serverless

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 created this article to help you in both of these cases.

AWS Lambda is one of the most popular serverless computing services out there. It is also the most popular provider used with the Serverless Framework.

In this article we'll cover everything we know about AWS Lambda, its upsides and downsides, and we'll provide a list of resources so you can learn more and get hands-on experience in using AWS Lambda.

Table of Contents

What is 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

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:

1. individual tasks run for a short time;
2. each task is generally self-contained;
3. 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.

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

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

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

you might be better off using AWS EC2 or a different cloud provider.

AWS Lambda pricing

A number of AWS Lambda executions are included with the AWS Free Tier with every AWS account. Unlike some other services, the Lambda free tier isn't limited to 12 months. Both existing and new accounts get 1 million AWS Lambda requests plus 400,000 GB-seconds per month — Lambda's measure of function runtime and the memory allocated to a function.

Beyond the free tier the pricing for AWS Lambda is as follows:

Aspect	Pricing	Comment
Requests	$0.20 per 1M requests	The free tier includes 1M requests.
Function memory and run time	$0.0000166667 per GB-second	The free tier includes 400,000 GB-seconds.
Inbound network traffic to the Lambda function	Free	
Outbound network traffic—within the same AWS region	$0.01/GB	
Outbound network traffic—to other AWS regions	$0.02/GB	
Outbound network traffic—to public internet	$0.09/GB	Lower pricing per GB applies starting at 10TB/month.

Amazon API Gateway	$3.50 per 1M requests	If you are using API Gateway with Lambda functions. Lower pricing from 333M requests per month and up.

For each execution, the total cost will be the sum of all applicable factors, including the cost per request, the memory and run time, and the network traffic.

Example cost calculations

Scheduled jobs. Imagine that you run cron jobs in your AWS infrastructure that perform database rollovers. The job runs every night and the average run time is 10 minutes. The function uses 1GB of memory while running. There is no outbound network traffic generated by the function as it connects to your RDS database in the same availability zone.

Total compute: 30 days x 600 GB-seconds (10 minutes) = 18,000 GB-seconds Total requests: 30 days x 1 request per day = 30 requests

This usage is within the AWS free tier, so you'll be charged $0 for AWS Lambda. **Note:** You'll still pay for the RDS usage according to the RDS price list.

HTTP API. Let's assume that you're building a web application based entirely on an AWS Lambda backend. Let's also assume that you're great at marketing, so after a few months you'll have 10,000 users in the app every day on average. Each user's actions within the app will result in 100 API requests per day, again, on average. Your API runs in Lambda functions that use 512MB of memory, and serving each API request takes 1 second.

Total compute: 30 days x 10,000 users x 100 requests x 0.5GB RAM x 1 second = 15,000,000 GB-seconds Total requests: 30 days x 10,000 users x 100 requests = 30,000,000 requests

For the 30M requests you'll pay 30 x $0.20/1M requests$ =6/month on AWS Lambda.

All these requests go through Amazon API Gateway, so there for the 30M requests you'll pay 30 x $3.50/1M requests$ =105/month on API Gateway.

For the monthly 15M GB-seconds of compute on AWS Lambda you'll pay 15M * $0.0000166667/GB - second$ =250/month.

So the total cost of the API layer will be around $360/month with this load.

Frequently asked questions (FAQ)

Is AWS Lambda open source? No, AWS Lambda itself is a proprietary system that's only available within AWS.

Is AWS Lambda an API? AWS Lambda provides an API that you can use to perform many kinds of operations on your functions, from deploying a new version of the function's code to seeing the function's layers to deleting functions. You can see the full description of the AWS Lambda API in the API reference docs. You can also use AWS Lambda to **build** your own APIs: check out the resources section of this article for more details.

Can AWS Lambda call other AWS Lambda functions? Yes. One option is to use the Lambda API (invoked via the AWS SDK in your function code) to call another function. Another option is to use a queue service like Amazon SNS to create an SNS message from one function and then use it as an event to trigger the second function. Yet another option is to use AWS Step Functions to build a workflow that includes multiple functions.

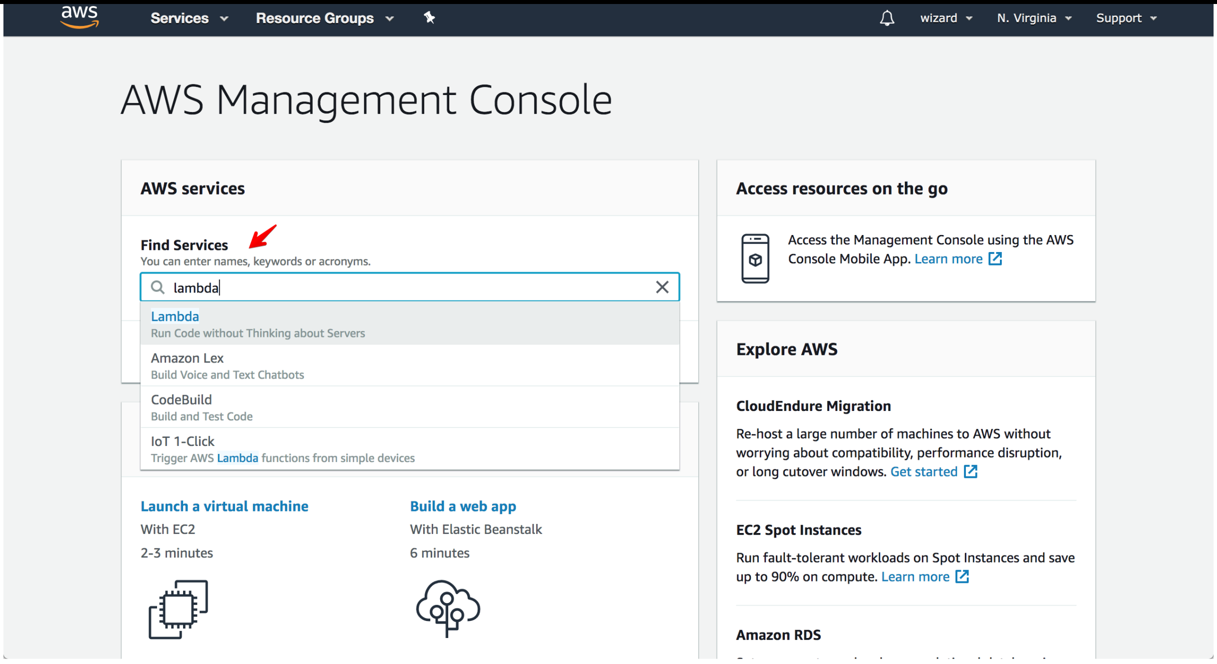
Getting started with AWS Lambda

In this section we cover the steps needed to create, deploy and secure AWS Lambda functions. We cover two approaches: using the AWS console and API directly, and using the Serverless Framework.

Creating AWS Lambda functions using the AWS console

If you wish, you can use the AWS Lambda console to create your first function. In the AWS console, choose Lambda:

serverless



AWS Management Console

AWS services

Find Services
You can enter names, keywords or acronyms.

lambda

Lambda
Run Code without Thinking about Servers

Amazon Lex
Build Voice and Text Chatbots

CodeBuild
Build and Test Code

IoT 1-Click
Trigger AWS Lambda functions from simple devices

Launch a virtual machine
With EC2
2-3 minutes

Build a web app
With Elastic Beanstalk
6 minutes

Access resources on the go

Access the Management Console using the AWS Console Mobile App. Learn more

Explore AWS

CloudEndure Migration

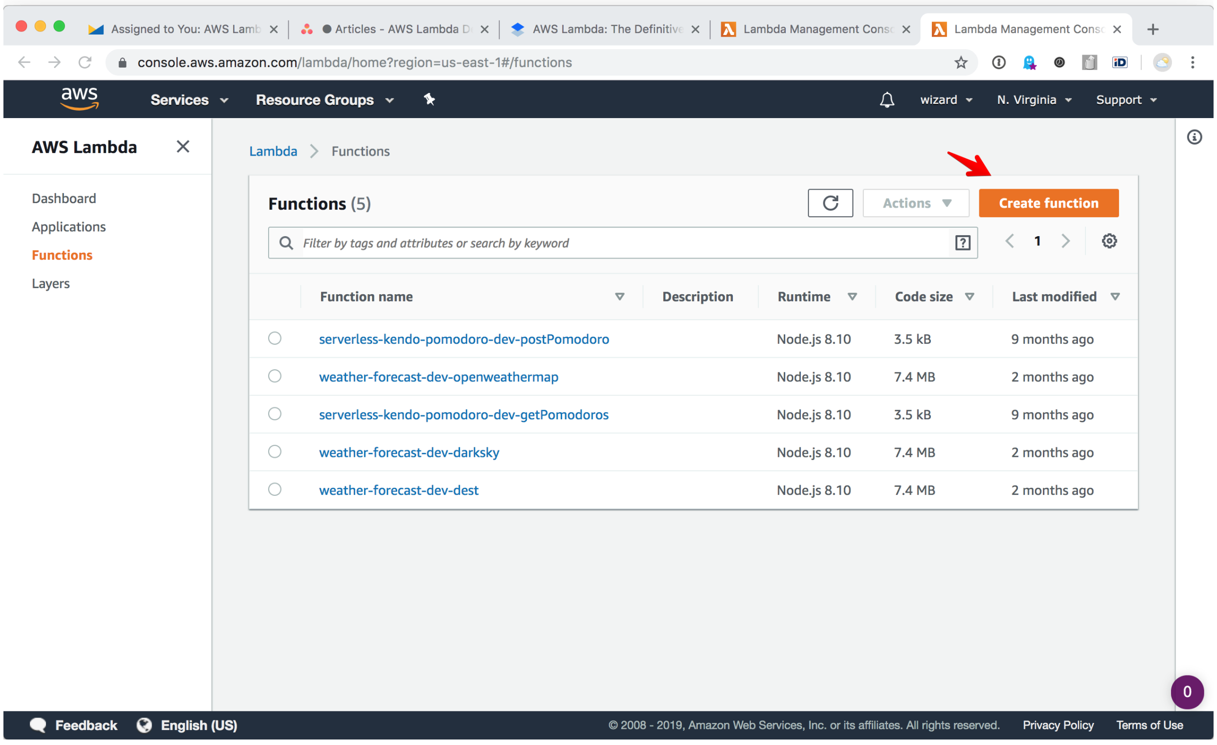
Re-host a large number of machines to AWS without worrying about compatibility, performance disruption, or long cutover windows. Get started

EC2 Spot Instances

Run fault-tolerant workloads on Spot Instances and save up to 90% on compute. Learn more

Amazon RDS

Once in the Lambda management console, click Create Function:



Functions (5)

Function name	Description	Runtime	Code size	Last modified
serverless-kendo-pomodoro-dev-postPomodoro		Node.js 8.10	3.5 kB	9 months ago
weather-forecast-dev-openweathermap		Node.js 8.10	7.4 MB	2 months ago
serverless-kendo-pomodoro-dev-getPomodoros		Node.js 8.10	3.5 kB	9 months ago
weather-forecast-dev-darksky		Node.js 8.10	7.4 MB	2 months ago
weather-forecast-dev-dest		Node.js 8.10	7.4 MB	2 months ago

Add a name for your new function and choose the desired runtime. After that, click "Create function" to confirm the settings:

We don’t recommend creating production Lambda functions via the management console for a two reasons:

- Your function code will be limited to 30MB.
- You’ll need to use the AWS web IDE to write the code in the browser. This IDE only provides very basic revision tracking and limits the collaboration on the code.

Creating AWS Lambda functions using the Serverless Framework

We recommend getting started with AWS Lambda by using Serverless Framework. With the Serverless Framework, you can create Lambda functions using the tools you’re familiar with on your local machine and deploy to AWS in seconds. With this approach, your function’s code and configuration are located in the same Git repository which makes collaboration, change tracking and deployment of Lambda functions easier.

1. Install Serverless Framework on your machine:

```
$ npm install serverless -g
```

2. Create a new service:

```
$ serverless
```

3. Add the resources your function needs to the `serverless.yml` file. Check out the AWS Intro doc for an example of this file and the list of options you can configure there.
4. Add code to your service. See the Serverless AWS provider docs for specific steps you can follow to create your functions.
5. Deploy to AWS by running the deploy step:

```
$ serverless deploy
```

That’s it! Your function will be deployed and you’ll see the URL of the function’s endpoint in your console.

Implementing AWS Lambda functions in production

Once your Lambda functions are deployed, you most likely want to secure and monitor them in order to serve production traffic adequately.

Monitoring Lambda functions with AWS CloudWatch

Once your function is deployed, you can use CloudWatch to set up the monitoring for your Lambda applications. CloudWatch provides a few default metrics that are available to you as soon as the function is deployed. To access these, head to the Lambda interface in the AWS console and choose Functions → Monitoring. You can view the metrics directly in the Lambda interface or head to CloudWatch to see more details. See the Lambda metrics doc for exact steps to follow. Keep in mind that the CloudWatch pricing applies for these default metrics. CloudWatch allows setting up basic alerts on the metrics using CloudWatch Alarms.

Securing the Lambda functions

Many of the functions you deploy on AWS Lambda are likely to require access to other services or your internal infrastructure. We provide a number of recommendations on how to secure these credentials in our article titled Secrets Management for AWS Powered Serverless Applications.

Out-of-the-box monitoring and security with the Serverless Framework

If you’ve deployed your function using Serverless Framework, your functions are already secured and monitored. You don’t need to do anything else. Just head to the Serverless Dashboard to see the up-to-date metrics and security events for your function.

Note: to take advantage of the metrics and the built-in secrets management mechanism you need to run `serverless login` before deploying your Serverless application.

Links and references

- Serverless Framework
- AWS provider reference — Serverless Framework

- Examples using AWS Lambda
  - A simple HTTP endpoint — Node.js
  - A simple HTTP endpoint — Ruby
  - A simple HTTP endpoint — Python
  - A REST API with DynamoDB — Python

New to serverless?

To get started, pop open your terminal & run:

```
npm install serverless -g
```

get started

documentation

examples