**AWS Lambda**

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). Users of AWS Lambda create functions, self-contained applications written in one of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner.  
  
The Lambda functions can perform any kind of computing task, from serving web pages and processing streams of data to calling APIs and integrating with other AWS services.  
  
The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so “serverless” doesn’t mean that there are no servers involved: it just means that the servers, the operating systems, the network layer and the rest of the infrastructure have already been taken care of, so that you can focus on writing application code.

**How does AWS Lambda work?**

Each Lambda function runs in its own container. When a function is created, Lambda packages it into a new container and then executes that container on a multi-tenant cluster of machines managed by AWS. Before the functions start running, each function’s container is allocated its necessary RAM and CPU capacity. Once the functions finish running, the RAM allocated at the beginning is multiplied by the amount of time the function spent running. The customers then get charged based on the allocated memory and the amount of run time the function took to complete.  
  
The entire infrastructure layer of AWS Lambda is managed by AWS. Customers don’t get much visibility into how the system operates, but they also don’t need to worry about updating the underlying machines, avoiding network contention, and so on—AWS takes care of this itself.  
  
And since the service is fully managed, using AWS Lambda can save you time on operational tasks. When there is no infrastructure to maintain, you can spend more time working on the application code—even though this also means you give up the flexibility of operating your own infrastructure.  
  
One of the distinctive architectural properties of AWS Lambda is that many instances of the same function, or of different functions from the same AWS account, can be executed concurrently. Moreover, the concurrency can vary according to the time of day or the day of the week, and such variation makes no difference to Lambda—you only get charged for the compute your functions use. This makes AWS Lambda a good fit for deploying highly scalable cloud computing solutions.

**Why is AWS Lambda an essential part of the Serverless architecture?**

When building Serverless applications, AWS Lambda is one of the main candidates for running the application code. Typically, to complete a Serverless stack you’ll need:

* a computing service;
* a database service; and
* an HTTP gateway service.

Lambda fills the primary role of the compute service on AWS. It also integrates with many other AWS services and, together with API Gateway, DynamoDB and RDS, forms the basis for Serverless solutions for those using AWS. Lambda supports many of the most popular languages and runtimes, so it’s a good fit for a wide range of Serverless developers.

**What are the most common use cases for AWS Lambda?**

Due to Lambda’s architecture, it can deliver great benefits over traditional cloud computing setups for applications where:

* individual tasks run for a short time;
* each task is generally self-contained;
* there is a large difference between the lowest and highest levels in the workload of the application.

Some of the most common use cases for AWS Lambda that fit these criteria are:  
Scalable APIs. When building APIs using AWS Lambda, one execution of a Lambda function can serve a single HTTP request. Different parts of the API can be routed to different Lambda functions via Amazon API Gateway. AWS Lambda automatically scales individual functions according to the demand for them, so different parts of your API can scale differently according to current usage levels. This allows for cost-effective and flexible API setups.  
‍  
Data processing. Lambda functions are optimized for event-based data processing. It is easy to integrate AWS Lambda with datasources like Amazon DynamoDB and trigger a Lambda function for specific kinds of data events. For example, you could employ Lambda to do some work every time an item in DynamoDB is created or updated, thus making it a good fit for things like notifications, counters and analytics.

**Task automation**

With its event-driven model and flexibility, AWS Lambda is a great fit for automating various business tasks that don’t require an entire server at all times. This might include running scheduled jobs that perform cleanup in your infrastructure, processing data from forms submitted on your website, or moving data around between different datastores on demand.

**Supported languages and runtimes**

As of now, AWS Lambda doesn’t support all programming languages, but it does support a number of the most popular languages and runtimes. This is the full list of what’s supported:

* Node.js 8.10
* Node.js 10.x (normally the latest LTS version from the 10.x series)
* Node.js 12.x (normally the latest LTS version from the 12.x series)
* Python 2.7
* Python 3.6
* Python 3.7
* Python 3.8
* Ruby 2.5
* Java 8 - This includes JVM-based languages that can run on Java 8’s JVM — the latest Clojure 1.10 and Scala 2.12 both run on Java 8 so can be used with AWS Lambda
* Java 11
* Go 1.x (latest release)
* C# — .NET Core 1.0
* C# — .NET Core 2.1
* PowerShell Core 6.0

All these runtimes are maintained by AWS and are provided in an Amazon Linux or Amazon Linux 2 environment. For each of the supported languages, AWS provides an SDK that makes it easier for you to write your Lambda functions and integrate them with other AWS services.  
‍  
A few additional runtimes are still in the pre-release stage. These runtimes are being developed as a part of AWS Labs and are not mentioned in the official documentation:

* Rust 1.31
* C++

The C++ runtime also serves as an example for creating custom runtimes for AWS Lambda. See the AWS docs for the details of how to create a custom runtime if your language isn’t supported by default.

**Benefits of using AWS Lambda**

AWS Lambda has a few unique advantages over maintaining your own servers in the cloud. The main ones are:  
  
Pay per use. In AWS Lambda, you pay only for the compute your functions use, plus any network traffic generated. For workloads that scale significantly according to time of day, this type of billing is generally more cost-effective.  
‍  
Fully managed infrastructure. Now that your functions run on the managed AWS infrastructure, you don’t need to think about the underlying servers—AWS takes care of this for you. This can result in significant savings on operational tasks such as upgrading the operating system or managing the network layer.  
‍  
Automatic scaling. AWS Lambda creates the instances of your function as they are requested. There is no pre-scaled pool, no scale levels to worry about, no settings to tune—and at the same time your functions are available whenever the load increases or decreases. You only pay for each function’s run time.  
‍  
Tight integration with other AWS products. AWS Lambda integrates with services like DynamoDB, S3 and API Gateway, allowing you to build functionally complete applications within your Lambda functions.

**Limitations of AWS Lambda**

While AWS Lambda has many advantages, there are a few things you should know before using it in production.

**Cold start time**

When a function is started in response to an event, there may be a small amount of latency between the event and when the function runs. If your function hasn’t been used in the last 15 minutes, the latency can be as high as 5-10 seconds, making it hard to rely on Lambda for latency-critical applications. There are ways to work around it, including a method we wrote about in our blog.

**Function limits**

The Lambda functions have a few limits applied to them:  
‍  
Execution time/run time. A Lambda function will time out after running for 15 minutes. There is no way to change this limit. If running your function typically takes more than 15 minutes, AWS Lambda might not be a good solution for your task.  
‍  
Memory available to the function. The options for the amount of RAM available to the Lambda functions range from 128MB to 3,008MB with a 64MB step.  
‍  
Code package size. The zipped Lambda code package should not exceed 50MB in size, and the unzipped version shouldn’t be larger than 250MB.  
‍  
Concurrency. By default, the concurrent execution for all AWS Lambda functions within a single AWS account are limited to 1,000. (You can request a limit increase for this number by contacting AWS support.)  
  
Any Lambda executions triggered above your concurrency limit will be throttled and will be forced to wait until other functions finish running.  
‍  
Payload size. When using Amazon API Gateway to trigger Lambda functions in response to HTTP requests (i.e. when building a web application), the maximum payload size that API Gateway can handle is 10MB.

**Not always cost-effective**

On AWS Lambda, you pay only for the used function runtime (plus any associated charges like network traffic). This can produce significant cost savings for certain usage patterns, for example, with cron jobs or other on-demand tasks. However, when the load for your application increases, the AWS Lambda cost increases proportionally and might end up being higher than the cost of similar infrastructure on AWS EC2 or other cloud providers.

**Limited number of supported runtimes**

While AWS Lambda allows adding custom runtimes, creating them can be a lot of work. So if the version of the programming language you are using isn’t supported on Lambda, you might be better off using AWS EC2 or a different cloud provider.