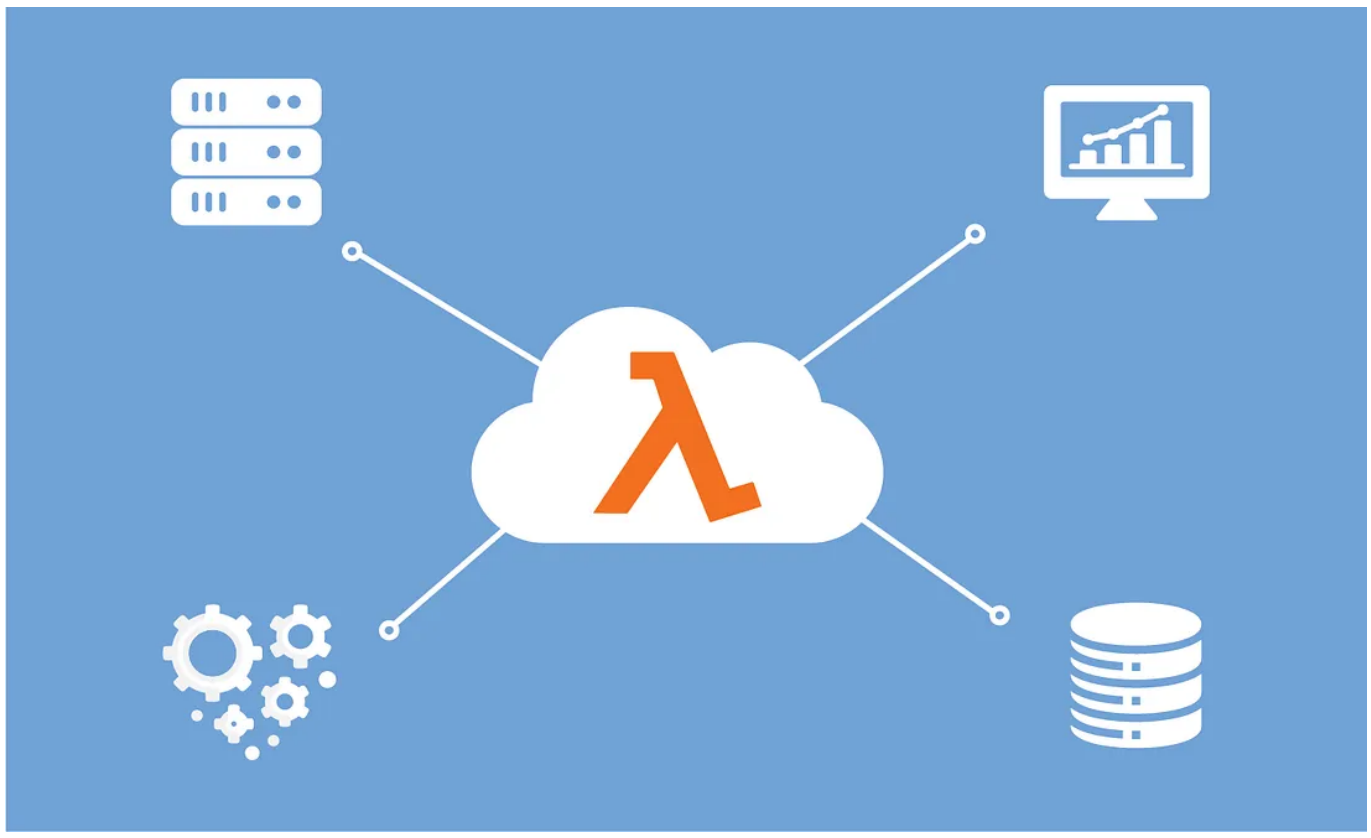
Member-only story

Everything You Need to Know About AWS Lambda

What is AWS Lambda, how does it work and why you should care.

Nikki Siapno · Follow

Published in The Startup · 11 min read · Aug 4, 2020



Lambda Illustration | Created by Author

"Serverless" has been the buzz word for several years now, with many applications choosing to implement the serverless approach. The term originated from the idea that the infrastructure used to run your backend code does not need to be provisioned and managed by you and your team. This significantly lessens the time it takes to get your application production-ready as well as the time and effort required to maintain your infrastructure. In 2014, Amazon Web Services released a product that would eventually become a gem in the wide pool of serverless solutions; that product is known as Lambda. In this article, we'll take a look at why Lambda is worth your attention as well as the disadvantages you'll want to consider, we'll walk through the most prominent features of this service and explore its inner workings.

. . .

What is AWS Lambda?

As a brief overview, AWS Lambda is a function-based computing service that takes the efforts of provisioning and maintaining its infrastructure out of your hands. With Lambda, you don't need to worry about scaling your infrastructure and removing unnecessary resources as this is all handled for you. We'll take a deeper dive into how this service works, but first let's take a look at why this tool is a worthy addition to your stack.

. . .

Advantages of AWS Lambda

Many of the advantages of using AWS Lambda relates to the advantages of adopting the serverless-approach in general. As mentioned in the intro, a major benefit of going serverless is the time and effort saved from creating and maintaining your infrastructure. AWS provisions and manages the infrastructure your Lambda functions run on, scales the instances to handle times of excessive load, and implements proper logging and error handling. Anyone that's been involved in the creation or maintenance of infrastructure will understand the gravity of this advantage. Not only is there a large amount of time involved in building a system that suits the needs of your application, there is also a considerable amount of time required to maintain that system as your application evolves. Time saved means quicker time to market for your application, greater agility as your team is able to most faster, and more time spent on more important tasks such as bug fixes or new features.

As for why AWS Lambda is one of the most popular serverless solutions, AWS has done a very good job of ensuring Lambda accommodates for applications at scale as well as applications in early stages. For applications with large amounts of load, AWS allows you to run your Lambda function simultaneously with other Lambda functions; meaning, you won't need to worry about clogged up queues. Not only that, multiple instances of the same Lambda function can be provisioned and run at the same time. Both advantages ensures that no matter how much load your application is under, Lambda will be able to handle it. Another advantage of using AWS Lambda is that you only pay for what you need; accomodating for applications that are not yet at scale or have widely differing loads. AWS charges you for the number of requests your Lambda functions recieve and the time it takes to execute those requests per 100ms. Despite its wide array of advantages, there isn't a single solution that exists without its share of disadvantages and AWS Lambda is no exception.

. . .

Disadvantages of AWS Lambda

Moving the task of maintaining your infrastructure away from your team and in the hands of a provider results in less control and flexibility, which is the biggest disadvantage of the serverless approach. On top of that, services that help implement the serverless approach come with their own set of infrastructure-related limitations; in Lambda's case, these limitations are the following:

- Functions will timeout after 15 minutes.
- The amount of RAM available ranges from 128MB to 3008MB with a 64MB increment between each option.
- The Lambda code should not exceed 250mb in size, and the zipped version should be no larger than 50mb
- There is a limit of 1,000 requests that can run concurrently, any request above this limit will be throttled and will need to wait for other functions to finish running.

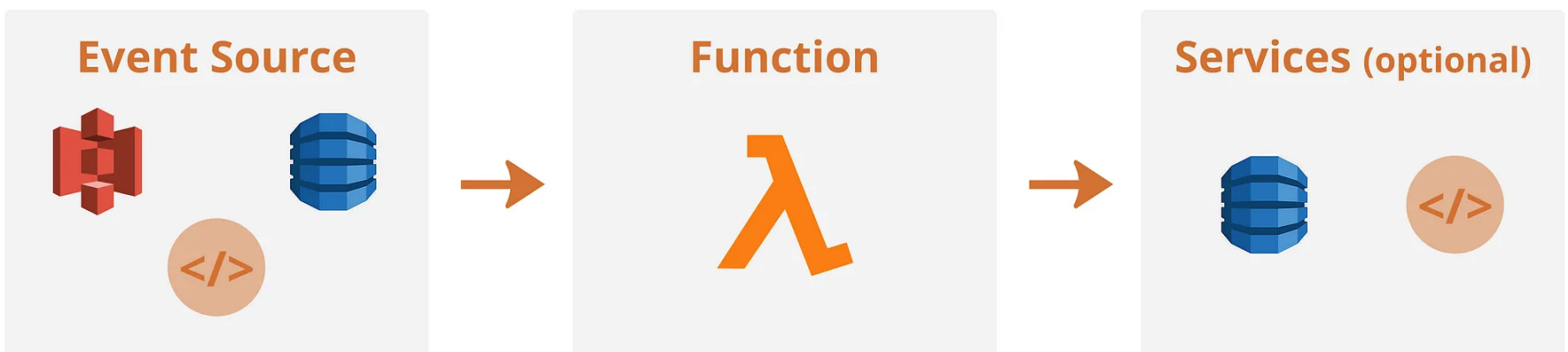
Whether or not these limitations will impact your application is dependant on the nature of your Lambda functions; usually, the solution is to refactor your Lambda functions to improve their efficiency. If any of these limitations begin to impact your Lambda functions, the first thing to do is to investigate why and whether your functions could be improved. For example, is the reason your function is timing out is because there's inefficient algorithms involved? Are there any unnecessary dependencies in your Lambda code, causing its size to exceed the limit?

Cost was mentioned in our list of advantages, but although you only pay for what your application requires this does not necessarily result in a cost effective solution; during times of high load, the cost of the same infrastructure on AWS EC2 or other services may be cheaper. The price of other services based on your application's needs should be considered especially if your application experiences consistently high load. The final disadvantage worth mentioning is the small latency time between when an event occurs and when the function runs. This small latency times only occurs in some cases — during a *cold start*. In most cases, these latency times are so miniscule that it's hardly an issue but it's still worth considering if your application is already bordering towards potential load problems; I'll talk about cold starts in more detail in a later section.

PROS	CONS
Less time spent maintaining infrastracture	Less flexibility and control
Built-in scalability	Not necessarily cheaper
Only pay for what you need	Potential cold starts
Integrates with other AWS services	

AWS Lambda Pros & Cons Table | Created by Author

If you've decided that Lambda may be a worthy addition to your stack, the next step is to understand the inner workings of a Lambda function. On a very basic level, serverless applications are made up of 2 or 3 components; these are event sources, functions and (in some cases) services. An event source encapsulates anything that can invoke a function, such as uploads to an S3 bucket, changes to data state or requests to an endpoint. When any one of the designated events occurs, your Lambda function will run in its own container. The resources allocated to that container and the number of containers used is determined by the size of the data and the computational requirements of the function, this is all handled by AWS. Once the request is completed, your Lambda function will either return a result back to the invocation source or a connected service, or it could make changes to a connected service (such as a database).



(note: this is not an extensive list of examples)

AWS Lambda Flow Chart Diagram | Created by Author

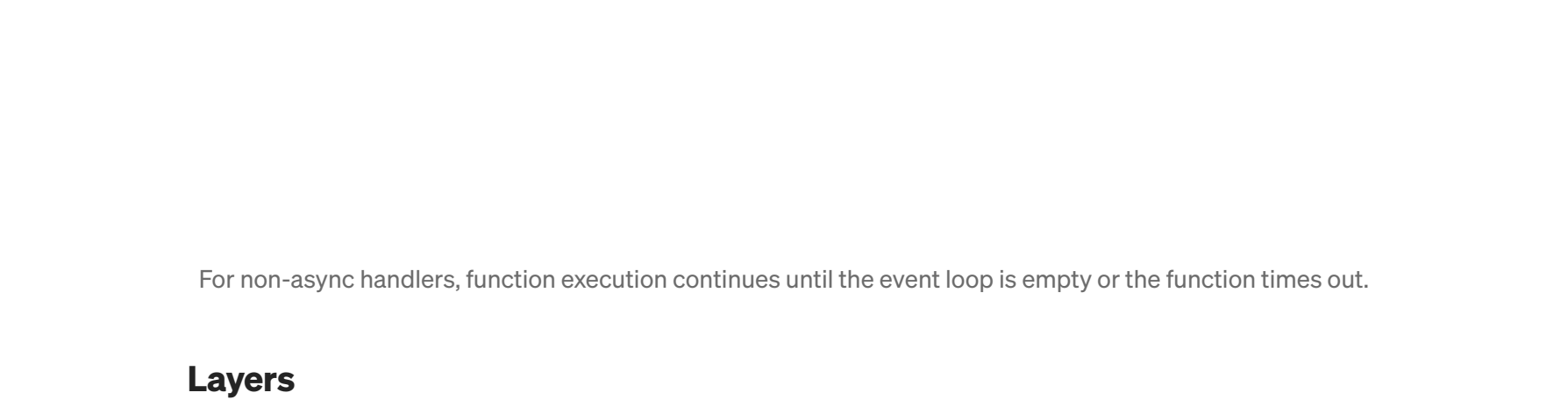
Before you can run a Lambda function, you'll need to create one and to successfully do so, you'll need a basic understanding of what's involved. A Lambda function consists of 3 or 4 parts; the handler function, the event object, the context object and in some cases a callback function. The handler function is the function that will be executed upon invocation, this can either be async or non-async. Asynchronous functions take an event object and a context object whereas non-asynchronous functions take both these objects and a callback function. The event object contains data that was sent when the function's event was triggered, this includes information such as the request body and the uri; the data that is passed through depends on the invocation service. The context object contains runtime information such as the function name, function version and log group. The callback function is only passed through to synchronous handlers and it takes two arguments: an

error and a response. Once the Lambda function is created and pushed up to AWS, it is compressed along with its dependencies and stored in an S3 bucket.

For non-async handlers, function execution continues until the event loop is empty or the function times out.

Layers

Once you start to build your Lambda functions you'll notice that there's pieces of logic that could be shared between multiple functions, this is when layers can come in handy. Layers allow you to reuse code across several functions without needing an additional invocation. Once you've identified a piece of code that could be reused, implement it as a layer and attach it to the functions that need it. A layer is created in the same fashion as a Lambda function, with slight configuration changes which depend on the method you've chosen to deploy your functions (via AWS's GUI, using the serverless framework, or AWS's CLI tool). This is also true for adding a layer to a



Alias Use Cases Diagram | Created by Author

Permissions

AWS not only allows you to save different versions of your Lambda functions but also allows these versions to coexist and run at the same time, this gives consumers of your functions the flexibility to upgrade to newer versions as

they please. Aliases are used as a pointer to a particular version of a Lambda function, there is a long list of use cases in which they can be utilized in; two of which is worth mentioning. Firstly, instead of updating the version of a client, you'll have to enable it to do so. Permissions surrounding your function everywhere it's called, you could use an alias in these areas and update the version that the alias points to. Secondly, aliases have the ability to point to two versions and give you the flexibility to determine the percentage of traffic to be sent to each version. This can be very useful if you and your team wanted to test a new version of a function with a small percentage of your traffic before releasing the new version universally. versions, aliases and layer versions.

Relatively speaking, your Lambda functions are considerably secure by default; your function can't talk to other services nor can it be invoked by any client, you'll have to enable it to do so. Permissions surrounding your Lambda functions fall into two buckets: execution policies and resource-based policies. Execution policies determine which services and resources a Lambda function has access to, as well as which event sources can trigger a Lambda function to run. Resource-based policies grant other accounts and AWS services access to your Lambda resources, these include functions, versions, aliases and layer versions.

Resilience

AWS helps to ensure that your Lambda function is able to handle faults without impacting your entire application using a set of features they've included into Lambda. The most notable features have already been mentioned in this article and those are Lambda's scalability, versioning and ability to run concurrently. A couple of other features that contribute to the service's resilience is their use of multiple availability zones and the ability to reserve concurrency. By default, AWS runs your Lambda functions in multiple availability zones, this ensures that your functions are not impacted if a single zone is down; the same cannot be said for services such as EC2 where this behaviour must be set by the developer. With Lambda, developers have the ability to set reserved concurrency for a particular function which ensures that it can always scale to (but not exceed past) a set number of concurrent invocations despite the number of requests other functions are consuming — note that AWS will still adhere to the upper limit of 1,000 requests, which means requests for other functions will be throttled.

Cold Starts

Cold starts occur when a function has been idle for a long enough period of time that its container has been completely terminated. A new container is provisioned when the function is invoked resulting in a small amount of latency. At times, an idle Lambda container is available to pick up new requests; if this is the case, provisioning a new container isn't necessary — this is called a *warm start*. The period of time a Lambda function can be idle for before it gets terminated isn't well-documented but an experiment in 2017 found that most functions were terminated after 45–60 minutes of inactivity; potentially earlier if resources are needed by other customers. The amount of time it takes for a function to start up is influenced by its scripting language, whether the function is outside of a VPC (if it is, start up time will be faster), how big the package size is and how much memory is allocated to the function. Whether or not your application will likely experience cold starts depends on the amount of variation between your load levels. A fairly constant amount of load will mean that your application will require the same number of containers most of the time, which results in more warm starts as a container will likely be available for most requests.

Final Thoughts

There is a great number of use cases in which AWS Lambda functions would thrive in. Tasks that are self-contained and run for a short period of time are great candidates for a Lambda function. These include processing operations (such as formatting an image), automation tasks where an entire server isn't required at all times (such as data backups) or functions used to support static websites (such as capturing emails). If you're in the MVP stage of an application, Lambda is a great tool to utilise to get your application market-ready quicker and test your product faster. I don't believe serverless must be an "all or nothing" approach; for example, it is very reasonable for teams to use services such as EC2 throughout most of their application while using Lambda in other areas (such as their landing page or other marketing efforts). Which ever use case it may be for you, I hope this article has convinced you to add AWS Lambda into your list of possible tools to utilise.

. . .

Thanks for reading, I hope there were gems in this article for you 💎
What are your thoughts on AWS Lambda and/or serverless functions? 🤔
Always happy to have a healthy discussion 🙂

If you enjoy reading articles like this, consider signing up to become a Medium member. It's only $5 a month, giving you unlimited access to articles on Medium. If you sign up using my link below, I'll earn a small commission.

Resources

AWS Lambda - Serverless Compute - Amazon Web Services

Run code without thinking about servers. Pay only for the compute time you consume. AWS Lambda lets you run code...

aws.amazon.com

AWS Lambda - The Ultimate Guide

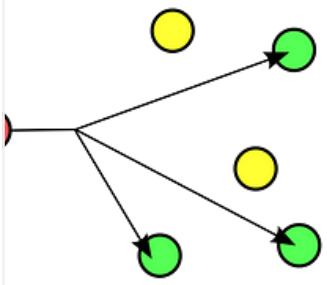
Are you looking to use AWS Lambda for the first time? Or are you evaluating its use for a production environment? We've...

www.serverless.com

AWS Lambda: Sync or Async? | Stackery

When you execute your code within AWS Lambda, the functions can either be invoked synchronously or asynchronously...

www.stackery.io



AWS Lambda function handler in Node.js

The handler is the method in your Lambda function that processes events. When you invoke a function, the runtime runs...

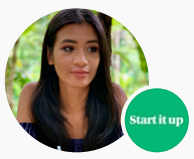
docs.aws.amazon.com

Introduction to AWS Lambda.

AWS Lambda is a Function-as-a-service (FaaS) computing platform provided by AWS. FaaS provides a computing platform to...

dashbird.io

DevOps | Software Development | Programming | AWS | AWS Lambda



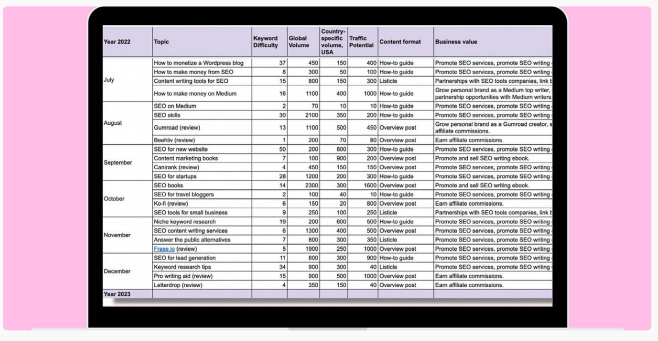
Written by Nikki Siapno

414 Followers · Writer for The Startup

Follow

Engineer Manager | Product Builder | Educator | Free Resources + My Work 👉
links.nikkiandchris.io | twitter.com/nikkisiapno

More from Nikki Siapno and The Startup



Nikki Siapno in Geek Culture

10 Fun Coding Project Ideas to Get Hired as a Junior Engineer

These projects allow you to showcase the attributes technical leaders are looking for,...

· 10 min read · Dec 16, 2021

-- 6

Tim Denning in The Startup

The Most High-Income Habit You Can Start in the Next 30 Days

No one talks about this. And no it's not "build an audience"...LOL.

· 5 min read · Oct 12

-- 86

Victoria Kurichenko in The Startup

My Content Writing Plan That Makes Me 4 Figures a Month

You can replicate it for your website.

· 4 min read · Oct 3

-- 87

Nikki Siapno in Geek Culture

These 7 CSS Generators Will Save You Hours of Work

Spend more time on other things while looking like a frontend champion.

· 4 min read · Nov 2, 2022

-- 5

See all from Nikki Siapno

See all from The Startup

Recommended from Medium

Vaishnav Manoj in DataX Journal

JSON is incredibly slow: Here's What's Faster!

Unlocking the Need for Speed: Optimizing JSON Performance for Lightning-Fast Apps...

16 min read · Sep 28

-- 68

Arslan Ahmad in Level Up Coding

Don't Just LeetCode; Follow the Coding Patterns Instead

What if you don't like to practice 100s of coding questions before the interview?

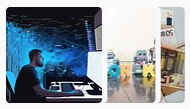
5 min read · Sep 15, 2022

-- 31

Lists

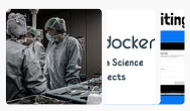
General Coding Knowledge

20 stories · 518 saves



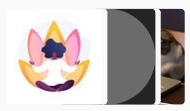
It's never too late or early to start something

15 stories · 189 saves



Coding & Development

11 stories · 248 saves



Stories to Help You Grow as a Software Developer

19 stories · 505 saves

Cloudmize in Towards AWS

5 Best Strategies I Employed to Pass All 12 AWS Certifications Within a...

Efficient approaches to mastering AWS certification exams in a short timeframe

16 min read · 1 day ago

-- 17

Hassan Jalil in AWS in Plain English

Ramping up AWS Lambda with Docker

How with help of Docker images, we can make much more powerful Lambda...

4 min read · May 24

--

Samsonk in AWS Tip

Run AWS on Your Laptop. Introduction to LocalStack.

What is LocalStack

5 min read · Aug 18

-- 12

subham polpagedar in Simform Engineering

Creating Lambda Layers Made Easy with Docker: A Developer's Guide

Streamline AWS Lambda development with Docker. Simplify dependencies, ensure...

6 min read · Jun 14

--

See more recommendations