


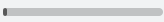


AWS Lambda Pricing: A Complete Guide

 blog.awsfundamentals.com/aws-lambda-pricing-a-complete-guide-to-understanding-the-cost-of-the-serverless-service

Sandro Volpicella



 Play this article

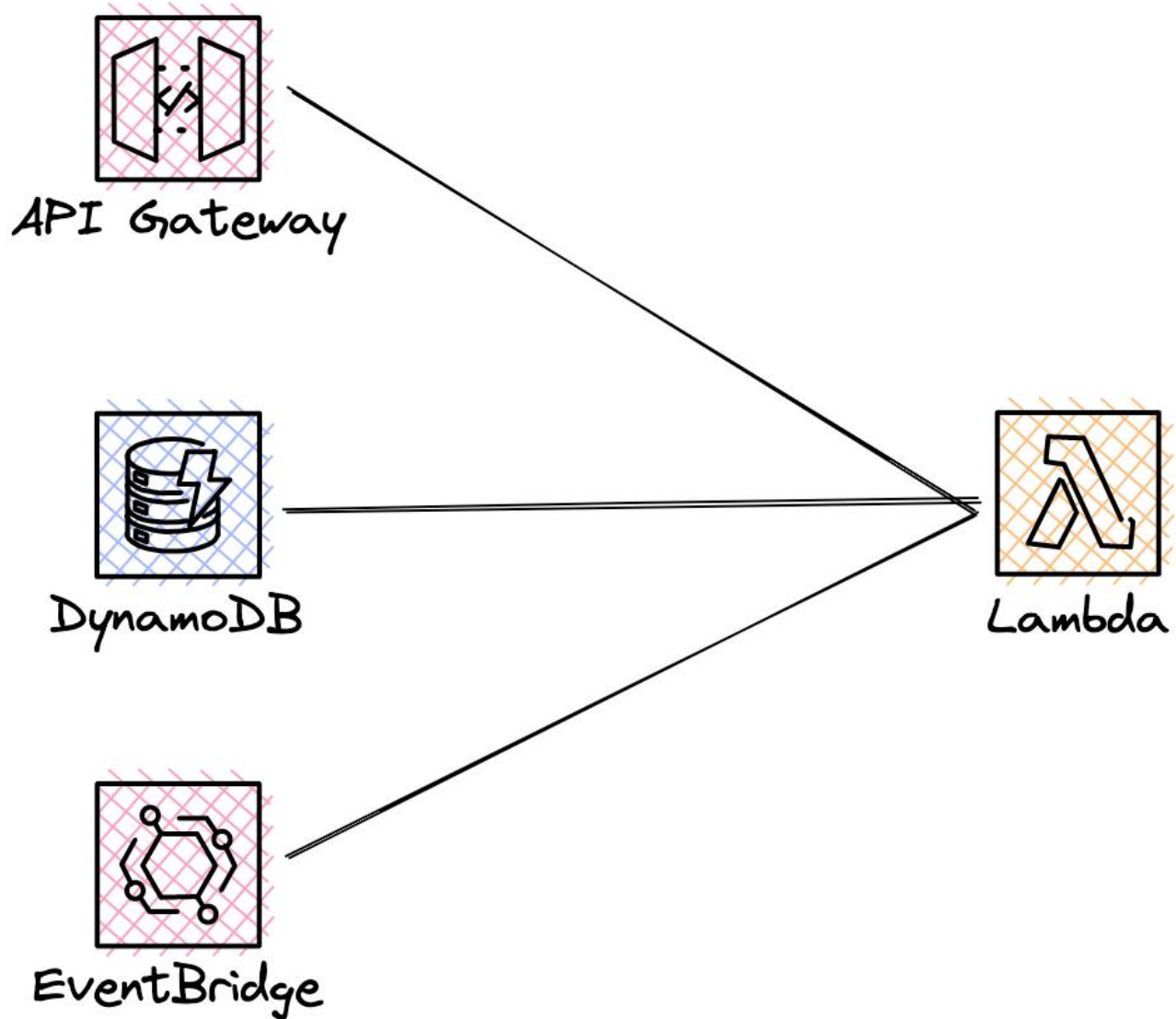
▶ 0:00 / 8:39   

Introduction to AWS Lambda and Its Pricing Structure

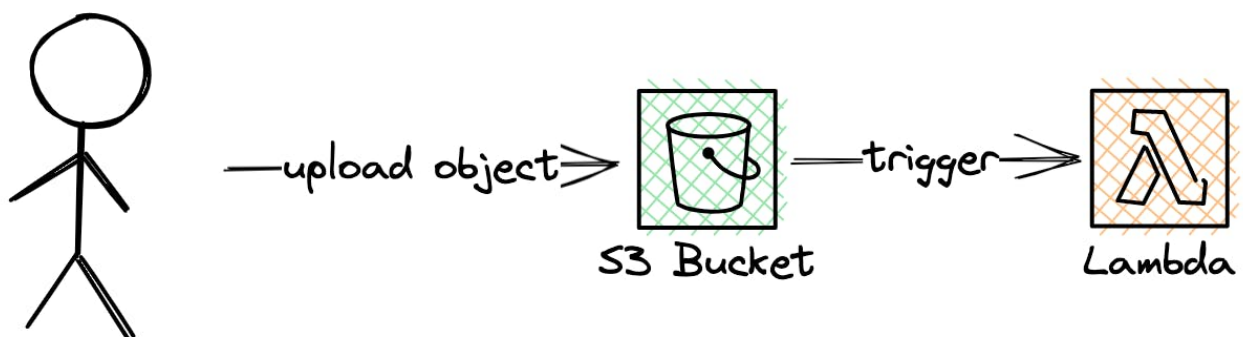
AWS Lambda is the **serverless service** by Amazon Web Services (AWS). It allows you to run code without worrying about any infrastructure. Lambda functions run based on trigger events. These events can come from a variety of services within AWS. For example:

- S3
- Amazon EventBridge
- Amazon DynamoDB
- etc.

Many of these services trigger Lambda functions automatically and run the code by that.



A typical example is S3 event notifications. Once an object is uploaded to S3 you can invoke a Lambda function. The Lambda function receives information from the newly created object as an event. It can then work on this event with custom business logic.



The great thing about Lambda is: You don't need to worry about scaling at all. Lambda **automatically** provisions the infrastructure it needs to serve your requests. This includes up- and downscaling of all your resources. You can only focus on writing your code.

Lambda follows the **pay-per-use pricing model**. That means you only pay for the actual runtime and the number of requests. You don't have to pay for any idle time that your server runs but isn't actually executing requests. The pricing can have some quirks so let's look into that in a bit more detail.

Pay-Per-Use in Lambda

Lambda is charged based on usage. If we look at the [official Lambda pricing page](#) we can see different factors that determine the pricing:

The Runtime in GB Seconds

First of all, we see the GB seconds.

Architecture	Duration	Requests
x86 Price		
First 6 Billion GB-seconds / month	\$0.0000166667 for every GB-second	\$0.20 per 1M requests
Next 9 Billion GB-seconds / month	\$0.000015 for every GB-second	\$0.20 per 1M requests
Over 15 Billion GB-seconds / month	\$0.0000133334 for every GB-second	\$0.20 per 1M requests
Arm Price		
First 7.5 Billion GB-seconds / month	\$0.0000133334 for every GB-second	\$0.20 per 1M requests
Next 11.25 Billion GB-seconds / month	\$0.0000120001 for every GB-second	\$0.20 per 1M requests
Over 18.75 Billion GB-seconds / month	\$0.0000106667 for every GB-second	\$0.20 per 1M requests

This means AWS charges you based on the **provisioned memory** of your function (the GBs, which also imply the number of vCPUs) and the **execution time** of your functions.

Let's see some example configurations:

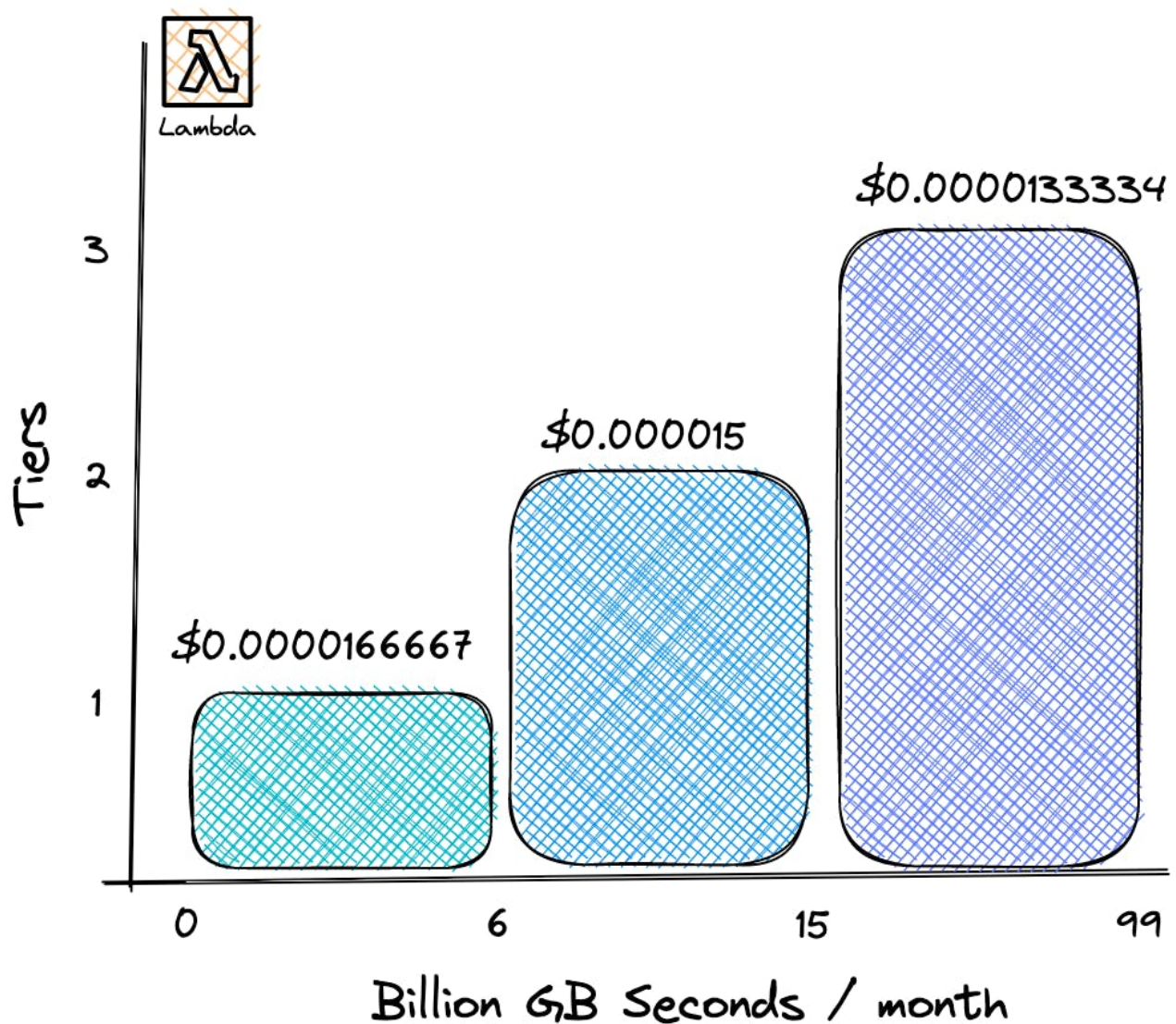
- 1.0 GB Memory -> 500,000 executions x 1 second per execution -> 500,000 **seconds** -> 500,000 **GB seconds** x 0.0000166667 USD = **8.33 USD**
- 10.0 GB Memory -> 1,500,000 executions x 1 second per execution -> 1,500,000 **seconds** -> 15,000,000 **GB seconds** x 0.0000166667 USD = **250.00 USD**

The formula is **executions x avg. execution time x GB per second**

Tiered Pricing

As you can see in the table above the pricing in Lambda is tiered. That means the more you use the cheaper it gets. The tiers are pretty high. You probably only reach those tiers once you are a heavy user of lambda. Let's see some of those tiers:

1. 0-6 billion GB seconds/month
2. 6-15 billion GB-seconds/month
3. > 15 billion GB-seconds/month



Paying for Ephemeral Storage

The second category we spot in the pricing documentation is the number of ephemeral storage. Your lambda function has storage applied. This storage is needed to load external libraries (e.g. node_modules), Lambda layers, and files you are saving in your lambda

function. By default, your Lambda function has 512 MB of storage. The default storage is free.

You can allocate more storage to your lambda function. Starting from 512 MB up to 10240 MB. You need to increase it in 1 MB increments. The price for Ephemeral Storage is **\$0.0000000309 for every GB-second** That means this applies **again for your runtime** in GB seconds.

Let's take another example. Our Lambda function has 2048 MB of ephemeral storage and 2 GB of memory. It runs for 800,000 seconds per month.

- **GB Second Runtime:** $800,000 \text{ seconds} \times 2 \text{ GB} = 1,600,000 \text{ GB seconds}$
- **Price for Memory Runtime:** $1,600,000 \text{ GB seconds} \times 0.0000166667 \text{ USD / GB second} = 26.67 \text{ USD}$
- **Price for ephemeral storage:** $2 \text{ GB} - 0.5 \text{ GB (free storage)} = 1.5 \text{ GB} \times 800,000 \text{ seconds} = 1,200,000 \text{ GB seconds} \times 0.0000000309 \text{ USD} = 0.0371 \text{ USD}$

As you can see the main driver for costs is most of the time the actual runtime and not the ephemeral storage.

You Are Billed per 1 ms

One word about the units you are gonna billed. Lambda runtime is billed by 1 ms. That means each millisecond in Lambda counts. This was introduced in 2020 and changed the billing for many companies and made it even cheaper. Before Lambda was calculated by every 100 ms, so it happened that you paid too much because it was rounded up.

Architectures ARM Vs. x86 - ARM Is Cheaper

When it comes to the architecture of the underlying infrastructure for your Lambda functions, there are two main options: **ARM and x86**.

Each architecture has its unique characteristics and performance capabilities, and as a result, they also have different pricing structures.

With Lambda you can use AWS's own Graviton processors. These processors are based on **ARM** architecture. The runtime is about 34% **cheaper** compared to the default x86 processors.

Before you actually switch test if all your workloads are working on this architecture. Some packages or libraries are not working on an ARM architecture. In our experience, most of the default packages work very well. This is cost efficiency for free!

The price decrease comes with implications for the ARM instances. ARM instances are typically less powerful and are, therefore, less expensive to run. While x86 instances are more powerful.

Free Tier in AWS Lambda

AWS offers a free tier for almost all services. Let's first have a look at the free tier description in the AWS documentation:

The AWS Lambda free tier includes one million free requests per month and 400,000 GB-seconds of compute time per month...

Let's have a look at 400,000 GB seconds per month in some different configurations:

- 0.5 GB Memory → $400,000 / 0.5 \text{ GB} \rightarrow 800,000 \text{ GB-seconds} \rightarrow \mathbf{9.2 \text{ days}}$ of execution
- 1.0 GB Memory → $400,000 / 1.0 \text{ GB} \rightarrow 400,000 \text{ GB-seconds} \rightarrow \mathbf{\sim 4.6 \text{ days}}$ of execution
- 10 GB Memory → $400,000 / 10 \text{ GB} \rightarrow 40,000 \text{ GB-seconds} \rightarrow \mathbf{\sim 0.5 \text{ days}}$ of execution

Additional charges apply if you increase the ephemeral storage to over 512 MB.

This is an amazing free tier. Many production applications with real users can be run within the whole free tier of AWS. It is really a great way to get started and see if Lambda is suitable for your use case.

Remember, that the free tier is only available for 12 months. After 12 months, you need to pay for your runtime.

Optimize Lambda Costs

As you can see there are many factors that influence the cost of running your Lambda functions. Including the amount of memory, execution time, and ephemeral storage.

However, there are some ways to reduce costs and optimize your services.

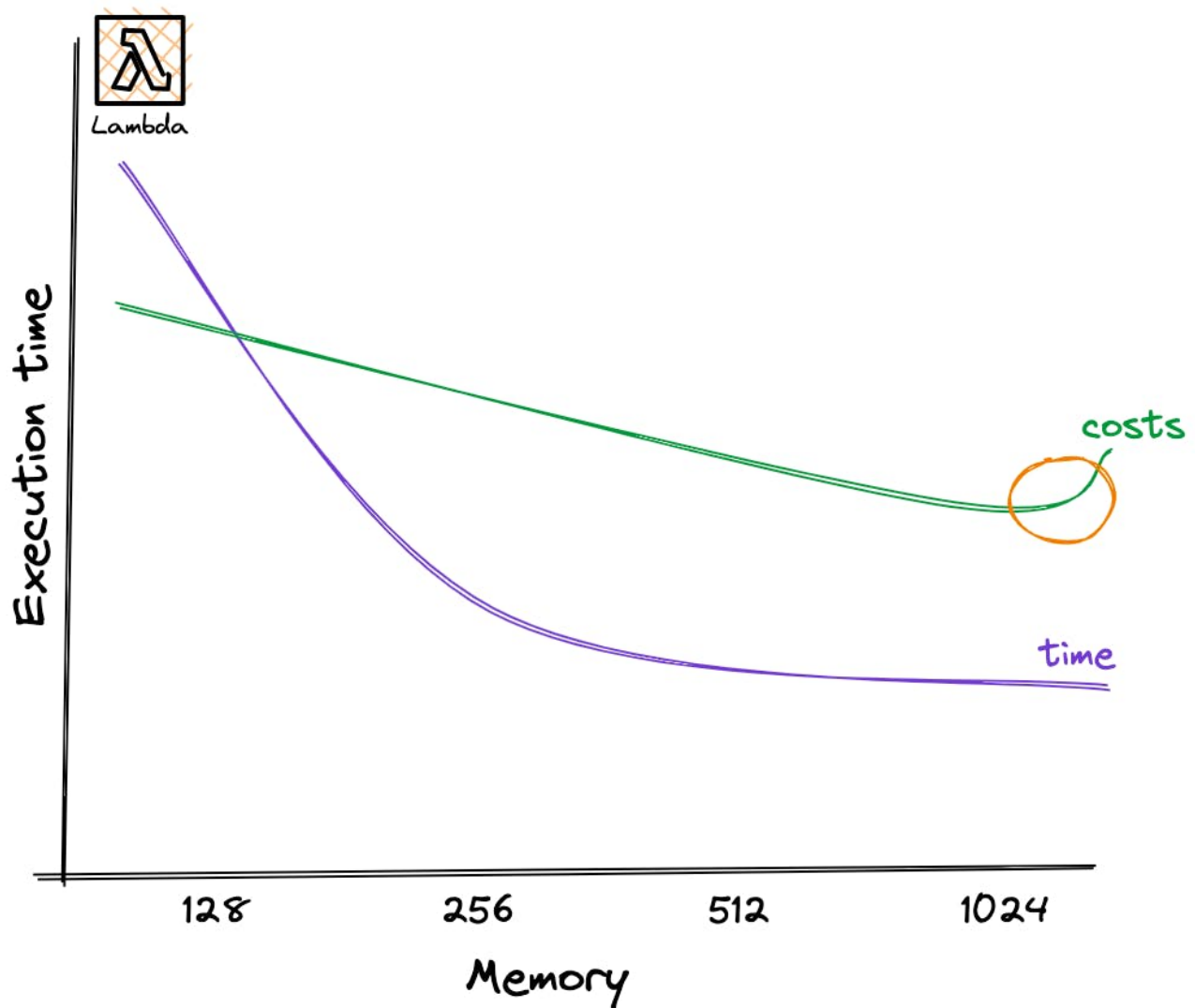
Start with Little Memory

There is no reason to give each of your Lambda functions a lot of memory. Test it out and see how it behaves. If 128 MB of memory is enough keep it that way.

Faster is cheaper

Contradicting to the first point: If more memory means your Lambda function runs faster, do that. A faster execution always means your Lambda costs are cheaper. There is a sweet spot between assigned memory and execution speed. In general focus on reducing your execution time.

For measuring your speed you can use a tool like [AWS Lambda Power Tooling](#). This tool allows you to find exactly this sweet spot.



Switch to ARM

Switching to ARM architecture can make a great difference. It is 34% more price efficient compared to the x86 architecture. Change it, make sure your workload is still running, and spend less money.

Using Provisioned Concurrency

Provisioned concurrency holds your lambda functions warm 24/7. In general, this approach is very good for tackling cold starts. But provisioned concurrency can also help reduce costs. The GB second is much cheaper (about 70%) as with normal Lambda functions. However, you need to pay for it all the time. This approach only works if your Lambda function is almost never idle. Check out this [article](#) from Yan Cui for more tips on reducing your Lambda costs.

Final Words

That's it about AWS Lambda Pricing. If you're interested in more about the fundamentals of AWS make sure to sign up for our [newsletter](#) and get bi-weekly emails about all fundamentals of AWS services and concepts.

Written by

Published on
