GUIDE

AWS Cloud Cost Allocation: The Complete Guide

www.nops.io

Table of Contents

Introduction

To get the most value out of your AWS resources, it is crucial for decision-makers across business leadership, engineering, product, and finance to understand what cloud costs are generated and who is generating them.

However, the complexities of dynamic cloud usage make it difficult to have a complete understanding of your cloud cost. AWS provides a monthly billing file called the Cost and Usage Report (CUR) which may have hundreds of thousands, or millions, of rows of granular data on your hourly resource use. In some cases, such as EC2 instances running Linux, billing is tracked on a per-second level. With so much information available, you need a method for translating cost data into business value. Otherwise, it's impossible to answer questions like:

- Why is the AWS bill $X this month?
- Are individual teams and business units using the right amount of resources?
- What value is $Y charge bringing to my organization?
- Who is responsible for shared costs?

In the 2023 re:Invent Keynote, Werner Vogels emphasized that unobserved systems lead to unknown costs. That's why the FinOps foundation's first phase of the journey starts with "Inform": in other words, visibility and cost allocation.

About this guide:

This ebook aims to show you how to allocate every dollar of your AWS spend. When you truly understand your cloud costs, you can get the right information to the right stakeholders — driving better visibility, accountability, and strategic decision-making.

Unpacking the AWS Cloud Cost Structure

Unpacking the AWS Cloud Cost Structure

AWS currently has over 200 services, with new offerings launched by the month. Its pricing system is notoriously complex, with over 700 instances just for EC2 alone.

Luckily, a solid grasp of the landscape and most important factors that determine pricing can go a long way in explaining your cloud bill. Here's the most important info you need to know about AWS pricing.

AWS Compute Options

The main four types of AWS compute are On-Demand, Reserved Instances (RI), Savings Plans (SP), and Spot. Let's compare.

1. On-Demand

On-Demand Instances let you pay for compute capacity by the hour or second (minimum of 60 seconds) with no long-term commitments. It is the most expensive form of AWS pricing.

2. Savings Plans

Savings Plans provide a flexible pricing model offering lower prices than On-Demand pricing in exchange for a specific usage commitment (measured in $/hour) for a one- or three-year period.

AWS offers three types of Savings Plans: Compute, EC2, and SageMaker.

Compute Savings Plans

Compute Savings Plans are the most flexible commitment option. They allow you to apply usage across Amazon EC2, (regardless of instance family, size, AZ, region, OS, or tenancy), AWS Lambda, and AWS Fargate.

EC2 Instance Savings Plans

EC2 SP provides the lowest prices, with savings up to 72% (compared to 66% for Compute SP). However, you need to commit to a specified region and instance family, though you can change instance size, OS, and tenancy.

SageMaker Savings Plans

SageMaker SP will apply to any ML instance or size, across any region, without manual modifications required.

What are the advantages of using Savings Plans?

- Compute Savings Plans can apply to any instance, maximizing the value and flexibility of the plan you purchase.
- Savings Plans offer the flexibility to make infrastructure modifications while still receiving discounts.
- A Savings Plan automatically applies to eligible usage, regardless of changes to your infrastructure.
- You purchase Savings Plans on a dollar-per-hour basis, not per instance. As a result, SP allows you to purchase compute capacity for as little as a fraction of a cent per hour - one-tenth of a cent per hour, to be exact ($0.001/hr).
- SPs offer multiple payment options, meaning you have the flexibility to pay upfront or monthly (with higher discounts for paying upfront).
- SPs can be scheduled for future purchase, allowing them to apply automatically at a later date (called “queuing”).
- SPs “float”, meaning they can be applied not just to the purchasing account, but to other linked AWS accounts.

3. Reserved Instances

AWS offers two types of RIs: Standard and Convertible. Let’s compare:

Standard RIs offer the highest discount (up to 75% off On-Demand) and are suited to steady-state workloads with predictable usage patterns. They require a commitment to a specific instance family and region, and you can't modify the instance class or operating system after purchase.

Convertible RIs offer less of a discount than Standard RIs (up to 54% off On-Demand), but allow greater flexibility. If your application's needs change, you can exchange Convertible RIs for other Convertible RIs to cover different families, operating systems, or tenancies, as long as they are of equal or greater value.

What are the advantages of using Reserved Instances?

- Standard RIs offer up to a 75% discount on On-Demand pricing
- RIs provide predictable compute power and pricing over a set period of time, ideal for predictable and consistent workloads
- Convertible RIs offer the flexibility to change instance size, AZ, scope, and networking type should your needs change. You still receive a discount, though not as great as for Standard RIs.
- Standard RIs (though not Convertible RIs, or RIs obtained at a discounted rate) can be sold in the Reserved Instances Marketplace, but only for EC2 RIs.
- While Savings Plans can only be purchased for EC2, Lambda and Fargate, Reservation models are available for a greater range of services including EC2, RDS, ElastiCache, OpenSearch, Redshift, and DynamoDB.

Reserved Instances versus Savings Plans

Here is a quick comparison of some common commitment options to consider.

	Standard Reserved Instances	Convertible Reserved Instances	EC2 Instances Savings Plans	Compute Savings Plans
Applicable to	EC2 only	EC2 only	EC2 only	EC2, Lambda, & Fargate
Commitment term	1 year, 3 years	1 year, 3 years	1 year, 3 years	1 year, 3 years
Best case scenario savings	75%	54%	72%	66%
Payment Options	All Upfront, Partial Upfront, and No Upfront	All Upfront, Partial Upfront, and No Upfront	All Upfront, Partial Upfront, and No Upfront	All Upfront, Partial Upfront, and No Upfront
Instance Family	Fixed	Any	Fixed	Any
Instance Operating System	Fixed	Any	Any	Any
Instance AZ	Any (Regional), Fixed (Zonal)	Any	Any	Any
Instance Tenancy	Fixed	Any	Any	Any
Need to reserve capacity?	Any (Regional), Fixed (Zonal)	No	No	No

What are the disadvantages of using Savings Plans & Reserved Instances?

- **SP & RI commitments can't cover spikes**. Savings Plans and Reserved Instances are ideal for a steady and predictable compute spend level. You'll commit to using the agreed capacity per hour. If you exceed this usage, AWS will bill this extra usage at the higher On-Demand rate.

- **Use it (all the time) or lose it.** Savings Plans and Reserved Instances apply on an hourly basis, meaning that you'll need to get fairly granular in your breakdown. If you commit to $10 per hour but only consume $6 of services in any hour, you will lose the remaining $4, known as unused commit.

- **Forecasting the exact amount you will need is difficult to do**. In the dynamic world of cloud infrastructure, systems are constantly evolving — your team is constantly modernizing and rightsizing, so the overall cloud cost fluctuates.

Ultimately, it comes down to the following dilemma: if you commit too heavily, every hour you're paying for unused commit. But if you under-commit, you'll have to pay the On-Demand premium for additional resources.

4. AWS Spot

AWS Spot Instances are spare AWS capacity that users can purchase at a heavy discount. It allows AWS to monetize idle time in their data center by offering it on the Spot market.

Spot can be challenging to utilize from an engineering standpoint. AWS gives you a discount on the instance, but without a guarantee that you'll be able to use it for the duration of your compute need.

AWS can terminate these instances with a two-minute warning (known as a Spot Instance Interruption).

The Spot Instance can be terminated if:

- The Spot price increases above the maximum price that you're willing to pay per hour per instance
- Capacity is no longer available
- The Spot request has constraints that can't be met

[source: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/interruption-reasons.html]

These unexpected interruptions can cause workloads to fail, potentially posing a major problem for production or mission-critical applications. That’s why many companies avoid using Spot, despite the significant potential cost savings.

To reduce the chance of Spot interruptions, it’s important to have diversity in your instance selection pools. Because each instance family and instance size, within each AZ, for every Region, is a separate Spot pool, having more instance options to run your application helps optimize Spot usage. We recommend a minimum of 5 instance families (inclusive of different generations), and encourage even more, provided the instances can handle the workloads.

The other challenge of using Spot is knowing the right amount to use with your existing Reserved Instance and Savings Plan commitments to maximize savings. While it is complex and time-consuming to continually analyze and rebalance your usage, automated solutions can help you cost-optimize your workloads without manual effort.

Common AWS Services and Pricing:

While AWS pricing is extremely complex and constantly changing based on demand and other factors, there are some key factors and considerations that determine overall pricing. Let's break down the most common AWS services and what you need to know.

Service	What it does	Main factors that determine pricing
EC2	Provides virtual machines (EC2 instances) for running applications, offering full control over the operating system and configuration.	• Instance type (these are optimized to fit different use cases, and comprise varying combinations of CPU, memory, storage, and networking capacity) • Number of instances and instance duration • Pricing model (On-Demand, Reserved Instance, Savings Plan, Spot)
Elastic Container Service (ECS)	Manages the deployment and scaling of containerized applications using Docker	• No additional charge for ECS — you pay only for AWS resources (for example, EC2 instances or attached EBS volumes) to run your application • With Fargate, you pay for the amount of vCPU and memory resources that your containerized application requests
Elastic Kubernetes Service (EKS)	Offers a managed Kubernetes service for container orchestration	• $0.10 per hour for each Amazon EKS cluster that you create • You pay for AWS resources (for example, EC2 instances or attached EBS volumes) created to run your Kubernetes worker nodes
Simple Storage Service (S3)	Serves as a highly scalable object storage service for data storage and retrieval	• Storage class (these are optimized to fit different use cases, such as general-purpose storage of frequently accessed data, intelligent-tiering for data with unknown or changing access patterns, etc.) • Number and size of objects stored • Quantity and type of requests and data retrieval • Data transfer between regions

Service	What it does	Main factors that determine pricing
Relational Database Service (RDS)	Provides managed relational database services	• Database instance hours • Database characteristics (vary depending on the database engine, size, and memory class) • Pricing model (On Demand or Reserved) • Number of database instances • Provisioned and additional storage • Outbound data transfer
Elastic Block Storage (EBS)	Provides block-level storage volumes that can be attached to EC2 instances.	• Volume storage (charged by the amount of GB you provision per month) • Snapshots (the amount of space your data consumes in Amazon S3) • Data transfer • Additional charge for EBS volumes that support additional input/output operations per second (IOPS) and throughput beyond baseline performance.
Lambda	“Serverless”, i.e. runs code without provisioning or managing infrastructure	• Number of requests: 1 million free, then $0.0000002 per request thereafter • Duration: 400,000 GB-seconds per month free, then $0.00001667 for every GB-second used thereafter • Data transfer
Fargate	Runs containers without having to manage servers or clusters of Amazon EC2 instances	• vCPU • Memory • Operating Systems • CPU Architecture • Storage volumes attached to containers

Fundamentals of Cost Allocation in AWS

Fundamentals of Cost Allocation in AWS

Spending money on the cloud is not a problem, but wasting money is. But how do you know how much money you are wasting? How do you make sure every dollar of your AWS bill is accounted for?

Optimizing cloud costs often starts with addressing the key issue of limited visibility into expenses. Without clear insights, it's hard to identify what and how much to cut. Key challenges typically include:

- It’s difficult to attribute shared costs to individual teams and business units
- It’s difficult to forecast spending and stay within the budget
- More visibility is needed to investigate unexpected charges or spikes in usage

Step one is to connect the functions of your business to what you’re spending in AWS each month. The goal is to fully allocate, analyze, and report cloud costs so that you understand how resources are being used and by whom. This contextualization, whether it's through different environments, business units, or teams, allows you to organize and understand your spending better.

Better visibility into your organization’s cloud spend and more granular cost and usage reporting provides actionable insight into where you are spending the most money — as well as opportunities for cost optimization.

It makes teams or business units accountable and answerable for their own cloud usage. Who is spending in the cloud? What are they spending money on? What value are you getting for the cost? With cost allocation, all of these questions are possible to answer.

Let’s go over some fundamental terms relating to AWS cost allocation.

Common AWS Services and Pricing:

Cost Allocation: The process of identifying and assigning the costs of shared cloud resources and services to different projects, applications, users or teams within an organization.

Cost and Usage Report: The Cost and Usage Report (CUR) provides a detailed breakdown of your cost and usage of AWS services. It can contain millions of lines depending on the scale of your AWS infrastructure, which can be broken down into time frame, service, resource, and user-defined tags.

Tags: Simple metadata attached to resources in order to make them easily identifiable. If your AWS account has hundreds or thousands of resources, tags help you to categorize, filter and organize depending on different criteria such as project, environment or purpose.

Chargebacks: Type of cost allocation in which the costs of cloud services and resources are billed to the business units or teams that use them.

Showbacks: Type of cost allocation in which the costs of cloud services are reported to the business units or teams that used them (but not billed).

AWS Tagging

AWS Tagging

Amazon Web Services lets customers label their resources with tags, comprising a user-defined key and an optional value. These tags can help you manage, identify, organize, search for, and filter resources by criteria like purpose, owner, project, or environment.

Tagging is immensely helpful for the categorization and allocation of cloud costs. While this section articulates the importance of tagging, and some strategies around it, understand you can still allocate costs while your tagging implementation remains a “work in progress.” There is other metadata that can be used in conjunction with the tagging you do have, to properly split up your cloud spend.

Tag Types

AWS categorizes tags into 4 groups: Technical, Automation, Business, and Security. Let’s take a look at the most important types of tags used for cost allocation.

Category	Tag Type	Purpose
Technical	Name	Identifies an individual resource
	Environment/Lifecycle	Specifies the environment in which the resource is operating, such as Development, Testing, Staging, or Production
	Application ID	A unique identifier for the application the resource is part of
	Application Role	Describes the role or function of the resource within an application, such as Web Server, Database, or Cache
	Cluster	Identifies cluster to which tag belongs
	Version	Indicates the version of the software or application running on the resource

Category	Tag Type	Purpose
Automation	Date-Time	Used to record timestamps related to the resources: when a resource was or should be started, stopped, deleted, or rotated
	Opt in	Indicates whether a resource is opted into certain automated processes, such as starting, stopping, or resizing instances
Business	Project	Associates resources with a specific project
	Team/Business Unit/Department	Assigns the responsibility for the resources to a Team, Business Unit, or Department
	Owner	Entered as a name or email address, this assigns the specific person who created (or updated) the resource
	Customer/Client	The customer/client for which the resources are needed, and possibly charged back to.
	Billing Code/Cost Center	For use in chargeback and showback, this helps the Finance and Accounting groups allocate costs for the business.
Security	Confidentiality	Indicates the level of sensitivity of the resource, such as Public, Internal, or Confidential
	Compliance	Identifies whether a resource needs to adhere to specific regulatory standards, such as HIPAA, PCI-DSS, or GDPR

How To Implement A Tagging Strategy?

Start by using the service console, API, or the AWS Tag Editor to manage tags for individual or multiple resources. Early implementation of tags is crucial, as resources cannot be retroactively tagged. Choose between user-defined tags for customized tagging or AWS-generated tags for standard categorizations, or a combination of both for flexibility.

Once you create your tags, you need to activate them in the Billing and Cost Management console. Once you have activated your tags, you can utilize AWS Cost Explorer to dissect costs by tags and attribute AWS costs accurately across various dimensions like project, team, or customer.

The following table covers the differences between AWS-created, User-created or Vendor-created tags and how to use them.

Source Type	Description	How to activate	Keep in mind:
AWS-defined tags	The 'createdBy' tag is an AWS-defined tag used for cost allocation. It captures data from specific API or console events, aiding in the accounting of otherwise uncategorized resources. These tags are generated and managed automatically by AWS and are exclusively available in Billing Consoles and reports.	1. Go to the AWS Billing Console 2. Choose cost allocation tags in the navigation pane 3. Choose the ‘created 4. By’ tag option under the AWS-generated cost allocation tags section 5. Click on Activate	• Management account access is required • You cannot edit, update, or delete AWS-defined tags • A maximum of 500 active tag keys are allowed • The reserved prefix for these tags is “aws:” • Null tag values are omitted in Cost Explorer and AWS budgets.
User-defined tag	User-defined tags in AWS are customizable tags that you create, define, update, and apply to AWS resources for detailed cost allocation. These tags will be visible on the AWS console after enabling Budgets, AWS Cost and Usage Reports (CURs), and Cost Explorer.You can use the AWS Tag editor, AWS Management Console or your API to create, update, and apply user-defined tags to different resources.	1. Go to the AWS Billing Console 2. Choose cost allocation tags in the navigation pane 3. Choose the user-defined tags you want option under the AWS-generated cost allocation tags section 4. Click on Activate	• It may take up to 24 hours for these tags to reflect in cost allocation reports after application. • User-defined tags typically use a 'user:' prefix. • Only one key is allowed per resource. • Be mindful of tag naming restrictions.
Vendor-provided tags	Some AWS vendors can also create tags and associate the tags with their specific software usage.	1. Navigate to the Billing and Cost Management console. 2. Activate the relevant tags.	• These do not count toward your total tag per resource quota • They have the prefix “aws:marketplace:isv”

As you create and implement your tagging initiative, keep the following best practices in mind:

- Implement a robust tagging strategy that sets out categories of tags; description of the tag, its purpose, and when to use it; and example tag values.
- Use a standardized, case-sensitive format for tags that you apply consistently across all resource types. To ensure a consistent and enforced tagging strategy, follow AWS Tag Policies.
- Use automated tools such as Tag Editor to help manage resource tags, making it easier to automatically manage, search, and filter tags and resources. Also use AWS Config to require tags and report resources without them.
- Use too many tags rather than too few tags — you want to have as few untagged resources as possible for an accurate cost picture. AWS allows for up to 50 user-generated tags per resource.
- Ensure that tags are assigned as close to the creation of the resource and source data as possible, so that they are passed with cost and usage data to downstream systems.

Common Challenges & Tagging Misconfigurations

As you design your tagging strategy, it's also worth keeping in mind some of the common challenges that may arise.

- **Lack of Standardization.** Inconsistent tagging practices can make managing and identifying resources difficult, leading to errors and mismanagement. Even when organizations have standardized tagging practices, enforcing them can be difficult. It is common for users to forget to add tags, use incorrect tags, or use different tags for the same type of resource. For a tag about "production," departments can use "prod," "production," "PROD," "prod1," or any other number of variations.
- **Undefined Resources:** Some resources cannot be assigned tags, such as business support and bandwidth costs. As a result, it can be difficult to allocate these costs to the right resources or cost centers, leading to inaccurate cost allocation and potential budget overruns.
- **Shared Cost Allocation:** Your AWS bill contains various line items, such as purchasing a saving plan or incurring bandwidth costs for business support. These costs are not associated with a particular resource or a particular team, making it difficult to allocate them accurately.

Tip: free tools can help you automatically tag resources and fix mis-tagged resources.

Showback and Cost Allocation

Showback and Cost Allocation

It’s hard to talk about cost allocation without showbacks to layer business context over your AWS spend — let’s break down what exactly they are, how to use them, and compare various approaches.

What is a showback? What is a chargeback?

Showback is an informative approach where users are shown (but not billed for) the costs associated with their IT consumption. It's quick and administratively simple to implement, often serving as a precursor to a full chargeback system.

Showback improves accountability by highlighting the services consumed, allowing users to understand and potentially challenge their IT usage before any actual billing takes place. It focuses on the outputs of IT services rather than the inputs like infrastructure or labor.

On the other hand, chargeback is a financial approach where users or departments are actually billed for their IT usage, turning IT into a cost center. This method requires more robust financial processes and often leads to a more cost-conscious culture within an organization as departments are directly accountable for their IT expenditure. Implementing a chargeback system is more complex and takes longer to put into use.

In this ebook, we’ll focus on Showbacks and how to implement them.

How To Create AWS Showbacks?

Here is the basic outline of creating Showbacks for AWS spend. We'll cover break down these high-level principles in more detail throughout this chapter.

1. **Define the scope**, i.e. which cloud services the model will cover and which business units it will apply to.
2. **Assign account owners** for each cloud service to monitor and track their business units' usage, generate usage reports, and present them to their business units.
3. **Establish tracking mechanisms** such as tags, billing accounts, and organization units.
4. **Distribute shared costs** to the remaining Showback Values to allocate costs more accurately.
5. **Create usage reports** with information about your business units' usage of each cloud service, including the amount of usage, the type of usage, and the costs.
6. **Present the AWS Showback reports** to departments or project heads to make them aware of the cloud resources being consumed, promoting greater cost awareness, financial responsibility, and alignment between finance and operations. These include usage reports, cost reports, service mix reports, and service dependency reports**.**

How To Categorize Resources?

Before allocating costs to Showback values (sometimes referred to as “buckets”), there should be some planning done around the hierarchy of values to group by.

Here are a few different ways of categorizing your resources to get the most out of Showbacks.

- **Tags** are the most recognized method of grouping assets. Key-value pairing is often the only way to create showback for older accounts that have mixed resources within them. Strategies have evolved in the world of segmentation of data, but tags have been a constant.
- **Organizational Units (OUs)** are a great way to create a separation at the account group level. One way of implementing this is to have pre-prod, prod, qa, and disaster recovery accounts for one entity in its own OU. That entity could be a product or application, a line of business, or a department. One example would be OUs based on a Product line.

Organization Name	Account Name
Widgets	Widgets Dev
	Widgets Prod
	Widgets Staging
Sprockets	Sprockets Dev
	Sprockets R&D
	Sprockets Prod

Organization Name	Account Name
Doodads	Doodads Non-Prod
	Doodads Prod

- **Account Name** - this goes hand in hand with an OU structure, but doesn't necessarily require creating OUs. More often than not, companies are leveraging AWS's feature of a canonical account name as a standard practice. Depending on the need, this can be enough information for solid Showback reporting.

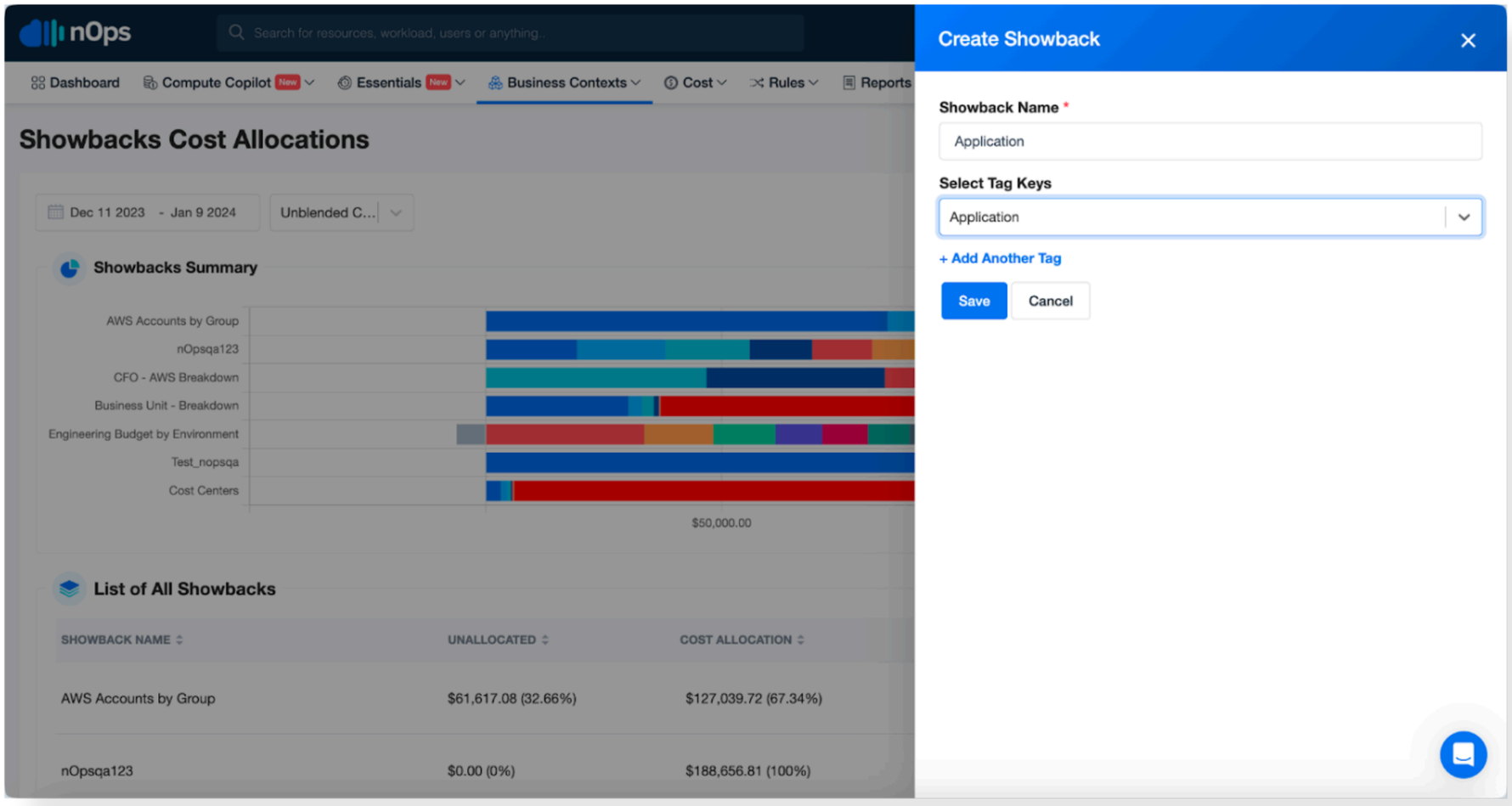
- **Resource Name** - much like the account name, a naming convention on resources can empower companies to create rich, detailed Showback reports. Some companies go so far as to create a multi-component naming convention. For example, the first 3 letters of the resource name might indicate the application. The second part could indicate a 2 digit numeric code to identify if it is production, dev, DR, etc. The third part might consist of 4 alphanumerics denoting the function such as webserver, back-end database, credentialing, and so on. This type of strategy requires an initial time investment, but it means that any engineer can look at the name of the resource and immediately know what it does.

A resource can only live in one Showback value. And, because this is a rule-based mechanism, the order in which you allocate your costs matter.

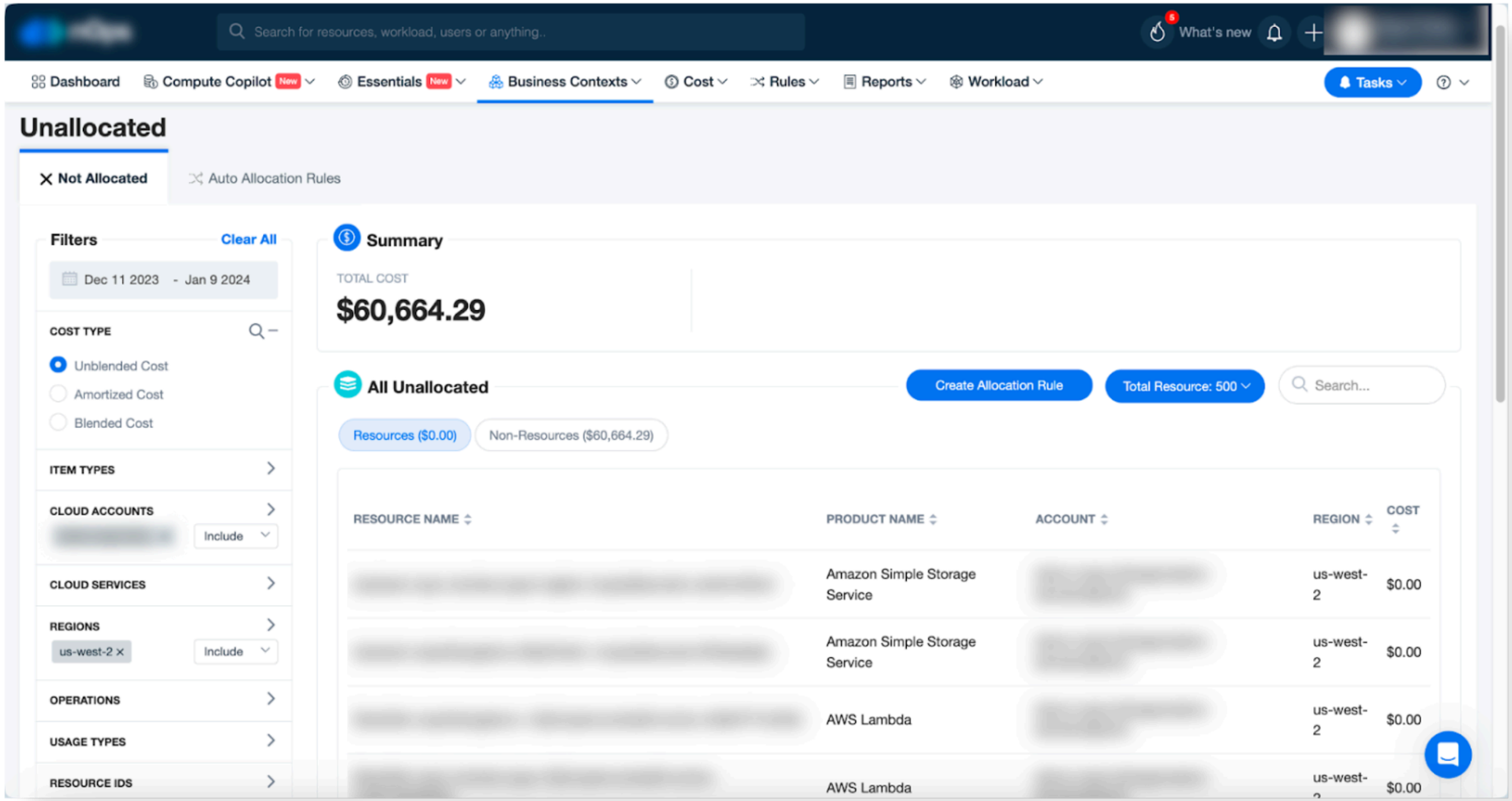
For example, resources tagged with Environment: Production can span across accounts. Do you want to allocate costs based firstly by account and secondly by tag? Or does the tag reign supreme, regardless of which account the resource lives in? What if a resource is named sprockets-webserver-prod, but is tagged Environment: Staging? Which bucket does it truly belong to? These are key considerations when determining your cost allocation strategy.

All of this metadata can be used together to create a cogent method of allocating costs. With the nOps feature of Showbacks, it can be done in a few simple clicks. As tags are so prevalent today, it's a common practice to start with a well-applied tag.

Simply give a name to your Showback and select a tag to start with, though you can skip the tag and use other metadata if you prefer.



By selecting a tag, nOps will automatically gather tagged resources into a Showback Value. You can use the filters on the left to quickly identify unallocated resources.

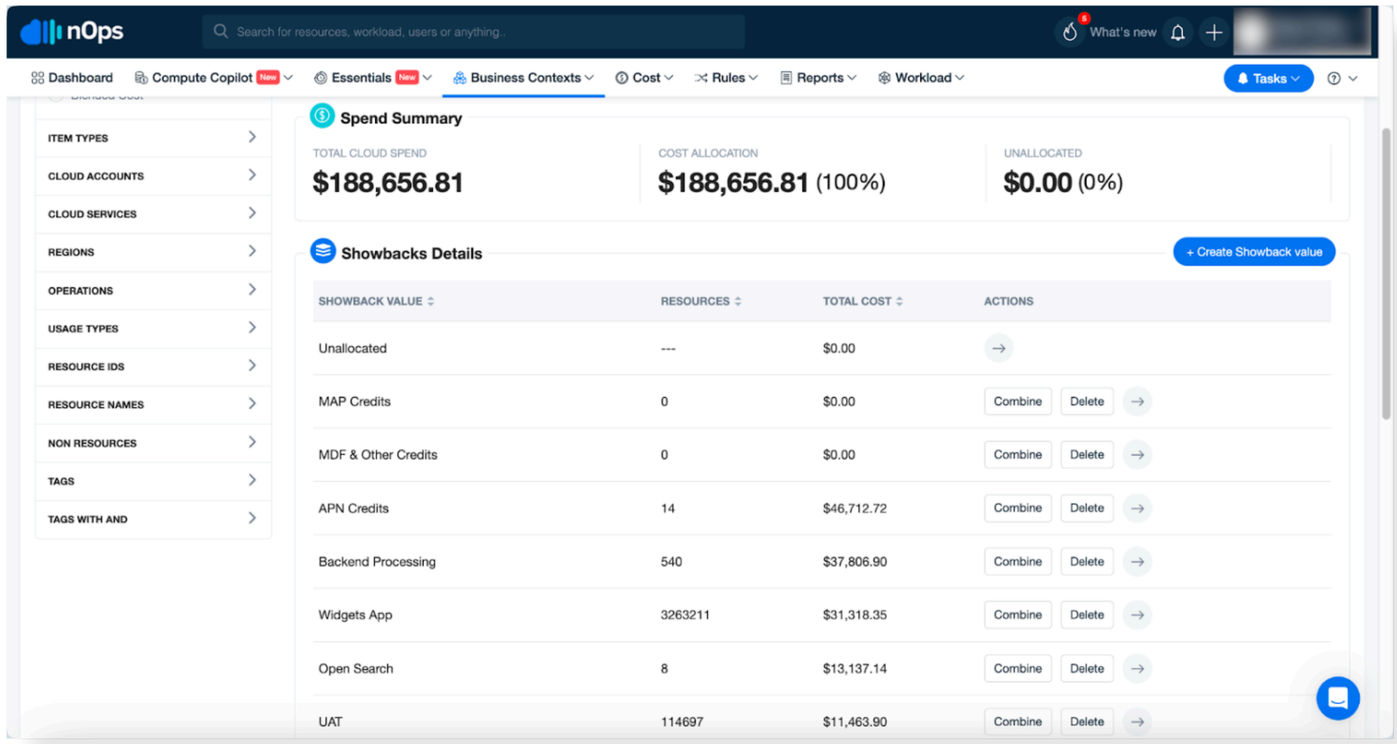


This collection of costs can then be put fully into a Showback value or distributed across the other values. The latter concept is known as shared cost allocation, which we'll talk about later.

After a few rounds of cleanup, you should have a well-allocated Showback. Generally speaking, to clean-up what wasn't originally tagged, you can leverage other tags that may mean the same thing or help identify where resources belong.

As an example, if the tag we start with is Application, there may also be a key of App that was applied outside of standard practice (or standardization is a work in progress).

From there, you can start to look at Owner. In many companies, an owner will only be working on one application, so looking for Owner:DeveloperJane can find resources to allocate to the Sprockets bucket.



Various Showback Strategies

In terms of the type of cost you want to allocate, there are primarily three different methods of allocating and presenting showback costs. Let’s compare.

1. Amortized Cost:

Amortized cost involves spreading the expense of a particular resource or service over a defined period. Instead of charging the full cost upfront, it is divided into equal or proportional portions across multiple time periods.

This approach is often used for costs associated with long-term assets or investments. By amortizing costs, it becomes possible to allocate the expense more evenly and accurately over time, aligning with the benefits or usage of the resource. It can also help provide a more predictable and manageable cost structure.

2. Blended Cost:

Blended cost refers to a combined or averaged rate used to represent the overall cost of a service or resource. It involves aggregating multiple cost components into a single value. This approach simplifies the cost allocation process by presenting a consolidated cost that encompasses various factors. The individual cost components may include things like infrastructure costs, labor costs, software licenses, maintenance fees, and other associated expenses.

Blended cost provides a straightforward and easily understandable cost value, which makes it convenient for comparison and decision-making.

3. Unblended Cost:

Unblended cost represents the detailed, itemized breakdown of all cost components associated with a specific service or resource. Instead of combining different cost factors, unblended cost provides transparency by presenting each cost component separately. It offers a granular view of the underlying expenses, allowing stakeholders to understand precisely how costs are derived. This breakdown can include individual resource costs, usage fees, data transfer charges, storage costs, and any other relevant cost elements.

Unblended cost provides a higher level of detail and accuracy, enabling more in-depth analysis, optimization, and cost attribution.

4. Allocating shared cost

Allocating cost is easy when the costs for a particular application or service have a single owner. However, some cloud costs (“shared costs”) cannot be assigned to a single business unit, application or cost center. These include data transfer costs, AWS Support fees, third party SaaS tools and services, custom built internal services, and network costs.

In the shared cost scenario, fairly dividing costs is more challenging. In the next chapter, we’ll go through some advanced strategies for handling such situations.

Advanced Cost Allocation Strategies

Advanced Cost Allocation Strategies

Any significant cloud-based organization will have costs that need to be split and allocated across multiple teams and business owners. Sharing cloud resources can increase scalability and efficiency. By procuring cloud infrastructure on a large scale, organizations benefit from lower unit costs through discounts and strategic purchasing.

However, for maximum efficiency, these costs should be reallocated to the specific business segments responsible for their usage. Effective tagging and allocation of shared costs are essential to prevent cost overruns and ensure proper management of shared cloud expenses.

According to the FinOps foundation, there are three common methods for splitting up costs:

1. Even split:

This strategy splits the total amount of shared costs evenly among departments (or Showback values).

Department	Percentage Split	Original Spend	Shared Costs	Total Spend
Sales	25%	$50,000	$10,000	$60,000
Marketing	25%	$50,000	$10,000	$60,000
Engineering	25%	$250,000	$10,000	$260,000
Product	25%	$150,000	$10,000	$160,000
Total	100%	$500,000	$40,000	$540,000

In this example, the shared costs of $40,000 are split evenly across the remaining Showback values. While this strategy is the simplest to implement, the disadvantage is that individual departments have no incentive to make their cloud consumption more efficient.

2. Fixed/Custom Percentage:

This strategy splits shared expenses in accordance with a relative, defined percentage of costs, usage, or some other set metric.

For example, say that you have analyzed historical usage and determined that sales generally incurs 20% of fixed costs, marketing 15%, and so on.

Department	Percentage Split	Original Spend	Shared Costs	Total Spend
Sales	20%	$50,000	$8,000	$58,000
Marketing	15%	$50,000	$6,000	$56,000
Engineering	45%	$250,000	$18,000	$268,000
Product	20%	$150,000	$8,000	$158,000
Total	100%	$500,000	$40,000	$540,000

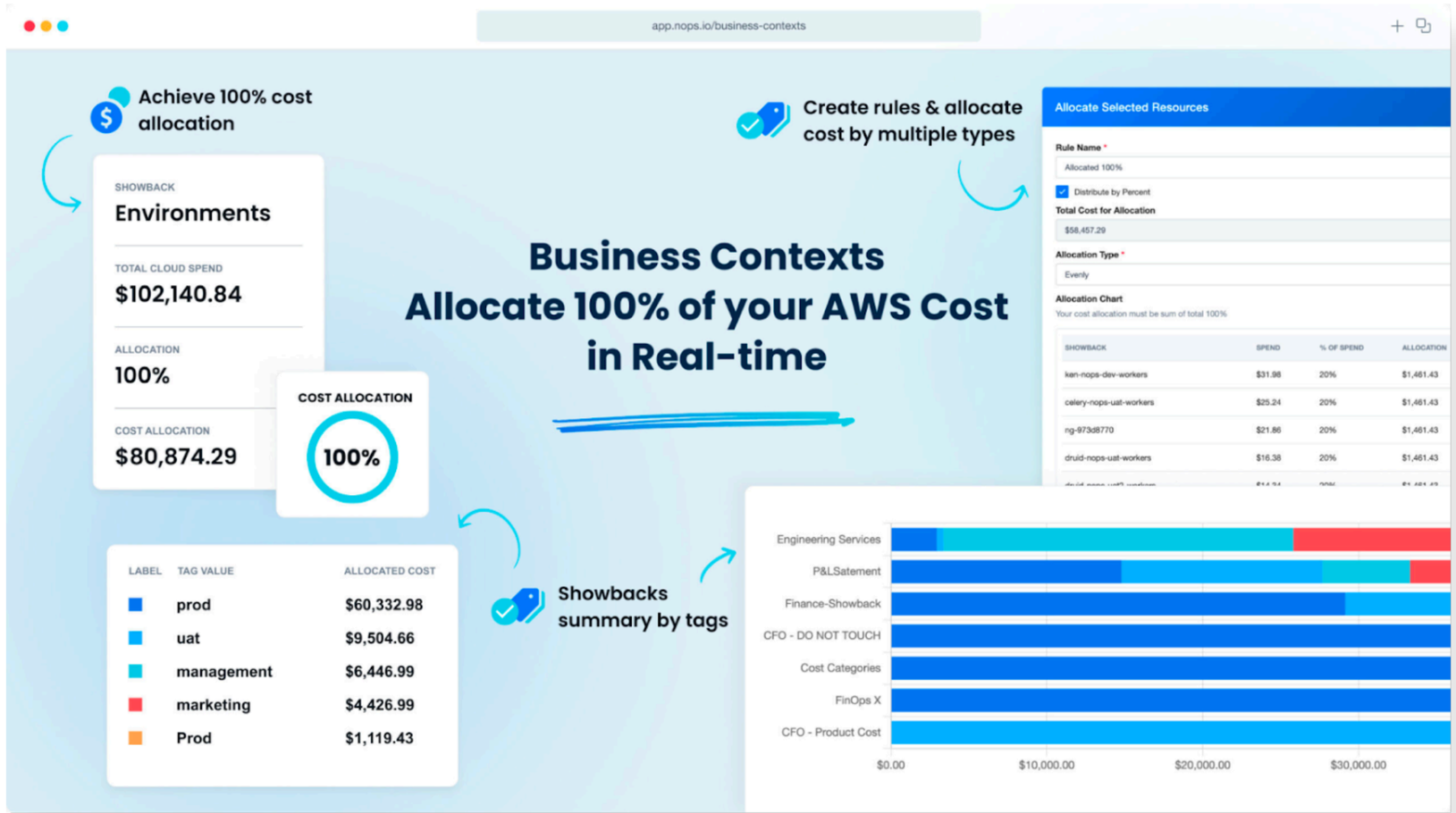
In this example, the shared costs of $40,000 are split across the remaining Showback values in accordance to the specified shared cost portion.

This strategy attempts to more fairly allocate costs, while keeping calculations simple. However, the disadvantage is that it does not account for changing usage patterns.

3. Variable Proportion:

This is the most advanced strategy, used by organizations with mature FinOps practices. It allocates shared costs based on relative percentage of costs, usage, or some other metric which is routinely used for determining the split of spend.

This method generally requires an advanced [monitoring, tagging and cost allocation solution] to effectively implement.



Container Cost Allocation

Container Cost Allocation

The rapid adoption of containers in modern cloud architectures has added a significant amount of complexity to financial management. In this chapter, we'll talk about the challenges that containers pose to cost allocation, and how to gain visibility into your container costs.

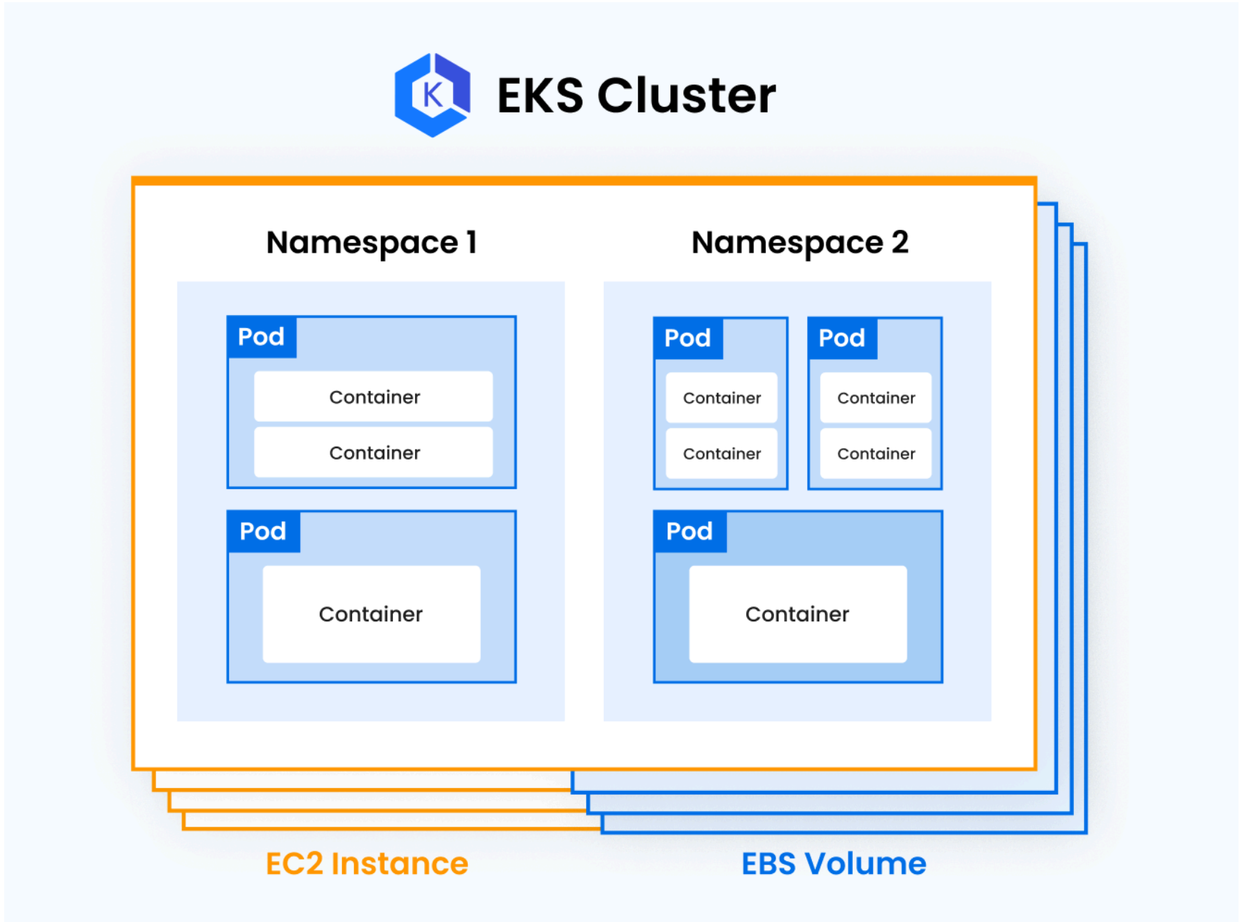
The Complexities of Container Costs

Containers are becoming increasingly popular for deploying software applications, due to advantages such as application consistency across different environments, improved resource efficiency compared to traditional virtual machines, and improved scalability and management.

However, the nature of containerized environments poses challenges to cost visibility and cost allocation. Mapping the cost of resources one-to-one back to specific teams, products or features with tags or labels is not possible, because workloads share resources. A resource may be running multiple containers, each of which may support a different application or cost center. As a result, many teams don't have a good understanding of container costs and how to allocate them.

Container Orchestration basics

Let’s start by briefly outlining the structure of an EKS cluster and the key terms necessary to understand container cost allocation.



Cluster	A collection of nodes that run containerized applications.
Server Instance/Node	A single machine within a cluster, such as an EC2 instance.
Namespace	A way to divide cluster resources between multiple users through a virtual abstraction. Each namespace can contain resources such as pods, services, and deployments, and serves to isolate these from resources in other namespaces.
Pod	A group of containers, treated as a single block of resources for scheduling and scaling. The pod is the smallest deployable unit in Kubernetes.
Container	A lightweight, stand-alone, executable package that includes software and all of its dependencies. Containers are isolated from each other and the host system, ensuring consistent operation across different computing environments.
Pod Labels	Labels are key-value pairs used to organize and select groups of pods within a namespace, facilitating management and resource allocation.

What is Container Cost Allocation?

Container cost allocation is the process of identifying and assigning the costs associated with running containers in a cloud-based cluster environment to specific teams, projects, or departments within an organization by containers. This cost allocation is crucial for organizations that utilize containerized applications within shared clusters, as it ensures transparent and fair billing based on actual resource usage.

In a cluster, costs are incurred from the moment a node is created, which includes the resources such as CPU, memory, storage, and networking required to support the node. To ensure fair billing, it is essential to accurately track how much of the node's resources each container utilizes. This tracking allows for the proportional allocation of the node's total costs to each container based on its resource consumption.

Beyond the direct costs of the server instances, container cost allocation also involves understanding and accounting for the ancillary costs associated with maintaining and managing the cluster, such as networking, storage, load balancing, monitoring, and the management layer.

Container cost allocation comes with some challenges, including:

- **Resource sharing.** When EKS, multiple containers and services may share the same underlying resources, such as CPU and memory, across different pods and containers. This shared ecosystem complicates the tracking and allocation of costs, as the resources are not dedicated to a single container or service. That means that tagging is not as simple as mapping one-to-one, and there’s not an easy way to map these charges to specific container usage.

- **The dynamic nature of containerization.** Containers are constantly created, scaled, and terminated in response to application demands. This fluidity presents a challenge in tracking resource usage over time, as the footprint of a container can change rapidly and frequently. Workloads can be rescheduled across instance types, zones or regions, making it complex and difficult to accurately keep track of these changes.

- **The way AWS pricing works.** AWS discounts such as Reserved Instances and Savings Plans apply hourly, on a use-it-or-lose-it basis, to the usage that will result in the highest discount. This adds a layer of complication to accurately tracking and calculating container costs.

How to allocate Container costs

Let’s discuss the steps required to successfully allocate your container costs.

1. **Implement a robust labeling and namespace strategy.** Use tags to label all of your cloud resources with metadata that identifies the owner (team, department, project) and the environment (production, development, staging). Tags can include information like CostCenter, Project, or Environment.
2. **Determine resource utilization.** Track resource consumption with resource management and monitoring tools. Tools like Kubernetes' built-in resource metrics or third-party solutions like Datadog, Prometheus, or Grafana can provide detailed insights into resource usage. Determining resource utilization can be complex; you might decide to base it off of CPU utilization, memory usage, or some combination of the two. Besides the cost of the core clusters, it's important to consider container management costs, edge services (such as load balancers), licensing costs, backup and data retrieval costs, observability costs, and security costs.
3. **Decide whether to base utilization on requests or actual consumption.** According to the FinOps foundation, each method has its pros and cons.

	Resource Requests	Actual Usage
Advantages	• Allocate all costs • Incentivize teams to only provision what they need • There are tools to help! (e.g. Vertical Pod Autoscaler)	• Each team / app only pays for what they use
Challenges	• Some organizations are not using resource request fields yet • May also incentivize under-specifying requirements	• Who pays for the rest (idle time / cycles)? • What do we do about overprovisioning? • Can incentivize teams to provision more just in case, and not pay for it • Can set unrealistic goal of 100% utilization

Source: FinOps Foundation

4. Address Static and Runtime costs. Static costs are the expenses associated with maintaining the infrastructure required to run containers, regardless of the containers' actual runtime. You have to consider how static costs affect the CPU, Network, and Storage when deployed. Examples of static costs in a containerized environment include:

- **Cluster Infrastructure:** The cost of the underlying physical or virtual machines (nodes) that form the cluster, including costs for reserved instances or dedicated hosts that are paid for regardless of usage.
- **Database Costs:** Persistent storage costs for container images, configuration files, or databases needed by your applications, which are incurred even when containers are not actively running.
- **Networking Infrastructure:** Fixed networking costs such as dedicated load balancers, VPNs, or static IP addresses that are necessary for the cluster's operation.
- **Licensing and Subscriptions:** Fees for any software licenses, management platforms, or support contracts required for container orchestration, monitoring, and security, which are typically not dependent on the number of containers or their usage.

Runtime costs are variable expenses that depend on the actual usage of resources by running containers. These costs fluctuate based on the scale, performance, and efficiency of the containerized applications. Managing runtime costs, on the other hand, requires optimizing container efficiency, scaling strategies, and monitoring to ensure that resources are used effectively based on demand. Runtime costs include:

- **Compute Resources:** The cost of CPU and memory utilization (EC2, EBS, etc).
- **Dynamic Storage and I/O:** Costs associated with read/write operations and dynamic provisioning of storage volumes used by running containers, which can vary significantly based on the application workload.
- **Network Traffic**: Expenses related to outbound data transfer or bandwidth utilization by running containers, especially important for applications with high external network traffic.

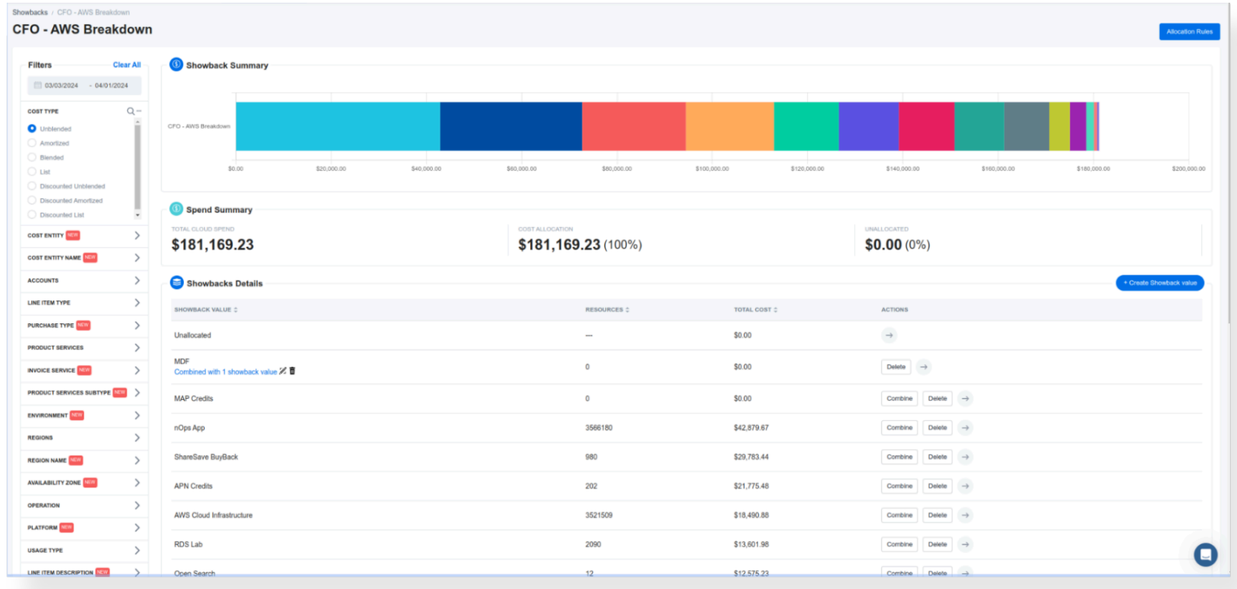
- **Container Orchestration and Management:** Operational costs associated with deploying, managing, and scaling containers in real-time, including the cost of automation tools and additional compute resources required for orchestration services.

For many organizations, it may not be worth building a dedicated proprietary system to handle your Kubernetes cost allocation, requiring significant time expenditures, a deep understanding of your resources and services, and deep expertise in Kubernetes.

Use an Existing Tool to Allocate your Container Costs

If your EKS spending is currently a black box, you can achieve precise EKS cost allocation at the container level with nOps.

nOps automatically and continuously analyzes your clusters and AWS CUR data to map your costs with complete accuracy, and link them back to the individual business units producing them.



nOps Showback Dashboard highlights your shared costs.

Easy integration of your various metadata

EKS containers do not have AWS tags, but they can use Kubernetes labels. nOps Business Contexts automatically combines and integrates your existing tags and labels, making it easy to assign container spend to existing tags or showback values through your existing tagging infrastructure. As a result, it’s quick and easy to start setting billing rules and allocate costs based on your unified AWS ecosystem.

Deep integration with showbacks

It is critical for product, finance, and engineering teams to be able to track service delivery costs by customer, product or feature. Container costs are fully integrated with showbacks making it easy to gain the visibility you need for budgeting, forecasting, accountability, and calculating business value.

Business Contexts is your complete solution for Cost Allocation and visibility

nOps Container Cost Allocation is integrated into Business Contexts, a complete solution that makes it easy to understand your AWS spending. It continuously, fully, and automatically allocates your AWS cost, handles tag misconfiguration, and spreads shared cost to multiple teams and business units for better accountability.

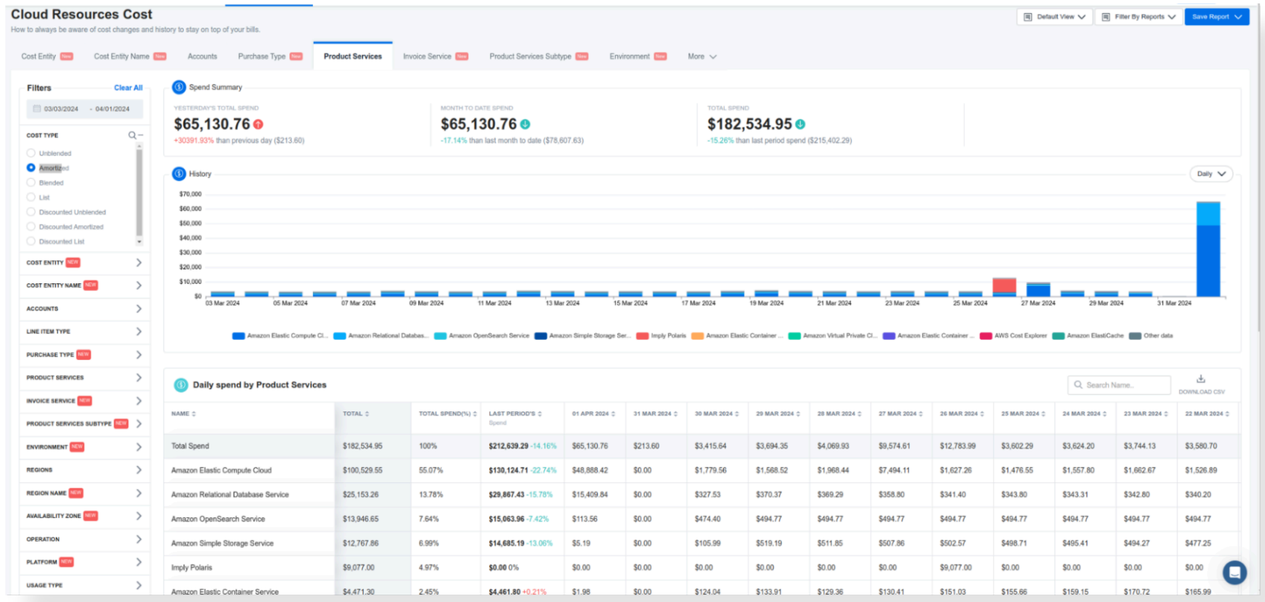
There’s no need for any other tools or services to gain visibility into your cloud costs. nOps transforms your CUR from millions of lines of data into the who, what, when and why of cloud spend.

See the hourly costs associated with each container, pod, and service within your clusters. You can slice spending by relevant dimensions such as workload, environment, resource type, team, pricing type, and more. Intuitive filters make it easy for engineering, finance, and business leaders to gain the insights they need, whether it’s pinpointing the workload spiking your EC2 bill or fairly splitting shared costs.

Understand your real hourly costs

It can be very difficult to understand the real-time impact of your commitments (Reserved Instance or Savings Plans) on the operational costs of your containers — these financial mechanisms are applied at runtime and reflected only when the bill is generated.

nOps Business Contexts automatically tracks your organization-wide commitments, showing you real hourly container costs after commitments are applied (amortized costs).



Click amortized view to see your real post-RI and SP costs

nOps is entrusted with over a 1.5 billion dollars of AWS spend, and was recently ranked #1 in G2's cloud cost management category. Learn more about the nOps cloud optimization platform by booking a demo today!

Book a Demo