests to ensure an app or service is allotted enough CPU cycles or memory to function properly.

Although getting these configurations right can be tricky, it does mean your developers have more control over what apps and services are allowed to gain a performance boost over others. The problem is, your developers have to completely change their performance mindset from the old-school ways.

# 03.

## Agility

Agility is one of the biggest benefits of cloud-native computing. With cloud-native computing, services are abstracted from the underlying infrastructure. By allowing developers to focus on the service, it is possible to innovate and deploy more quickly, instead of having to be concerned with how a particular service will perform on a particular infrastructure. It is this portability that makes cloud-native so agile, but it also adds new levels of complexity to manage as we'll see.

Containers can be deployed and taken down in seconds. Your developers could theoretically create manifests for multiple business types and deploy each at will, whenever the demand arises. You cannot achieve that level of agility with traditional applications and services.

# 04.

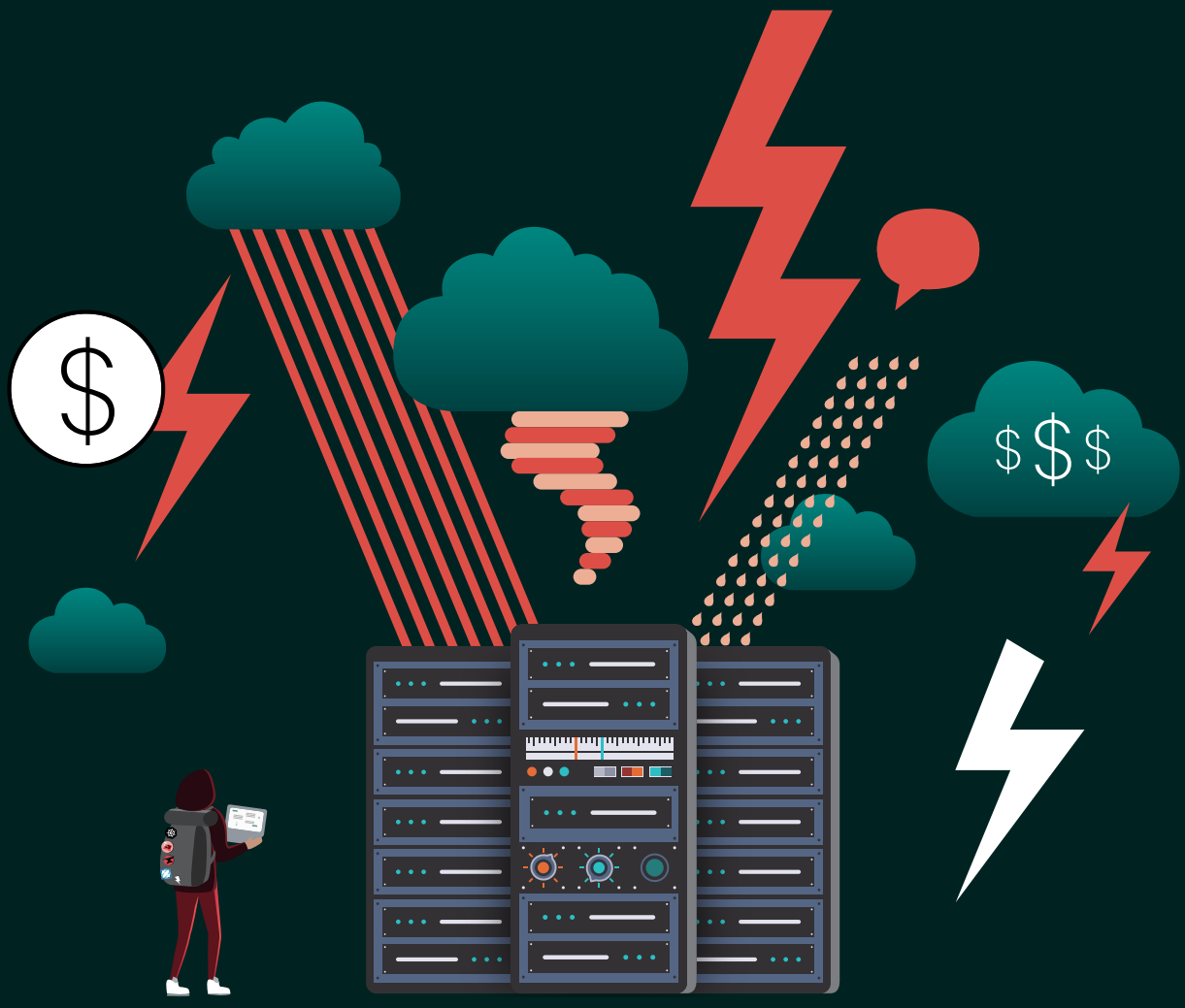
## Scalability

This is where cloud-native deployments really shine. Your business will undergo variable demand from day to day and even hour to hour, and containerized applications can automatically grow and shrink to meet those demands. Say, for instance, you know your business is always hit with an exponentially higher demand between the hours of 7PM and 10PM during weekdays. With the help of an orchestrator, you can have those containerized applications automatically meet those demands during those hours. When demand decreases, those apps and services will no longer need the added resources.

Not only does this ensure your containerized applications will be able to more than meet the rise in demand, it can also save you money, as those containers will discard the extra resources when demand drops. Again though, only if your application and autoscaling are configured optimally.

CHAPTER 4

# The Problem and Impact of Cloud Waste



Now that you understand the benefits of cloud-native, it's very important to grasp the problem and impact that cloud waste can have on your business. We break this down into four categories.

# 01.

## Costs

At first blush, cloud-native seems like an absolute victory over standard application deployment. After all, you don't have to pay for costly hardware, and usage tends to be dedicated by demand. Problem is (as we've already stated), cost savings will greatly depend on how well your developers have configured the container/pod manifests. And even if you have the best cloud-native developers on the planet, when the YAML files get too complicated, they'll still struggle to configure them in such a way to seriously put a dent in costs. This is especially true when you have containerized applications that are hit with serious demand. If those pods aren't configured perfectly, you'll waste resources (which will, in the end, cost you).

The good news is, there are services and application layers that can be used with or added to your container deployments that can detect if resources are poorly configured and even optimize them for you. These types of services are especially useful when your manifests contain numerous containers for a single deployment.

# 02.

## Time and effort

This is a problem that can quickly snowball and before you know it, you've wasted serious resources. The time and effort your developers invest in properly configuring container manifests can become significant. This is compounded as those deployments become more complicated. But more than anything, the challenges your developers face when trying to optimize container deployments can not only lead to cloud waste, but also wasted time and effort that could be spent on other tasks. Given how this job can (and should) be automated, the more time your developers spend attempting to optimize those containers, you'll find the waste becomes exponential.

Think about it this way: Your developers are already working long hours to either roll out new services and apps, or update current software. When they're given the almost impossible task of perfect container YAML configurations, they'll get buried under a mountain of duties. And because the cost-effectiveness of your cloud-native development can easily become a priority, you'll likely pull them from their core focus on developing new features to help curtail runaway spending on cloud resources.

Instead of allowing this, you should turn to a service like StormForge, which is capable of automatically perfecting resource allocation in container manifests. Not only will this free up your developers from a task no human could possibly achieve, they'll be able to spend their time and effort on other important jobs (ones they can successfully complete).

# 03.

## Performance

Container optimization is incredibly challenging, especially when you're working in massive scale, enterprise computing environments and demands. And even when your developers toss their hands in the air and throw every resource they can at the configurations, performance can still suffer.

Why? There are so many moving parts involved with complex deployments that knowing exactly what tweaks to make to eke out the best performance is next to impossible for a human to get right. The biggest problem with performance is that (when a deployment isn't performing as expected) developers start handing over more and more resources. Sometimes giving a container more resources isn't nearly as performant as giving it the right combination of resources. In other words, adding more CPU cores might not be as effective as the same number of cores and just a bit more memory. Or maybe it's not the NGINX container, but the MySQL container that needs more resources. Getting these combinations right can be incredibly complex. Getting them wrong can lead to significant cloud waste.

# 04.

## Environmental impact

Finally, we have the impact cloud waste has on the environment. This is a serious issue that continues to grow more and more daunting. Consider this: The more cloud waste you create, the more a cloud host's servers will have to work. The harder those servers work, the more negative impact they'll have on the environment. With every business on the planet generating cloud waste, that impact becomes significant. **Greenpeace** estimates that by 2025, the technology sector could consume 20% of the world's total electricity, with the increase from 7% today partly due to the growth of cloud computing.

The importance of minimizing environmental impact cannot be overstated. When you optimize your container deployments to minimize cloud waste, you're not just helping your company's bottom line, you're lessening the carbon footprint of your cloud host. According to **Forbes**, only 60% of companies have a sustainability strategy.

That same article indicates that implementing a successful sustainability strategy would reap benefits beyond just altruistic ones, including:

- ☑ Adding brand value and a competitive advantage.
- ☑ Attracting more consumers who see sustainability as a value.
- ☑ Increasing operational efficiency.
- ☑ Attracting better development talent.
- ☑ Creating new opportunities in more markets.

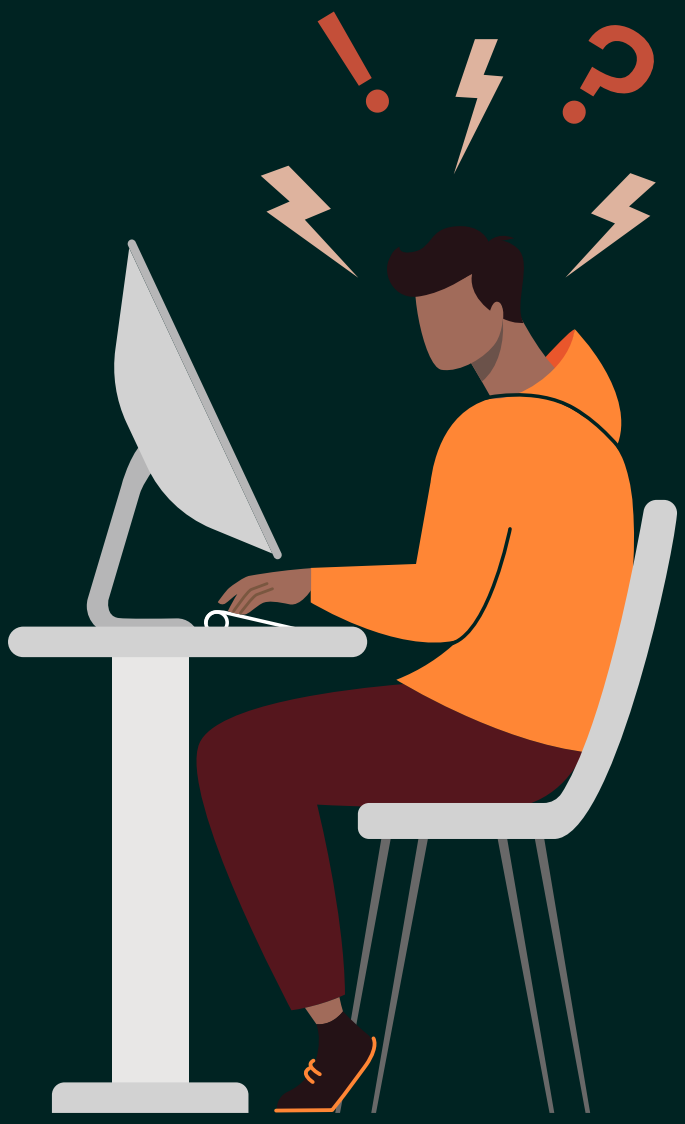
GREENPEACE

"By 2025, the technology sector could consume 20% of the world's total electricity; this increase from 7% currently is attributed to the expansion of cloud computing and the further development of new technologies, such as artificial intelligence, which require a great deal of computing power."

**READ THE ARTICLE**

CHAPTER 5

# The Sources of Cloud Waste



So what are the sources of cloud waste? By now you're probably pretty keen on what it is and even have a good idea of where the waste stems. But what specifically in your containers is the primary source of cloud waste? Let's examine the two primary candidates.

# 01.

## Idle resources

Idle resources are virtual machines and cloud-native instances being paid for by the hour (or minute, or second) that are not being used. These resources are either dedicated to development, staging, and QA, but can also land in production, where you've paid for resources and haven't optimized your containers well enough such that those resources are used wisely.

If you're spending a considerable amount for idle resources, you should revisit this and either consider using in-house (on-premises data center) servers for development and testing, or make sure your container manifests are perfectly optimized.

# 02.

## Over-provisioned resources

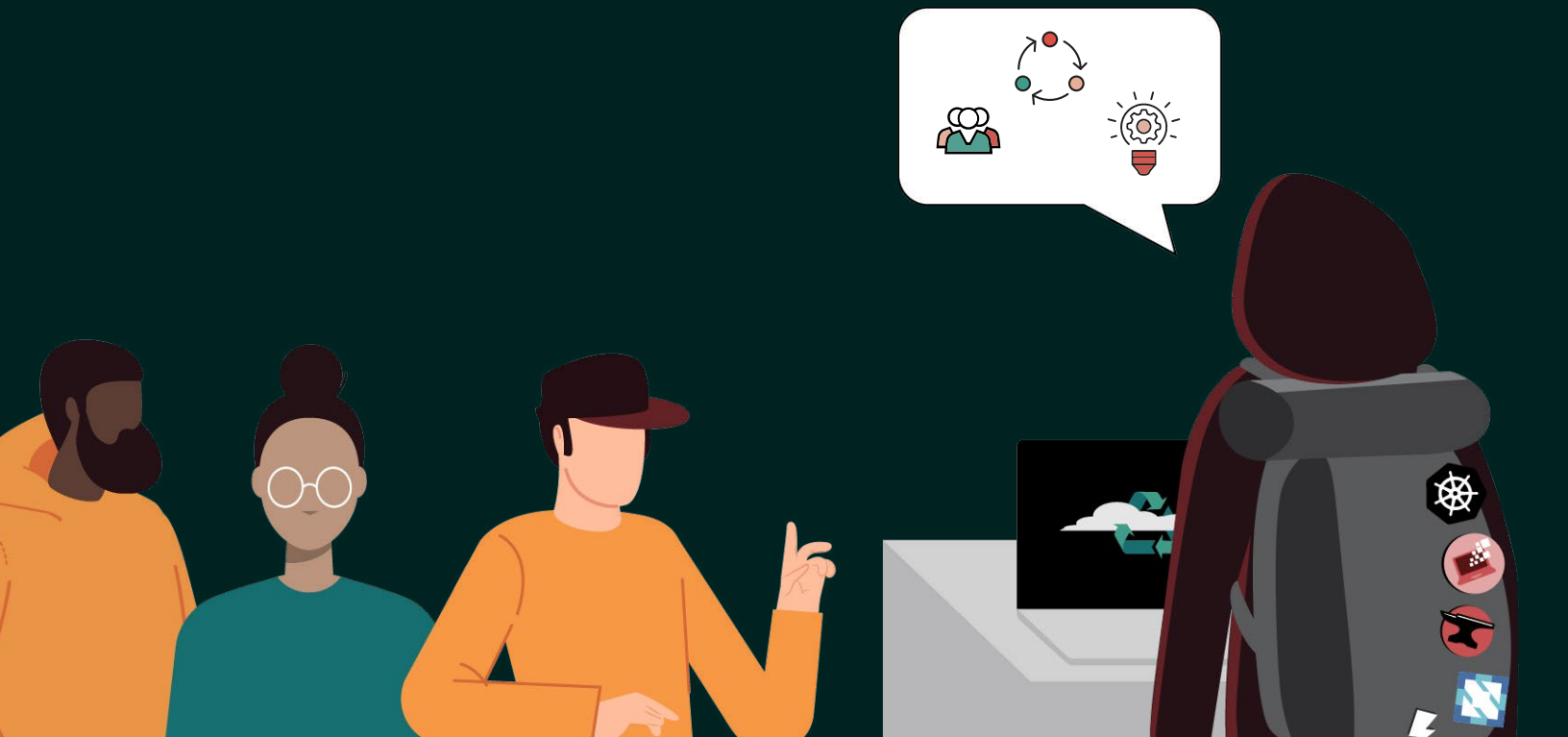
The other source of cloud waste is over-provisioned resources. This is when you pay for more resources than are necessary for a deployment. If you find you consistently only use 50% of your resources (even during times of heavy load), then you shouldn't be paying for that waste. This is when it becomes crucial to not only spend the time and effort in perfecting your configurations but keeping a close watch on those deployments over extended periods, so you are aware of how resources are used.

**Never** assume more is better.

**Always** assume optimized is best.

CHAPTER 6

# Addressing the Issue of Cloud Waste



What areas can you focus your energy on to help eliminate cloud waste? As with any holistic solution, you have to consider people, process and technology.

# 01.

## People

The first thing you must do is ensure your developers and anyone within your DevOps teams is taking full ownership of the problem. When you empower your teams with the tools and knowledge necessary, you not only make it possible for developers to deliver faster, but you give them visibility into the efficiency of the apps and services they create. When developers can readily see the impact of this efficiency, they can more easily make changes to mitigate cloud waste.

But cloud waste isn't just about developers, it's also about those who are charged with monitoring and paying for cloud resources. Everyone responsible must play a part in the process of reducing cloud waste, otherwise, a reduction will never occur.

There are also new roles emerging that directly impact your company's cloud waste. For example, Cloud Center Of Excellence (CCoE) teams collaborate together on the issues of:

- Cloud adoption (solution architects)
- Cloud strategy (program and project managers)
- Cloud governance
- Cloud platform
- Cloud automation

With such a team in place, you'll have a much better chance of keeping cloud waste at a minimum.

Another example is FinOps, which strives to:

- Combine Finance, Technology, and Business leadership for cross-company collaboration, innovation, and decision making.
- Focuses on cloud costs and finds areas to make trade-offs for the optimization of spending.
- Helps to make smarter tech investment decisions that will bring about significant cost savings without compromising deployments and/or deliveries.

# 02.

## Process

This is where you help everyone involved take ownership of cloud waste. You need a regular, repeatable process that takes cloud waste into consideration at every step. This means you'll be integrating the likes of DevOps and FinOps into the software lifecycle and giving all of those involved the agency to effect change.

It's very important that you create a cloud-native process that is not only repeatable but is optimized for both performance and cost-effectiveness. That means you'll be relying on your operations teams to be able to work together in ways that might never have before. And those team members need to be willing to make compromises and understand the goal is both high performance and the avoidance of cloud waste.

# 03.

## Technology

Finally, you need to integrate the necessary technology into your pipeline. This isn't just about Kubernetes, Helm, Flux, and Istio, but about third-party services that can help optimize your deployments in ways humans aren't capable of. Machine Learning is one of the best tools available for optimized container resource allocation. Such services can be automated such that with every update to your deployments, the configurations are optimized to prevent cloud waste. With ML integrated such that it's automated, you can be sure that not only are you reducing cloud waste, but you won't have poorly configured resources creeping in with manifest updates.

CHAPTER 7

# Conclusion

Cloud waste is a bigger problem than you think it is. It's not until you dive into your deployments and get full visibility that you realize how much money you are spending on resources that aren't used or are poorly used.

To make matters worse, cloud waste impacts more than your budget and your deployments. If you're serious about cloud-native, then it is imperative that you focus your efforts on ridding your containers of cloud waste. Your budget, your apps/services, your customers, and the environment will thank you for your effort.


+1 (857) 233-9831 | info@stormforge.io | stormforge.io