**The Default Heuristic**

The Default Heuristic technique helps solve our struggle for searching the best AWS services and features.

**We'll cover the following**

* [Optimization fallacy](https://www.educative.io/module/lesson/the-good-parts-of-aws/BnXwMv9zXON#Optimization-fallacy)
* [The default heuristic](https://www.educative.io/module/lesson/the-good-parts-of-aws/BnXwMv9zXON#The-default-heuristic)
* [Our default choice](https://www.educative.io/module/lesson/the-good-parts-of-aws/BnXwMv9zXON#Our-default-choice)

**Optimization fallacy**[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/BnXwMv9zXON#Optimization-fallacy)

Chasing the best tool for the job is a particularly insidious trap when it comes to making progress—especially at the beginning of a project. We consider the relentless search for the best tool to be an **optimization fallacy**—in the same category as any other premature optimization.

Searching for the optimal option is:

* Almost always expensive.
* Any belief that we can easily discover the best option by exhaustively testing each one is delusional.

To make matters worse:

* We developers tend to enjoy tinkering with new technology and figuring out how things work, and this only amplifies the vicious cycle of such a pursuit.

**The default heuristic**[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/BnXwMv9zXON#The-default-heuristic)

Instead of searching for the best option, we recommend a technique we call the default heuristic. The premise of this heuristic is that when the cost of acquiring new information is high and the consequence of deviating from a default choice is low, sticking with the default will likely be the optimal choice.

But what should your default choice be?

* It should be any option that gives you very high confidence that it will work.
* It is something you’ve used before.
* Something you understand well.
* Something that has proven itself to be a reliable way for getting things done in the space you’re operating in.
* It doesn’t necessarily have to be the theoretical best choice.
* It doesn’t have to be the most efficient. Or the latest and greatest.
* It simply needs to be a reliable option to get you to your ultimate desirable outcome.
* It should be very unlikely to fail you; you have to be confident that it’s a very safe bet. In fact, that’s the only requirement.

With this heuristic, you start making all your choices based on your defaults. You would only deviate from your defaults if you realize you absolutely have to.

Let’s take a small quiz to check your understanding of the default heuristic.

## Our default choice[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/BnXwMv9zXON#Our-default-choice)

When you start with little experience, you might not have a default choice for everything you want to do. In this course:

* We’re going to share our own default choices when it comes to AWS services and features.
* We’re going to explain why some things became our defaults, and why other things we don’t even bother with.

We hope this information will help you build or supplement your basket of default choices so that when you take on your next project you will be able to make choices quickly and confidently.

In the next chapter, we will look at all the good parts of AWS one by one.

# Database: DynamoDB

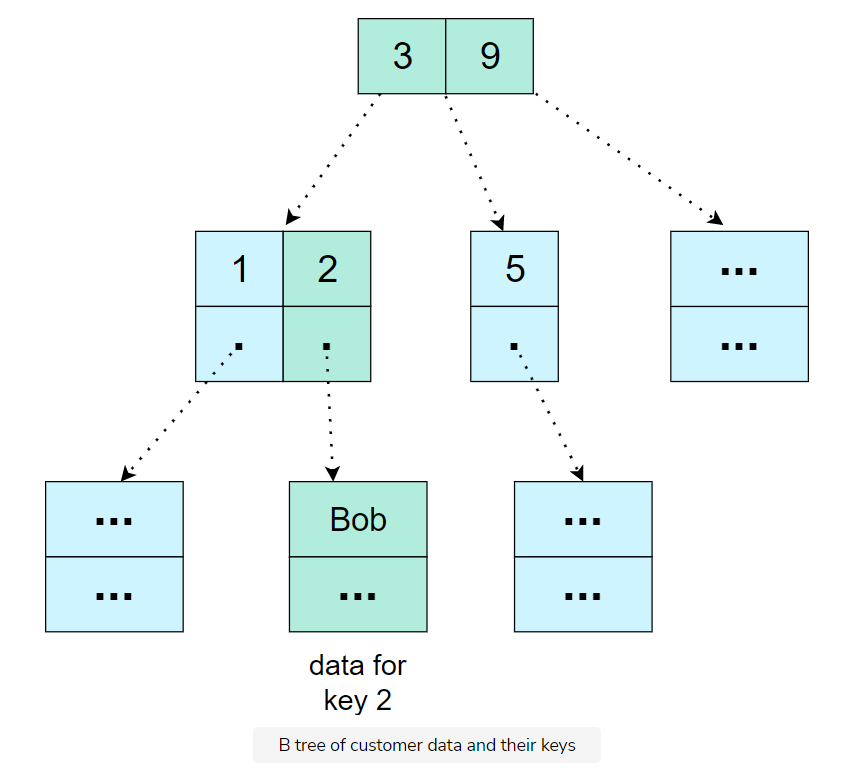
DynamoDB features, the pros and cons of DynamoDB indexes and when it is a great default choice for a database will be discussed in this lesson. To conclude, we will list down our recommendations on the use of DynamoDB indexes.

**We'll cover the following**

* [DynamoDB vs relational database](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#DynamoDB-vs-relational-database)
* [Query processing](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Query-processing)
* [Storage cost](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Storage-cost)
* [Request pricing](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Request-pricing)
  + [On-demand](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#On-demand)
  + [Provisioned capacity](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Provisioned-capacity)
* [DynamoDB indexes](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#DynamoDB-indexes)
  + [Local indexes](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Local-indexes)
  + [Global indexes](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Global-indexes)

**Data structure in the cloud**

Amazon describes DynamoDB as a database, but it’s best seen as a highly-durable data structure in the cloud. A partitioned B-tree data structure, to

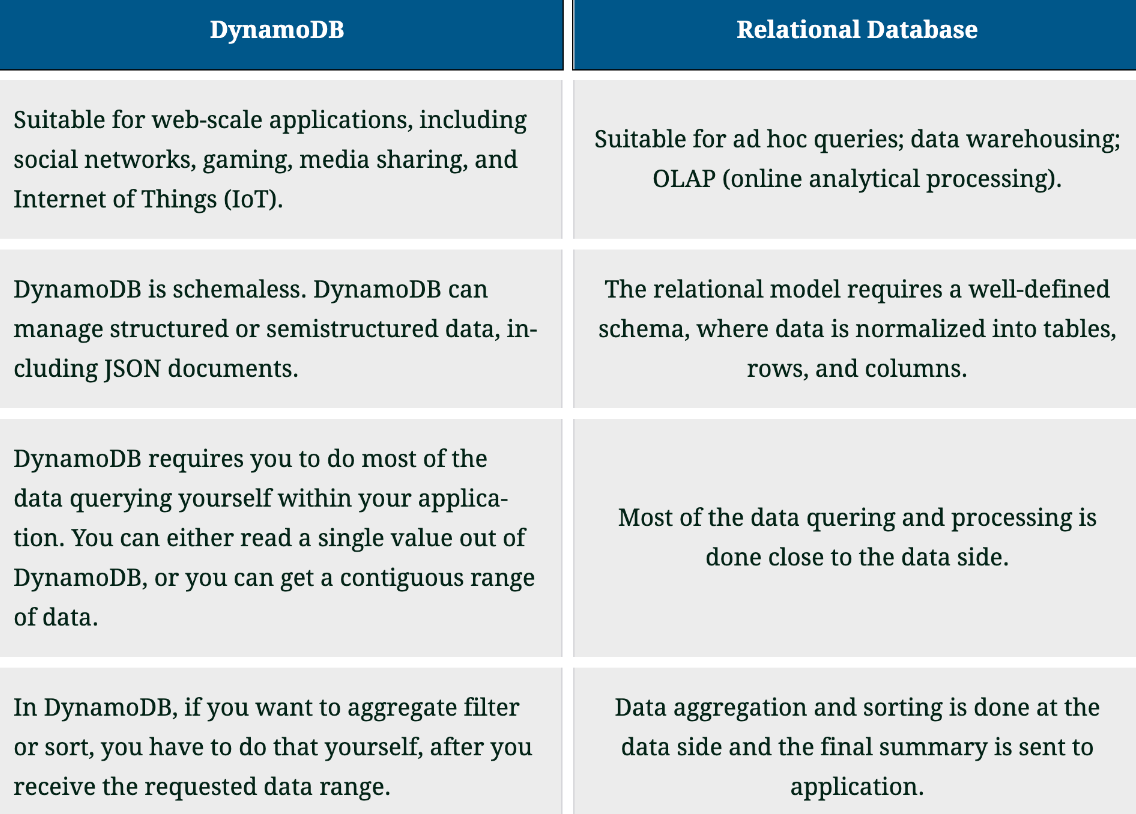


DynamoDB is much more similar to a **Redis** than it is to a **MySQL**. But, unlike **Redis**, it is immediately consistent and highly-durable, centered around that single data structure. If you put something into DynamoDB, you’ll be able to read it back immediately and, for all practical purposes, you can assume that what you have put will never get lost.

It is true that DynamoDB can replace a relational database, but only if you think you can get away with storing all your data in a primitive B-tree. If so, then DynamoDB makes a great default choice for a database.

**DynamoDB vs relational database**[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#DynamoDB-vs-relational-database)

The following table shows some high-level differences between DynamoDB and relational database:

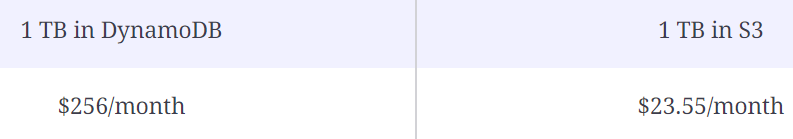


**Query processing**[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Query-processing)

Having to do most query processing on the application side isn’t just inconvenient. It also comes with performance implications. Relational databases run their queries close to the data, so if you’re trying to calculate the sum total value of orders per customer, then that rollup gets done while reading the data, and only the final summary (one row per customer) gets sent over the network. However, if you were to do this with DynamoDB, you’d have to get all the customer orders (one row per order), which involves a lot more data over the network, and then you have to do the rollup in your application, which is far away from the data. This characteristic will be one of the most important aspects of determining whether DynamoDB is a viable choice for your needs.

**Storage cost**[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Storage-cost)

Another factor to consider is cost. As you can see in the table below, storing 1 TB in DynamoDB costs more as compared to storing 1 TB in S3.



Data can also be compressed much more efficiently in S3, which could make this difference even bigger.

However, *storage cost is rarely a large factor when deciding whether DynamoDB is a viable option. Instead, it’s generally****request pricing****that matters most.*

## Request pricing[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Request-pricing)

### On-demand[**#**](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#On-demand)

By default, you should start with DynamoDB’s on-demand pricing and only consider the provisioned capacity as cost optimization. On-demand costs $1.25 per million writes, and $0.25 per million reads. Now, since DynamoDB is such a simple data structure, it’s often not that hard to estimate how many requests you will need. You will likely be able to inspect your application and map every logical operation to a number of DynamoDB requests. For example, you might find that serving a web page will require four DynamoDB **read** requests. Therefore, if you expect to serve a million pages per day, your DynamoDB requests for that action would cost $1/day.

### Provisioned capacity[**#**](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Provisioned-capacity)

If the performance characteristics of DynamoDB are compatible with your application and the on-demand request pricing is in the ballpark of acceptability, you can consider switching to provisioned capacity. On paper, that same workload that cost $1/day to serve 1 million pages would only cost $0.14/day with provisioned capacity, which seems like a very spectacular cost reduction. However, this calculation assumes that:

* Requests are evenly distributed over the course of the day
* There is absolutely zero capacity headroom. (You would get throttled if there were a million and one requests in a day.)

Obviously, both of these assumptions are impractical. In reality, you’re going to have to provide abundant headroom in order to deal with the peak request rate, as well as to handle any general uncertainty in demand. With provisioned capacity, you will have the burden to monitor your utilization and proactively provision the necessary capacity.

In general:

* You will almost always want to start with on-demand pricing (no capacity management burden).
* Then, if your usage grows significantly, you will almost always want to consider moving to provisioned capacity (significant cost savings).

However, if you believe that on-demand pricing is too expensive, then DynamoDB will very likely be too expensive, even with provisioned capacity. In that case, you may want to consider a relational database, which will have very different cost characteristics than DynamoDB.

It is important to note that:

* With on-demand pricing, the capacity you get is not perfectly on-demand. Behind the scenes, DynamoDB adjusts a limit on the number of reads and writes per second, and these limits change based on your usage. However, this is an opaque process and, if you want to ensure that you reserve capacity for big fluctuations in usage, you may want to consider using provisioned capacity for peace of mind.

**NOTE:** It is very important to note that the prices mentioned above are only for the purpose of understanding cost comparisons. These prices are subject to change anytime by AWS. Therefore, most current prices should be referenced for final business decisions.

## DynamoDB indexes[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#DynamoDB-indexes)

A final word about DynamoDB indexes. They come in two flavors:

* Local indexes
* Global indexes

Local indexes came first in early 2013, and global indexes were added just a few months later.

### Local indexes[**#**](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Local-indexes)

The only advantage of local indexes is that they’re immediately consistent, but they do come with a very insidious downside. Once you create a local index on a table, the property that allows a table to keep growing indefinitely goes away. Local indexes come with the constraint that all the records that share the same partition key need to fit in 10 GB, and once that allocation gets exhausted, all write operations with that partition key will start failing.

### Global indexes[**#**](https://www.educative.io/module/lesson/the-good-parts-of-aws/my1KWXq6JwE#Global-indexes)

On the other hand, global indexes don’t constrain your table size in any way, but reading from them is eventually consistent (although the delay is almost always unnoticeable). Global indexes also have one insidious downside, but for most scenarios, it is much less worrisome than that of local indexes.

DynamoDB has an internal queue-like system between the main table and the global index, and this queue has a fixed (but opaque) size. Therefore, if the provisioned throughput of a global index happens to be insufficient to keep up with updates on the main table, then that queue can get full. When that happens, disaster strikes:

* All write operations on the main table start failing.

The most problematic part of this behavior is that there’s no way to monitor the state of this internal queue. So, the only way to prevent it is:

* To monitor the throttled request count on all your global indexes, and then to react quickly to any throttling by provisioning additional capacity on the affected indexes.

Nevertheless, this situation tends to only happen with highly active tables, and short bursts of throttling rarely cause this problem. Global indexes are still very useful, but keep in mind the fact that they’re eventually consistent and that they can indirectly affect the main table in a very consequential manner if they happen to be under provisioned.

In the next lesson, we will take a look at when should we use S3.

# Storage: S3

The features and attributes of S3 make it a top choice for storage.

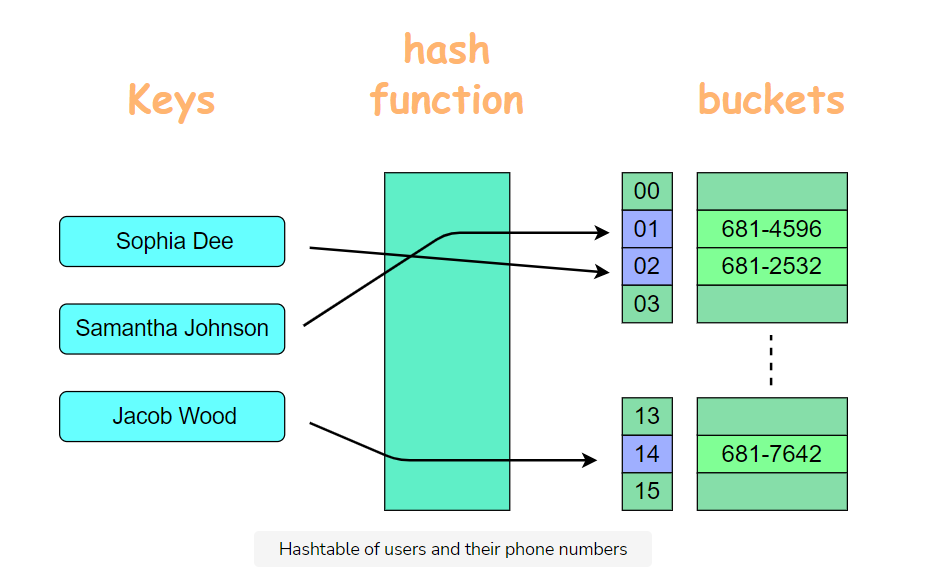
**We'll cover the following**

* [Features of S3](https://www.educative.io/module/lesson/the-good-parts-of-aws/B6Ok3w6v8BN#Features-of-S3)
* [Storage cost](https://www.educative.io/module/lesson/the-good-parts-of-aws/B6Ok3w6v8BN#Storage-cost)
* [Request pricing](https://www.educative.io/module/lesson/the-good-parts-of-aws/B6Ok3w6v8BN#Request-pricing)
* [Limitations](https://www.educative.io/module/lesson/the-good-parts-of-aws/B6Ok3w6v8BN#Limitations)
  + [Hosting static websites](https://www.educative.io/module/lesson/the-good-parts-of-aws/B6Ok3w6v8BN#Hosting-static-websites)
  + [Bucket names](https://www.educative.io/module/lesson/the-good-parts-of-aws/B6Ok3w6v8BN#Bucket-names)

widget

**Hashtable in the cloud**

Fundamentally, you can think of S3 as a highly-durable hash table in the cloud. The key can be any string, and the value any blob of data up to 5 TB. When you upload or download S3 objects, there’s an initial delay of around 20 ms before the data gets streamed at a rate of around 90 MB/s. You can have as many parallel uploads and downloads as you want, thus, the infinite bandwidth. You can also store as many objects as you want and use as much volume as you need, without either having to provision capacity in advance or experiencing any performance degradation as the scale increases.



Following are the different features of S3:

## Features of S3



**Storage cost**

S3 storage costs $23.55/TB/month using the default storage class. It’s almost impossible to beat S3’s storage costs when you factor in the in-built redundancy.

At first:

* You can start with the default storage class and ignore all the other classes.

Unless you’re storing several terabytes in S3, it is almost never worth bothering with them. In general, you can spare yourself the trouble of understanding all the implications of the different storage classes until you really need to start saving money from S3 storage costs.

Apart from the storage classes:

* S3 also offers a reduced redundancy option, but this one should definitely *never* be used.

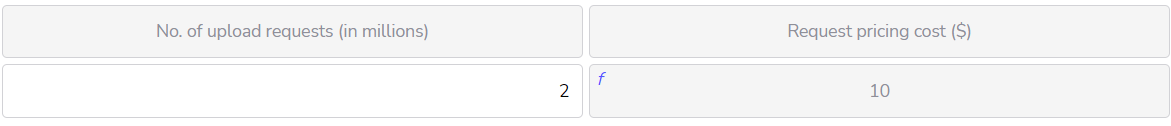
This is a legacy feature that has been around since 2010 (before storage classes existed), and it is currently more expensive than the default storage class, but with no benefits and lower availability (in theory).

**Request pricing**

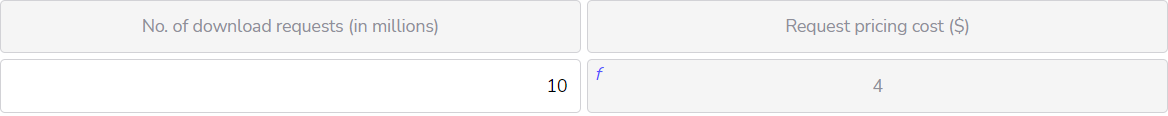
While S3 storage costs are practically unbeatable, request pricing can become expensive in certain situations. With S3, you pay $5/million uploads and $0.40/million downloads.

* When your S3 usage is the result of a human operation—such as somebody uploading a file, or requesting an image from a website—this cost tends to be acceptable. Serving a million people is a big deal, and paying $5 or $0.40 for that level of scale is hardly expensive.
* However, when your S3 usage is driven by other computers, this can change quickly. If you’re touching S3 objects at a high frequency (millions of times a day), request pricing becomes an important aspect of S3’s viability for your use case.

**Let’s do a fun exercise:** Use the below widget to calculate the request pricing cost when $5 is the cost of a million uploads. You just have to put the *number* of **upload** requests (in millions) required for serving a web page, in the first cell. It will automatically generate the cost in the next cell.

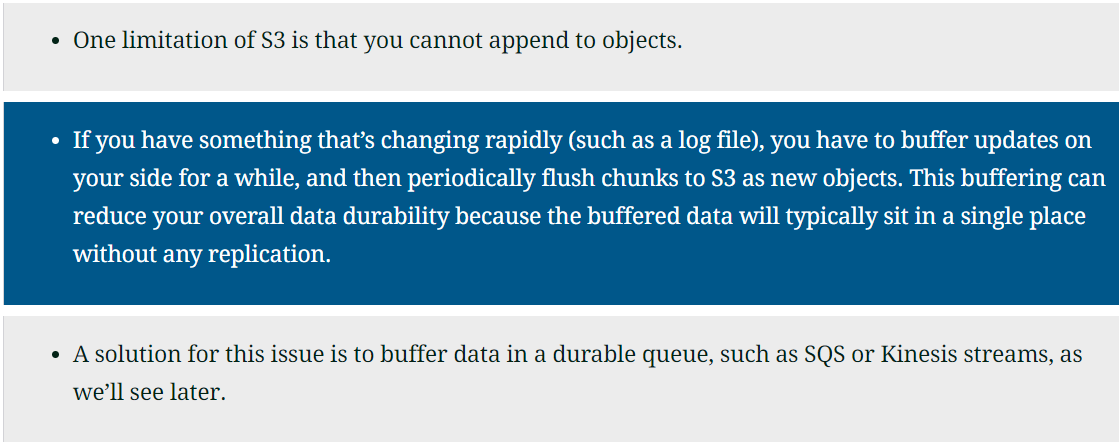


**Here’s another activity:** Use the below widget to calculate the request pricing cost when $0.40 is the cost of a million downloads. You just have to put the number of **download** requests (in millions) required for serving a web page, in the first cell. It will automatically generate the cost in the next cell.



**NOTE:** It is very important to note that the prices mentioned above are only for the purpose of understanding cost comparisons. These prices are subject to change anytime by AWS. Therefore, most current prices should be referenced for final business decisions.

**Limitations**



S3 is a highly-durable hash table in the cloud. The key can be any string, and the value any blob of data up to \_\_\_\_\_\_ TB?

###### D)5TB

### Hosting static websites

As far as hosting static websites on S3 is concerned:

* It doesn’t support HTTPS when used as a static website host, which is a problem.

Web browsers will display a warning, and search engines will penalize you in the rankings.

* You could set up HTTPS using CloudFront, but it’s probably much more trouble than it’s worth.

Nowadays, there are plenty of static website hosts outside of AWS that offer a much better hosting experience for static websites.

### Bucket names

Finally, a note on one of S3’s quirks.

* Bucket names are globally unique across all AWS customers and across all AWS regions.

This can be quite inconvenient because, if you try to rely on a naming convention for your bucket names, you might find that someone else has already taken the name you want.

* A common mitigation is to always add your AWS account ID to the bucket name, which makes conflicts much less likely.

Apart from the inconvenience, the global bucket namespace also has an important security implication.

* If your application tries to use an S3 bucket without checking its owner, you might find yourself uploading data to someone else’s bucket.

Luckily, S3 has an API to check if you own the bucket, and this should always be done before interacting with an existing S3 bucket.

The following example sets bucket\_exists to true if a bucket with the name my-bucket already exists. The region: parameter to Resource has no effect on the result.

s3 = Aws::S3::Resource.new(region: 'us-west-2')

bucket\_exists = s3.bucket('my-bucket').exists?

In the next lesson, we will take a look at EC2 and it’s functionalities.

# Compute: EC2

**We'll cover the following**

* [Instance types](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#Instance-types)
* [Pricing model](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#Pricing-model)
  + [Reserved instances](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#Reserved-instances)
  + [Saving plans](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#Saving-plans)
  + [Spot instances](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#Spot-instances)
* [EC2—network security](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#EC2%E2%80%94network-security)
  + [Security group](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#Security-group)
  + [VPC ACL](https://www.educative.io/module/lesson/the-good-parts-of-aws/mE0ELynqDw9#VPC-ACL)

EC2 allows you to get a complete computer in the cloud in a matter of seconds. The nice thing about EC2 is that the computer you get will be very similar to the computer you use to develop your software. If you can run your software on your computer, you can almost certainly run it on EC2 without any changes. This is one of EC2’s main advantages compared to other types of compute platforms (such as Lambda): you don’t have to adapt your application to your host.

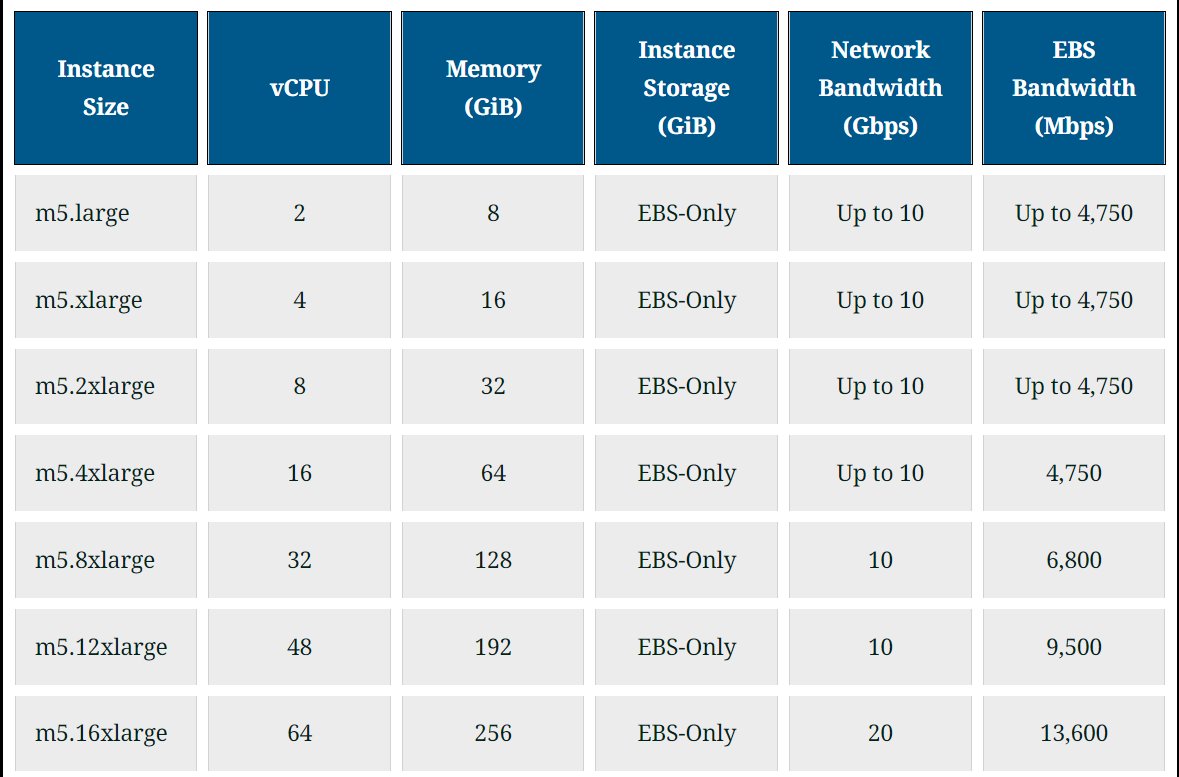
## Instance types

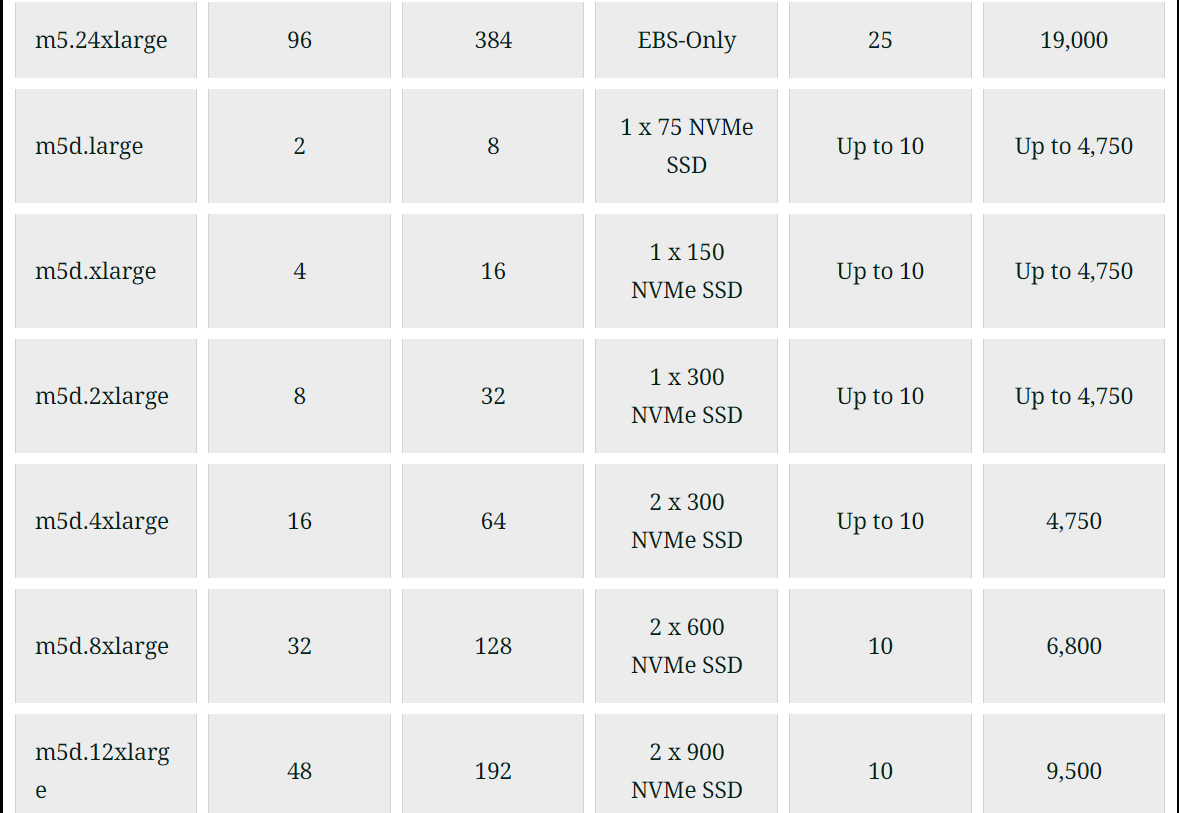
EC2 is a sophisticated service with dozens of options that you will likely never need. This is the result of the highly varied workloads and use cases serviced by EC2. Nevertheless, the defaults that EC2 comes with are good default choices, and the most consequential decision you will have to make is selecting an instance type. As of the time of writing, EC2 offers 256 different instance types, but they can be narrowed down to a few categories defined by what they’re optimized for:

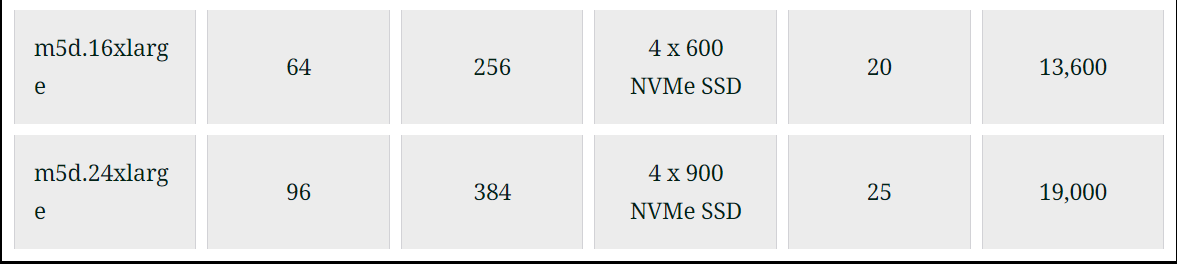
* CPU
* Memory
* Network
* Storage
* etc.

with different instance sizes for each category.

A snapshot of some of the AWS EC2 latest generation General Purpose (GP) Instance types is shown in the below table.







If you were building your own server, there would be an infinite number of ways to configure it, but with EC2 you get to pick an instance type from its catalog. Sometimes this may seem inefficient because the instance type you settle for might come with resources you don’t need. But this commoditization of server types is what makes it possible for EC2 to exist as a service and to have servers available to be provisioned in a matter of seconds.

## Pricing model

One of the most compelling features of EC2 is that you only pay for the number of seconds your instance is running. If you choose to terminate your instance because you don’t need it anymore or because you want to use a different instance type, you immediately stop getting charged for the terminated instance.

### Reserved instances

EC2 also offers you the option to commit to a long period in exchange for a price reduction. The way this has been done for many years was through reserved instances, where you make 1- or 3-year commitments on a specific instance type in exchange for a substantial price reduction.

### Saving plans

However, a recently released option called savings plans offers equivalent cost savings with some additional flexibility in switching instance types during the period under contract.

With the introduction of savings plans, we don’t see any reason to use reserved instances anymore.

The biggest benefit in using EC2 instances is: you pay for the number of seconds an instance is being used and not by the number of seconds an instance is running.

###### B)False

### Spot instances

Spot instances are another cost-saving option, where instead of saving money by reserving long-term usage, you save money by allowing EC2 to take away your instance whenever it wants. The cost savings with spot can be even more significant than with reserved instances, but of course not every use case can tolerate having compute capacity taken away from it randomly.

In general, you should think of savings plans, reserved instances, and spot instances as just cost optimization features. You can ignore them and all their implications until you need to start improving your EC2 efficiency.

Remember that these are not free discounts. They come at the cost of more complicated capacity management and less optionality.

## EC2—network security

Let’s end with what’s probably one of the most daunting aspects of EC2—network security. It is complicated because there are many options, but once again the defaults are a very reasonable starting point. There are two important concepts that you will likely have to modify:

* The security group
* The VPC ACL.

### Security group

You can think of security groups as individual firewalls for your instances. With security groups, you can control what goes in and out of your instances.

### VPC ACL

You can think of VPC ACL as a network firewall. With the VPC ACL, you can control what goes in and out of your network.

In Part 2: The Bootstrap Guide, we’ll show you how to set up a robust EC2 setup with all the necessary networking resources.

In the next lesson, we will take a look at EC2 Auto Scaling and it’s different features.

# Compute: EC2 Auto Scaling

**We'll cover the following**

* [When to use Auto Scaling?](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#When-to-use-Auto-Scaling?)
  + [Capacity headroom](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#Capacity-headroom)
  + [Cost usage ratio](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#Cost-usage-ratio)
  + [Demand fluctuations](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#Demand-fluctuations)
* [Our Recommendations](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#Our-Recommendations)
  + [Secondary features of Auto Scaling](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#Secondary-features-of-Auto-Scaling)
    - [Replace unhealthy instances](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#Replace-unhealthy-instances)
    - [Ease of adding/removing instances](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVW58xk4MJ7#Ease-of-adding/removing-instances)

Amazon will tell you that Auto Scaling allows you to automatically add or remove EC2 instances based on the fluctuating demands of your application. This sounds great in theory, and while we’ve certainly seen that work successfully in a few places, it’s almost never useful except in very specific situations. You will almost never need to use the auto part of Auto Scaling for the reason it exists.

## When to use Auto Scaling?

Let’s start by seeing how you should decide how many EC2 instances to run.

### Capacity headroom

You obviously need to have enough instances to meet your expected peak demand. But you probably don’t want your capacity to exactly match the demand with no leeway. You will want to have some headroom too. This headroom is not waste—it will act as a safety buffer that can absorb many types of unpredictable events. For example:

* If an availability zone were to go down and you lost half of your instances, the headroom in the remaining instances can compensate for the lost capacity.
* Or if there were to be a sudden increase in demand, the same headroom will be immediately available to take it.
* Or if for some reason the performance of your system were to degrade abruptly (due to a software bug, a bad instance, etc.), that same headroom may help compensate the excess load.

So, a capacity headroom is a wonderful thing. You definitely need some. And if you can afford to, it’s probably wise to have a lot of it. It can help you sleep well at night—sometimes in the literal sense.

### Cost usage ratio

The main premise of Auto Scaling is that once you decide how much headroom you want, you’ll be able to make that headroom a constant size, even as the demand for your instances fluctuates. Therefore, a simpler way to look at Auto Scaling is to see it as just a cost reduction tool. Because what’s wrong with having excess headroom during off-peak periods? Absolutely nothing, except cost.

Therefore, the first question you should ask yourself is:

* Are your EC2 costs high enough that any reduction in usage will be materially significant?
* As a thought experiment, consider if your EC2 bill were to go down by 30%—would that be a big deal for your business?

If not, the effort and complexity of getting Auto Scaling working properly is probably not going to be worth it. You might as well just keep the extra headroom during off-peak periods and let it work for you in case of an emergency.

Headroom acts as a safety buffer that can absorb many types of unpredictable events.

###### A)True

### Demand fluctuations

The other thing to consider is:

* Does your EC2 demand vary enough for Auto Scaling even to matter?

If the fluctuations are not significant, or they are too abrupt, or they are not very smooth, Auto Scaling will almost certainly not work well for you.

Nevertheless, in some cases, Auto Scaling can deliver exactly what it says on the tin.

* If you run a business where your EC2 costs are a significant percentage of your expenses and your demand patterns are compatible with Auto Scaling’s capabilities, then this can be a handy tool to help you improve your business margins.

## Our Recommendations



### Secondary features of Auto Scaling

Auto Scaling has a few secondary features that quite frankly should have been part of EC2 itself.

#### Replace unhealthy instances

* One of these features is a setting that allows Auto Scaling to automatically replace an instance if it becomes unhealthy.
* If you are already using a load balancer, you can use the same health checks for both the load balancer and Auto Scaling.
* You can also send health check signals using the Auto Scaling API, either directly from your instances (which isn’t necessarily a reliable way to send unhealthy signals) or from something that monitors your instances from the outside.

#### Ease of adding/removing instances

The other nice thing that comes with Auto Scaling is the ability to simply add or remove instances just by updating the desired capacity setting.

* Auto Scaling becomes a launch template for your EC2 instances, and you get a dial that you can turn up or down depending on how many running instances you need.

There is no faster way to add instances to your fleet than with this method.

One of the limitations of Auto Scaling is that you have to manually remove an instance from the auto scaling group if it becomes unhealthy.

###### B) False

In the next lesson, we will take a look at Lambda and it’s usage.

# Compute: Lambda

Lambda service as a code runner in the cloud or a plugin service for other AWS services is discussed in this lesson together with our rule of thumb in using Lambda service.

**We'll cover the following**

* [Misuse of Lambda](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Misuse-of-Lambda)
* [Lambda as a plugin system for other AWS services](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Lambda-as-a-plugin-system-for-other-AWS-services)
  + [S3](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#S3)
  + [Application Load Balancer (ALB)](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Application-Load-Balancer-(ALB))
  + [CloudFront](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#CloudFront)
  + [CloudWatch](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#CloudWatch)
  + [Kinesis](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Kinesis)
  + [CloudFormation](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#CloudFormation)
* [Limitations](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Limitations)
  + [Short term limitations](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Short-term-limitations)
    - [Cold start](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Cold-start)
    - [Limit on code bundle](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Limit-on-code-bundle)
  + [Inherent Limitations](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Inherent-Limitations)
    - [Stateless](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Stateless)
* [Rule of Thumb](https://www.educative.io/module/lesson/the-good-parts-of-aws/qZq0x5YAwxy#Rule-of-Thumb)

**Code runner in the cloud**

If EC2 is a complete computer in the cloud, Lambda is a code runner in the cloud. With EC2 you get an operating system, a file system, access to the server’s hardware, etc. But with Lambda, you just upload some code and Amazon runs it for you. The beauty of Lambda is:

* It’s the simplest way to run code in the cloud.
* It abstracts away everything except for a function interface, which you get to fill in with the code you want to run.

## Misuse of Lambda

We think Lambda is great—definitely one of the good parts of AWS—as long as you treat it as the simple code runner that it is. A problem we often see is that people sometimes mistake Lambda for a general-purpose application host. Unlike EC2, it is very hard to run a sophisticated piece of software on Lambda without making some very drastic changes to your application and accepting some significant new limitations from the platform.

## Lambda as a plugin system for other AWS services

Lambda is most suitable for small snippets of code that rarely change. We like to think of Lambda functions as part of the infrastructure rather than part of the application. In fact, one of our favorite uses for Lambda is to treat it as a plugin system for other AWS services. Let’s look at a few examples of it.

### S3

* S3 doesn’t come with an API to resize an image after uploading it to a bucket, but with Lambda, you can add that capability to S3.

### Application Load Balancer (ALB)

* Application load balancers come with an API to respond with a fixed response for a given route, but they can’t respond with an image. Lambda lets you make your load balancer do that.

### CloudFront

* CloudFront can’t rewrite a request URL based on request cookies (which is useful for A/B testing), but with Lambda, you can make CloudFront do that with just a little bit of code.

### CloudWatch

* CloudWatch doesn’t support regex-based alerting on application logs, but you can add that feature with a few lines of Lambda code.

### Kinesis

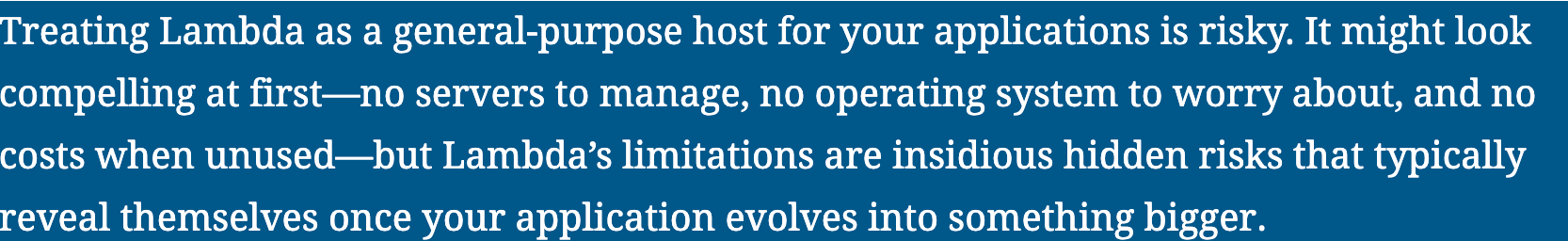
* Kinesis doesn’t come with an API to filter records and write them to DynamoDB, but this is very easy to do with Lambda.

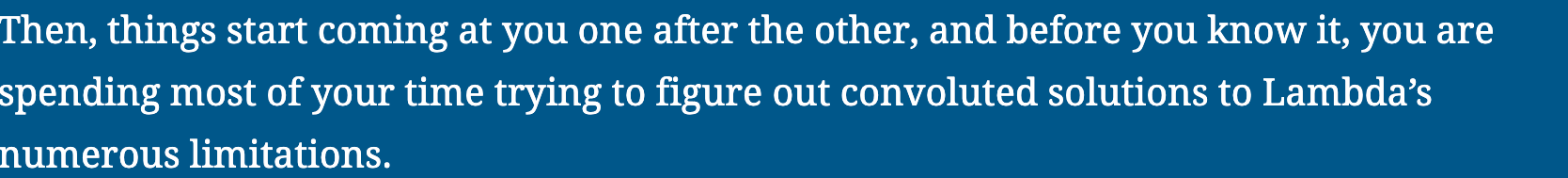
### CloudFormation

* CloudFormation’s native modeling language has many limitations and, for example, it can’t create and validate a new TLS certificate from the AWS Certificate Manager. Using Lambda, you can extend the CloudFormation language to add (almost) any capability you want.

And so on—you get the idea. Lambda is a great way to extend existing AWS features.

## Limitations





Unlike S3, it is very hard to run a sophisticated piece of software on Lambda without making some very drastic changes to your application.

###### B) False

### Short term limitations

We hope that some limitations will likely improve or go away over time. Let’s look at a few examples.

#### Cold start

A very annoying issue is the cold start when a function is invoked after a period of inactivity or when Lambda decides to start running your function on new backend workers.

#### Limit on code bundle

Another problem is the limit of 250 MB for your code bundle, including all your dependencies. Depending on the programming language you’re using, you can find yourself quickly exhausting this limit. And the network bandwidth from Lambda functions seems to be very limited and unpredictable. These can all be problematic, depending on your use case, but we’re quite confident that these issues will improve over time.

### Inherent Limitations

As opposed to short term limitations that we expect to go away, there are other limitations that are inherent to the way Lambda works and which are less likely to go away.

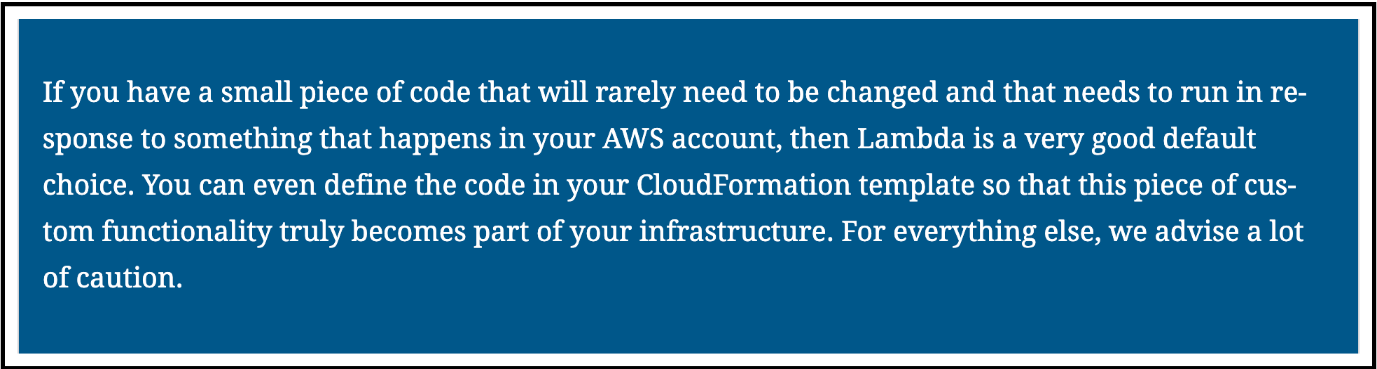
#### Stateless

* For example, you have to assume that every Lambda invocation is stateless. If you need to access some state, you have to use something like S3 or DynamoDB.

While this works fine for a demo, it can quickly become prohibitively expensive in the real world.

* For example, handling a WebSocket connection on Lambda will likely require a read and write to DynamoDB for every exchanged packet, which can quickly result in a spectacularly large DynamoDB bill, even with modest activity.

## Rule of Thumb



That said, we are very optimistic about the future of serverless computing. The idea of abstracting away everything in the stack beneath your code is a phenomenal advance in software development. However, when building software in the present, we have to assess the options available to us today, and while Lambda has its place, it is certainly not a substitute for EC2.

In the next lesson, we will take a look at ELB and it’s different attributes.

# Networking & Content Delivery: ELB

ELB service and its varations will be discussed in this lesson, followed by our recommendations on the use of ELB service of AWS.

**We'll cover the following**

* [Classic](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#Classic)
* [ALB](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#ALB)
* [NLB](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#NLB)
* [Support for TLS/HTTPS](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#Support-for-TLS/HTTPS)
* [Amazon as a man-in-the-middle](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#Amazon-as-a-man-in-the-middle)
* [NLB vs ALB](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#NLB-vs-ALB)
  + [TCP passthrough](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#TCP-passthrough)
  + [Single vs Multi-tenant system](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#Single-vs-Multi-tenant-system)
  + [Cost effective](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#Cost-effective)
* [Our Recommendations](https://www.educative.io/module/lesson/the-good-parts-of-aws/gxVZ7QpDgkD#Our-Recommendations)

ELB is a load balancer service and comes in three variants:

* **Classic**
* **Application (ALB)**
* **Network (NLB)**



## Classic

Classic is a legacy option and remains there only because it works with very old AWS accounts, where you can still run EC2 instances outside of a VPC. For any new setup, you should choose one of the other two variants.

## ALB

ALBs are proper reverse proxies that sit between the internet and your application. Every request to your application gets handled by the load balancer first. The load balancer then makes another request to your application and finally forwards the response from your application to the caller. ALBs have lots of features, and they support sophisticated routing rules, redirects, responses from Lambda functions, authentication, sticky sessions, and many other things.

## NLB

On the other hand, NLBs behave like load balancers, but they work by routing network packets rather than by proxying HTTP requests. An NLB is more like a very sophisticated network router. When a client connects to a server through an NLB, the server would see the client as if it were connected to the client directly.

## Support for TLS/HTTPS

Both ALBs and NLBs support TLS/HTTPS, and they integrate very well with the AWS Certificate Manager. This lets you set up TLS certificates and forget about them. The certificates get renewed automatically and deployed to your load balancers without any downtime. And all the certificates are free.

To have end-to-end TLS from the caller to your application, you will also have to enable TLS on your application. Otherwise, the traffic from the load balancer to your application will travel unencrypted on part of the network. Unfortunately, certificates from the Certificate Manager cannot be exported, so you can’t use them for your application. Instead, common practice is to create a self-signed certificate on your host and use that for your application. The load balancers do not validate the server’s certificate (neither the name, nor the expiry), so in this case, a self-signed certificate works fine.

## Amazon as a man-in-the-middle

The fact that ALBs and NLBs don’t validate certificates might seem concerning. However, since these load balancers run in a VPC, Amazon authenticates each network packet and guarantees that the packets go only to the hosts you configured in your load balancer. The protection from spoofing and man-in-the-middle is provided by Amazon.

That said, keep in mind that:

by installing TLS certificates on your load balancers, you’re letting Amazon become a man-in-the-middle itself. Amazon’s hardware and software will be decrypting your network traffic and re-encrypting it when forwarding it to your application (if you enable TLS on your application).

If you’d rather not trust Amazon with this responsibility, you must use:

an NLB with TCP passthrough (without enabling TLS on the load balancer). But in that case, you must keep a valid TLS certificate on your application host and deal with certificate renewals yourself.

## NLB vs ALB

### TCP passthrough

Only NLBs support TCP passthrough, but since NLBs work on the network layer, they also lack support for many of the features found in ALBs. So, unless you need TCP passthrough, why would you ever want to use an NLB?

Well, ALBs have two main disadvantages:

* First, their proxy approach adds a few milliseconds to each request, so they’re slightly slower than NLBs.
* Second, they may not scale quickly enough to handle a big burst of traffic.

### Single vs Multi-tenant system

An ALB behaves like a single-tenant system. AWS keeps track of your request rates and then automatically scales your ALB up or down based on the demand it sees. The exact logic of this behavior is opaque, so the only way to be assured that your ALB’s elasticity meets your demands is:

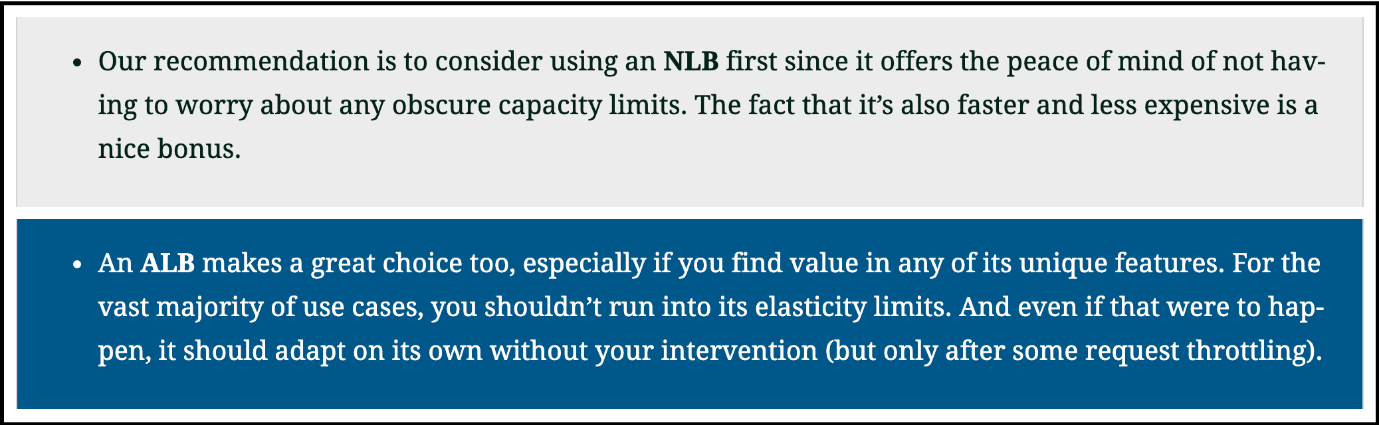
* To test it yourself.
* Or, if you know a certain amount of traffic is on the way, you can ask AWS (through a support ticket) to preemptively provide sufficient capacity for you.

On the other hand, NLBs behave like a multi-tenant system and are scaled up and down in aggregate, rather than per individual load balancer. Therefore, in theory, you should never have an NLB fail to scale to your needs (unless you exhaust all of Amazon’s capacity).

### Cost effective

NLBs are also slightly less expensive than ALBs. But a single ALB can be used to handle multiple domains (via host-based routing) while an NLB cannot, so in some situations, an ALB can be more cost-effective. Nevertheless, cost is unlikely to be the deciding factor when choosing between an ALB and an NLB.

## Our Recommendations



NLBs are proper reverse proxies that sit between the internet and your application.

###### B) False

In the next lesson, we will take a look at Route 53 and it’s usage.

# Networking & Content Delivery: Route 53

Route 53: The DNS service of AWS with its limitations and benefits are presented in this lesson.

**We'll cover the following**

* [Benefits of using Route 53](https://www.educative.io/module/lesson/the-good-parts-of-aws/BnWlA1nJ9PY#Benefits-of-using-Route-53)

**A DNS service**

Route 53 is a DNS service. It lets you translate domain names to IP addresses.

**Limitations**

There’s nothing particularly special about Route 53’s DNS capabilities. In fact, it has a few annoying (but mostly minor) limitations such as the lack of support for ALIAS records (unless they point to AWS resources).

## Benefits of using Route 53

However, the reason we stick to using Route 53 is that:

* First of all, it’s good enough.
* Secondly, it integrates very well with ELB.

There is a significant benefit in having CloudFormation automatically set up your load balancer together with the DNS records for your custom domain. Route 53 makes this possible, whereas if you were to use a different DNS provider, you’d likely have to manage your DNS records manually.

Route 53 provides support for ALIAS records as long as they point to AWS resources.

###### True

In the next lesson, we will take a look at CloudFormation and it’s different functionalities.

# Management & Governance: CloudFormation

Defining AWS resources with CloudFormation service and our rule of thumb of what services to manage by Cloudformation and what services should be avoided is going to be discussed in this lesson.

**We'll cover the following**

* [Defining AWS resources with CloudFormation](https://www.educative.io/module/lesson/the-good-parts-of-aws/xo2j8A52A1E#Defining-AWS-resources-with-CloudFormation)
* [Things to avoid in CloudFormation](https://www.educative.io/module/lesson/the-good-parts-of-aws/xo2j8A52A1E#Things-to-avoid-in-CloudFormation)
* [Rule of thumb](https://www.educative.io/module/lesson/the-good-parts-of-aws/xo2j8A52A1E#Rule-of-thumb)
* [Our Recommendations](https://www.educative.io/module/lesson/the-good-parts-of-aws/xo2j8A52A1E#Our-Recommendations)

When using AWS, you almost always want to use some CloudFormation (or a similar tool). It lets you create and update the things you have in AWS without having to click around on the console or write fragile scripts. It takes a while to get the hang of it, but the time savings pay off the initial investment almost immediately. Even for development, the ability to tear down everything cleanly and recreate your AWS set up in one click is extremely valuable



## Defining AWS resources with CloudFormation

With CloudFormation;

* You define your AWS resources as a YAML script (or JSON, but we find YAML to be much easier to read and modify).
* Then you point CloudFormation to your AWS account, and it creates all the resources you defined.
* If you run the script again without making any changes, CloudFormation won’t do anything (it’s idempotent).
* If you make a change to one resource, it will change only that resource, plus any other resources that depend on the modified one (if necessary).
* If you change your mind about an update, you can safely tell CloudFormation to roll it back.
* You can also tell CloudFormation to tear down everything it created, and it will give you your AWS account back in the original state (with a few exceptions).

**Q** CloudFormation lets you create and update the things you have in AWS without having to click around on the console or write fragile scripts.

###### A) True

## Things to avoid in CloudFormation

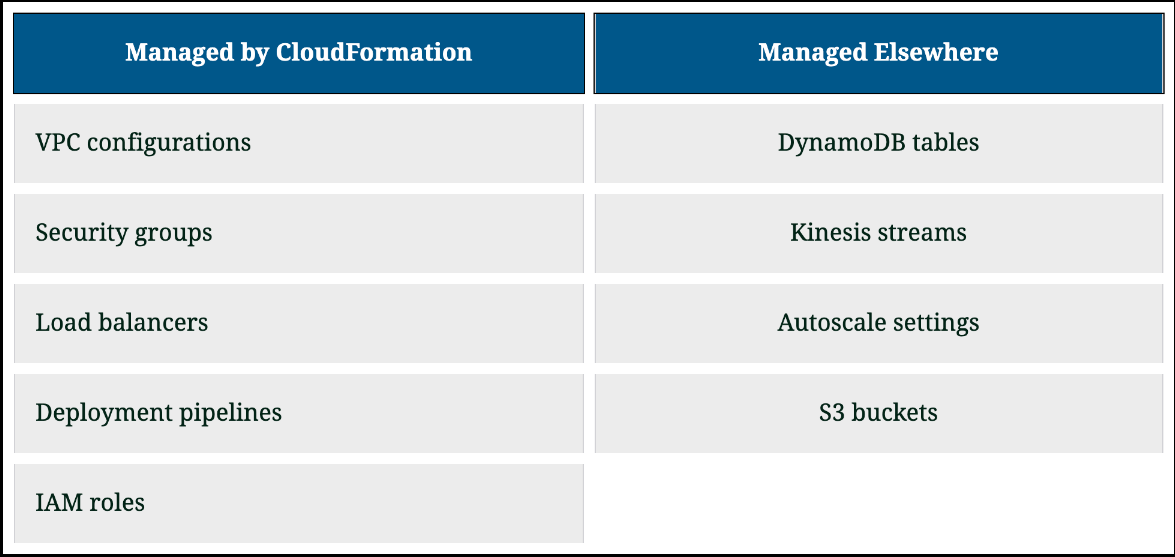
All of that works exceptionally well. However, a trap that people often fall into is to use CloudFormation a little bit too much! There are some things you will likely want to keep out of there. The problems arise when you modify something manually that should be under CloudFormation’s control because when you do that, you can expect unpredictable behavior. Sometimes it’s okay. Sometimes it’s an unrecoverable outcome with catastrophic consequences.

When you’ve touched something manually, and you run your CloudFormation script again, it will often try to revert your changes back to how they were. Sometimes it will manage to do so, but you wouldn’t have wanted it to. Sometimes it will try to reconcile but become stuck in an endless loop.

## Rule of thumb

Our rule of thumb is to let CloudFormation deal with all the AWS things that are either static or change very rarely. The table below lists down the things that should be managed by CloudFormation and the things that are better managed elsewhere. You may want to handle some of these things directly from your application, or you could have another simple script that sets them up separately.

## Our Recommendations[#](https://www.educative.io/module/lesson/the-good-parts-of-aws/xo2j8A52A1E#Our-Recommendations)



Then there are some things that are so infrequently touched and so hard to automate that it just doesn’t make sense to script them. For example:

* Route 53 domain registrations and hosted zones
* Certificate creation and validation from the Certificate Manager.
* and so on.

The test for whether your infrastructure-as-code is good enough is whether you feel confident that you can tear down your stack and bring it all up again in a few minutes without any mistakes. Spending an unbounded amount of time in pursuit of scripting everything is not advisable.

In the next lesson, we will take a look at SQS and it’s different attributes.

# Application Integration: SQS

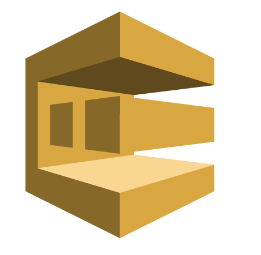
SQS: A queue in the cloud and its features are discussed in this lesson.

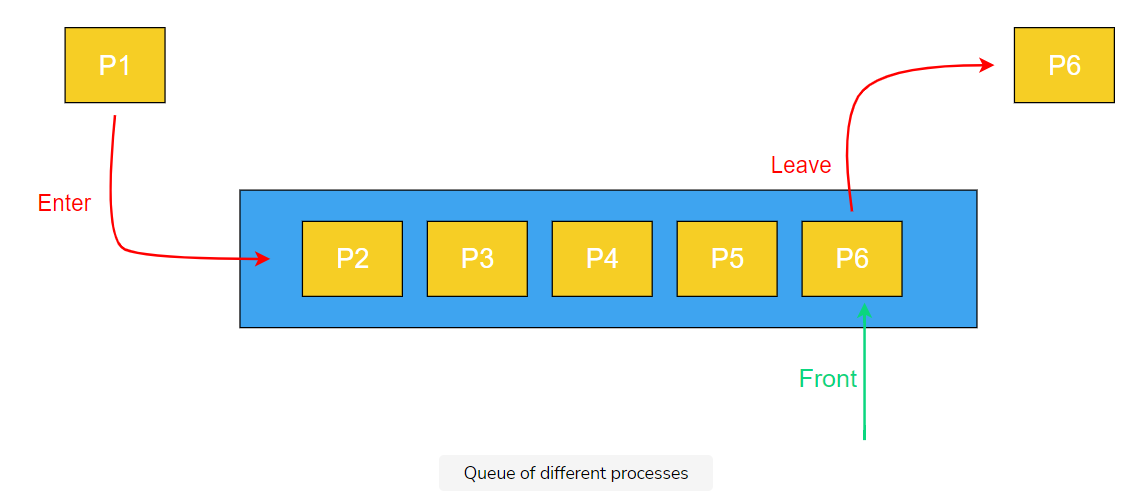
**We'll cover the following**

* [Zero capacity management](https://www.educative.io/module/lesson/the-good-parts-of-aws/39jvz92wlq4#Zero-capacity-management)

**A queue in the cloud**

SQS is a highly-durable queue in the cloud. You put messages on one end, and a consumer takes them out from the other side. The messages are consumed in almost first-in-first-out order, but the ordering is not strict. The lack of strict ordering happens because your SQS queue is actually a bunch of queues behind the scenes. When you enqueue a message, it goes to a random queue, and when you poll, you also poll a random queue. In addition, duplicate messages can emerge within SQS, so your consumers should be prepared to handle this situation.





**Zero capacity management**

Like S3, SQS is one of the few AWS services that requires zero capacity management.

* There is no limit on the rate of messages enqueued or consumed, and you don’t have to worry about any throttling limits.
* The number of messages stored in SQS (the backlog size) is also unlimited.

As long as you can tolerate the lack of strict ordering and the possibility of duplicates, this property makes SQS a great default choice for dispatching asynchronous work.

If you really need strict ordering and exactly-once delivery (no duplicates):

* SQS has an option to enable this property by marking your queue as FIFO.
* But these FIFO queues come with a throughput limit of 300 messages per second, so they’re only viable if you’re certain that your message rate will remain well clear of that limit.

**Q.** In SQS, there is no strict ordering because SQS queue is actually a bunch of queues behind the scenes and duplicate messages can emerge within SQS too.

###### A) True

In the next lesson, we will take a look at Kinesis and it’s different features.

# Media Service: Kinesis

Kinesis: A linked list in the cloud and it's pros and cons will be discussed in this lesson.

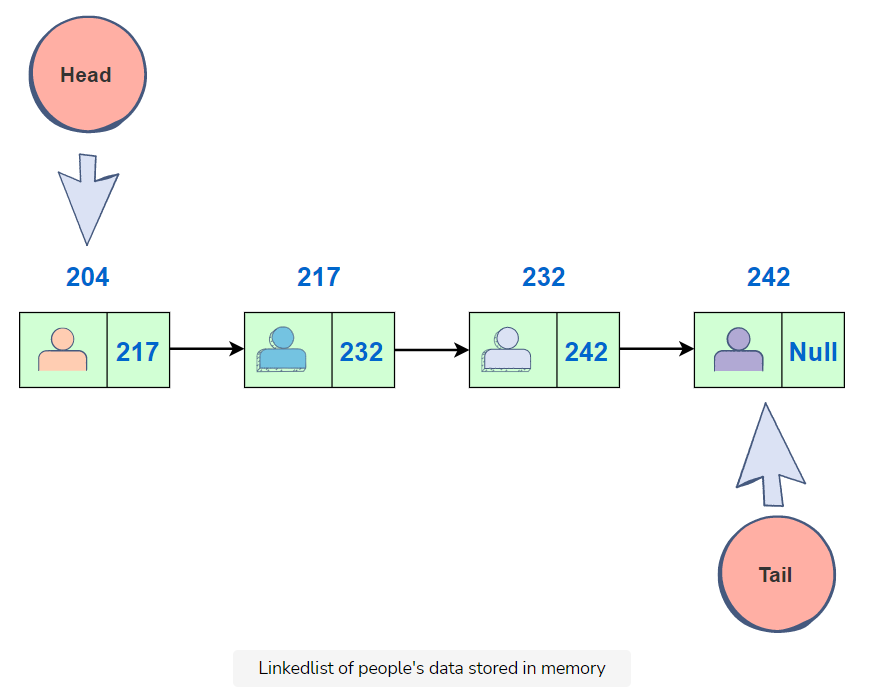
**We'll cover the following**

* [Kinesis vs SQS](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVBjqWZD0xr#Kinesis-vs-SQS)
* [Cheaper/cost effective](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVBjqWZD0xr#Cheaper/cost-effective)
* [Not easy to use](https://www.educative.io/module/lesson/the-good-parts-of-aws/qVBjqWZD0xr#Not-easy-to-use)

**A linked list in the cloud**

You can think of a Kinesis stream as a highly-durable linked list in the cloud. The use cases for Kinesis are often similar to those of SQS—you would typically use either Kinesis or SQS when you want to enqueue records for asynchronous processing.





**Kinesis vs SQS**

The main difference between the two services is:

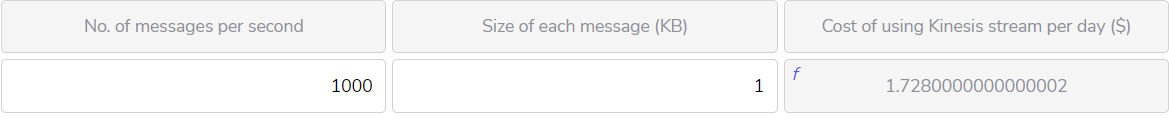
* SQS can only have one consumer, while Kinesis can have many.
* Once an SQS message gets consumed, it gets deleted from the queue. But Kinesis records get added to a list in a stable order, and any number of consumers can read a copy of the stream by keeping a cursor over this never-ending list.
* Multiple consumers don’t affect each other, and if one falls behind, it doesn’t slow down the other consumers.
* Whenever consumers read data out of Kinesis, they will always get their records in the same order.

**Cheaper/cost effective**

In addition to supporting multiple consumers, another benefit of Kinesis over SQS is that it can be a lot cheaper. For example:

* Putting 1 KB messages in SQS at an average rate of 500 messages per second will cost you $34.56 per day. A Kinesis stream with 50% capacity headroom can handle that same volume for just $0.96 per day. So there can be about a massive difference in cost.

**Let’s do a fun exercise:** Use the below widget to calculate the cost of putting a certain size (KB) message at an average rate of certain number of messages per second per day to a Kinesis stream. $0.02 is the cost of putting a million messages to a Kinesis stream per day. You just have to put the *number* of **messages** per second, and the size of each message in KB. It will automatically generate the cost per day in the next cell.



The reason for this cost profile is simple:

* Kinesis streams are optimized for sequential reads and sequential writes.
* Records get added to the end of a file and read always happens sequentially from a pointer on that file.
* Unlike SQS, records in a Kinesis stream don’t get deleted when consumed, so it’s a pure append-only data structure behind the scenes.
* Data simply ages out of a Kinesis stream once it exceeds its retention period, which is 24 hours by default.

**NOTE:** It is very important to note that the prices mentioned above are only for the purpose of understanding cost comparisons. These prices are subject to change anytime by AWS. Therefore, the most current prices should be referenced for final business decisions.

**Not easy to use**

On the other hand, Kinesis is not as easy to use as SQS. While with SQS you can simply enqueue as many messages as you want without having to worry about capacity or throttling limits, a Kinesis stream is made up of slices of capacity called shards, and it’s up to you to figure out how many shards you need, monitor shard utilization, add shards, and figure out the best way to route records to shards so that they get approximately an equivalent amount of traffic. And unfortunately, all of this is a significant operational burden.

That said, Kinesis can still be much easier to use when compared to other self- hosted technologies that provide the same streaming properties.

**Q** Once a Kinesis message gets consumed, it gets deleted from the queue as opposed to SQS where it is appended to the end of the file.

###### False

In the next lesson, we will take a small quiz to test the concepts you have learned so far in this first chapter.

# Quiz - Basic AWS Services

To wrap up part 1 of this course, let's check your understanding of AWS services that you have studied so far.

**1.**An \_\_\_\_\_\_ is more like a very sophisticated network router.

###### D)NLB

Lambda’s limitations are insidious hidden risks that typically reveal themselves once your application evolves into something bigger.

###### A)True

\_\_\_\_\_\_ abstracts away everything except for a function interface, which you get to fill in with the code you want to run.

###### A)Lambda

The fact that ALBs and NLBs don’t validate certificates might seem concerning. However, since these load balancers run in a VPC, Amazon authenticates each network packet and guarantees that the packets go only to the hosts you configured in your load balancer.

A. True

The number of messages stored in SQS (the backlog size) is limited.

False

NLB behaves like a single-tenant system. AWS keeps track of your request rates and then automatically scales your NLB up or down based on the demand it sees.

False

The nice thing about EC2 is that the computer you get will be very similar to the computer you use to develop your software.

True

\_\_\_\_\_\_ doesn’t support regex-based alerting on application logs.

Cloudwatch

Every Lambda invocation is stateless. If we need to access some state, we have to use something like S3 or DynamoDB.

True

Unlike SQS, records in a Kinesis stream don’t get deleted when consumed, so it’s a pure append-only data structure behind the scenes.

True

Check all AWS services that requires zero capacity management.

###### A)SQS

###### D)S3