AWS Lambda

General

* Pay for only compute time that is used
* No charge when code is not running
* Run code with zero administration
* Runs and scale code with high availability
* Set code to automatically trigger from other services or call directly from any web or mobile app
* Serverless computing allows you to build and run apps and services without thinking about servers
* Serverless computing – server management done by AWS
* Event sources that can trigger lambda –
* AWS Lambda vs Amazon EC2
  + Amazon EC2 – instance types, customize operating system network and security settings and the entire software stack, provisioning capacity monitoring fleet health and performance designing for fault tolerance and scalability
  + AWS Elastic Beanstalk – easy to use service for deploying and scaling web apps in which you retain ownership and full control over the underlying EC2 instance
  + Amazon EC2 Container Service – scalable management service that supports docker containers and allows easily run distributed applications on a managed cluster of amazon ec2 instances
  + AWS lambda – build mobile back-ends that retrieve and transform data from amazon dynamodb, handlers that compress or transform objects as the are uploaded to Amazon S3, auditing and reporting of api calls made to any amazon web servce and server-less processing of string data using amazon kinesis
* Aws lambda supported languages – java, go powershell,node.js c# python ruby
* You cannot access the infrastructure that AWS lambda runs on
* AWS lambda function runs in its own isolated environment with its own resources and file system view, same techniques as EC2 to provide security and separation at infra and execution level

AWS Lambda Functions

Each function has associated configuration information, such as its name, description, entry point, and resource requirements

The code must be written in a “stateless” style i.e. it should assume there is no affinity to the underlying compute infrastructure.

any persistent state should be stored in Amazon S3, Amazon DynamoDB, or another Internet-available storage service.

Lambda functions can include libraries, even native ones.

To improve performance, AWS Lambda may choose to retain an instance of your function and reuse it to serve a subsequent request, rather than creating a new copy.

Each Lambda function receives 500MB of non-persistent disk space in its own /tmp directory.

Keeping functions stateless enables AWS Lambda to rapidly launch as many copies of the function as needed to scale to the rate of incoming events

While AWS Lambda’s programming model is stateless, your code can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.

 AWS Lambda allows you to use normal language and operating system features, such as creating additional threads and processes.

Resources allocated to the Lambda function, including memory, execution time, disk, and network use, must be shared among all the threads/processes it uses

You can launch processes using any language supported by Amazon Linux.

Inbound network connections are blocked by AWS Lambda, and for outbound connections only TCP/IP and UDP/IP sockets are supported, and ptrace (debugging) system calls are blocked.

TCP port 25 traffic is also blocked as an anti-spam measure.

You can also package the code (and any dependent libraries) as a ZIP and upload it using the AWS Lambda console from your local environment or specify an Amazon S3 location where the ZIP file is located. Uploads must be no larger than 50MB (compressed).

You can use the AWS Eclipse plugin to author and deploy Lambda functions in Java. You can use the Visual Studio plugin to author and deploy Lambda functions in C#, and Node.js.

You can easily create and modify environment variables from the AWS Lambda Console, CLI or SDKs.

For sensitive information, such as database passwords, we recommend you use client-side encryption using [AWS Key Management Service](http://docs.aws.amazon.com/kms/latest/developerguide/overview.html) and store the resulting values as ciphertext in your environment variable.

You can easily list, delete, update, and monitor your Lambda functions using the dashboard in the AWS Lambda console.

You can package any code (frameworks, SDKs, libraries, and more) as a [Lambda Layer](https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html) and manage and share them easily across multiple functions.

AWS Lambda automatically monitors Lambda functions on your behalf, reporting real-time metrics through Amazon CloudWatch, including total requests, account-level and function-level concurrency usage, latency, error rates, and throttled requests. You can view statistics for each of your Lambda functions via the Amazon CloudWatch console or through the AWS Lambda console.

AWS Lambda automatically integrates with Amazon CloudWatch logs, creating a log group for each Lambda function and providing basic application lifecycle event log entries, including logging the resources consumed for each use of that function.

you can easily insert additional logging statements into your code. You can also call third-party logging APIs in your Lambda function.

Every time an event notification is received for your function, AWS Lambda quickly locates free capacity within its compute fleet and runs your code. Since your code is stateless, AWS Lambda can start as many copies of your function as needed without lengthy deployment and configuration delays. There are no fundamental limits to scaling a function. AWS Lambda will dynamically allocate capacity to match the rate of incoming events.

In the AWS Lambda resource model, you choose the amount of memory you want for your function, and are allocated proportional CPU power and other resources. For example, choosing 256MB of memory allocates approximately twice as much CPU power to your Lambda function as requesting 128MB of memory and half as much CPU power as choosing 512MB of memory.

You can set your memory in 64MB increments from 128MB to 3GB

AWS Lambda functions can be configured to run up to 15 minutes per execution. You can set the timeout to any value between 1 second and 15 minutes.

AWS Lambda is priced on a pay per use basis.

 In addition to saving money on Amazon EC2 and AWS Fargate, you can also use Compute Savings Plans to save money on AWS Lambda.

Compute Savings Plans offer up to 17% discount on Duration, Provisioned Concurrency, and Duration (Provisioned Concurrency).

Compute Savings Plans do not offer a discount on Requests in your Lambda bill. However, your Compute Savings Plans commitment can apply to Requests at regular rates.

By default, each AWS Lambda function has a single, current version of the code. Clients of your Lambda function can call a specific version or get the latest implementation.

Deployment times may vary with the size of your code, but AWS Lambda functions are typically ready to call within seconds of upload

you can include your own copy of a library (including the AWS SDK) in order to use a different version than the default one provided by AWS Lambda

Using AWS Lambda to process AWS events

An event source is an AWS service or developer-created application that produces events that trigger an AWS Lambda function to run.

Lambda can pull records from an Amazon Kinesis stream or an Amazon SQS queue and execute a Lambda function for each fetched message.

Many other services, such as AWS CloudTrail, can act as event sources simply by logging to Amazon S3 and using S3 bucket notifications to trigger AWS Lambda functions.

Events are passed to a Lambda function as an event input parameter. For event sources where events arrive in batches, such as Amazon SQS, Amazon Kinesis, and Amazon DynamoDB Streams, the event parameter may contain multiple events in a single call, based on the batch size you request.

From the AWS Lambda console, you can select a function and associate it with notifications from an Amazon S3 bucket. Alternatively, you can use the Amazon S3 console and configure the bucket’s notifications to send to your AWS Lambda function.

You can trigger a Lambda function on DynamoDB table updates by subscribing your Lambda function to the DynamoDB Stream associated with the table.

You can associate a DynamoDB Stream with a Lambda function using the Amazon DynamoDB console, the AWS Lambda console or Lambda’s registerEventSource API.

From the AWS Lambda console, you can select a Lambda function and associate it with an Amazon Kinesis stream owned by the same account.

The Amazon Kinesis and DynamoDB Streams records sent to your AWS Lambda function are strictly serialized, per shard. This means that if you put two records in the same shard, Lambda guarantees that your Lambda function will be successfully invoked with the first record before it is invoked with the second record. If the invocation for one record times out, is throttled, or encounters any other error, Lambda will retry until it succeeds (or the record reaches its 24-hour expiration) before moving on to the next record. The ordering of records across different shards is not guaranteed, and processing of each shard happens in parallel.

From the AWS Lambda console, you can select a Lambda function and associate it with an Amazon SNS topic.

From the Amazon SES Console, you can set up your receipt rule to have Amazon SES deliver your messages to an AWS Lambda function.

First, configure the alarm to send Amazon SNS notifications. Then from the AWS Lambda console, select a Lambda function and associate it with that Amazon SNS topic.

rom the AWS Lambda console, you can select a function to trigger when any datasets associated with an [Amazon Cognito](https://aws.amazon.com/cognito/) identity pool are synchronized.

You can invoke a Lambda function using a custom event through AWS Lambda’s invoke API. Only the function’s owner or another AWS account that the owner has granted permission can invoke the function.

AWS Lambda is designed to process events within milliseconds. Latency will be higher immediately after a Lambda function is created, updated, or if it has not been used recently.

You upload the code you want AWS Lambda to execute and then invoke it from your mobile app using the AWS Lambda SDK included in the AWS Mobile SDK. You can make both direct (synchronous) calls to retrieve or check data in real time as well as asynchronous calls. You can also define a custom API using Amazon API Gateway and invoke your Lambda functions through any REST compatible client.

You can invoke a Lambda function over HTTPS by defining a custom RESTful API using Amazon API Gateway. This gives you an endpoint for your function which can respond to REST calls like GET, PUT and POST.

When called through the AWS Mobile SDK, AWS Lambda functions automatically gain insight into the device and application that made the call through the ‘context’ object.

When your app uses the Amazon Cognito identity, end users can authenticate themselves using a variety of public login providers such as Amazon, Facebook, Google, and other OpenID Connect-compatible services. User identity is then automatically and secured presented to your Lambda function in the form of an Amazon Cognito id, allowing it to access user data from Amazon Cognito, or as a key to store and retrieve data in Amazon DynamoDB or other web services.

AWS Lambda is integrated with the Alexa Skills Kit, a collection of self-service APIs, tools, documentation and code samples that make it easy for you to create voice-driven capabilities (or “skills”) for Alexa. You simply upload the Lambda function code for the new Alexa skill you are creating, and AWS Lambda does the rest, executing the code in response to Alexa voice interactions and automatically managing the compute resources on your behalf.

For Amazon S3 bucket notifications and custom events, AWS Lambda will attempt execution of your function three times in the event of an error condition in your code or if you exceed a service or resource limit.

For ordered event sources that AWS Lambda polls on your behalf, such as Amazon DynamoDB Streams and Amazon Kinesis streams, Lambda will continue attempting execution in the event of a developer code error until the data expires. You can monitor progress through the Amazon Kinesis and Amazon DynamoDB consoles and through the Amazon CloudWatch metrics that AWS Lambda generates for your function. You can also set Amazon CloudWatch alarms based on error or execution throttling rates.

## Using AWS Lambda to build applications

You can deploy and manage your serverless applications using the AWS Serverless Application Model (AWS SAM). AWS SAM is a specification that prescribes the rules for expressing serverless applications on AWS. This specification aligns with the syntax used by AWS CloudFormation today and is supported natively within AWS CloudFormation as a set of resource types (referred to as "serverless resources"). These resources make it easier for AWS customers to use CloudFormation to configure and deploy serverless applications, using existing CloudFormation APIs.

You can choose from a collection of serverless applications published by developers, companies, and partners in the AWS community with the [AWS Serverless Application Repository](https://aws.amazon.com/serverless/serverlessrepo/).

You can automate your serverless application’s release process using AWS CodePipeline and AWS CodeDeploy. CodePipeline is a continuous delivery service that enables you to model, visualize and automate the steps required to release your serverless application. CodeDeploy provides a deployment automation engine for your Lambda-based applications. CodeDeploy lets you orchestrate deployments according to established best-practice methodologies such as canary and linear deployments, and helps you establish the necessary guardrails to verify that newly-deployed code is safe, stable, and ready to be fully released to production.

You can use [AWS Step Functions](https://aws.amazon.com/step-functions/) to coordinate a series of AWS Lambda functions in a specific order. You can invoke multiple Lambda functions sequentially, passing the output of one to the other, and/or in parallel, and Step Functions will maintain state during executions for you.

You can enable your Lambda function for tracing with [AWS X-Ray](https://aws.amazon.com/xray/) by adding X-Ray permissions to your Lambda function’s execution role and changing your function’s “tracing mode” to “active. ” When X-Ray is enabled for your Lambda function, AWS Lambda will emit tracing information to X-Ray regarding the Lambda service overhead incurred when invoking your function. This will provide you with insights such as Lambda service overhead, function init time, and function execution time. In addition, you can include the X-Ray SDK in your Lambda deployment package to create your own trace segments, annotate your traces, or view trace segments for downstream calls made from your Lambda function. X-Ray SDKs are currently available for Node.js and Java.

Yes. You can build highly scalable, secure, Lambda-based serverless applications that connect to relational databases using [Amazon RDS Proxy](https://aws.amazon.com/rds/proxy/), a highly available database proxy that manages thousands of concurrent connections to relational databases. Currently, RDS Proxy supports MySQL and Aurora databases. You can begin using RDS Proxy through the Amazon RDS console or the AWS Lambda console. Serverless applications that use fully managed connection pools from RDS Proxy will be billed according to [RDS Proxy Pricing](https://aws.amazon.com/rds/proxy/pricing/).

## Provisioned Concurrency

Provisioned Concurrency gives you greater control over the performance of your serverless applications. When enabled, Provisioned Concurrency keeps functions initialized and hyper-ready to respond in double-digit milliseconds.

The simplest way to benefit from Provisioned Concurrency is by using AWS Auto Scaling. You can use Application Auto Scaling to configure schedules, or have Auto Scaling automatically adjust the level of Provisioned Concurrency in real time as demand changes.

You don’t need to make any changes to your code to use Provisioned Concurrency.

Provisioned Concurrency adds a pricing dimension, of ‘Provisioned Concurrency’, for keeping functions initialized. When enabled, you pay for the amount of concurrency that you configure and for the period of time that you configure it. When your function executes while Provisioned Concurrency is configured on it, you also pay for Requests and execution Duration.

Provisioned Concurrency is ideal for building latency-sensitive applications, such as web or mobile backends, synchronously invoked APIs, and interactive microservices. You can easily configure the appropriate amount of concurrency based on your application's unique demand. You can increase the amount of concurrency during times of high demand and lower it, or turn it off completely, when demand decreases.

If the concurrency of a function reaches the configured level, subsequent invocations of the function have the latency and scale characteristics of regular Lambda functions. You can restrict your function to only scale up to the configured level. Doing so prevents the function from exceeding the configured level of Provisioned Concurrency. This is a mechanism to prevent undesired variability in your application when demand exceeds the anticipated amount.

## Lambda@Edge

[Lambda@Edge](https://aws.amazon.com/lambda/edge/) allows you to run code across AWS locations globally without provisioning or managing servers, responding to end users at the lowest network latency.

To use Lambda@Edge, you just upload your code to AWS Lambda and associate a function version to be triggered in response to Amazon CloudFront requests. Your code must satisfy the Lambda@Edge service limits. Lambda@Edge supports Node.js and Python for global invocation by CloudFront events at this time.

Lambda@Edge is optimized for latency sensitive use cases where your end viewers are distributed globally.

You can associate existing Lambda functions with CloudFront events for global invocation if the function satisfies the Lambda@Edge service requirements and limits.

* Viewer Request - This event occurs when an end user or a device on the Internet makes an HTTP(S) request to CloudFront, and the request arrives at the edge location closest to that user.
* Viewer Response - This event occurs when the CloudFront server at the edge is ready to respond to the end user or the device that made the request.
* Origin Request - This event occurs when the CloudFront edge server does not already have the requested object in its cache, and the viewer request is ready to be sent to your backend origin webserver (e.g. Amazon EC2, or Application Load Balancer, or Amazon S3).
* Origin Response - This event occurs when the CloudFront server at the edge receives a response from your backend origin webserver.

Using [Lambda@Edge](https://aws.amazon.com/lambda/edge/" \t "_blank) and [Amazon CloudFront](https://aws.amazon.com/cloudfront/)allows you to execute logic across multiple AWS locations based on where your end viewers are located.

## Scalability and availability

AWS Lambda is designed to use replication and redundancy to provide high availability for both the service itself and for the Lambda functions it operates. There are no maintenance windows or scheduled downtimes for either.

When you update a Lambda function, there will be a brief window of time, typically less than a minute, when requests could be served by either the old or the new version of your function

AWS Lambda is designed to run many instances of your functions in parallel. However, AWS Lambda has a default safety throttle for number of concurrent executions per account per region (visit [here](http://docs.aws.amazon.com/lambda/latest/dg/concurrent-executions.html#concurrent-execution-safety-limit) for info on default safety throttle limits). You can also control the maximum concurrent executions for individual AWS Lambda functions which you can use to reserve a subset of your account concurrency limit for critical functions, or cap traffic rates to downstream resources.

n exceeding the throttle limit, AWS Lambda functions being invoked synchronously will return a throttling error (429 error code). Lambda functions being invoked asynchronously can absorb reasonable bursts of traffic for approximately 15-30 minutes, after which incoming events will be rejected as throttled. In case the Lambda function is being invoked in response to Amazon S3 events, events rejected by AWS Lambda may be retained and retried by S3 for 24 hours. Events from Amazon Kinesis streams and Amazon DynamoDB streams are retried until the Lambda function succeeds or the data expires. Amazon Kinesis and Amazon DynamoDB Streams retain data for 24 hours.

No, the default limit only applies at an account level.

On failure, Lambda functions being invoked synchronously will respond with an exception. Lambda functions being invoked asynchronously are retried at least 3 times. Events from Amazon Kinesis streams and Amazon DynamoDB streams are retried until the Lambda function succeeds or the data expires. Kinesis and DynamoDB Streams retain data for a minimum of 24 hours.

On exceeding the retry policy for asynchronous invocations, you can configure a “dead letter queue” (DLQ) into which the event will be placed; in the absence of a configured DLQ the event may be rejected. On exceeding the retry policy for stream based invocations, the data would have already expired and therefore rejected.

You can configure an Amazon SQS queue or an Amazon SNS topic as your dead letter queue.

You grant permissions to your Lambda function to access other resources using an IAM role. AWS Lambda assumes the role while executing your Lambda function, so you always retain full, secure control of exactly which AWS resources it can use

When you configure an Amazon S3 bucket to send messages to an AWS Lambda function a resource policy rule will a be created that grants access.

Access controls are managed through the Lambda function’s role. The role you assign to your Lambda function also determines which resource(s) AWS Lambda can poll on its behalf.

Access controls can be managed by the Lambda function’s role or a resource policy setting on the queue itself. If both policies are present, the more restrictive of the two permissions will be applied

You can access resources behind Amazon VPC.

To enable VPC support, you need to specify one or more subnets in a single VPC and a security group as part of your function configuration. To disable VPC support, you need to update the function configuration and specify an empty list for the subnet and security group.

Lambda functions provide access only to a single VPC. If multiple subnets are specified, they must all be in the same VPC. You can connect to other VPCs by peering your VPCs.

Lambda functions configured to access resources in a particular VPC will not have access to the internet as a default configuration. If you need access to external endpoints, you will need to create a [NAT](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-nat-gateway.html) in your VPC to forward this traffic and configure your security group to allow this outbound traffic

## AWS Lambda functions in Java

You can use standard tools like Maven or Gradle to compile your Lambda function. Your build process should mimic the same build process you would use to compile any Java code that depends on the AWS SDK. Run your Java compiler tool on your source files and include the AWS SDK 1.9 or later with transitive dependencies on your classpath.

Lambda provides the Amazon Linux build of openjdk 1.8.

## AWS Lambda functions in Node.js

You can use NPM packages as well as custom packages.

Lambda’s built-in sandbox lets you run batch (“shell”) scripts, other language runtimes, utility routines, and executables.

Any statically linked native module can be included in the ZIP file you upload, as well as dynamically linked modules compiled with an rpath pointing to your Lambda function root directory.

You can use Node.js' child\_process command to execute a binary that you've included in your function or any executable from Amazon Linux that is visible to your function. Alternatively several NPM packages exist that wrap command line binaries such as node-ffmpeg.

To deploy a Lambda function written in Node.js, simply package your Javascript code and dependent libraries as a ZIP

## AWS Lambda functions in Python

You can use pip to install any Python packages needed.

## AWS Lambda functions in C#

You can create a C# Lambda function using the Visual Studio IDE by selecting "Publish to AWS Lambda" in the Solution Explorer. Alternatively, you can directly run the "dotnet lambda publish" command from the dotnet CLI which has the [# Lambda CLI tools patch] installed, which creates a ZIP of your C# source code along with all NuGet dependencies as well as your own published DLL assemblies, and automatically uploads it to AWS Lambda using the runtime parameter “dotnetcore1.0”

## AWS Lambda functions in PowerShell

A PowerShell Lambda deployment package is a ZIP file that contains your PowerShell script, PowerShell modules that are required for your PowerShell script, and the assemblies needed to host PowerShell Core. You then use the *AWSLambdaPSCore* PowerShell module that you can install from the [PowerShell Gallery](https://www.powershellgallery.com/packages/AWSLambdaPSCore/1.1.0.0) to create your PowerShell Lambda deployment package.

## AWS Lambda functions in Go

Upload your Go executable artifact as a ZIP file through the AWS CLI or Lambda console and select the go1.x runtime. With Lambda, you can use Go's native tools to build and package your code.

## AWS Lambda functions in Ruby

To deploy a Lambda function written in Ruby, package your Ruby code and gems as a ZIP. You can upload the ZIP from your local environment, or specify an Amazon S3 location where the ZIP file is located

## Other topics

AWS Lambda offers a single version of the operating system and managed language runtime to all users of the service.

AWS Lambda is integrated with AWS CloudTrail. AWS CloudTrail can record and deliver log files to your Amazon S3 bucket describing the API usage of your account

You can use Amazon Step Functions to coordinate multiple invoking Lambda functions. You can invoke multiple Lambda functions serially, passing the output of one to the other, or in parallel.