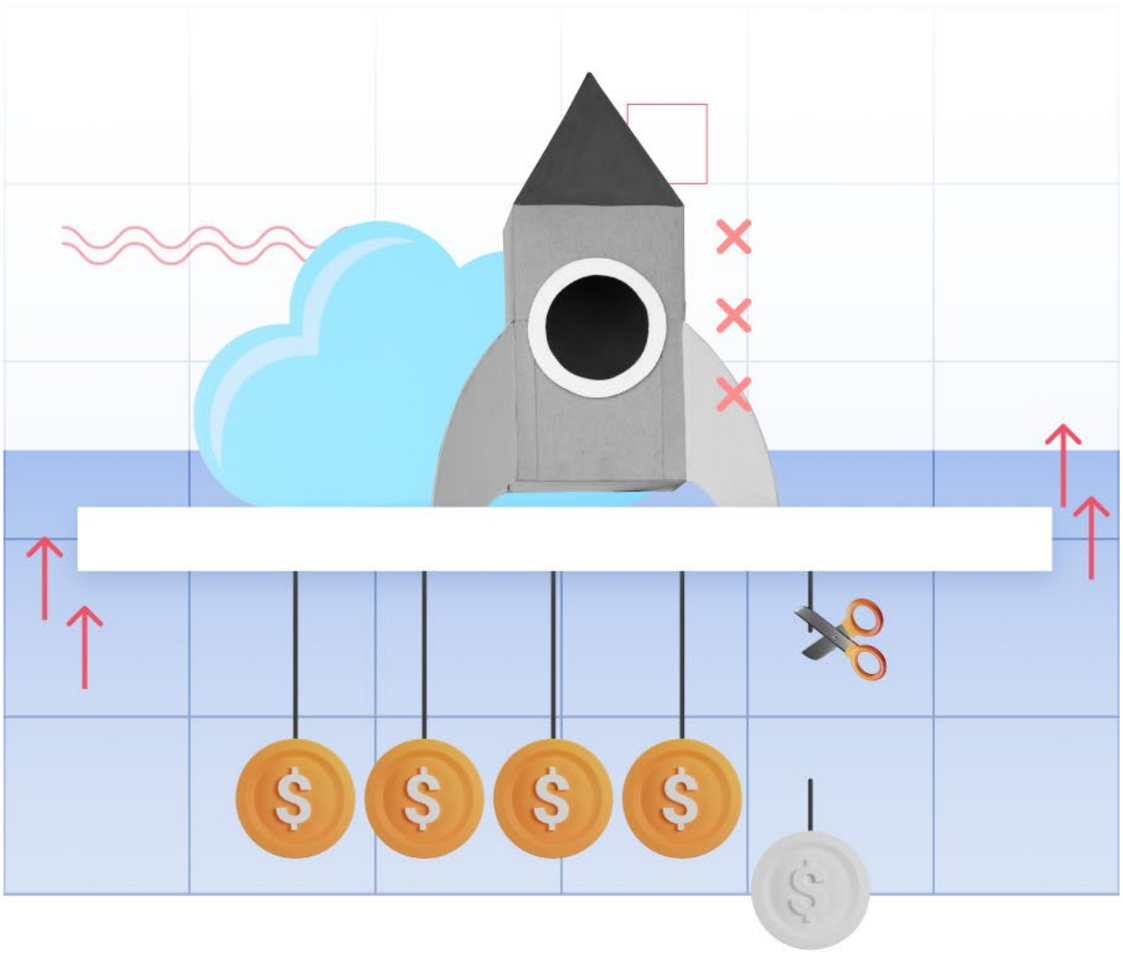
SIMFORM

AWS Cloud Cost Optimization Best Practices:

Learn how to save your IT costs from 10+ real-life examples



Contents

Segment, a popular customer data platform, empowers over 20,000+ businesses by enabling them with customer data to make better business decisions. However, in September 2018, Segment realized that substantial steps towards infrastructure cost reduction were required to boost their gross margin. The business metrics looked impressive, including revenue growth, customer churn, and new product attachments. However, the gross margin was a cause of concern when they were contemplating a new funding round.

One of the board members commented, "Your gross margin is a bit of a black eye on the whole economics of the business."

So how did Segment pull it off?
The company witnessed many cost-saving wins from pure infrastructural enhancements, including network, NSQs, CPU, and memory costs.
In addition, it found several places where systems were contributing significantly to the expenditures.

According to Segment's engineering leaders, more than 6000 of the total sources were either not connected to any destination or were directed to destinations with expired credentials. So they started a campaign to reduce the total number of sources without any associated destinations. Moreover, they asked their customers if they wanted to keep running any underutilized sources, and with that, they were able to save more.

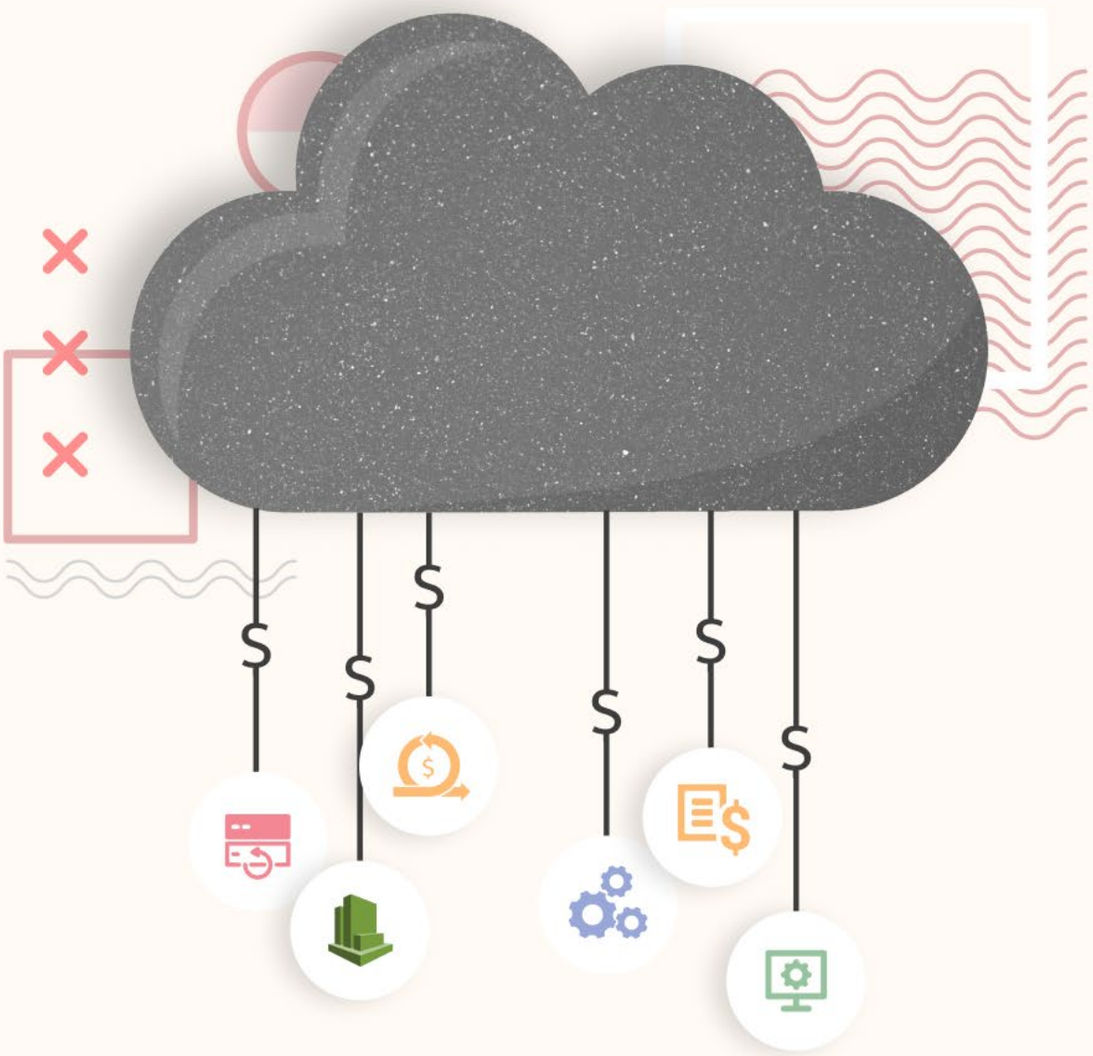
Here's what we learned from Segment's example:

- It's crucial to identify all areas of cost savings. It's impossible to act without measuring things.
- Once you identify the places, make a plan. Then, make everyone accountable for cost-cutting efforts.
- Build a repeatable monitoring process instead of a one-time fix.

What did Segment finally achieve? After all the optimizations, the gross margin increased by 20% over 90 days. The company put systems and monitoring to forecast the biggest spending areas instead of focusing on a one-time fix.

This ebook looks at real-world examples of adopting practices that focus on "cloud cost optimization," and practical ways to avoid skyrocketing your cloud bills.

01



Six common AWS cloud cost optimization mistakes

1. Keeping redundant backups

Amazon EBS snapshots are used by cloud administrators to back up data to Amazon S3 by taking point-in-time snapshots. Even though AWS has reduced the time and cost of creating snapshots by retaining the most recent versions, users still pay more by keeping unnecessary backups. As a result, snapshot storage costs have increased. So it's recommended to investigate modern backup capabilities based on your requirements.

2. Not utilizing AWS CloudWatch and Smart Alerts for AWS Costs Monitoring

AWS CloudWatch is a monitoring service that allows you to keep track of large datasets created by your AWS resources. Monitoring your infrastructure regularly allows you to visualize the data and take quick actions. Unfortunately, cloud practitioners frequently overlook the importance of integrating such critical monitoring tools with third-party performance monitoring tools.

CloudWatch allows you to create custom metrics, in addition to the standard metrics, like CPU utilization and network traffic. With AWS, it's also possible to set smart alarms that monitor resource usage thresholds and shut down resources when the metric crosses the threshold.

3. Not putting enough emphasis on developing a cost-conscious culture

Businesses often want to lower their cloud costs. However, they seem to miss the mark when recognizing scope and creating a cost-conscious culture. The well-architected framework of AWS focuses on fostering a culture of cost-cutting initiatives. Such frameworks aid in developing financial management practices and a better understanding of cloud costs.

It is recommended that organizations keep publicly visible cloud usage and cost dashboards and emphasize cloud cost optimization learning practices.

Case Example

Spotify decided to align engineering and business goals after discovering that its infrastructure costs were quite high and outweighed user acquisition. It decided to focus on cloud spending by cultivating a cost-conscious culture.

When engineering teams at Spotify realized that cloud costs were growing faster than income and revenue, they saw it as a major engineering problem and decided to build a specific product to handle it. They created an internal product to control cloud spending as part of it.

The engineering leaders at Spotify assist software engineers in making better resource allocation decisions, saving millions of dollars in cloud costs.

They created the Backstage platform as a developer portal to manage the entire software delivery supply chain, including all components, data, pipelines, and services, from concept to completion. Moreover, they put Backstage in charge of all engineering tasks, removing complexity, such as launching a Kubernetes cluster or provisioning a pipeline.

Because it is based on a service catalog model, Backstage quickly became a place to bootstrap cost insights; these cost insights also support the labels of cloud provider resources. The platform was created so that organizations with shared infrastructure services used by cross-functional teams can properly represent billing and differentiate costs.

As a result, Backstage became the central communication tool for the engineering teams. Spotify wanted cost optimization where developers already spend most of their time rather than encouraging them to use a third-party system, so it became the home for cost insights.

4. Underestimating automation needs

Manual tracking of resources, configurations, and technical aspects in large-scale projects often results in complex and time-consuming processes. On the other hand, automation services are designed to bring more visibility into processes in a short time.

Organizations that want to increase revenue should focus on practices that allow them to spend less time managing infrastructure and more time developing core products. Automating application deployment, configuration monitoring, and other time-consuming tasks, for example, could help you create more efficient products. You can also use auto-scaling capabilities to save money on applications that require on-demand resources by utilizing automated mechanisms.

Case Example

Compared to other organizations that set budgets and impose strict spending limits, Netflix, the world's largest streaming service, took a different approach to managing its data infrastructure costs. Netflix's data engineers wanted to bring cost transparency as close to decision-makers as possible. As a result, they developed a custom data efficiency dashboard that serves as a single source of truth for cost and usage patterns.

The cloud costs were aggregated across platforms to create usage and cost visibility and get a holistic view of costs for each team.
Netflix engineers used a customized approach to break down AWS costs because AWS billing is divided into services (EC2, S3, and so on).Although AWS tags can

NETFLIX

separate these services, that wasn't enough to provide granular visibility into infrastructure costs.

For EC2-based platforms, they looked at the platform's bottleneck metrics, such as memory, storage, and IO. In addition, platform logs and various REST APIs were used to identify the consumption of bottleneck metrics per data resource.

They also created a dashboard view that provides context for taking data-driven actions. As a result, when dealing with multiple data platforms, Netflix emphasizes consolidating cost and usage patterns to create feedback loops using dashboards. This proves to be providing a great outcome in tackling efficiency.

5. Underestimating automation needs

The AWS platform has a lot to offer organizations regarding cloud services, storage, and pricing model flexibility. Frequently, businesses lack active management and knowledge resources about the best pricing plans for their needs. Predicting the right pricing models is difficult without knowing what resources you'll need and how you'll use them.

AWS offers a wider range of pricing options such as pay-as-you-go, save when you commit, and volume-based discounts. It has various pricing models based on services and database types. For example, for Amazon EC2 instances, you can choose instances types from the on-demand model, such as spot instances, saving plans, dedicated hosts, and reserved instances.

You can choose on-demand instances based on your business needs; for example, users who prefer the flexibility of EC2 at a low cost with no upfront payment or long-term commitment should choose the on-demand model. It's also appropriate for applications with erratic traffic.

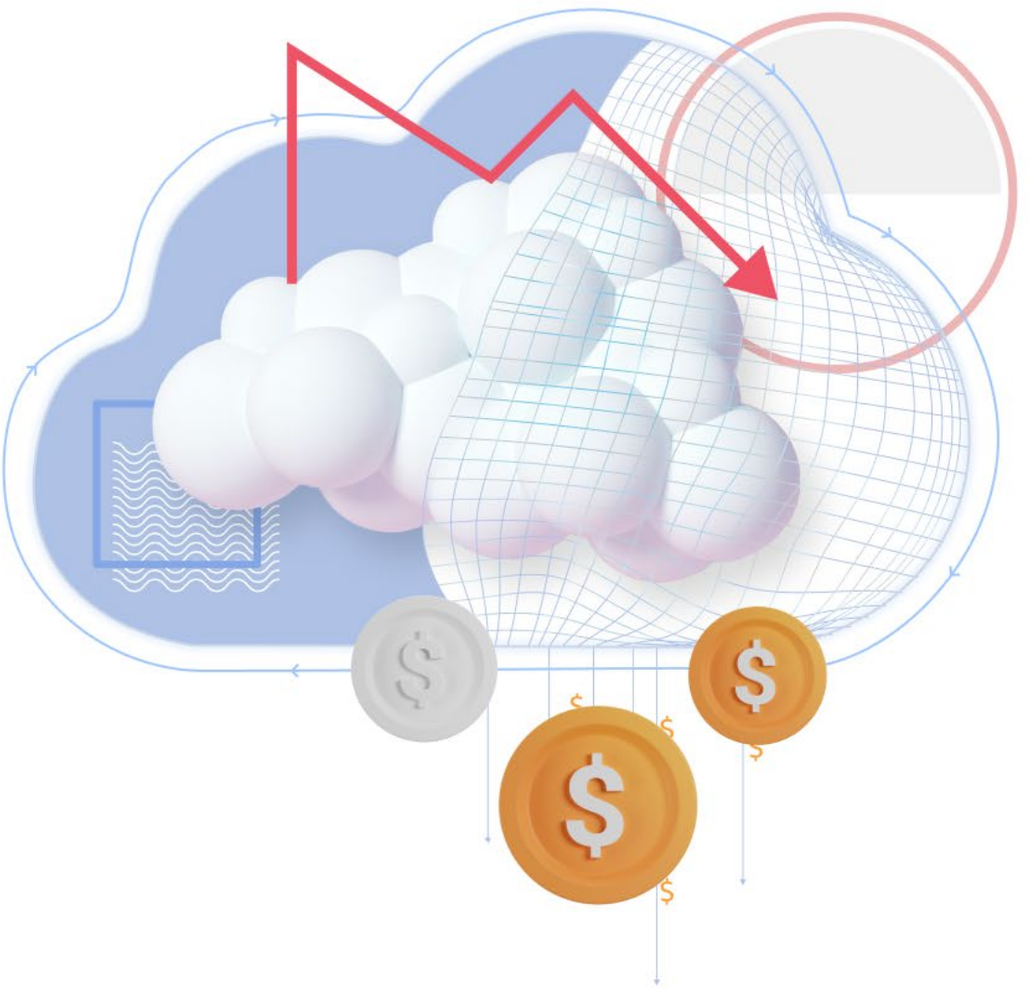
Other models, meanwhile, are tailored to specific use cases, such as applications that require flexible start and stop times, applications that require reserved instances, and so forth.

6. Not monitoring resources properly

Not utilizing the pay-as-you-go model is a common blunder made by AWS customers, particularly when performing application testing and in temporary development environments. They also undervalue the AWS platform's automation tools and auto-scaling capabilities, resulting in the waste of valuable financial resources. Moreover, they frequently use large storage pools and database workloads without a proper monitoring mechanism, which harms the bottom line.

For example, some idle EC2 instances used in the past for testing are left in an active state and no longer in use.

02



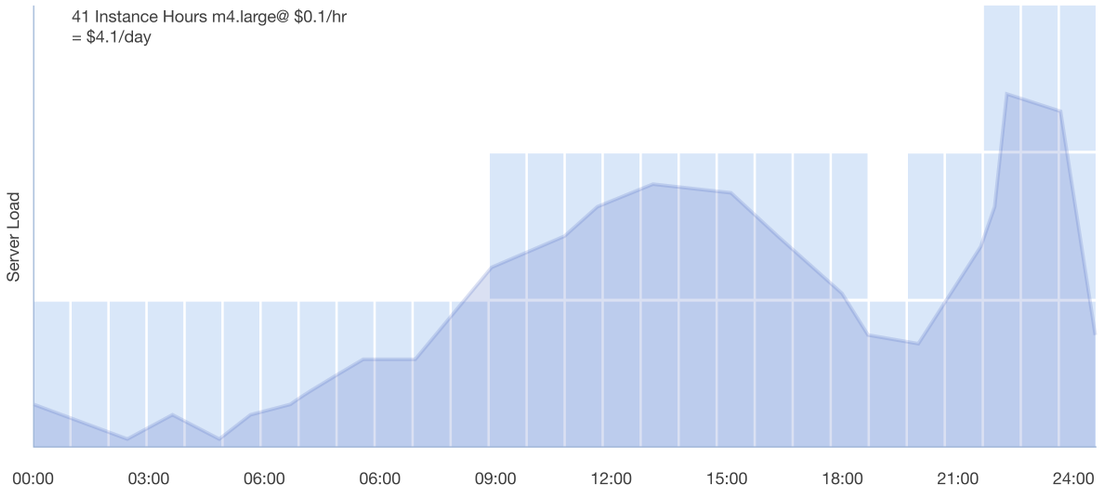
Best practices for optimizing your cloud infrastructure costs

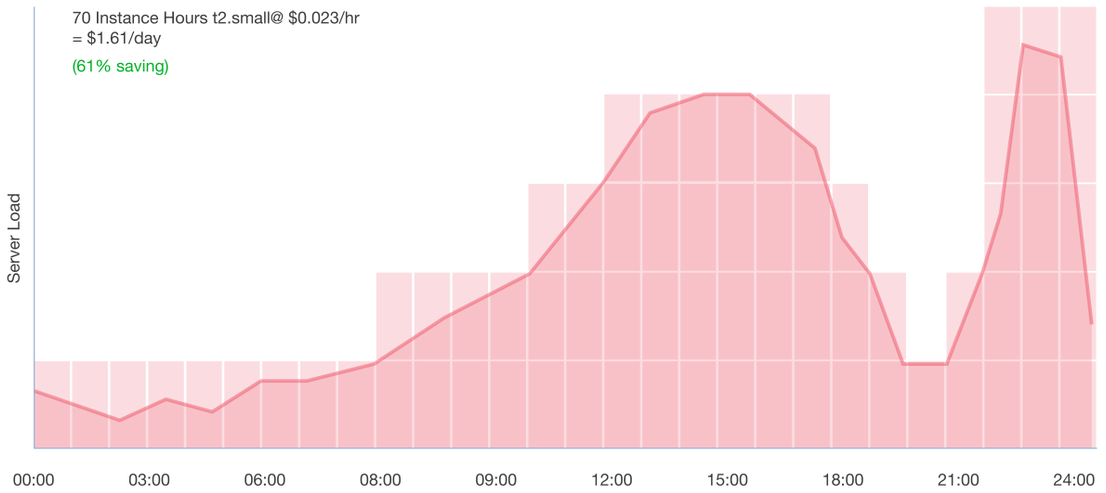
> Good intentions never work, you need good mechanisms to make anything happen”

Jeff Bezos
Founder and Chief Executive Officer Amazon.com, Inc.

1. Downsize under-utilized instances

Downsizing one size in an instance family reduces costs by 50 percent. Let’s take one example of m4.large and t2.small.





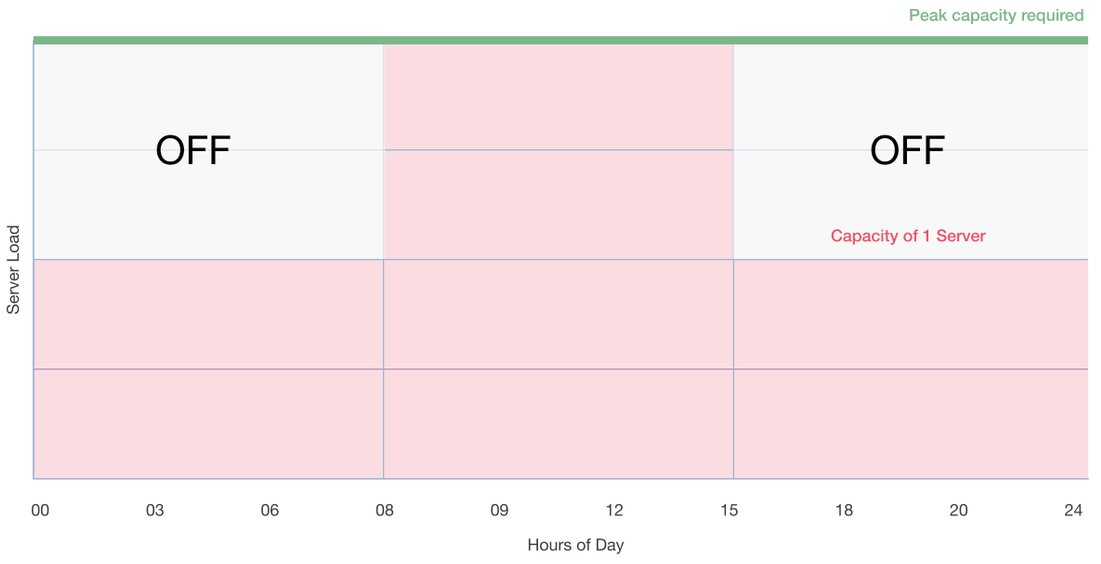
The image shows that m4.large is utilized less than 50% between 0 to 9 am. Server usage increases from 9 am to 2 pm. Here it requires two m4.large instances. Again, usage starts decreasing till 6 pm and increases the night. As per the graph, it is clear that a single m4.large can't be utilized even 50% of its capacity during some specific period. It costs $4.1/day. Under-utilized instances should be considered as candidates for downsizing either one or two instance sizes.

Remember that downsizing one size in an instance family optimizes costs by 50 percent.

According to the server load graph, we can use t2.small for utmost utilization. As t2.small has a half compute capacity as m4.large, 70 t2.small instances are required instead of 41 m4.large instances. Using t2.small instances, decision-makers can save more than 60% of the instance cost.

2. Turn off idle resources

Organizations use instances according to the highest peak of requirement, which is not constant. Due to variations in the requirement, organizations have to pay for non-utilized instances. To fully optimize cloud spend, turn unused instances off. The below image shows how servers load varies on a particular day and then turn off instances.



As you can see in the above image, server load can be handled by 1 instance, so you can turn off 2nd instance to save the cost. During the next 8 hours, server load increases and requires two instances. Again, server load decreases from 16 to 24, and you can turn off 2nd instance.

Production instances should be auto-scaled to meet the demand. Shutting down development and test instances in the evenings and on weekends when developers are no longer working can save up to 65 percent or more of costs. Also, instances for training, demos, and development must be terminated upon projects completion.

Case Example

Union Bank of the Philippines will be the first major bank to be fully cloud-hosted. **"We're a bank, but we're a technology company first,"** said Dennis Omila, UnionBank's chief information and operations officer, on the importance of scalability and cost optimization for cloud-based IT models.

> We no longer worry about capacity, slowdowns, or potential data loss with SAP on AWS."

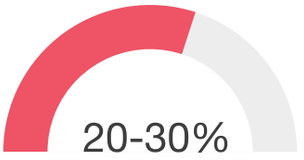
Dennis Omila
Chief Information and Operations Officer, Union Bank of the Philippines

The Bank realized that running SAP on AWS would be more cost-effective as it aimed to serve 50 million Filipinos by providing digital payment and cash management solutions.

It also followed AWS's Well-architected framework and security practices to ensure cost optimization and set up the SAP infrastructure as part of the project.

The company focuses on application and platform management through continuous improvement, automation, and cost reduction.
It also focused on right-sizing compute instances in AWS as part of cost optimization, which reduced the cost of running SAP by 20-30%.

- SAP operating costs were reduced by 20-30% thanks to cost optimization strategies and well-architected framework best practices.



- System uptime and access have improved, and the system scales to support 100,000 new accounts per month.

3. Delete unused EBS volumes

Keeping track of unused EBS volumes and deleting those contributing to AWS spending is another way to keep costs down for underutilized resources. Because EBS volumes are independent of Amazon EC2 compute instances, they tend to persist even if the associated EC2 instance is terminated.

As EBS volumes are physically linked to EC2 instances as storage devices, you will be charged even if the associated instances are no longer in use.

You must choose the “Delete on Termination” option when launching them in order for them to be terminated. In addition, if no workflows are in place to automatically delete EBS volumes, the instances being spun up and down may leave some EBS volumes.

Though this is a great way to stop before it starts charging you, what if you already have volumes attached to the instances in your AWS account?

You can identify and delete unused EBS volumes in a variety of ways, including

- Using AWS’s well-designed framework lens
- Using the Amazon Web Services Console
- Detecting unused EBS volumes with a Python script
- In Java, using the AWS Lambda function

Before deleting a volume with AWS Console, you must first detach it and check its status in the console.

- When a volume is attached to an instance, it will be marked as “in-use.”
- If you detach a volume from an instance, it will appear in the “available” state, and you can delete it.

For example, if you are planning to do it with Python script, you can identify and delete the unused volumes in three steps:

➡ List and describe EBS volumes in an AWS account and region

```
import boto3

sess = boto3.Session(
    # TODO: Supply your AWS credentials & specified region
here
  aws_access_key_id='MYAWSACCESSKEYID',
  aws_secret_access_key='MYSECRETACCESSKEY',
  region_name='us-east-1', # Or whatever region you want
)
```

Next, we make a call to get all EC2 volumes:

```
ec2 = sess.resource(‘ec2’)
volumes = ec2.volumes.all()
```

This will return the list of all EC2 volumes in the specified region and account. This will return a collection of EBS volume objects.

Filter the unattached EBS volumes

Next, we can filter this list and add the ones that are unattached to a termination list

`to_terminate.` Check out the code below:

```
to_terminate=[]
for volume in volumes:
    print('Evaluating volume {0}'.format(volume.id))
    print('The number of attachments for this volume is
{0}'.format(len(volume.attachments)))

    # Here's where you might add other business logic for
deletion criteria
    if len(volume.attachments) == 0:
        to_terminate.append(volume)
You can also decouple the logic of filtering the deletion
logic and take different actions with different attributes.
```

For example, you might delete all the unattached volumes and send a Slack notification for all the volumes that aren't encrypted.

Delete the unattached volumes

To delete the volumes, you need to check for the empty condition as each volume has a delete() method and then delete the volumes on the termination list.

`to_terminate.` Check out the code below:

```
if len(to_terminate) == 0:
    print ("No volumes to terminate! Exiting.")
    exit()

for volume in to_terminate:
    print('Deleting volume {0}'.format(volume.id))
    volume.delete()
```

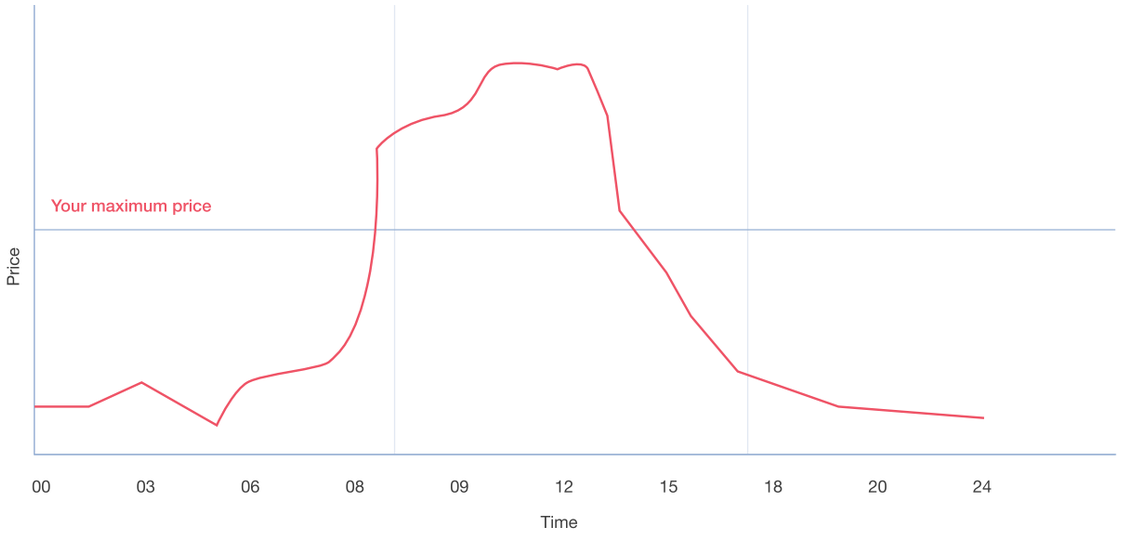
Source

4. AWS spot instances

On the AWS spot market, the traded products are Amazon EC2 instances, and they're delivered by starting an EC2 instance. It is very common to see 15-60% savings, and in some cases, you can save up to 90% with spot instances.

How does it work?

You set a maximal bid price and optionally a period up to 6 hours. The price you pay is the spot price each hour. When this price goes above your specified maximum bid price, the instance is terminated. Be careful that the new console for spot creates you a fleet, and even terminating the instance yourself, the fleet remains open. You need to cancel the "spot request" (the fleet), as well as terminate the instance(s) when you are done. The instances are guaranteed to remain active for the specified amount of time up to 6 hours. See the below image for more understanding.



There are many different spot markets available. A spot market is defined by:

- Instance type (e.g. m3.medium)
- Region (e.g. eu-west-1)
- Availability Zone (e.g. eu-west-1a)

Each spot market is offering a separate current price. So when using spot instances, it is an advantage to use different instance types in different availability zones or even regions, as this allows you to noticeably lower costs.

Case example

Ula, an Indonesian B2B eCommerce marketplace application, used Amazon EC2 spot instances to implement a cost-effective and scalable solution for small businesses.

ula

> AWS continues to guide us on how to build our business while optimizing cost along the way.”

Samuel Pamudji
Engineering Manager, Ula

The goal of this app was to make the user experience as simple and useful as possible, taking into account a small business owner's basic mobile devices and their struggle with poor network connections.

Ula was launched in January 2020 and has accrued a user base of over 100,000 in the last two years. When the company's founders decided to use Amazon Web Services (AWS) because of their previous success with the platform, they prioritized the application's scalability and high availability while keeping cost-effectiveness in mind.

Ula planned for further expansion across Indonesia and other Southeast Asian countries and adopted a fully containerized approach, given that business volume had increased by 300 times since the company's launch. All of it was successfully implemented using Amazon Elastic Container Service (Amazon ECS) Spot instances in development, staging, and production environments.

Aside from that, the startup made use of AWS's well-architected tool to ensure the high availability and reliability of applications in the production environment, resulting in a 99 percent uptime.

Case Example

Fork Media Group(FMG), a media-tech company specializing in contextual advertising, deployed artificial intelligence and machine learning tools to analyze content sentiments before presenting an ad to their customers.

It works with top international and local online publications in India, Southeast Asia, and the Gulf Cooperation Council to serve ads to 7-9 million unique web visitors per day (GCC).

FMG chose AWS to host its ad server databases in order to maximize the platform's capabilities and visibility. It dynamically scaled its capacity to meet the demands of traffic spikes using Amazon EC2 and Amazon EC2 spot instances, resulting in a 30% cost reduction and a 50% decrease in ad delivery latency.

5. Minimizing data transfer costs

Make sure your Object Storage and Compute Services are in the same region because Data transfer is free in the same region. For example, AWS charges $0.02/GB to download the file from another AWS region. If you perform a lot of cross-region transfers, it may be cheaper to replicate your Object Storage bucket to a different region than download each between regions each time.

Let's understand it with AWS S3's example.

1GB data in us-west-2 is anticipated to be transferred 20 times to EC2 in us-east-1. If you initiate an inter-region transfer, you will pay $0.20 for data transfer (20 * 0.02). However, if you first download it to mirror the S3 bucket in us-east-1, you pay $0.02 for transfer and $0.03 for storage over a month. It is 75% cheaper. This feature is built into S3 called cross-region replication. You will also get better performance along with cost-benefit.

Use AWS content delivery network called CloudFront If there are a lot of downloads from the servers stored in S3 (e.g., images on consumer site).

There are CDN providers such as Cloudflare that charge a flat fee. If you have a lot of static assets, then CDN can save money over S3, as just a tiny percent of original requests will hit your S3 bucket.

6. Use AWS compute savings plans

Compute Savings Plans are applied to all EC2 and Lambda instances, regardless of instance family, region, size, or tenancy. If you use AWS for one year, you can save up to 54% compared to on-demand pricing, and you don't have to pay anything upfront.

Use AWS Cost Explorer to keep track of your AWS cloud costs and usage monthly or daily. By signing up for these plans, your compute usage is automatically charged at the discounted Saving Plans prices, and any use beyond the commitment is charged at regular on-demand rates.

Case Example

What Airbnb accomplished in just nine months is truly remarkable and deserves to be called a cost-conscious culture in the proper sense. Airbnb reduced its hosting costs by $63.5 million over the years, resulting in a 26 percent decrease in its cost of revenue.

It established a cost-conscious culture from the top down, as well as various organizational efforts to control AWS costs.

What made Airbnb a model for companies focusing on "cost-consciousness"?

- Its emphasis is on generating cost data through the use of specialized exploration and visualization platforms such as Apache Superset. Terraform was also used as a configuration tool to ensure that AWS resources were assigned to the projects.
- Furthermore, it implemented cost-conscious practices, such as ingesting the Cost and Usage Report, which is widely regarded as the most accurate and comprehensive source of AWS billing.
- AWS Saving Plan was used in the majority of their computing resources, to optimize on-demand charges and keep utilization appropriate by allowing certain workloads to be moved on and off Saving Plan.

7. Design workloads for scalability

For any public cloud, scalability is critical. Scalability uses event-driven compute instances like AWS Lambda or containers like Google Container Engine to scale core services for important workloads, such as microservices. These techniques are intended to utilize more computing when necessary . Once the requirement increases, those related resources are released for reuse.

Suppose you are running a dynamic platform with various products and services. In that case, you need an auto-scaling capacity that delivers faster solutions and a system that doesn't crash with unpredictable website traffic. Or imagine a streaming platform like Spotify where artists post their music albums, and the platform plays a critical role in generating revenue.

You can design your workloads for scalability with AWS EC2, Elastic Load Balancing, and auto-scaling. It's easy to serve billions of users a day in a few months and deploy new features and code on a daily basis using these instances and services.

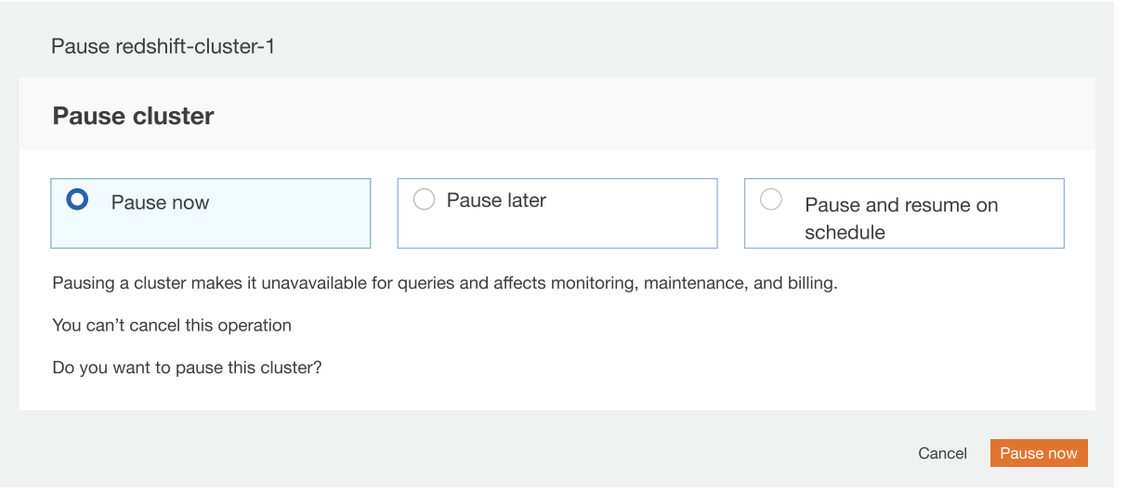
For example, Beat, a top ride-hailing app in Latin America, started experiencing outages in its system after reaching hypergrowth in 2019. It had over 700,000 drivers and more than 20 million users across 6 countries. Initially, the company managed the architecture with Amazon EC2, Amazon Aurora, AWS Auto Scaling, and Amazon ElastiCache. But at some point, it reached an end where the teams could only scale vertically, which led to frequent bottlenecks and hours-long downtime.

Beat reevaluated its architecture by dividing the traffic into as many instances as possible and used Amazon ElasticCache for Redis to distribute the load balance. By using Amazon ElastiCache for Redis and AWS Enterprise Support, Beat enabled the cluster mode, improved the operations, performed horizontal scaling, reduced compute load by 90%, and achieved zero time virtually.

8. Identify less utilized Amazon RDS, Redshift instances

Identify DB instances that have no connection over the last 7 days using the Trusted Advisor Amazon RDS Idle DB instances check. You can reduce costs significantly by stopping these DB instances using the DB instance stop and start feature in Amazon RDS for up to 7 days. This is one of the cost-effective ways to use Amazon RDS databases, as you can limit the costs when you are not using them.

For Redshift, AWS allows you to pause the clusters which have had no connections for the last 7 days, and there is less than 5% cluster-wide average CPU utilization for 99% of the previous 7 days. Identify underutilized clusters over the previous 7 days using Redshift clusters check.



Pause Underutilized Redshift Clusters

Let's understand cost optimization with this example case of before and after optimization is performed.

Before Cost Optimization

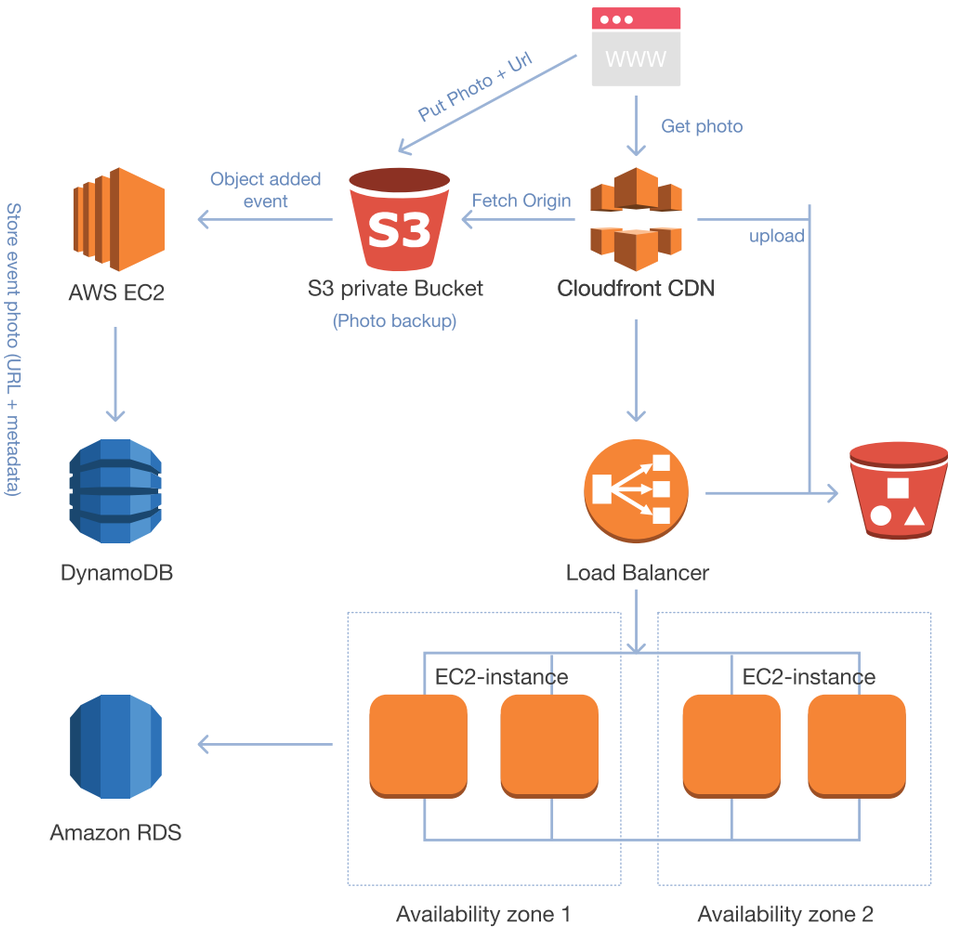
This section will analyze the actual cost of running AWS services for our demo application.

Let's assume our example is an image upload processing application in which users can upload images, resize, crop, and take backups. Our goal is to optimize cloud costs by making some modifications and choosing a suitable instance family.

Following AWS services are used in the application architecture.

- Cloud front CDN
- AWS EC2 (10 instances)
- Amazon RDS
- Amazon S3
- Amazon DynamoDB
- AWS application load balancer
- AWS EC2 (1 instance)

Below is the architecture of our image upload application:



We'll consider the application architecture parts that contribute and are relevant for specific cost calculations. Finally, we'll try to keep the calculation simple for better understanding.

Suppose the application is in the US-EAST-1 region, all prices are calculated for this geographic region.

Consider that the application handles approximately 3 million requests per month. Therefore, the free tier from all the services has been excluded, and pricing is within the US-EAST-1 region.

CloudFront CDN

We'll require 100 GB storage for storing the images, for which the CloudFront CDN pricing is 0.085 per 100 GB, which will cost $ 8.5.

AWS EC2

AWS allows users to select pricing models based on their demand.

- In the case of our application, images are going to be uploaded in bulk. So we are using 10 large T2 instances which costs $0.032/ hr each.
- If we are using it for 24 hours and 30.5 days, it will cost us= 10 x $ 0.038 x 24 x 30.5= $ 278.16

Amazon S3

For storage, we will be using S3 to store static content like HTML, CSS and Java.

The S3 costs $0.02 for standard storage of 1 GB. In our case, the storage is 100 GB so it will cost $2.

Amazon RDS

Here are pricing examples (applicable to MySQL DB instances that are deployed in a single Availability Zone (AZ) within the US East region):

- **No upfront payment**—$0 upfront cost, $8.76 monthly rate (effective hourly rate of $0.012), 29% saving compared to on-demand
- **Partial upfront payment**—$50 upfront cost, $4.161 monthly rate (effective hourly rate of $0.011), 33% saving compared to on-demand
- **Full upfront payment**—$98 upfront cost, $0 monthly rate (effective hourly rate of $0.011), 34% saving compared to on-demand

In our example, we are utilizing No upfront payment cost which is $8.76/ month.

AWS Application Load Balancer

- AWS pricing for Application Load Balancer:
- $0.0252 per ALB-hour (or partial hour)

$0.008 per LCU-hour (or partial hour)

Here, the LCU measures dimensions on which the Application Load balancer processes traffic.

- The main dimensions considered are new connections, active connections, processed bytes, and rule evaluations.
- New connections are described as newly established connections per second.
- Active connections are the number of active connections per minute.
- Processed bytes are the number of processed bytes by the load balancer in GBs for requests and responses.
- Rule evaluations are the number of rules processed by the load balancer and the request rate. These rules are evaluated based on priorities from the lowest value to the highest. It's used for evaluating the priorities of requests and the availability of servers.

For our use case, at 35,000GB per month (the sum of ingress and egress), this averages at 49 GB per hour so an average of 49 LCUs.

That gives an approximate monthly cost of $300 per month: ($0.0252 + $0.008 * 49) * 24 hours * 30 days.

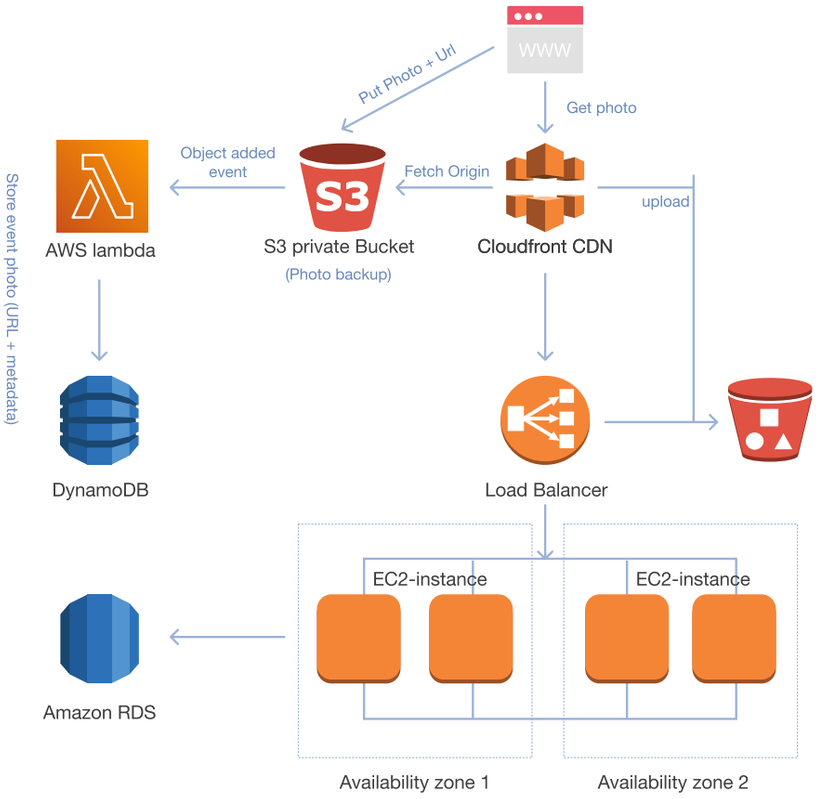
AWS EC2 (1 instance)

In our example, we are using 1 large T2 instance of EC2 which costs $0.038/ hr. If we use it for 24 hours and 30.5 days, the total cost will be $27.82.

Total Costs Before Optimization	
Cloudfront CDN	$ 0.085 x 100 = 8.5
AWS EC2 (10 instances)	10 x $ 0.038 x 24 x 30.5= $ 278.16
Amazon S3	$0.02 x 100= $2
Amazon RDS	$8.76
AWS Application Load Balancer	$300
AWS EC2 (1 instance)	1 x $0.038 x 24 x 30.5 = $27.82
Total	**$ 625.24**

After cost optimization

We will take each of the services one by one and see how we can optimize costs with some changes in our approach.



CloudFront CDN

One of the most common uses of CloudFront is delivering web and media content stored in an S3 bucket or EC2 instance to clients worldwide.

While optimizing CloudFront services, we have the following options to save costs:

WWWTotal cost optimization for CDN (30%) = **$ 5.44**

AWS EC2

In EC2, we have two options to reduce costs:

- We can run instances for 100% of the time and use 10 x T2 Large instances = **$120**
- Or we can autoscale instances from 2 to 10, 2 x large T2@100% utilization + 8 x small T2@15 % utilization = **$98**

Amazon S3

For S3 also, we can optimize costs using two ways:

- In our example, if we use Infrequent Access Storage (IAS) to cut the storage cost by a factor of 2, we can save upto 50 % costs **= $1**
- If we use Amazon Glacier storage class, which has same durability as S3, the costs can be cut upto 30% **= $1.40**

Amazon RDS

By rightsizing Amazon RDS instances,in our application, we can get 30% cost savings **= $ 6.13**

AWS Application Load Balancer

The only opportunity to save cost on AWS application load balancer is to remove any idle resources.

We can save costs around 20% by eliminating idle resources

= 240 $

AWS Lambda

Instead of using AWS EC2 T2 large instances to handle small and moderate workloads, we'll replace them with AWS Lambda since EC2 is costly. We can save additional costs by choosing AWS Lambda.

For simplicity, let's assume your application processes 3 million requests per month. The average function execution duration is 120 ms. You have configured your function with 1536 MB of memory, on an x86 based processor.

Monthly compute charges	
The monthly compute price is $0.0000166667 per GB-s and the free tier provides 400,000 GB-s.	
Total compute (seconds)	= 3 million * 120ms = 360,000 seconds
Total compute (GB-s)	= 360,000 * 1536MB/1024 MB = 540,000 GB-s
Total compute – Free tier compute	monthly billable compute GB- s
540,000 GB-s – 400,000 free tier GB-s	140,000 GB-s
Monthly compute charges	140,000 * $0.0000166667 = $2.33

Monthly request charges

The monthly request price is $0.20 per 1 million requests and the free tier provides 1 million requests per month.

Total requests – Free tier requests = monthly billable requests

3 million requests – 1 million free tier requests = 2 million monthly billable requests

Monthly request charges = 2M * $0.2/M = $0.40

Total monthly charges

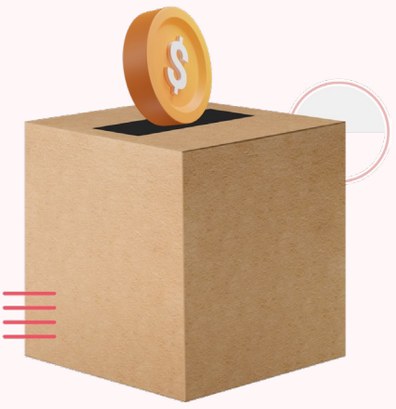
Total charges = Compute charges + Request charges = $2.33 + $0.40 = $2.73 per month (98% costs savings)

Total cost optimization for CDN (30%)	$ 5.44
AWS EC2	$98
Amazon S3	$1.40
Amazon RDS	$6.13
Application load balancer	240 $
AWS Lambda	$2.33 + $0.40 = $2.73 per month (98% costs savings)
Total cost saved	**$ 353.17**

Cost-saving tips

➜ Use discounted instances

For Amazon Web Services, it is critical to comprehend the discount options for instances in order to optimize cloud costs. AWS Reserved Instances offers a discount in exchange for a one-year or three-year commitment, with the longer commitment providing a greater discount. Depending on the RI(Reserved Instance) term, instance type, and region, discounts range from 24 to 75 percent.



➜ Delete or migrate unwanted files after a certain date

Data deletion and migration between storage types can be configured programmatically by cloud architects. Long-term storage costs are drastically reduced as a result of this. Lifecycle Management is a feature that all of the major cloud vendors offer. Active data, for example, can be stored in Azure Blob Standard storage. However, if certain data begins to show signs of infrequent access, users can set up rules to move that data to Azure Cool Blob storage, which has a lower storage rate.

Many deployments use Object Storage for log collection. You may automate deletion using life cycles. Delete objects 7 days after their creation time. E.g., if you use S3 for backups, it makes sense to delete them after a year. Similarly, you can use the Object Lifecycle management feature in Google cloud and the expiration of Blobs in Azure.

➜ Compress data before storage.

Compressing data reduces your storage requirements. Subsequently, reducing the cost of storage. Using fast compression algorithms such as LZ4 gives better performance. LZ4 is a lossless compression algorithm, providing compression speed at 400 MB/s per core (0.16 Bytes/cycle). It features an extremely fast decoder, with a speed of multiple GB/s per core (0.71 Bytes/cycle). In many use cases, it makes sense to use compute-intensive compressions such as GZIP or ZSTD. So, compressing data will reduce your cloud waste significantly.

➜ Take care of incomplete multipart uploads

Many partial objects were uploaded to Object Storage but were interrupted during the process. Even 1% of incomplete uploads can waste terabytes of space if you have a petabyte Object Storage bucket. After 7 days, you should clean up any incomplete uploads.

➜ Reduce the cost of API access

The cost of API access can be reduced by using batch objects and avoiding a large number of small files. Since API calls are charged per object, regardless of its size. Uploading 1-byte costs the same as uploading 1GB. So usually, small objects can cause API costs to soar.

For example, in AWS S3, PUT calls cost $0.005/1000 calls. So uploading a 10GB file in 5MB chunks will cost roughly $0.01. Whereas for 10KB file chunks, it will cost approximately $5.00.

Similarly, it is bad to use CALL/PUT options with tiny files using S3. It makes sense to use batch objects or a database such as DynamoDB or MySQL instead of S3. 1 writes per second in DynamoDB costs $0.00065/hour. Assuming 80% utilization, DynamoDB will cost $0.000226/1000 calls. This is 95% cheaper to use DynamoDB compared to AWS S3.

The S3 file names are not a database. Relying too much on S3 LIST calls is not the right design and using a proper database can typically be 10-20 times cheaper.

➜ Cache storage strategically

Use memory-based caching, such as AWS ElastiCache. Caching improves data accessibility by moving important or frequently accessed in-memory instead of retrieving data from storage instances. This can reduce the expense of higher-tier cloud storage and improve the performance of some applications and websites.

It gives the best results for performance-sensitive workloads which run in remote regions or when efficient replication is needed for resilience.

➜ Use auto-scaling to reduce spending during off-hours.

Most apps have busier and slower periods throughout the week or day. Taking advantage of auto-scaling could save you some money during slow periods.

These deployment types all support auto-scaling:

Cloud Services

App Services

VM Scale Sets

(Including Batch, Service Fabric, Container Service)

Scaling could also mean shutting your app down completely. You can use “AlwaysOn” feature provided by App Services that controls if the app should shut down due to no activity. You could also schedule shutting down your dev/QA servers with something like DevTest Labs.

➜ Analyze Amazon S3 usage and reduce costs

Leverage recommendations of S3 analytics and analyze storage access patterns on the object data set for 30 days or longer. AWS offers you Amazon S3 Glacier Storage classes which are purpose-built for archiving the data, and it provides you the lowest cost archive storage in the cloud. Utilize S3 Infrequently Accessed for practical recommendations of reducing storage costs effectively.

Concluding remarks!

While cloud vendors are constantly evolving processes, technologies, and platforms, businesses should invest more in cloud platforms and recognize the importance of best practices and tailored cloud strategies.

Furthermore, if you're already an AWS cloud customer, you should take advantage of its programs, which include a well-architected review framework and immersion days organized by Advanced Consulting Partners like Simform.

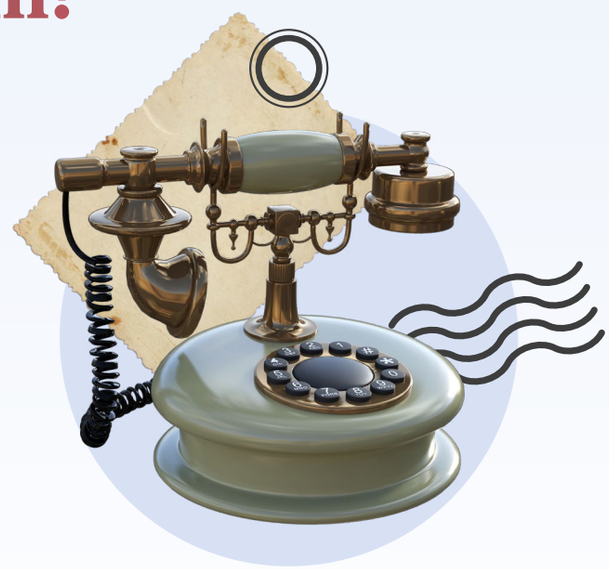
What does it mean to be an AWS customer in general?

- Cloud migration strategy that is tailored to your needs
- AWS Partner Programs is being used.
- AWS Certified Partners are a great way to get started.
- Making a business case for cloud computing (aligning business goals with cloud strategies)
- Using various pricing models
- Implementation of DevOps and Agile methodologies

We are Simform!

With over 10+ years of experience under our belt, we are more than ready to supercharge your project with extraordinary code. 10 years ago, Simform was one person. Today, we're over 600+ people strong and growing.

Simform is a custom software development powerhouse. Let's get in touch to discuss your next project!



Contact Us

- Managed software engineering teams
- Quality Engineering and Testing
- Cloud native development and modernization
- DevOps