
AWS Lambda

Developer Guide



AWS Lambda: Developer Guide

Copyright © 2015 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

The following are trademarks of Amazon Web Services, Inc.: Amazon, Amazon Web Services Design, AWS, Amazon CloudFront, AWS CloudTrail, AWS CodeDeploy, Amazon Cognito, Amazon DevPay, DynamoDB, ElastiCache, Amazon EC2, Amazon Elastic Compute Cloud, Amazon Glacier, Amazon Kinesis, Kindle, Kindle Fire, AWS Marketplace Design, Mechanical Turk, Amazon Redshift, Amazon Route 53, Amazon S3, Amazon VPC, and Amazon WorkDocs. In addition, Amazon.com graphics, logos, page headers, button icons, scripts, and service names are trademarks, or trade dress of Amazon in the U.S. and/or other countries. Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon.

All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is AWS Lambda?	1
When should I use it?	1
Are you a first-time user of AWS Lambda?	2
How it Works	3
Lambda Function and Event Source	3
Lambda Function	3
Event Source	4
Related Topics	4
Lambda Function Invocation Options	4
The Pull Model	5
The Push Model	5
Related Topics	7
Permission Model	7
Execution Permissions	7
Invocation Permissions	8
Related Topics	9
Programming Model	9
The <code>context</code> Object: Methods and Properties	10
Related Topics	11
Resource Model	12
Next Step	12
Supported Versions	12
Next Step	12
Setup an AWS Account	13
Sign up for AWS	13
Create an IAM User	13
Next Step	15
Getting Started	16
Preparing for the Getting Started	16
Getting Started 1: Invoking Lambda Functions from User Applications	17
Step 1: Create a Lambda Function	17
Step 2: Invoke the Lambda Function Manually	19
Next Step	20
Getting Started 2: Handling Amazon S3 Events	20
Next Step	20
Step 1: Create a Lambda Function and Invoke it Manually	20
Step 2: Configure Amazon S3 as the Event Source	23
Getting Started 3: Handling Amazon Kinesis Events	25
Next Step	25
Step 1: Create a Lambda Function and Invoke It Manually (Using the Console)	25
Step 2: Configure an Amazon Kinesis Stream as the Event Source	27
Walkthroughs	30
Walkthrough 1: Handling User Application Events	30
Next Step	31
Step 1: Prepare for the Walkthrough	31
Step 2: Create a Lambda Function	32
Step 3: Invoke the Lambda Function	34
Step 4: Try More CLI Commands	36
Step 5: Delete the Lambda Function and IAM Role	38
Walkthrough 2: Handling Amazon S3 Events	38
Scenario	38
Implementation Summary	38
Next Step	40
Step 1: Preparing for the Walkthrough	40
Step 2: Create and Test the Function	41

Step 3: Configure Amazon S3 to Publish Events	47
Walkthrough 3: Handling Amazon DynamoDB Stream Events	48
Scenario	48
Implementation Summary	48
Next Step	49
Step 1: Preparing for the Walkthrough	49
Step 2: Create a Lambda Function and Invoke it Manually Using Sample Event Data	50
Step 3: Create AWS Lambda Event Source Mapping and Test the Setup	54
Walkthrough 4: Handling Amazon Kinesis Stream Events	59
Next Step	60
Step 1: Preparing for the Walkthrough	60
Step 2: Create a Lambda Function and Invoke it Manually	60
Step 3: Add an AWS Lambda Event Source and Test	64
Walkthrough 5: Handling AWS CloudTrail Events	65
Scenario	65
Implementation Summary	66
First Walkthrough Step	68
Step 1: Preparing for the Walkthrough	68
Step 2: Create and Invoke a Lambda Function	69
Step 3: Configure Amazon S3 to Publish Events	76
Troubleshooting and Monitoring	79
Troubleshooting Scenarios	80
Troubleshooting Function not working	80
Troubleshooting	80
Accessing CloudWatch Metrics	81
Accessing CloudWatch Logs	82
Metrics	83
Metrics	84
Dimensions	84
API Logging with AWS CloudTrail	86
AWS Lambda Information in CloudTrail	86
Understanding AWS Lambda Log File Entries	87
Best Practices	89
Limits	90
Increasing Limits	90
Appendix: API Updates	92
API Reference	95
Actions	95
AddPermission	96
CreateEventSourceMapping	99
CreateFunction	103
DeleteEventSourceMapping	108
DeleteFunction	110
GetEventSourceMapping	112
GetFunction	114
GetFunctionConfiguration	116
GetPolicy	119
Invoke	121
InvokeAsync	124
ListEventSourceMappings	126
ListFunctions	128
RemovePermission	130
UpdateEventSourceMapping	132
UpdateFunctionCode	135
UpdateFunctionConfiguration	138
Data Types	141
EventSourceMappingConfiguration	142
FunctionCode	143

FunctionCodeLocation 143

FunctionConfiguration 144

Document History 146

AWS Glossary 147

What Is AWS Lambda?

AWS Lambda is a compute service that makes it easy for you to build applications that respond quickly to new information. AWS Lambda runs your code in response to events such as image uploads, in-app activity, website clicks, or outputs from connected devices. You can use AWS Lambda to extend other AWS services with custom logic, or create your own back-end that operates at AWS scale, performance, and security. With AWS Lambda, you can easily create discrete, event-driven applications that execute only when needed and scale automatically from a few requests per day to thousands per second.

AWS Lambda runs your code on a high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code and security patch deployment, and code monitoring and logging. All you need to do is supply your Node.js code.

When should I use AWS Lambda?

Amazon Web Services lets you choose from a range of compute services to meet your needs.

- The Amazon Elastic Compute Cloud (Amazon EC2) web service offers flexibility and a wide range of Amazon EC2 instance types to choose from. It gives you the option to customize operating systems, network and security settings, and the entire software stack, but you are responsible for provisioning capacity, monitoring fleet health and performance, and using Availability Zones for fault tolerance.
- Elastic Beanstalk offers an easy-to-use service for deploying and scaling applications onto EC2 in which you retain ownership and full control over the underlying Amazon EC2 instances.

The higher-level abstraction that AWS Lambda offers is the convenience of your being responsible only for your code. AWS Lambda manages the compute fleet that offers a balance of memory, CPU, and other network resources. This is in exchange for flexibility, which means you cannot log in to compute instances, customize the operating system or language runtime, or store data on local drives after your function completes execution. But this allows AWS Lambda to perform operational and administrative activities on your behalf, including capacity provisioning, monitoring fleet health, applying security patches, deploying your code, running a web service front end, and monitoring and logging your functions.

AWS Lambda is a great alternative to using these other AWS compute services if you can write your application code in languages supported by AWS Lambda, and run within the standard runtime environment and resources provided by the service.

Are you a first-time user of AWS Lambda?

If you are a first-time user of AWS Lambda, we recommend you read the following sections in order:

1. For a product overview and pricing information, go to the AWS Lambda [product detail page](#).
2. Read [AWS Lambda: How it Works \(p. 3\)](#).
3. Follow the console-based exercises in the getting started section (see [Getting Started with AWS Lambda \(p. 16\)](#)).
4. To explore the AWS Lambda API, follow the Lambda CLI-based walkthroughs:
 - [AWS Lambda Walkthrough 1: Invoking Lambda Functions from User Applications \(Using the AWS CLI\) \(p. 30\)](#)
 - [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\) \(p. 38\)](#)
 - [AWS Lambda Walkthrough 3: Processing Events from an Amazon DynamoDB Stream \(Using the AWS CLI\) \(p. 48\)](#)

Refer to the following topics to learn more about the service:

- [Troubleshooting and Monitoring AWS Lambda Functions with Amazon CloudWatch \(p. 79\)](#)
- [Best Practices for Working with AWS Lambda Functions \(p. 89\)](#)
- [AWS Lambda Limits \(p. 90\)](#)

AWS Lambda: How it Works

AWS Lambda is a compute service that makes it easy for you to build applications that respond quickly to new information and user activity. AWS Lambda runs your custom code on a high-availability compute infrastructure and administers all of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code, and security patches.

AWS Lambda can execute your custom code, also referred to as a "Lambda function," in response to events in other AWS services or events generated by user applications.

The following sections provides overview of various AWS Lambda service components and how they interact.

Note

After you read the introduction, try the getting started exercises and the walkthroughs, which provide step-by-step instructions for you to explore AWS Lambda.

Topics

- [Core Components: Lambda Function and Event Source \(p. 3\)](#)
- [Lambda Function Invocation Options \(p. 4\)](#)
- [Permission Model \(p. 7\)](#)
- [Programming Model \(p. 9\)](#)
- [Resource Model \(p. 12\)](#)
- [Supported Versions \(p. 12\)](#)

Core Components: Lambda Function and Event Source

A Lambda function and an event source are the core components. Event sources publish events, and a Lambda function is the custom code that you write to process the events. AWS Lambda executes your Lambda function on your behalf in response to events that event sources publish.

Lambda Function

After you upload your custom code to AWS Lambda, we refer to it as a Lambda function. Currently, you can author your Lambda function code in Node.js.

To create a Lambda function you first create a Lambda function deployment package, a zip file consisting of your code and any dependencies. At the time you create a Lambda function you provide the zip file and other configuration information such as name, description, and run-time requirements like memory allocation.

Depending on the resources your custom code uses, you have the following options when creating a Lambda function:

- Simple scenario—If your custom code requires only the AWS SDK library, then you can use the inline editor in the AWS Lambda console. Using the console, you can edit and upload your code to AWS Lambda. The console will zip up your code with the relevant configuration information into a deployment package that the Lambda service can run.

You can also test your code in the console by manually invoking it using sample event data.

Note

The Lambda service has preinstalled the AWS SDK for Node.js.

- Advanced scenario—If you are writing code that uses other resources, such as a graphics library for image processing, or you want to use the AWS CLI instead of the console, you need to first create the Lambda function deployment package, and then use the console or the CLI to upload the package.

Event Source

Event sources publish events that cause the Lambda function to be invoked (either by the event source or by AWS Lambda, discussed in the next section). Event sources can be AWS services. For example, Amazon S3 can publish object created events, Amazon DynamoDB can publish table updates to a stream associated with the table, and AWS CloudTrail can record all API calls made in your account.

Currently, AWS Lambda supports events from Amazon S3, Amazon DynamoDB, Amazon Kinesis, Amazon SNS, and Amazon Cognito. You can write Lambda function to process events published by these AWS services. You can also use Lambda functions with other AWS services that publish data to one of these listed event sources. For example, if you turn on CloudTrail, it records all API access events to an S3 bucket; Amazon CloudWatch can also publish events to an Amazon SNS topic; and AWS Lambda supports processing events from both Amazon S3 and Amazon SNS.

In addition to AWS services generating events, user applications can also generate events. User applications such as client, mobile, or web applications can publish events and invoke Lambda functions using the AWS SDKs or AWS Mobile SDK.

Related Topics

[AWS Lambda: How it Works \(p. 3\)](#)

Lambda Function Invocation Options

How a Lambda function is invoked depends on the event source it is used with. There are two models:

- The “pull event” model
- The “push event” model

In this model, AWS Lambda polls the event source and invokes your Lambda function when it detects an event. For example, AWS Lambda polls your Amazon Kinesis stream, or Amazon DynamoDB stream, and invokes your Lambda function when it detects new records on the stream.

In this model, AWS Lambda manages the event source. That is, it provides an API (see [CreateEventSourceMapping](#) (p. 99)) for you to create event source mappings that associate your Lambda function with a specific event source. You will need to grant AWS Lambda necessary permissions to access the event source via an IAM role. The permission model is discussed in the next section.

AWS Account

The diagram illustrates the architecture within an AWS account:

- Custom App**: Represented by a blue icon with a white atomic symbol.
- Amazon Kinesis**: Represented by an orange icon of stacked blocks.
- Stream**: Represented by five vertical bars of increasing height.
- Execution Role**: Represented by a dashed orange outline of a person holding a document.
- AWS Lambda**: Represented by an orange icon of a server rack.
- Lambda Function**: Represented by a code snippet showing constants and file paths.

The flow is as follows:

- The **Custom App** sends data to the **Amazon Kinesis Stream**.
- The **Amazon Kinesis Stream** triggers the **Execution Role**, which then invokes the **AWS Lambda Function**.

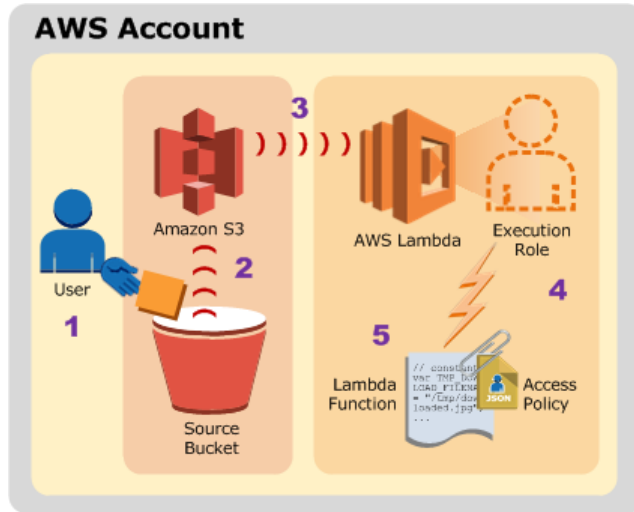
```
// constants  
var TMP_DIR =  
    IAM_PATH + "/tmp/" +  
    "lambda-role-  
    loaded.jpg";  
...
```

In this model, the event source directly invokes a Lambda function when it detects an event. When invoking the function, the event source passes the event as a parameter.

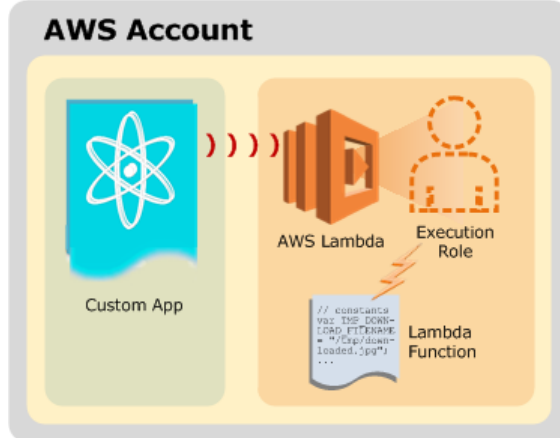
When the event source is an AWS service such as Amazon S3, it will invoke the function asynchronously by specifying the "event" invocation type in the `invoke` API call. This applies when the event source is Amazon S3, Amazon SNS, or Amazon Cognito. When the event source is a user application, you have an option to invoke the Lambda function either by using the "Event" invocation type or the "RequestResponse" invocation type. When invocation type is "RequestResponse", AWS Lambda executes the function and returns results immediately to the calling application. For more information, see [Invoke \(p. 121\)](#).

AWS Lambda does not explicitly manage event sources in this model. That is, you do not use the AWS Lambda APIs to map your Lambda function to its event source. Instead, you use APIs from the event source to configure this mapping. For example, Amazon S3 supports bucket notification configuration. You can use the relevant Amazon S3 APIs to add notification configuration on a bucket (as an event source) to publish events to AWS Lambda identifying a specific Lambda function to invoke.

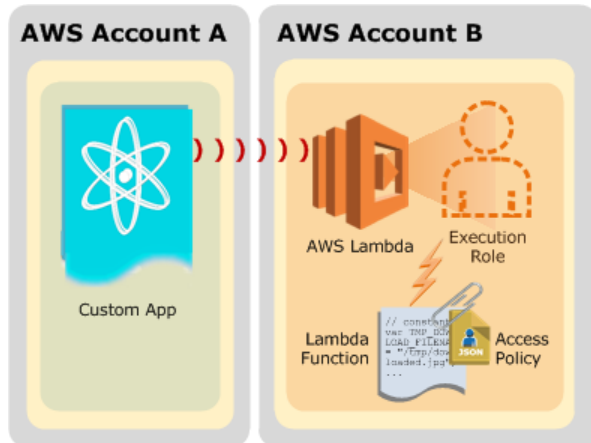
The following diagram shows Amazon S3 invoking a Lambda function upon detecting an object-created event in a bucket. AWS Lambda executes the function. The Lambda function owner must grant Amazon S3 permission to invoke the function by adding a permission in the access policy associated with the Lambda function. Access policies are discussed in the next section.



The following diagram shows a custom application in your account invoking your Lambda function.



In the following diagram, the user application and Lambda function are owned by different AWS accounts. In this case, the Lambda function owner must add a permission in the access policy associated with the Lambda function to grant cross-account access.



Related Topics

[AWS Lambda: How it Works \(p. 3\)](#)

Permission Model

Here we discuss two things:

- Execution permissions – These refer to the permissions your Lambda function needs to access other AWS resources in your account. You grant these permission by creating an IAM role, known as an execution role.
- Invocation permissions – These refer to the permissions the event source needs to communicate with your Lambda function. Depending on the invocation model, you can grant these permissions using either the execution role or resource policies (the access policy associated with your Lambda function).

Execution Permissions

When your Lambda function executes, it can access other AWS resources in your account. For example:

- Read objects from an S3 bucket and write objects to an S3 bucket
- Write logs to Amazon CloudWatch Logs.
- Write items to an Amazon DynamoDB table.

You must grant necessary permissions for the Lambda function to access these resources.

You grant these permissions via an IAM role, referred to as an execution role. You create the IAM role granting AWS Lambda permission to assume the role and attach an access policy to the role granting all the needed permissions.

When you create a Lambda function, you provide the execution role so AWS Lambda can assume the role to execute your Lambda function on your behalf.

Each IAM role has two policies attached.

- Access policy – This policy grants your Lambda function the resource permissions it needs.

For example, if your Lambda function writes logs for your function to Amazon CloudWatch Logs, you grant permissions for the CloudWatch Logs action. The following example policy allows permission for all log actions (`logs:*`) on CloudWatch Logs.

```
{
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

If your function uploads an object to an S3 bucket, you grant permission for relevant Amazon S3 actions in this policy. For a list of actions, go to [Specifying Permissions in a Policy](#) in *Amazon Simple Storage Service Developer Guide*.

- Trust policy – A trust policy identifies who can assume the role. Because you want AWS Lambda to assume the role, you specify the AWS Lambda service principal as shown:

Important

The user who is creating the IAM role is passing permission to AWS Lambda to assume this role. This requires the user to have permission for the `iam:PassRole` action to be able to grant this permission. If an administrator user is creating this role, the user has full permissions including the `iam:PassRole`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For more information about IAM roles, go to [Roles \(Delegation and Federation\)](#) in *Using IAM*.

Invocation Permissions

In the pull model, AWS Lambda polls the event sources (Amazon Kinesis or DyanamoDB streams) for new events. When new records are found on the stream, AWS Lambda reads those records and passes them to your Lambda function. You will need to grant AWS Lambda permissions for the necessary Amazon Kinesis and DynamoDB actions. You add the required permissions to the execution role associated with the function.

In the push model, the event source (such as Amazon S3 or a user application) directly invokes a Lambda function. Both Amazon S3 and the user application (assuming it is owned by another AWS account) will need permission from the Lambda function owner.

You grant these permissions by using a resource access policy attached to your Lambda function. AWS Lambda provides the `AddPermission` API for this purpose. For more information, see [AddPermission \(p. 96\)](#).

Note

If the user application and the Lambda function it invokes belong to the same AWS account, you don't need to grant explicit permissions.

The following are two example policies:

- Grant Amazon S3 permission to invoke your Lambda function – The following example policy grants Amazon S3 `Principal` permission for the `lambda:InvokeFunction` action provided the event source is `examplebucket` bucket and the bucket is owned by a specific AWS account.

```
{
  "Statement": {
    "StatementId": "Id-1",
    "Action": "lambda:InvokeFunction",
    "Principal": "s3.amazonaws.com",
    "SourceArn": "arn:aws:s3:::examplebucket",
    "SourceAccount": "account-id",
  }
}
```

Grant cross-account permission to a user application created by some other AWS account – To grant cross-account permission to another AWS account, you specify `Principal` as shown:

```
"Principal": "account-id"
```

For more information, go to [Principal](#) in the *Using IAM*.

Related Topics

[AWS Lambda: How it Works \(p. 3\)](#)

Programming Model

Your Lambda function code must be written in a stateless style, and have no affinity with the underlying compute infrastructure. Your code should expect local file system access, child processes, and similar artifacts to be limited to the lifetime of the request, and store any persistent state in Amazon S3, Amazon DynamoDB, or another cloud storage service. Requiring functions to be stateless enables AWS Lambda to launch as many copies of a function as needed to scale to the incoming rate of events and requests. These functions may not always run on the same compute instance from request to request, and a given instance of your Lambda function may be used more than once by AWS Lambda.

The following example skeleton code shows the format in which you write your custom Node.js code:

```
exports.handler_name = function(event, context) {
  console.log("value1 = " + event.key1);
  console.log("value2 = " + event.key2);
  ...
  context.succeed("some message");
}
```

In the code:

- `handler_name` – The handler parameter name you provided when you create a Lambda function (see `CreateFunction`). This tells Lambda which Node.js function to call.
- The `event` parameter – Lambda passes event data to the function via this parameter. The structure of event data depends on the event source. For example, if Amazon S3 is the event source, the event data will provide, among other things, bucket name and object key. For more information about Amazon S3's event data structure, go to [Event Message Structure](#) in the *Amazon Simple Storage Service Developer Guide*.
- The `context` parameter – See the following section.

The `context` Object: Methods and Properties

You interact with AWS Lambda execution environment via the `context` parameter. The `context` object allows you to specify when the function and any callbacks have completed execution. It also allows you to access useful information available within the Lambda execution environment. For example, you can use the `context` parameter to determine the CloudWatch log stream associated with the function, or use the `clientContext` property of the `context` object to learn more about the application calling the Lambda function (when invoked through the AWS Mobile SDK).

You can use the `Context` object methods as follows:

`context.succeed()` method

Indicates the Lambda function execution and all callbacks completed successfully. Here's the general syntax:

```
context.succeed (Object result);
```

where

`Object result` – provides the result of the function execution. The `result` provided must be `JSON.stringify` compatible. This parameter is optional. You can call this method without any parameters (`succeed()`) or pass a null value (`succeed(null)`). If AWS Lambda fails to stringify or encounters another error, an "unhandled" error is thrown, with the `X-Amz-Function-Error` response header set to "Unhandled".

The method behavior depends on the invocation type specified when the Lambda function is invoked (see [Invoke](#) (p. 121)).

- If the Lambda function is invoked using the `Event` invocation type, this `succeed` method returns HTTP status 200 (OK) and automatically logs the string representation of non-null values of the result to the Amazon CloudWatch Logs stream associated with the Lambda function.
- If the Lambda function is invoked using the `RequestResponse` invocation type, this `succeed` method will return HTTP status 202 and set the response body to the string representation of result, and also write to logs similar to when invoked with the `Event` invocation type.

`context.fail()` method

Indicates the Lambda function execution and all callbacks completed unsuccessfully, resulting in a handled exception. Here's the general syntax:

```
context.fail (Error error);
```

where

`Error error` – provides the result of the Lambda function execution.

The parameter is optional. You can call this method without the parameter or pass null as a parameter value. Non-null error values will populate the response body.

The `fail` method will set the response body to the string representation of error and also write to logs.

If AWS Lambda fails to stringify or encounters another error, an "unhandled" error, with the `X-Amz-Function-Error` header set to `Unhandled`.

`context.done()` method

Causes the Lambda function execution to terminate. This method complements the `context.succeed()` and `context.fail()` methods by allowing the use of the "error first" callback design pattern. It provides no additional functionality.

Here's the general syntax:

```
context.done (Error error, Object result);
```

where

`Error error` – provides an error representing the results of the failed Lambda function execution.

`Object result` – provides the result of a successful function execution. The result provided must be `JSON.stringify` compatible. If an error is provided, this parameter is expected to be null.

Both the parameters are optional. You can call this method without any parameters or pass null as a parameter value.

AWS Lambda treats any non-null value for the `error` parameter as a managed exception.

The function behavior depends on the invocation type specified when the Lambda function is invoked (see [Invoke](#) (p. 121)).

- If the Lambda function was invoked using the `Event` or the `RequestResponse` invocation type, this `done` method automatically logs the string representation of non-null values of `error` and `result` to the Amazon CloudWatch Logs stream associated with the Lambda function.
- If the Lambda function was invoked using the `RequestResponse` invocation type, the `done` method will do the following:
 - If the error is null, set the response body to the string representation of `result`, similar to the `context.fail()`.
 - If the error is not null, set the response body to `error`.
 - If the function is called with a single argument of type error, the error value will be populated in the response body.

`context` Object Properties

The `context` object properties are:

- `awsRequestId` – The request ID for the Lambda function invocation request currently being executed.
- `logStreamName` – The CloudWatch log stream name associated with the Lambda function invoked.
- `clientContext` – Information about the client application and device when invoked through the AWS Mobile SDK. It can be null.
- `identity` – Information about the Amazon Cognito identity provider when invoked through the AWS Mobile SDK. It can be null.

Related Topics

[AWS Lambda: How it Works](#) (p. 3)

Resource Model

In the AWS Lambda resource model, you choose the amount of memory you want allocated for your function. AWS Lambda then allocates CPU power proportional to the memory by using the same ratio as a general purpose Amazon EC2 instance type, such as an M3 type. For example, if you allocate 256 MB to your Lambda function, it will receive twice the CPU share than if you had allocated 128 MB. You can request additional memory in 64 MB increments up to 1024 MB. For a full list of AWS Lambda limits, see [AWS Lambda Limits \(p. 90\)](#).

Next Step

[Setup an AWS Account and Create an Administrator User \(p. 13\)](#)

Supported Versions

The AWS Lambda runtime supports the following versions:

Item	Version
Public Amazon Linux AMI version	AMI Id: ami-dfc39aef in the US West (Oregon) region. For information about using an AMI, see Amazon Machine Images (AMI) in the <i>Amazon EC2 User Guide for Linux Instances</i> .
Linux kernel version	3.14.35-28.38.amzn1.x86_64
Node.js	v0.10.33
ImageMagick	Installed with default settings. For versioning information, go to imagemagick nodejs wrapper and ImageMagick native binary (search for "ImageMagick").
AWS SDK version	aws sdk version 2.1.22

Next Step

[Setup an AWS Account and Create an Administrator User \(p. 13\)](#)

Setup an AWS Account and Create an Administrator User

Before you use AWS Lambda for the first time, complete the following tasks:

1. [Sign up for AWS](#) (p. 13)
2. [Create an IAM User](#) (p. 13)

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including AWS Lambda. You are charged only for the services that you use.

With AWS Lambda, you pay only for the resources you use. For more information about Amazon Lambda usage rates, see the [AWS Lambda product page](#). If you are a new AWS customer, you can get started with AWS Lambda for free; for more information, see [AWS Free Usage Tier](#).

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

1. Open <http://aws.amazon.com/>, and then click **Sign Up**.
2. Follow the on-screen instructions.

Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Note your AWS account ID, because you'll need it for the next task.

Create an IAM User

Services in AWS, such as AWS Lambda, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The console requires

your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or and grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console.

The getting started and walkthrough exercises in this guide assume you have a user ("adminuser") with administrator privileges. So when you follow the procedure, create a user with name "adminuser".

To create the Administrators group

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, click **Groups**, then click **Create New Group**.
3. In the **Group Name** box, type **Administrators** and then click **Next Step**.
4. In the list of policies, select the check box next to the **AdministratorAccess** policy. You can use the **Filter** menu and the **Search** box to filter the list of policies.
5. Click **Next Step**, then click **Create Group**.

Your new group is listed under **Group Name**.

To create an IAM user for yourself, add the user to the Administrators group, and create a password for the user

1. In the navigation pane, click **Users** and then click **Create New Users**.
2. In box **1**, enter a user name. Clear the check box next to **Generate an access key for each user**, then click **Create**.
3. In the list of users, click the name (not the check box) of the user you just created. You can use the **Search** box to search for the user name.
4. In the **Groups** section, click **Add User to Groups**.
5. Select the check box next to the **Administrators** group, then click **Add to Groups**.
6. Scroll down to the **Security Credentials** section. Under **Sign-In Credentials**, click **Manage Password**.
7. Select **Assign a custom password**, then enter a password in the **Password** and **Confirm Password** boxes. When you are finished, click **Apply**.

To sign in as this new IAM user, do the following:

1. Sign out of the AWS Management Console.
2. Use the following URL

`https://aws_account_number.signin.aws.amazon.com/console/`

The *aws_account_number* is your AWS account ID without hyphen. For example, if your AWS account ID is 1234-5678-9012, your AWS account number is 123456789012.

For information about finding your account number, go to [Your AWS Account ID and Its Alias](#) in the Using IAM.

3. Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "*your_user_name* @ *your_aws_account_id*".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias.

To create or remove an account alias

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the navigation pane, select **Dashboard**.
3. Find the IAM users sign-in link.
4. To create the alias, click **Customize**, enter the name you want to use for your alias, and then click **Yes, Create**.
5. To remove the alias, click **Customize**, and then click **Yes, Delete**. The sign-in URL reverts to using your AWS account ID.

To sign in after you create an account alias, use the following URL:

```
https://your_account_alias.signin.aws.amazon.com/console/
```

To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link**: on the dashboard.

For more information about IAM, see the following:

- [Identity and Access Management \(IAM\)](#)
- [IAM Getting Started Guide](#)
- [Using IAM](#)

Next Step

[Getting Started with AWS Lambda \(p. 16\)](#)

Getting Started with AWS Lambda

The getting started section consists of the following three exercises that use the AWS Lambda console.

[Getting Started 1: Invoking Lambda Functions from User Applications \(Using the AWS Lambda Console\)](#) (p. 17)

[Getting Started 2: Handling Amazon S3 Events \(Using the AWS Lambda Console\)](#) (p. 20)

[Getting Started 3: Handling Amazon Kinesis Events \(Using the AWS Lambda Console\)](#) (p. 25)

Before you explore the getting started exercises, we recommend you read introductory information including the core components of AWS Lambda, the programming model, and the permissions model. For more information, see [AWS Lambda: How it Works](#) (p. 3)

Note that the console does many things for you as you create and configure Lambda functions. To help you understand the AWS Lambda API, the documentation also provides AWS CLI–based equivalent walkthroughs. For more information, see [AWS Lambda Walkthroughs](#) (p. 30).

Preparing for the Getting Started

First, you need to sign up for an AWS account and create an administrator user in your account. For instructions, see [Setup an AWS Account and Create an Administrator User](#) (p. 13).

Important

AWS Identity and Access Management (IAM) recommends not using the root credentials of your AWS account to make requests. Instead, create an IAM user (called "adminuser"), grant that user full access, and then use that user's credentials to interact with AWS. We refer to this user as an administrator user. For more information, go to [Root Account Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in *Using IAM*.

Getting Started 1: Invoking Lambda Functions from User Applications (Using the AWS Lambda Console)

Topics

- [Step 1: Create a Lambda Function \(p. 17\)](#)
- [Step 2: Invoke the Lambda Function Manually \(p. 19\)](#)
- [Next Step \(p. 20\)](#)

One of the use cases for using AWS Lambda is to process events generated by a user application. For demonstration purposes, you don't need to write a user application that will invoke your Lambda function. Instead, in this getting started exercise, you will use sample event data and invoke your Lambda function manually.

When a user application invokes a Lambda function as shown in this exercise, it's an example of the AWS Lambda "request-response" model in which an application invokes a Lambda function and receives a response in real time. For more information, see [AWS Lambda: How it Works \(p. 3\)](#).

In this exercise, you will do the following:

- Create a Lambda function by using the following JavaScript code (the console provide this as template code) to process input data it receives as a parameter. In the code, the "console.log()" statements generate log events in CloudWatch.

```
console.log('Loading function');

exports.handler = function(event, context) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    context.succeed(event.key1); // Echo back the first key value
    // context.fail('Something went wrong');
};
```

- Invoke the function manually using sample event data as shown.

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

The function is simple. It processes incoming event data and redirects the `console.log` and `console.error` output to Amazon CloudWatch. AWS Lambda executes the function each time it is invoked.

Step 1: Create a Lambda Function

Follow the steps to create a Lambda function.

1. Sign in to the AWS Management Console and open the AWS Lambda console.
2. Click **Get Started Now**.



Note

The console shows the **Get Started Now** page only if you do not have any Lambda functions created. If you have created functions already, you will see the **Lambda: Function List** page. On the list page, choose **Create a Lambda function** to go to the **Lambda: New function** page.

3. On the **Lambda: New function** page, do the following:
 - a. For **Lambda Function Name**, enter the function name ("HelloWorld") in the **Name** box and an optional description.
 - b. In the **Lambda Function Code** section, you do two things:
 - Provide code for the Lambda function you are creating.
 - Specify the IAM role (execution role, see [Execution Permissions \(p. 7\)](#)) that AWS Lambda can assume when it executes the function on your behalf. The example Lambda function only writes logs to Amazon CloudWatch. So you will use the **Basic execution role** template that grants permissions for CloudWatch actions.
 - i. Choose **Edit code inline** (in this console-based exercise, you will use console-provided example code to create a Lambda function).
 - ii. From the **Code Template** drop-down, select the **Hello World** code template.

Note

The name in the **Handler name** box matches the `exports.handler` in the code (see [Programming Model \(p. 9\)](#)).

- iii. From the **Role** drop-down, select "Basic Execution Role" from the **Create New Role** section (you might need to enable pop-ups to see the role selector drop-down).

This action simply indicates that you are going to associate the following predefined access policy for the role you are about to create.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:*"
      ]
    }
  ]
}
```

AWS Lambda Developer Guide

Step 2: Invoke the Lambda Function Manually

```
    "Resource": "arn:aws:logs:*:*:*"
  }
]
}
```

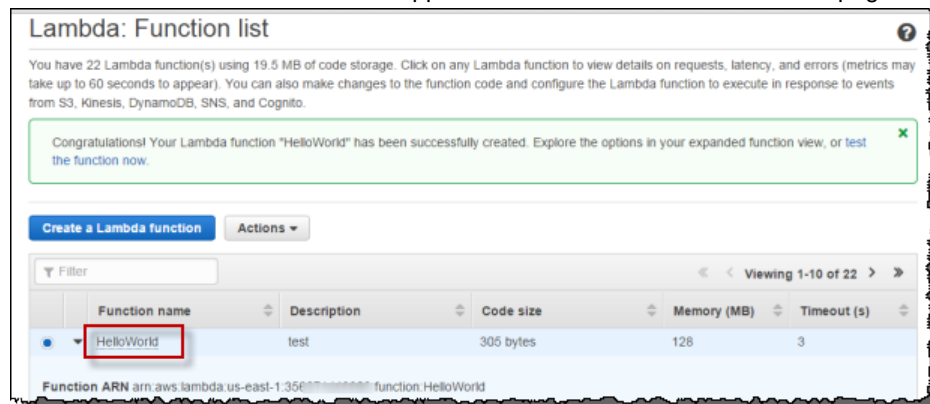
On the **AWS Lambda requires access to your resource** page, do the following:

- Select **Create a new IAM Role** from the **IAM Role** drop-down.
- For **Role Name**, enter a role name. For example, "lambda_basic_execution", the execution role you are creating for the first getting started exercise.
- Choose **View Policy Document** to review the access policy.
- Choose **Allow**.

The browser tab will close, and the new role name appears on the **Lambda: New Function** page.

- In the **Advanced settings** section, leave the default values.
- Choose **Create Lambda Function** to create a Lambda function.

The console saves the code into a file and then zips the file, which is the deployment package. The console then uploads the deployment packages to AWS Lambda creating your Lambda function. The Lambda function then appears on the **Lambda: Function List** page as shown:



Step 2: Invoke the Lambda Function Manually

Follow the steps to invoke your Lambda function using console-provided sample event data.

- In the **Lambda: Function List** page, click the radio button or the triangle to the left of the function name to view expanded function details.
- From the **Actions** drop-down, select **Edit/Test** to invoke the function.

The console opens the **Lambda: Edit/Test HelloWorld** page.

- Choose **Hello World** from the **Sample event** drop-down so you have appropriate sample event data to send to the function. Now you see both the function and the sample event it will process.
- Click **Invoke**.

AWS Lambda assumes the execution role and uses it to execute the code.

5. View results in the console. Note the following:
 - The **Execution result** section shows the object passed to the `context.succeed()` method in your code.
 - The **Summary** section shows the REPORT line in the execution log.
 - The **Execution log** section shows the log AWS Lambda generates for each execution.

Note the **Click here** link shows logs in the CloudWatch console. The function then adds logs to Amazon CloudWatch, to the log group that corresponds to the Lambda function.

Next Step

[Getting Started 2: Handling Amazon S3 Events \(Using the AWS Lambda Console\) \(p. 20\)](#)

Getting Started 2: Handling Amazon S3 Events (Using the AWS Lambda Console)

In this getting started exercise, you create a Lambda function to consume events published by Amazon S3. For each object uploaded to a bucket, Amazon S3 will invoke your Lambda function by passing event information as function parameters. AWS Lambda then executes the function. As the function executes, it reads the incoming S3 event data and, for illustration, logs some of the event information to Amazon CloudWatch. This is an example of the "push" model where Amazon S3 invokes the Lambda function. For more information on the "push" model, see [AWS Lambda: How it Works \(p. 3\)](#). In this exercise, you will do the following:

- Create a Lambda function to process Amazon S3 events and test it by invoking it manually by using sample Amazon S3 event data. The function reads incoming event data and writes logs to Amazon CloudWatch.
- Configure an S3 bucket notification so that Amazon S3 can publish object-created events to AWS Lambda by invoking your Lambda function.

You can then upload objects to the bucket. Each object upload will cause Amazon S3 to invoke your Lambda function. You can verify AWS Lambda executed your function by reviewing the logs.

Important

Both the Lambda function and the S3 bucket must be in the same AWS region. This exercise assumes the `us-west-2` (Oregon) region.

Next Step

[Step 1: Create a Lambda Function and Invoke it Manually \(Using the Console\) \(p. 20\)](#)

Step 1: Create a Lambda Function and Invoke it Manually (Using the Console)

In this step, you create and manually invoke a Lambda function:

- You will use the following JavaScript code that console provides to create a Lambda function. The code simply reads the Amazon S3 event data it receives as parameter and logs some of the information to Amazon CloudWatch Logs. The "console.log()" statements in the code generate log events in CloudWatch.

```
console.log('Loading function');
var aws = require('aws-sdk');
var s3 = new aws.S3({apiVersion: '2006-03-01'});

exports.handler = function(event, context) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    // Get the object from the event and show its content type
    var bucket = event.Records[0].s3.bucket.name;
    var key = event.Records[0].s3.object.key;
    s3.getObject({Bucket: bucket, Key: key}, function(err, data) {
        if (err) {
            console.log("Error getting object " + key + " from bucket " +
bucket +
            ". Make sure they exist and your bucket is in the same region
as this function.");
            context.fail ("Error getting file: " + err)
        } else {
            console.log('CONTENT TYPE:', data.ContentType);
            context.succeed();
        }
    });
};
```

Note that the AWS Lambda runtime already has the AWS SDK for JavaScript in Node.js. So you only need to provide the preceding code for the console to create the deployment package.

- As part of creating a Lambda function, the console first creates an IAM role (execution role) and provides that role at the time it sends the request to create a Lambda function. AWS Lambda can assume this role to execute the function on your behalf. For more information, see [Component: Execution Role](#). You will use predefined "S3 execution role" template in the console that attaches the following access policy to the role you will create. This policy grants permissions for specific Amazon S3 actions and CloudWatch actions. These are the permissions your Lambda function will need when it executes.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:*"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:*:*:*"
      ]
    }
  ]
}
```

```
    ]
  }
}
```

- You will use sample Amazon S3 event data that the console provides to first invoke the Lambda function manually. The sample event shown is only a fragment. It shows an example bucket name where the event occurred, and the example key name of the object created.

```
{
  "Records": [
    {
      ...
    },
    {
      "s3": {
        ...
        "bucket": {
          "name": "sourcebucket",
          ...
          "arn": "arn:aws:s3:::sourcebucket"
        },
        "object": {
          "key": "object keyname",
          "size": 1024,
          ...
        }
      }
    }
  ]
}
```

- You will then configure notification on an S3 bucket to publish object-created events to AWS Lambda and invoke your Lambda function.

Step 1.1: Create a Lambda Function

Follow the procedure in getting started 1 to create a Lambda function with the following differences:

- Select the "S3 Get Object" code template.
- Create a new IAM **Role** and select "S3 execution role" from the drop-down. This causes the console to use the preceding access policy when creating the new role. Create a new IAM role and assign a name, for example, "lambda_s3_exec_role", as the execution role for getting started 2).
- In the **Advanced settings**, increase the timeout to 5 seconds to avoid any timeout issues, and increase memory to 512 MB. The amount of memory and timeout is configurable and depends on the size of objects you are creating in your bucket.

For instructions, see [Step 1: Create a Lambda Function \(p. 17\)](#).

Step 1.2: Invoke the Lambda Function Manually

Follow getting started 1 to manually invoke the function and review the results. Note that you will need to select "S3 Put" from the **Sample event** drop-down so you get the sample S3 event that you will pass to the function when you invoke it.

Note

You must manually update the sample S3 event and provide your bucket name and an existing object key. Because the Lambda function retrieves the object and logs its content type.

For instructions, see [Step 2: Invoke the Lambda Function Manually](#) (p. 19).

Next Step

[Step 2: Configure Amazon S3 as the Event Source \(Using the Console\)](#) (p. 23)

Step 2: Configure Amazon S3 as the Event Source (Using the Console)

In this step, you add Amazon S3 as the event source for your Lambda function. You do this by identifying a bucket and type of S3 event that you want Amazon S3 to publish. For this example, you want Amazon S3 to invoke the Lambda function when an object is created in the bucket (regardless of the S3 API that was used to create the object). This is an example of a "push" model where Amazon S3 invokes your Lambda function. When you configure Amazon S3 as the event source, the console does the following things:

- Adds a permission, in the access policy associated with the Lambda function, to allow Amazon S3 to invoke the function.
- Add notification configuration on the S3 bucket identifying the event type (object-created events) and Lambda function. When Amazon S3 detects the event of the specified type on the bucket, it invokes the Lambda function by passing event information as function parameters. For more information about Amazon S3 notifications, go to [Configuring Notifications for Amazon S3 Events](#) in the *Amazon Simple Storage Service Developer Guide*.

For more information, see [Lambda Function Invocation Options](#) (p. 4).

You can then test the end-to-end experience; upload an object to the bucket and verify function executed by reviewing the logs. The Lambda function simply reads the incoming event data and logs some of the information to Amazon CloudWatch Logs. You can review the logs in the AWS Lambda console or click a link in the console to review logs in Amazon CloudWatch Logs.

Step 2.1: Add Amazon S3 as the Event Source

Follow the steps to add Amazon S3 as the event source for your Lambda function. You will first create a bucket in Amazon S3.

1. In the Amazon S3 console, create a bucket.

In the next step, you will add notification configuration to this bucket.

Important

The bucket must be in the same region where you created your Lambda function.

For instructions, go to [Create a Bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*.

2. In the AWS Lambda console, add an event source for your Lambda function.
 - a. On the **Lambda: Function List** page, click the triangle to the left of the function name to view expanded function details.
 - b. From the **Actions** drop-down, select **Add Event Source**.
 - c. In the dialog box that opens, select "S3" from the **Event source type** drop-down.
 - d. Select a bucket from the **Bucket** drop-down.

- e. Select **Object Created** from the **Event type** drop-down.
- f. Click **Submit**.

This will add Amazon S3 as the event source for the Lambda function. The console will also do the following to complete the setup:

- Add a permission in the function access policy to grant Amazon S3 principal permission to invoke the function.
- Add notification configuration on the S3 bucket so that Amazon S3 can publish object-created events to AWS Lambda by invoking the specific function.

This completes the setup.

Step 2.2: Test the Setup

To test the end-to-end setup, do the following:

1. Upload an object to the S3 bucket. Amazon S3 will detect the object-created event and invoke your Lambda function.

AWS Lambda will then execute your function. As the function executes, it reads Amazon S3 event data it received as parameters, and logs some of the event information to CloudWatch Logs.

2. You can verify your logs by using the following steps:
 - a. Review function logs in the AWS Lambda console. For the specific function, the logs appear under the **CloudWatch Metrics at a glance** heading.
 - b. You can click the **logs** links to open review the logs in the CloudWatch console.

Step 2.3: Cleanup

If you don't need the resources you created, you can remove them.

- Remove your S3 bucket using the Amazon S3 console.
- Remove the Lambda function using the AWS Lambda console.
- Remove IAM roles using the IAM console.

Related Topics

In this exercise, you used the AWS Lambda console for the setup. This guide also provides a follow-up walkthrough in which you do all these steps using the AWS Command Line Interface (CLI). For more information, see [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\)](#) (p. 38).

Getting Started 3: Handling Amazon Kinesis Events (Using the AWS Lambda Console)

In this exercise, you create a Lambda function to consume events from an Amazon Kinesis stream. This is an example of the "pull" model (see [AWS Lambda: How it Works \(p. 3\)](#)) where AWS Lambda polls the Amazon Kinesis stream and invokes your Lambda function when it detects new data on the stream. In this case, AWS Lambda both invokes and executes the Lambda function. As the function executes, it reads the incoming Amazon Kinesis event data and, for illustration, logs some of the event information to Amazon CloudWatch. In this getting started exercise, you will do the following:

- Create a Lambda function to process Amazon Kinesis stream events. You will first invoke the function manually by passing sample Amazon Kinesis event data to the function, and verify logs the function writes to Amazon CloudWatch.

When you are creating the Lambda function, you will also create an IAM role (execution role) that AWS Lambda can assume to both invoke and execute the Lambda function you are creating. (For an Amazon Kinesis stream as the event source, AWS Lambda uses the "pull" model, so you must grant the role permission for AWS Lambda actions to both invoke and execute the function.)

- Create an Amazon Kinesis stream and add it as the event source for your Lambda function.

As soon as you add the Amazon Kinesis stream as an event source, AWS Lambda starts polling the stream. You will then add a sample event record to the Amazon Kinesis stream and verify AWS Lambda executed your Lambda function on your behalf.

Important

Both the Lambda function and the Amazon Kinesis stream must be in the same AWS region. This exercise assumes the `us-west-2` (Oregon) region.

Next Step

[Step 1: Create a Lambda Function and Invoke It Manually \(Using the Console\) \(p. 25\)](#)

Step 1: Create a Lambda Function and Invoke It Manually (Using the Console)

In this step, you take the following actions:

- Create a Lambda function, using console-provided JavaScript code, to process Amazon Kinesis stream events. The function simply reads the event data it receives as parameters and logs some of the information to Amazon CloudWatch Logs. In the code, the `"console.log()"` statements generate log events in CloudWatch.

```
console.log('Loading function');

exports.handler = function(event, context) {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');

        console.log('Decoded payload:', payload);
    });
}
```

AWS Lambda Developer Guide

Step 1: Create a Lambda Function and Invoke It Manually (Using the Console)

```
});  
context.succeed();  
};
```

- When creating a Lambda function, the console first creates an IAM role (execution role) and provides that role at the time it sends the request to create a Lambda function. AWS Lambda can assume this role to execute the function on your behalf. For more information, [Component: Execution Role](#). Note that the console attaches the following access policy granting various permissions to the role:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "lambda:InvokeFunction"  
      ],  
      "Resource": [  
        "*"   
      ]  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "kinesis:GetRecords",  
        "kinesis:GetShardIterator",  
        "kinesis:DescribeStream",  
        "kinesis:ListStreams",  
        "logs:CreateLogGroup",  
        "logs:CreateLogStream",  
        "logs:PutLogEvents"  
      ],  
      "Resource": "*"   
    }  
  ]  
}
```

The access policy grants the following permissions to the role:

- Permission for Amazon Kinesis actions.

With Amazon Kinesis streams, AWS Lambda uses the "pull" model where it polls the stream, reads any records, and passes them to the Lambda function. So you grant permissions for relevant Amazon Kinesis actions.

- Permission for the Lambda action to allow function invocation.

In the "pull" model, AWS Lambda invokes the function when it detects new records in the stream. So the access policy grants permission for Lambda actions to invoke the function.

- Permission for Amazon CloudWatch Logs actions.

As your Lambda function executes, it writes logs to CloudWatch Logs. So you grant permissions for the relevant CloudWatch actions.

- Invoke the Lambda function manually by passing the following sample Amazon Kinesis stream event provided by the console.

```
{
  "Records": [
    {
      "kinesis": {
        "partitionKey": "partitionKey-3",
        "kinesisSchemaVersion": "1.0",
        "data": "SGVsbG8sIHRobXMgaXMGYSB0ZXN0IDEyMy4=",
        "sequenceNumber": "value"
      },
      "eventSource": "aws:kinesis",
      "eventID": "shardId-id",
      "invokeIdentityArn": "arn:aws:iam::account-id:role/testLEBRole",
      "eventVersion": "1.0",
      "eventName": "aws:kinesis:record",
      "eventSourceARN": "arn:aws:kinesis:us-east-1:account-id:stream/ex
amplestream",
      "awsRegion": "us-east-1"
    }
  ]
}
```

Step 1.1: Create a Lambda Function to Process Amazon Kinesis Stream Events

Follow the procedure in getting started 1 to create a Lambda function with the following differences:

- Select the "Kinesis Process Record" code template.
- Create a new IAM **Role** and select "Kinesis execution role..." from the drop-down. This causes the console to use the preceding access policy when creating the new role. Create a new IAM role and assign a name, for example, "lambda-exe-role-gs-3", as the execution role for getting started 3).

For instructions, see [Step 1: Create a Lambda Function \(p. 17\)](#).

Step 1.2: Invoke the Lambda Function Manually

Follow getting started 1 to manually invoke the function and review the results. Note that you will need to select **Kinesis** from the **Sample event** drop-down so you get the sample Amazon Kinesis event that your Lambda function is designed to process.

For instructions, see [Step 2: Invoke the Lambda Function Manually \(p. 19\)](#).

Next Step

[Step 2: Configure an Amazon Kinesis Stream as the Event Source \(Using the Console\) \(p. 27\)](#)

Step 2: Configure an Amazon Kinesis Stream as the Event Source (Using the Console)

In this step, you create an Amazon Kinesis stream and associate it with your Lambda function by creating an AWS Lambda event source. You then test the end-to-end setup.

Topics

AWS Lambda Developer Guide

Step 2: Configure an Amazon Kinesis Stream as the Event Source

- [Step 2.1: Add an Amazon Kinesis Stream as the Event Source](#) (p. 28)
- [Step 2.2: Test the Setup](#) (p. 28)
- [Step 2.3: Cleanup](#) (p. 29)
- [Related Topics](#) (p. 29)

Step 2.1: Add an Amazon Kinesis Stream as the Event Source

Follow the steps to create an Amazon Kinesis stream, and add an event source in AWS Lambda to associate the stream with your Lambda function.

1. In the Amazon Kinesis console, create a stream.

For this exercise, set the number of shards to 1.
2. In the AWS Lambda console, add an event source for your Lambda function.
 - a. On the **Lambda: Function List** page, click the triangle to the left of the function name to view expanded function details.
 - b. From the **Actions** drop-down, click the **Add Event Source** link for the specific function.
 - c. On the **Lambda: Publish event notifications** page, select "Kinesis" from the **Event source type** drop-down, and select your Kinesis stream from the **Kinesis stream** drop-down. You can leave defaults in other fields.

Lambda: Publish event notifications to ExampleKinesisEventHandler

Configure your Lambda function to respond to AWS events. You may also call your Lambda function directly using the AWS [mobile SDK for Android and iOS](#).

Event source type ⓘ

Kinesis stream ⓘ

Batch size ⓘ

Starting position ⓘ

Your execution role must have proper access to read the Kinesis stream. [Click here to add a new policy to your execution role with Kinesis permissions.](#)

[Cancel](#) [Submit](#)

- d. Click **Submit**.

This will add the Amazon Kinesis stream as the event source for the Lambda function.

This completes the setup.

Step 2.2: Test the Setup

Test the end-to-end experience.

You cannot use the console to add sample records to the stream, but you can upload sample records using the AWS CLI. Using the following AWS CLI command, add event records to your Amazon Kinesis stream. The `--data` value is a base64-encoded value of the "Hello, this is a test." string .

AWS Lambda Developer Guide

Step 2: Configure an Amazon Kinesis Stream as the Event Source

```
aws kinesis put-record \  
--stream-name examplestream \  
--data "Hello, this is a test" \  
--partition-key shardId-0000000000000000 \  
--region us-west-2 \  
--profile adminuser
```

The guide provides example walkthroughs that use AWS CLI for setting up and testing. So for instructions to set up AWS CLI, see [Step 1: Prepare for the Walkthrough \(p. 31\)](#).

You can run the same command more than once to add multiple records to the stream. AWS Lambda does the following:

- It polls the stream, and when it detects updates, invokes your Lambda function by passing in the event data as function parameters.
- It executes your Lambda function.

As the function executes, it reads incoming event data and writes some of the information as logs to Amazon CloudWatch Logs. You have the following options to review the logs:

- You can verify the logs in CloudWatch, make sure you check logs in the same AWS region where you created the Lambda function and stream.
- You can review function logs in the AWS Lambda console. For the specific function, the logs appear under the **Cloudwatch Metrics at a glance** heading.
- You can click the **logs** links to open review the logs in the CloudWatch console.

Step 2.3: Cleanup

If you don't need the resources you created, you can remove them.

- Remove the Amazon Kinesis stream.
- Remove the Lambda function using the AWS Lambda console.
- Remove the IAM role using the IAM console.

Related Topics

In this exercise, you complete the setup using the console. This guide also provides a follow-up walkthrough in which you do all these steps using the AWS CLI. For more information, see [AWS Lambda Walkthrough 4: Processing Events from an Amazon Kinesis Stream \(Using the AWS CLI\) \(p. 59\)](#).

AWS Lambda Walkthroughs

This section contains several AWS CLI-based examples that walk through how to use AWS Lambda. The walkthroughs show you how to create Lambda functions and invoke them in response to Amazon S3 event notifications, Amazon DynamoDB table updates, and Amazon Kinesis streams. They also illustrate how to use Lambda functions with a user application.

Topics

- [AWS Lambda Walkthrough 1: Invoking Lambda Functions from User Applications \(Using the AWS CLI\) \(p. 30\)](#)
- [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\) \(p. 38\)](#)
- [AWS Lambda Walkthrough 3: Processing Events from an Amazon DynamoDB Stream \(Using the AWS CLI\) \(p. 48\)](#)
- [AWS Lambda Walkthrough 4: Processing Events from an Amazon Kinesis Stream \(Using the AWS CLI\) \(p. 59\)](#)
- [AWS Lambda Walkthrough 5: Handling AWS CloudTrail Events \(Using the AWS CLI\) \(p. 65\)](#)

AWS Lambda Walkthrough 1: Invoking Lambda Functions from User Applications (Using the AWS CLI)

One of the use cases for using AWS Lambda is to process events generated by a user application. For demonstration purposes, you don't need to write a user application that will invoke your Lambda function. Instead, in this walkthrough, you will use sample event data and invoke your Lambda function manually.

When a user application invokes a Lambda function as shown in this walkthrough, it's an example of the AWS Lambda "request-response" model in which an application invokes a Lambda function and receives a response in real time. For more information, see [AWS Lambda: How it Works \(p. 3\)](#).

Note

In this walkthrough, you will use the AWS CLI to create and invoke a Lambda function and explore other AWS Lambda APIs.

Here's an overview of what you'll be doing:

- Create a Lambda function to process an event it receives as a parameter. You will use the following example JavaScript code to create your Lambda function.

```
console.log('Loading function');

exports.handler = function(event, context) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    context.succeed(event.key1); // Echo back the first key value
    // context.fail('Something went wrong');
};
```

The function is simple. It processes incoming event data by logging it; these logs are available in Amazon CloudWatch, and in the "request-response" model, you can request the log data be returned in the response.

- Simulate a user application sending an event to your Lambda function by invoking your Lambda function manually using the following sample event data.

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

Note

This example is similar to the getting started example (see [Getting Started 1: Invoking Lambda Functions from User Applications \(Using the AWS Lambda Console\)](#) (p. 17)). The difference is that the getting started exercises provide a console-based experience. The console does many things for you, simplifying your experience. When using the CLI, you get the raw experience of making the API calls, which can help you familiarize yourself with the AWS Lambda operations. In addition to creating and invoking a Lambda function, you will explore other Lambda API.

Next Step

[Step 1: Prepare for the Walkthrough](#) (p. 31)

Step 1: Prepare for the Walkthrough

You will need to set up the AWS CLI to test this walkthrough. The exercise assumes you are using administrator user credentials. We refer to the administrator user as *adminuser* in this walkthrough. For instructions on creating an administrator user in your AWS account, see [Setup an AWS Account and Create an Administrator User](#) (p. 13).

Set Up the AWS CLI

Before you can start the example walkthrough, you need to download and configure the AWS Command Line Interface (CLI).

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*.

- [Getting Set Up with the AWS Command Line Interface](#)
 - [Configuring the AWS Command Line Interface](#)
2. Add a named profile for the administrator user in the CLI config file. You will use this profile when executing the CLI commands.

```
[profile adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

For a list of available AWS regions, go to [Regions and Endpoints](#) in the *AWS General Reference*.

3. Verify the setup by entering the following commands at the command prompt.

- Try the help command to verify that the AWS CLI is installed on your computer:

```
aws help
```

- Try a Lambda command to verify the user can reach AWS Lambda. This command lists Lambda functions in the account, if any. The AWS CLI uses the `adminuser` credentials to authenticate the request.

```
aws lambda list-functions --profile adminuser
```

Next Step

[Step 2: Create a Lambda Function \(p. 32\)](#)

Step 2: Create a Lambda Function

In this section, you first do the following:

- Create a deployment package — A deployment package is a .zip file containing your code and any dependencies. For this walkthrough there are no dependencies, you only have a simple example code.
- Create an IAM role (execution role) — At the time you upload your deployment package to create your Lambda function, you must specify an IAM role. AWS Lambda uses this role when executing your function.

You also grant this role the permissions your Lambda function needs. The code in this walkthrough writes logs to Amazon CloudWatch Logs. So you grant permission for CloudWatch actions. For more information, see [Amazon LambdaWatch Logs](#).

You will then create a Lambda function ("HelloWorld") using the `create-function` CLI command. For more information about the underlying API and related parameters, see [CreateFunction \(p. 103\)](#).

Step 2.1: Create a Lambda Function Deployment Package

Follow the instructions to create an AWS Lambda function deployment package.

1. Open a text editor, and copy the following code.

```
console.log('Loading function');

exports.handler = function(event, context) {
  console.log('value1 =', event.key1);
  console.log('value2 =', event.key2);
  console.log('value3 =', event.key3);
  context.succeed(event.key1); // Echo back the first key value
  // context.fail('Something went wrong');
};
```

2. Save the file as `helloworld.js`.
3. Zip the `helloworld.js` file as `helloworld.zip`.

Step 2.2: Create an IAM Role (execution role)

When the Lambda function in this exercise executes, it will need permissions to write logs to Amazon CloudWatch. You grant these permission by creating an IAM role (also referred as an execution role). AWS Lambda assumes this role when executing your Lambda function on your behalf. In this section, you create an IAM role using the following predefined role type and access policy:

- AWS service role of the "AWS Lambda" type. This role grants AWS Lambda permission to assume the role.
- "AWSLambdaBasicExecutionRole" access policy that you attach to the role. This existing policy grants permissions that include permissions for Amazon CloudWatch actions that your Lambda function needs.

For more information about IAM roles, go to [Roles \(Delegation and Federation\)](#) in *Using IAM*.

To create an IAM role (executionrole)

1. Sign in to the AWS Management Console.
2. In the IAM console, create an IAM role, `executionrole`. As you follow the steps to create a role, note the following:
 - In **Select Role Type**, click **AWS Service Roles**, and then select **AWS Lambda**.
 - In **Attach Policy**, select the policy named **AWSLambdaBasicExecutionRole**.

For instructions, go to [Creating a Role for an AWS Service \(AWS Management Console\)](#) in *Using IAM*.

3. Write down the Amazon Resource Name (ARN) of the IAM role. You will need this value when you create your Lambda function in the next step.

Step 2.3: Create a Lambda Function

Execute the following Lambda CLI `create-function` command to create a Lambda function. You provide the deployment package and IAM role ARN as parameters.

```
$ aws lambda create-function \
--region us-west-2 \
--function-name helloworld \
```

```
--zip-file fileb://file-path/helloworld.zip \  
--role role-arn \  
--handler helloworld.handler \  
--runtime nodejs \  
--profile adminuser
```

For more information, see [CreateFunction \(p. 103\)](#). AWS Lambda creates the function and returns function configuration information as shown in the following example:

```
{  
  "FunctionName": "helloworld",  
  "CodeSize": 351,  
  "MemorySize": 128,  
  "FunctionArn": "function-arn",  
  "Handler": "helloworld.handler",  
  "Role": "arn:aws:iam::356671443308:role/LambdaExecRole",  
  "Timeout": 3,  
  "LastModified": "2015-04-07T22:02:58.854+0000",  
  "Runtime": "nodejs",  
  "Description": ""  
}
```

Next Step

[Step 3: Invoke the Lambda Function \(p. 34\)](#)

Step 3: Invoke the Lambda Function

In this section, you invoke your Lambda function manually using the invoke CLI command.

```
$ aws lambda invoke \  
--invocation-type RequestResponse \  
--function-name helloworld \  
--region us-west-2 \  
--log-type Tail \  
--payload '{"key1":"value1", "key2":"value2", "key3":"value3"}' \  
--profile adminuser \  
outputfile.txt
```

If you want you can save the payload to a file (say "input.txt") and provide the file name as a parameter.

```
--payload file://input.txt \  

```

The preceding `invoke` command specifies "RequestResponse" as the invocation type, which returns a response immediately in response to the execution. You can alternatively specify "Event" as the invocation type to invoke the function asynchronously.

By specifying the `--log-type` parameter, the command also requests the tail end of the log produced by the function. The log data in the response is base64-encoded as shown in the following example response:

```
{  
  "LogResult": "base64-encoded-log",  
}
```

```
    "StatusCode": 200
}
```

On Linux and Mac, you can use the base64 command to decode the log.

```
$ echo base64-encoded-log | base64 --decode
```

The following is a decoded version of an example log.

```
START RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
2015-04-01T18:44:12.323Z    16d25499-d89f-11e4-9e64-5d70fce44801    value1 =
value1
2015-04-01T18:44:12.323Z    16d25499-d89f-11e4-9e64-5d70fce44801    value2 =
value2
2015-04-01T18:44:12.323Z    16d25499-d89f-11e4-9e64-5d70fce44801    value3 =
value3
2015-04-01T18:44:12.323Z    16d25499-d89f-11e4-9e64-5d70fce44801    result:
"value1"
END RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
REPORT RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
Duration: 13.35 ms        Billed Duration: 100 ms    Memory Size: 128 MB
Max Memory Used: 9 MB
```

For more information, see [Invoke \(p. 121\)](#).

Because you invoked the function using the "RequestResponse" invocation type, the function executes and returns the object you passed to the `context.succeed()` in real time when it is called. In this example, you will see the following text written to the `outputfile.txt` you specified in the CLI command:

```
"value1"
```

Note

You are able to execute this function because you are using the same AWS account to create and invoke the Lambda function. However, if you want to grant cross-account permission to another AWS account or an AWS service permission to execute the function, you must add a permission to the access policy associated with the function. The walkthrough that uses Amazon S3 as the event source (see [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\)](#) (p. 38)) grants such permission to Amazon S3 to invoke the function.

You can monitor the activity of your Lambda function in the AWS Lambda console.

- The AWS Lambda console shows a graphical representation of some of the CloudWatch metrics in the **Cloudwatch Metrics at a glance** section for your function. Sign in to the AWS Management Console at <https://console.aws.amazon.com/>.
- For each graph, you can also click the **logs** link to view the CloudWatch logs directly.

Next Step

[Step 4: Try More CLI Commands \(p. 36\)](#)

Step 4: Try More CLI Commands

Step 4.1: List the Lambda Functions in Your Account

In this section, you try AWS Lambda list function operations. Execute the following CLI `list-functions` command to retrieve a list of functions you uploaded.

```
$ aws lambda list-functions \
--max-items 10 \
--profile adminuser
```

To illustrate the use of pagination, the command specifies the optional `--max-items` parameter to limit the number of functions returned in the response. For more information, see [ListFunctions](#) (p. 128). The following is an example response.

```
{
  "Functions": [
    {
      "FunctionName": "helloworld",
      "MemorySize": 128,
      "CodeSize": 412,
      "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
      "Handler": "ProcessKinesisRecords.handler",
      "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
      "Timeout": 3,
      "LastModified": "2015-02-22T21:03:01.172+0000",
      "Runtime": "nodejs",
      "Description": ""
    },
    {
      "FunctionName": "ProcessKinesisRecords",
      "MemorySize": 128,
      "CodeSize": 412,
      "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
      "Handler": "ProcessKinesisRecords.handler",
      "Role": "arn:aws:iam::account-id:role/lambda-execute-test-kinesis",
      "Timeout": 3,
      "LastModified": "2015-02-22T21:03:01.172+0000",
      "Runtime": "nodejs",
      "Description": ""
    },
    ...
  ],
  "NextMarker": null
}
```

In response, Lambda returns a list of up to 10 functions. If there are more functions you can retrieve, `NextMarker` provides a marker you can use in the next `list-functions` request; otherwise, the value is null. The following `list-functions` CLI command is an example that shows the `--next-marker` parameter.

```
$ aws lambda list-functions \
--max-items 10 \
--marker value-of-NextMarker-from-previous-response \
--profile adminuser
```

Step 4.2: Get Metadata and URL to Download Previously Uploaded Lambda Function Deployment Package

The Lambda CLI `get-function` command returns Lambda function metadata and a presigned URL that you can use to download the function's .zip file (deployment package) that you uploaded to create the function. For more information, see [GetFunction \(p. 114\)](#).

```
$ aws lambda get-function \
--function-name helloworld \
--region us-west-2 \
--profile adminuser
```

The following is an example response.

```
{
  "Code": {
    "RepositoryType": "S3",
    "Location": "pre-signed-url"
  },
  "Configuration": {
    "FunctionName": "helloworld",
    "MemorySize": 128,
    "CodeSize": 287,
    "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld",

    "Handler": "helloworld.handler",
    "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
    "Timeout": 3,
    "LastModified": "2015-04-07T22:02:58.854+0000",
    "Runtime": "nodejs",
    "Description": ""
  }
}
```

If you want only the function configuration information (not the presigned URL), you can use the Lambda CLI `get-function-configuration` command.

```
$ aws lambda get-function-configuration \
--function-name helloworld \
--region us-west-2 \
--profile adminuser
```

Next Step

[Step 5: Delete the Lambda Function and IAM Role \(p. 38\)](#)

Step 5: Delete the Lambda Function and IAM Role

Execute the following `delete-function` command to delete `helloworld` function.

```
$ aws lambda delete-function \  
  --function-name helloworld \  
  --region us-west-2 \  
  --profile adminuser
```

Delete IAM Role

After you delete the Lambda function you can also delete the IAM role you created in the IAM console. For information about deleting a role, see [Deleting Roles or Instance Profiles](#) in *Using IAM*.

AWS Lambda Walkthrough 2: Handling Amazon S3 Events (Using the AWS CLI)

Scenario

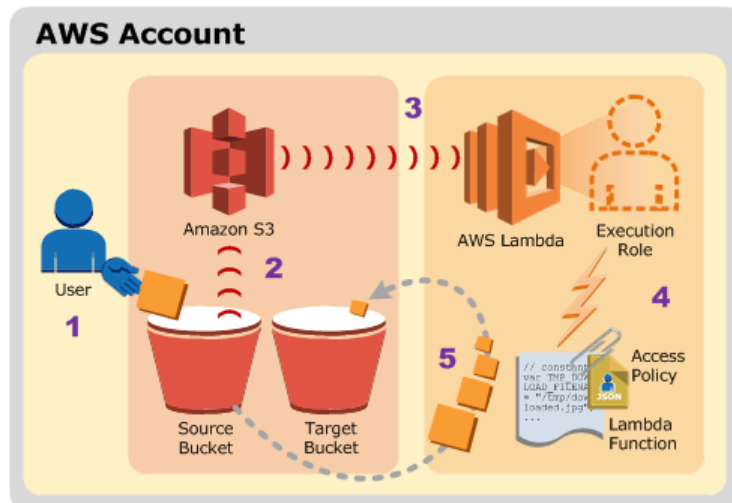
Suppose you have two buckets in Amazon S3. You store images (.jpg and .png objects) in one bucket (*sourcebucket*), and for each object created in the bucket, you want AWS Lambda to execute a Lambda function to create a thumbnail in the *sourcebucket*resized bucket. You will use Amazon S3's bucket notification configuration feature to request Amazon S3 to publish object-created events to AWS Lambda. In the notification configuration, you will identify your Lambda function ("CreateThumbnail") that you want Amazon S3 to invoke.

Important

You must use two buckets. If you use the same bucket as source and target, each thumbnail uploaded to the source bucket will trigger another object-created event, which will invoke the Lambda function creating the unwanted recursion.

Implementation Summary

The following diagram illustrates the application flow:



1. A user uploads an object to the source bucket.
2. Amazon S3 detects the object-created event.
3. Amazon S3 publishes the `s3:ObjectCreated:*` event to AWS Lambda by invoking the Lambda function by passing event data as function parameter.
4. Lambda executes the function by assuming the execution role.
5. From the event data it received, the Lambda function knows the source bucket name and object key name. The Lambda function reads the object and creates a thumbnail using graphics libraries, and saves it to the target bucket.

Note that upon completing this exercise, you will have the following S3, Lambda, and IAM resources in your account.



As the diagram shows, you create Amazon S3, AWS Lambda, and AWS Identity and Access Management (IAM) resources in your account:

In Lambda:

- A Lambda function – Amazon S3 invokes this function by passing event data such as the source bucket name and object key as parameter. The function reads the object and, if the object is a .jpg or .png object, creates a thumbnail in the target bucket.
- An access policy – In the access policy associated with the Lambda function, you will add a permission granting Amazon S3 permission to invoke the Lambda function. You will also restrict the permission so that Amazon S3 can invoke the Lambda function only for object-created events from a specific bucket owned by a specific AWS account.

Note

It is possible for an AWS account to delete a bucket and some other AWS account to later create a bucket with same name. The additional conditions ensure that Amazon S3 can invoke the Lambda function only if Amazon S3 detects object-created events from a specific bucket owned by a specific AWS account.

For more information, see [AWS Lambda: How it Works \(p. 3\)](#).

In IAM:

- Administrator user – Called *adminuser*. Using root credentials of an AWS account is not recommended. Instead, use the *adminuser* credentials to perform steps in this exercise.
- An IAM role (execution role) – When you create this role, you will trust AWS Lambda to assume this role. You will also grant sufficient permissions that your Lambda function needs—for example, permission to read objects from the source bucket and create a thumbnail in the target bucket.

In Amazon S3:

- Two buckets – We refer to these as the source bucket and the target bucket.
- Notification configuration – You will add notification configuration on your source bucket identifying the type of events (object-created events) you want Amazon S3 to publish to AWS Lambda and the Lambda function to invoke. For more information about the Amazon S3 notification feature, go to [Setting Up Notification of Bucket Events](#).

Now you are ready to try the steps. Note that after the initial preparation the walkthrough is divided into two main sections:

- First, you do the necessary setup to create a Lambda function and invoke it manually using Amazon S3 sample event data. This intermediate testing verifies that the function works.
- Then you will add a notification configuration to your source bucket so that Amazon S3 can then invoke your Lambda function when it detects object-created events.

Next Step

[Step 1: Preparing for the Walkthrough](#) (p. 40)

Step 1: Preparing for the Walkthrough

In this section, you do the following:

- Create an administrator user, *adminuser*. For instructions, see [Setup an AWS Account and Create an Administrator User](#) (p. 13). If you have an administrator user, you can skip this step.
- Create two buckets with a sample .jpg object (`HappyFace.jpg`) in the source bucket.
- Set up the AWS CLI.

Step 1.1: Create Buckets and Upload a Sample Object

Follow the steps to create buckets and upload an object.

Important

Both the source bucket and your Lambda function must be in the same AWS region. In addition, the example code used for the Lambda function also assumes both the buckets are in the same region. This exercise assumes the `us-west-2` (Oregon) region.

1. Using the IAM User Sign-In URL, sign in to the Amazon S3 console as *adminuser*.
2. Create two buckets. The target bucket name must be `sourcebucket` followed by "resized". For example, "mybucket" and "mybucketresized".

For instructions, go to [Create a Bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*.

3. In the source bucket, upload a .jpg object, `HappyFace.jpg`.

When you invoke the Lambda function manually, before hooking up Amazon S3, you will pass sample event data to the function that specifies the source bucket and `HappyFace.jpg` as the newly created object. So you need to create this sample object.

Step 1.2: Set Up the AWS CLI

You use the AWS CLI to upload your Lambda function deployment package. You will test the resulting Lambda function by manually invoking it using the CLI. Walkthrough 1 provides instructions to set up the AWS CLI and add the *adminuser* profile. For instructions, see [Step 1: Prepare for the Walkthrough \(p. 31\)](#).

Next Step

[Step 2: Create and Test the Lambda Function \(p. 41\)](#)

Step 2: Create and Test the Lambda Function

In this section, you do the following:

- Create a Lambda function deployment package using the sample Node.js code provided.
- Create an IAM role (execution role) — At the time you upload the deployment package, you will need to specify an IAM role (execution role) that Lambda can assume to execute the function on your behalf.
- Create the Lambda function and test.

Follow the instructions in the following sections:

[Step 2.1: Create a Lambda Function Deployment Package \(p. 41\)](#)

[Step 2.2: Create an IAM Role \(execution role\) \(p. 44\)](#)

[Step 2.3: Upload the Deployment Package and Test \(p. 45\)](#)

Step 2.1: Create a Lambda Function Deployment Package

The deployment package is a zip file containing your function code and dependencies. So you will first create a directory to save the JavaScript function code and dependencies. After you complete the steps, you will have the following folder structure:

```
CreateThumbnail.js
/node_modules/gm
/node_modules/async
```

You will then zip the directory content, which is your Lambda function deployment package.

To create a Lambda function deployment package

1. Create a folder (`examplefolder`). After creating the folder, create a subfolder (`node_modules`) in it.
2.
 - a. Install the Node.js platform. For more information, go to nodejs.org.
 - b. Install dependencies. The code examples uses the following libraries.
 - AWS SDK for JavaScript in Node.js
 - gm, "GraphicsMagick for node.js"
 - Async utility module

The AWS Lambda runtime already has the AWS SDK for JavaScript in Node.js. So you need only the other two. Open a command prompt, navigate to the `examplefolder`, and install the libraries using the `npm` command, which is part of Node.js.

```
npm install async gm
```

3. Save example code to the folder. Open a text editor, and copy the following code.

```
// dependencies
var async = require('async');
var AWS = require('aws-sdk');
var gm = require('gm')
    .subclass({ imageMagick: true }); // Enable ImageMagick integration.
var util = require('util');

// constants
var MAX_WIDTH = 100;
var MAX_HEIGHT = 100;

// get reference to S3 client
var s3 = new AWS.S3();

exports.handler = function(event, context) {
    // Read options from the event.
    console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
    var srcBucket = event.Records[0].s3.bucket.name;
    // Object key may have spaces or unicode non-ASCII characters.
    var srcKey = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));

    var dstBucket = srcBucket + "resized";
    var dstKey = "resized-" + srcKey;

    // Sanity check: validate that source and destination are different buckets.

    if (srcBucket == dstBucket) {
        console.error("Destination bucket must not match source bucket.");
        return;
    }

    // Infer the image type.
    var typeMatch = srcKey.match(/\.[^\.]*$/);
    if (!typeMatch) {
        console.error('unable to infer image type for key ' + srcKey);
        return;
    }
    var imageType = typeMatch[1];
    if (imageType != "jpg" && imageType != "png") {
        console.log('skipping non-image ' + srcKey);
        return;
    }

    // Download the image from S3, transform, and upload to a different S3
```

```
bucket.
  async.waterfall([
    function download(next) {
      // Download the image from S3 into a buffer.
      s3.getObject({
        Bucket: srcBucket,
        Key: srcKey
      },
      next);
    },
    function tranform(response, next) {
      gm(response.Body).size(function(err, size) {
        // Infer the scaling factor to avoid stretching the image unnaturally.
        var scalingFactor = Math.min(
          MAX_WIDTH / size.width,
          MAX_HEIGHT / size.height
        );
        var width = scalingFactor * size.width;
        var height = scalingFactor * size.height;

        // Transform the image buffer in memory.
        this.resize(width, height)
          .toBuffer(imageType, function(err, buffer) {
            if (err) {
              next(err);
            } else {
              next(null, response.ContentType, buffer);
            }
          });
      });
    },
    function upload(contentType, data, next) {
      // Stream the transformed image to a different S3 bucket.
      s3.putObject({
        Bucket: dstBucket,
        Key: dstKey,
        Body: data,
        ContentType: contentType
      },
      next);
    }
  ], function (err) {
    if (err) {
      console.error(
        'Unable to resize ' + srcBucket + '/' + srcKey +
        ' and upload to ' + dstBucket + '/' + dstKey +
        ' due to an error: ' + err
      );
    } else {
      console.log(
        'Successfully resized ' + srcBucket + '/' + srcKey +
        ' and uploaded to ' + dstBucket + '/' + dstKey
      );
    }
  });

  context.done();
}
```



```
} ;  
};
```

4. Review the preceding code and note the following:
 - The function knows the source bucket name and the key name of the object from the event data it receives as parameters. If the object is a .jpg, the code creates a thumbnail and saves it to the target bucket.
 - The code assumes the destination bucket exists and its name is a concatenation of the source bucket name followed by the string "resized". For example, if the source bucket identified in the event data is "examplebucket", the code assumes you have an "examplebucketresized" destination bucket.
 - For the thumbnail it creates, the code derives its key name as the concatenation of the string "resized-" followed by the source object key name. For example, if the source object key is "sample.jpg", the code creates a thumbnail object that has the key "resized-sample.jpg".
5. Save the file as `CreateThumbnail.js` in `examplefolder`.
6. Zip the folder content as `CreateThumbnail.zip`.

Important

You zip the folder content, not the folder itself.

This is your Lambda function deployment package.

Next Step

[Step 2.2: Create an IAM Role \(execution role\)](#) (p. 44)

Step 2.2: Create an IAM Role (execution role)

In this section, you create an IAM role using the following predefined role type and access policy:

- AWS service role of the "AWS Lambda" type. This role grants AWS Lambda permission to assume the role.
- "AWSLambdaExecute" access policy that you attach to the role.

For more information about IAM roles, go to [Roles \(Delegation and Federation\)](#) in *Using IAM*. Use the following procedure to create the IAM role.

To create an IAM role (execution role)

1. Sign in to the AWS Management Console.
2. Create an IAM role.

For instructions on creating the role, go to [Creating a Role for an AWS Service \(AWS Management Console\)](#) in *Using IAM*. As you follow the steps to create a role, note the following:

- In **Select Role Type**, click **AWS Service Roles**, and then select **AWS Lambda**. This will grant AWS Lambda service permission to assume the role.
 - In **Attach Policy** select `AWSLambdaExecute`.
3. Write down the role ARN. You will need it in the next step when you create your Lambda function.

Step 2.3: Upload the Deployment Package and Test

In this section, you do the following:

- Create a Lambda function by uploading the deployment package.
- Test the Lambda function by invoking it manually, by passing sample Amazon S3 event data as parameter.

Step 2.3.1: Create a Lambda Function (Upload the Deployment Package)

1. At the command prompt, run the following Lambda CLI `create-function` command using the `adminuser` profile.

You will need to update the command by providing the .zip file path and the execution role ARN.

```
$ aws lambda create-function \
--region us-west-2 \
--function-name CreateThumbnail \
--zip-file fileb://file-path/CreateThumbnail.zip \
--role role-arn \
--handler CreateThumbnail.handler \
--runtime nodejs \
--profile adminuser \
--timeout 10 \
--memory-size 1024
```

2. Write down the function ARN. You will need this in the next section when you add notification configuration to your Amazon S3 bucket.
3. (Optional) The preceding command specifies a 10-second timeout value as the function configuration. Depending on the size of objects you upload, you might need to increase the timeout value using the following CLI command.

```
$ aws lambda update-function-configuration \
--function-name CreateThumbnail \
--region us-west-2 \
--timeout timeout-in-seconds \
--profile adminuser
```

Step 2.3.2: Test Lambda Function (Invoke Manually)

Invoke the function manually using sample Amazon S3 event data.

1. Save the following Amazon S3 sample event data in a file, `input.txt`.

You will need to update the JSON by providing your *sourcebucket* and a .jpg object key.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-west-2",
```

```
"eventTime": "1970-01-01T00:00:00.000Z",
"eventName": "ObjectCreated:Put",
"userIdentity": {
  "principalId": "AIDAJDPLRKL7UEXAMPLE"
},
"requestParameters": {
  "sourceIPAddress": "127.0.0.1"
},
"responseElements": {
  "x-amz-request-id": "C3D13FE58DE4C810",
  "x-amz-id-2": "FMMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWer
MUE5JgHvANOjpd"
},
"s3": {
  "s3SchemaVersion": "1.0",
  "configurationId": "testConfigRule",
  "bucket": {
    "name": "sourcebucket",
    "ownerIdentity": {
      "principalId": "A3NL1KOZZKExample"
    },
    "arn": "arn:aws:s3:::sourcebucket"
  },
  "object": {
    "key": "HappyFace.jpg",
    "size": 1024,
    "eTag": "d41d8cd98f00b204e9800998ecf8427e",
    "versionId": "096fKKXTRTtl3on89fVO.nfljtsv6qko"
  }
}
}
```

2. Run the following Lambda CLI `invoke` command to invoke the function. Note that:

- The command requests asynchronous execution. You can optionally invoke it synchronously by specifying "RequestResponse" as the invocation-type parameter value.

```
$ aws lambda invoke \
--invocation-type Event \
--function-name CreateThumbnail \
--region us-west-2 \
--payload file:///file-path/inputfile.txt \
--profile adminuser \
outputfile.txt
```

Note

You are able to invoke this function because you are using your own credentials to invoke your own function. In the next section, you configure Amazon S3 to invoke this function on your behalf which requires you to add a permission to the access policy associated with your Lambda function to grant Amazon S3 permission to invoke your function.

3. Verify results:

- Verify the thumbnail was created in the target bucket.
- You can monitor the activity of your Lambda function in the AWS Lambda console.

- The AWS Lambda console shows a graphical representation of some of the CloudWatch metrics in the **Cloudwatch Metrics at a glance** section for your function.
- For each graph, you can also click the **logs** link to view the CloudWatch logs directly.

Next Step

[Step 3: Configure Amazon S3 to Publish Events \(p. 47\)](#)

Step 3: Configure Amazon S3 to Publish Events

In this section, you add the remaining configuration so Amazon S3 can publish object-created events to AWS Lambda and invoke your Lambda function. You will do the following:

- Add permission to the Lambda function access policy to allow Amazon S3 to invoke the function.
- Add notification configuration to your source bucket — In the notification configuration, you provide:
 - Event type for which you want Amazon S3 to publish events. For this exercise, you will specify the `s3:ObjectCreated:*` event type so that Amazon S3 publishes events when objects are created.
- Lambda function to invoke.

Step 3.1: Add Permission to the Lambda Function Access Policy

1. Run the following Lambda CLI `add-permission` command to grant Amazon S3 service principal ("s3.amazonaws.com") permission for the `lambda:InvokeFunction` action. Note that permission is granted to Amazon S3 to invoke the function only if the following conditions are met:
 - An object-created event is detected on a specific bucket.
 - The bucket is owned by a specific AWS account.

If a bucket owner deletes a bucket, some other AWS account can create bucket with the same name. This condition ensures that only a specific AWS account can invoke your Lambda function.

```
$ aws lambda add-permission \
--function-name CreateThumbnail \
--region us-west-2 \
--statement-id some-unique-id \
--action "lambda:InvokeFunction" \
--principal s3.amazonaws.com \
--source-arn arn:aws:s3:::sourcebucket \
--source-account bucket-owner-account-id \
--profile adminuser
```

2. Verify the access policy of your function by calling the CLI `get-policy` command.

```
$ aws lambda get-policy \
--function-name function-name \
--profile adminuser
```

Step 3.2: Configure a Notification on the Bucket

Add notification configuration on the source bucket to request Amazon S3 to publish object-created events to Lambda. In the configuration, you specify the following:

- Event type — For this exercise, select the "ObjectCreated (All)" Amazon S3 event type.
- Lambda function — This is your Lambda function Amazon S3 will invoke.

For instructions on adding notification configuration to a bucket, go to [Enabling Event Notifications](#) in the *Amazon Simple Storage Service Console User Guide*.

Step 3.3: Test the Setup

You are all done! Now *adminuser* can test the setup as follows:

1. Upload .jpg or .png objects to the source bucket using the Amazon S3 console.
2. Verify that the thumbnail was created in the target bucket by the `CreateThumbnail` function.
3. The *adminuser* user can also verify the logs reported by Amazon CloudWatch Logs.

You can monitor the activity of your Lambda function in the AWS Lambda console. For example, choose the logs link in the console to view logs, including logs your function wrote to CloudWatch Logs.

AWS Lambda Walkthrough 3: Processing Events from an Amazon DynamoDB Stream (Using the AWS CLI)

Scenario

Suppose you have a stream associated with a DynamoDB table. As the table is updated you want to process the updates that DynamoDB publishes to the corresponding stream, so you write a Lambda function to analyze the updates. We create a simple Lambda function in this walkthrough that reads incoming event data and logs some of the information to Amazon CloudWatch.

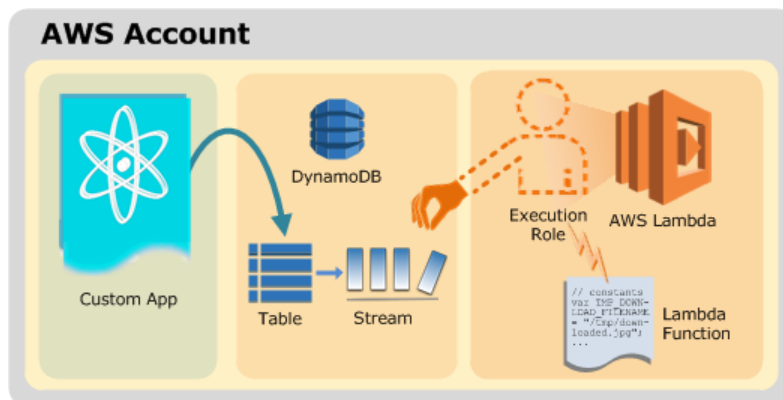
Important

To explore this Amazon DynamoDB and AWS Lambda integration walkthrough, you will need to sign up for the DynamoDB streams because this feature is currently in preview mode. For more information, go to [DynamoDB Streams Preview](#) in the *Amazon DynamoDB Developer Guide*.

Implementation Summary

This is an example of the "pull" model (see [AWS Lambda: How it Works \(p. 3\)](#)) where AWS Lambda polls the Amazon DynamoDB stream and invokes your Lambda function when it detects new data on the stream. In the "pull" model, AWS Lambda both invokes and executes the Lambda function.

The following diagram illustrates the application workflow:



In this walkthrough, you will do the following:

- Create a Lambda function to process Amazon DynamoDB events.
- Invoke a Lambda function manually by using sample Amazon DynamoDB event data.
- Create a stream-enabled DynamoDB table.

Using the AWS SDK for Java, you will create a table with a stream. You will then perform simple add, update, and delete operations. Amazon DynamoDB then sends stream records to the stream.

- Create an event source mapping in AWS Lambda associating the stream and your Lambda function.

As soon as you create the event source mapping, AWS Lambda starts polling the stream.

- Test the setup

You will then perform table operations that will cause Amazon DynamoDB to write updates the stream. You then verify AWS Lambda executed your Lambda function on your behalf.

Important

Both the Lambda function and the Amazon DynamoDB stream must be in the same AWS region. This exercise assumes the US East (N. Virginia) region (us-east-1). You must use the DynamoDB endpoints available for this preview.

Next Step

[Step 1: Preparing for the Walkthrough \(p. 49\)](#)

Step 1: Preparing for the Walkthrough

In this section, you do the following:

- If you don't have an administrator user in your account, create it. For instructions, see [Setup an AWS Account and Create an Administrator User \(p. 13\)](#).
- Set up the AWS CLI. For instructions, see [Step 1: Prepare for the Walkthrough \(p. 31\)](#).

You will use AWS CLI to perform the AWS Lambda activities, such as create a Lambda function, initially invoke it manually, and add an event source mapping.

Next Step

[Step 2: Create a Lambda Function and Invoke it Manually Using Sample Event Data \(p. 50\)](#)

Step 2: Create a Lambda Function and Invoke it Manually Using Sample Event Data

Topics

- [Step 2.1: Create a Lambda Function \(p. 50\)](#)
- [Step 2.2: Invoke Lambda Function Manually \(p. 51\)](#)
- [Next Step \(p. 54\)](#)

In this section, you create a Lambda function and manually invoke it by passing a sample DynamoDB event.

Step 2.1: Create a Lambda Function

To create a Lambda function you need to first create the following:

- A deployment package (a .zip file) containing your code and dependencies.

After you upload the deployment package to AWS Lambda we refer it as your Lambda function. The code for this exercise is a Node.js example code, and there are no dependencies.

- An IAM role (execution role).

At the time you create Lambda function you specify an IAM role that AWS Lambda can assume to execute the function on your behalf.

You must grant this execution role necessary permissions. For example AWS Lambda will need permission for Amazon DynamoDB actions so it can poll the stream and read records from the stream. In the "pull" model you must also grant AWS Lambda service permission to invoke your Lambda function. The example Lambda function writes some of the event data to Amazon CloudWatch so your function will need permissions for necessary Amazon CloudWatch actions.

For more information, see [Execution Permissions \(p. 7\)](#).

You provide both the deployment package and the IAM role at the time of creating a Lambda function. You can also specify other configuration information such as the function name, memory size, runtime environment (nodejs) to use, and the handler. For more information about these parameters, see [CreateFunction \(p. 103\)](#).

After creating the Lambda function you will invoke it using sample Amazon DynamoDB event data.

Step 2.1.1: Create a Lambda Function Deployment Package

Follow the instructions to create AWS Lambda function deployment package.

1. Open a text editor, and copy the following code.

```
console.log('Loading function');

exports.handler = function(event, context) {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        console.log(record.eventID);
        console.log(record.eventName);
        console.log('DynamoDB Record: %j', record.dynamodb);
    });
};
```

```
context.succeed("Processing successful");  
};
```

2. Save the file as `ProcessDynamoDBStream.js`.
3. Zip the `ProcessDynamoDBStream.js` file as `ProcessDynamoDBStream.zip`.

Step 2.1.2: Create an IAM Role (execution role)

In this section you create an IAM role using the following predefined role type and access policy:

- AWS service role of the "AWS Lambda" type. This role grants AWS Lambda permission to assume the role.
- "AWSLambdaDynamoDBExecutionRole" access policy that you attach to the role.

For more information about IAM roles, go to [Roles \(Delegation and Federation\)](#) in *Using IAM*. Create IAM role using the following procedure:

To create an IAM role (execution role)

1. Sign in to the AWS Management Console.
2. Create an IAM role.

For instructions on creating the role, go to [Creating a Role for an AWS Service \(AWS Management Console\)](#) in *Using IAM*. As you follow the steps to create a role, note the following:

- In **Select Role Type**, click **AWS Service Roles**, and then select **AWS Lambda**. This will grant AWS Lambda service permission to assume the role.
- In **Attach Policy** select `AWSLambdaDynamoDBExecutionRole`.

3. Write down the role ARN. You will need the ARN in the next step when you create Lambda function.

Step 2.1.3: Create Lambda Function

Execute the following Lambda CLI `create-function` command to create a Lambda function. You provide the deployment package and IAM role ARN as parameters.

```
$ aws lambda create-function \  
--region us-east-1 \  
--function-name ProcessDynamoDBStream \  
--zip-file fileb://file-path/ProcessDynamoDBStream.zip \  
--role role-arn \  
--handler ProcessDynamoDBStream.handler \  
--runtime nodejs \  
--profile adminuser
```

For more information, see [CreateFunction \(p. 103\)](#). AWS Lambda creates the function and return function configuration information.

Step 2.2: Invoke Lambda Function Manually

In this section you invoke your Lambda function manually using the `invoke` CLI command.

AWS Lambda Developer Guide

Step 2: Create a Lambda Function and Invoke it Manually Using Sample Event Data

Save the following JSON in a file, `input.txt`.

```
{
  "Records": [
    {
      "EventName": "INSERT",
      "EventVersion": "1.0",
      "EventSource": "aws:dynamodb",
      "Dynamodb": {
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "SizeBytes": 26,
        "StreamViewType": "NEW_AND_OLD_IMAGES",
        "SequenceNumber": "111",
        "Keys": {
          "Id": {
            "N": "101"
          }
        }
      },
      "EventID": "1",
      "eventSourceARN": "arn:aws:dynamodb:us-east-1:acct-id:table/ExampleTableWithStream/stream/stream-id/",
      "AwsRegion": "us-east-1"
    },
    {
      "EventName": "MODIFY",
      "EventVersion": "1.0",
      "EventSource": "aws:dynamodb",
      "Dynamodb": {
        "NewImage": {
          "Message": {
            "S": "This item has changed"
          },
          "Id": {
            "N": "101"
          }
        },
        "SizeBytes": 59,
        "StreamViewType": "NEW_AND_OLD_IMAGES",
        "SequenceNumber": "222",
        "OldImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "Keys": {
          "Id": {
            "N": "101"
          }
        }
      }
    }
  ]
}
```

AWS Lambda Developer Guide

Step 2: Create a Lambda Function and Invoke it Manually Using Sample Event Data

```
    }
  },
  "EventID": "2",
  "eventSourceARN": "arn:aws:dynamodb:us-east-1:acct-id:table/ExampleTableWithStream/stream/stream-id/",
  "AwsRegion": "us-east-1"
},
{
  "EventName": "REMOVE",
  "EventVersion": "1.0",
  "EventSource": "aws:dynamodb",
  "Dynamodb": {
    "SizeBytes": 38,
    "StreamViewType": "NEW_AND_OLD_IMAGES",
    "SequenceNumber": "333",
    "OldImage": {
      "Message": {
        "S": "This item has changed"
      },
      "Id": {
        "N": "101"
      }
    },
    "Keys": {
      "Id": {
        "N": "101"
      }
    }
  },
  "EventID": "3",
  "eventSourceARN": "arn:aws:dynamodb:us-east-1:acct-id:table/ExampleTableWithStream/stream/stream-id/",
  "AwsRegion": "us-east-1"
}
]
```

Execute the following `invoke` command.

```
$ aws lambda invoke \
--invocation-type RequestResponse \
--function-name ProcessDynamoDBStream \
--region us-east-1 \
--payload file://file-path/input.txt \
--profile adminuser
outputfile.txt
```

Note the `invoke` command specifies the "RequestResponse" as the invocation type which requests synchronous execution. For more information, see [Invoke \(p. 121\)](#). The function returns the string message (message in the `context.succeed()` in the code) in the response body. In the preceding example, it will be saved in `outputfile.txt`.

```
"Processing successful"
```

You can monitor the activity of your Lambda function in the AWS Lambda console.

- The AWS Lambda console shows a graphical representation of some of the CloudWatch metrics in the **Cloudwatch Metrics at a glance** section for your function. Sign in to the AWS Management Console at <https://console.aws.amazon.com/>.
- For each graph you can also click the **logs** link to view the CloudWatch logs directly.

Note

The DynamoDB stream and Lambda function must be in the same AWS account.

Next Step

[Step 3: Create AWS Lambda Event Source Mapping and Test the Setup \(p. 54\)](#)

Step 3: Create AWS Lambda Event Source Mapping and Test the Setup

In this section, you create an Amazon DynamoDB table with a stream. You also create an event source mapping in AWS Lambda to associate the stream with your Lambda function. After you create the event source mapping, AWS Lambda will start polling the stream. You will then test the setup by performing table updates, which will cause DynamoDB to write event records to the stream. You then verify that AWS Lambda executed your Lambda function on your behalf and review the CloudWatch logs.

Step 3.1: Create a DynamoDB Table with a Stream

Using the following Java code, create a table, `exampletable`, with a stream. The code also performs three table actions (create an item, update the same item, and delete the item). Amazon DynamoDB publishes these actions to the stream.

After the code executes, note down the stream ARN. The code uses the `DescribeTable` API to get the ID. You will need the stream ARN when you create an event source mapping in AWS Lambda to associate the stream with a Lambda function.

For information about the AWS SDK for Java with DynamoDB Streams, go to [DynamoDB Streams Preview](#) in the *Amazon DynamoDB Developer Guide*.

```
package com.amazonaws.codesamples;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.log4j.PropertyConfigurator;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreamsClient;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamResult;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
```

AWS Lambda Developer Guide

Step 3: Create AWS Lambda Event Source Mapping and Test the Setup

```
import com.amazonaws.services.dynamodbv2.model.GetRecordsRequest;
import com.amazonaws.services.dynamodbv2.model.GetRecordsResult;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorRequest;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorResult;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.Record;
import com.amazonaws.services.dynamodbv2.model.ResourceInUseException;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import com.amazonaws.services.dynamodbv2.model.Shard;
import com.amazonaws.services.dynamodbv2.model.ShardIteratorType;
import com.amazonaws.services.dynamodbv2.model.TableDescription;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;

public class CreateStreamForLambdaConsumption {

    private static AmazonDynamoDBClient dynamoDBClient = new AmazonDynamoDBClient(
        new ProfileCredentialsProvider());

    private static AmazonDynamoDBStreamsClient streamsClient = new AmazonDynamoDBStreamsClient(
        new ProfileCredentialsProvider());

    public static void main(String args[]) {
        //PropertyConfigurator.configure("conf/log4j.properties");

        dynamoDBClient.setServiceNameIntern("dynamodb");

        streamsClient.setEndpoint("http://streams.preview-dynamodb.us-east-1.amazonaws.com");
        streamsClient.setServiceNameIntern("dynamodb");

        String tableName = "ExampleTableWithStream"; // Test table for streams.

        TableDescription tableDesc = createTableIfNecessary(tableName);

        // Determine the Streams settings for the table
        StreamSpecification myStreamSpec = tableDesc.getStreamSpecification();

        String myStreamId = tableDesc.getLatestStreamId();
        System.out.println("Current stream Id for " + tableName + ": " + myStreamId);
        System.out.println("Stream enabled: " + myStreamSpec.getStreamEnabled());

        System.out.println("Update view type: " + myStreamSpec.getStreamViewType());

        // Add a new item
        int numChanges = 0;
        System.out.println("Making some changes to table data");
        Map<String, AttributeValue> item = new HashMap<String, AttributeValue>();

        item.put("Id", new AttributeValue().withN("101"));
        item.put("Message", new AttributeValue().withS("New item!"));
        dynamoDBClient.putItem(tableName, item);
    }
}
```

AWS Lambda Developer Guide

Step 3: Create AWS Lambda Event Source Mapping and Test the Setup

```
numChanges++;

// Update the item
Map<String, AttributeValue> key = new HashMap<String, AttributeValue>();

key.put("Id", new AttributeValue().withN("101"));
Map<String, AttributeValueUpdate> attributeUpdates = new HashMap<String,
AttributeValueUpdate>();
attributeUpdates.put(
    "Message",
    new AttributeValueUpdate().withAction(AttributeAction.PUT)
        .withValue(
            new AttributeValue()
                .withS("This item has changed")));
dynamoDBClient.updateItem(tableName, key, attributeUpdates);
numChanges++;

// Delete the item
dynamoDBClient.deleteItem(tableName, key);
numChanges++;

// Get the shards in the stream

DescribeStreamResult describeStreamResult = streamsClient
    .describeStream(new DescribeStreamRequest()
        .withStreamId(myStreamId));
String streamId = describeStreamResult.getStreamDescription()
    .getStreamId();
List<Shard> shards = describeStreamResult.getStreamDescription()
    .getShards();

// Process each shard

for (Shard shard : shards) {
    String shardId = shard.getShardId();
    System.out.println("Processing " + shardId + " from stream "
        + streamId);

    // Get an iterator for the current shard
    GetShardIteratorRequest getShardIteratorRequest = new GetShardIter
atorRequest()
        .withStreamId(myStreamId)
        .withShardId(shardId)
        .withShardIteratorType(ShardIteratorType.TRIM_HORIZON);
    GetShardIteratorResult getShardIteratorResult = streamsClient.get
ShardIterator(getShardIteratorRequest);
    String nextItr = getShardIteratorResult.getShardIterator();

    while (nextItr != null && numChanges > 0) {
        // Use the iterator to read the data records from the shard
        GetRecordsResult getRecordsResult = streamsClient.getRecords(new
GetRecordsRequest().withShardIterator(nextItr));
        List<Record> records = getRecordsResult.getRecords();
        System.out.println("Getting records...");
        for (Record record : records) {
            System.out.println(record);
            numChanges--;
        }
    }
}
```

AWS Lambda Developer Guide
Step 3: Create AWS Lambda Event Source Mapping and
Test the Setup

```
        nextItr = getRecordsResult.getNextShardIterator();
    }
}

// Delete the table
// System.out.println("Deleting the table...");
// dynamoDBClient.deleteTable(tableName);

System.out.println("Demo complete");

}

private static TableDescription createTableIfNecessary(String tableName){
    // Check if the table is already available - either in ACTIVE or UPDATING
    state.
    boolean shouldCreateTable = true;
    try {
        DescribeTableResult descResult = dynamoDBClient.describeTable(tableName);
        if (!descResult.getTable().getTableStatus().equals("DELETING")) {
            System.out.println("Table already exists.");
            shouldCreateTable = false;
        }
    } catch (ResourceNotFoundException e) {
        // We will create the table now.
    }

    // If it is not, then create the table.
    while (shouldCreateTable) {
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<AttributeDefinition>();
        attributeDefinitions.add(new AttributeDefinition().
            withAttributeName("Id").withAttributeType("N"));

        ArrayList<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
        keySchema.add(new KeySchemaElement().withAttributeName("Id")
            .withKeyType(KeyType.HASH));

        StreamSpecification streamSpecification = new StreamSpecification();

        streamSpecification.setStreamEnabled(true);
        streamSpecification.setStreamViewType(StreamViewType.NEW_AND_OLD_IMAGES);

        CreateTableRequest createTableRequest = new CreateTableRequest()
            .withTableName(tableName)
            .withKeySchema(keySchema)
            .withAttributeDefinitions(attributeDefinitions)
            .withProvisionedThroughput(
                new ProvisionedThroughput().withReadCapacityUnits(1L)
                    .withWriteCapacityUnits(1L))
            .withStreamSpecification(streamSpecification);

        System.out.println("Issuing CreateTable request for " + tableName);

        try {
```

AWS Lambda Developer Guide

Step 3: Create AWS Lambda Event Source Mapping and Test the Setup

```
        dynamoDBClient.createTable(createTableRequest);
        shouldCreateTable = false;
    } catch (ResourceInUseException e) {
        // Possible if the table is being deleted.
    }
}

// Make sure that the table is ready to take writes.
while (true) {
    System.out.println("Waiting for the table to become ready.");
    try {
        DescribeTableResult descResult = dynamoDBClient.de
scribeTable(tableName);
        if (descResult.getTable().getTableStatus().equals("ACTIVE") ||

            descResult.getTable().getTableStatus().equals("UPDAT
ING")) {
            return descResult.getTable();
        }
    } catch (ResourceNotFoundException e) {
        // Possible for a little time.
    }
}
}
```

Step 3.2: Create Event Source Mapping in AWS Lambda

Run the following AWS CLI `create-event-source-mapping` command. After the command executes, note down the UUID. You'll need this UUID to refer to the event source mapping in any commands, for example, when deleting the event source mapping.

```
$ aws lambda create-event-source-mapping \
--region us-east-1 \
--function-name ProcessDynamoDBStream \
--event-source DynamoDB-stream-arn \
--batch-size 100 \
--starting-position TRIM_HORIZON \
--profile adminuser
```

Note

This creates a mapping between the specified DynamoDB stream and the Lambda function. You can associate a DynamoDB stream with multiple Lambda functions, and associate the same Lambda function with multiple streams. However, the Lambda functions will share the read throughput for the stream they share.

You can get list of event source mappings.

```
$ aws lambda list-event-source-mappings \
--region us-east-1 \
--function-name ProcessDynamoDBStream \
--event-source DynamoDB-stream-arn
```

In the response, you can verify the `LastProcessingResult` value is "OK" (you might see value "No records processed", instead of "OK", if AWS Lambda has not polled the stream at the time you ran this command).

Step 3.3: Test the Setup

You are all done! Now *adminuser* can test the setup as follows:

1. Add, update, or delete items in the `ExampleTableForStream` table.

You can run the Java code to execute the table operations. For each update, Amazon DynamoDB adds to the stream.

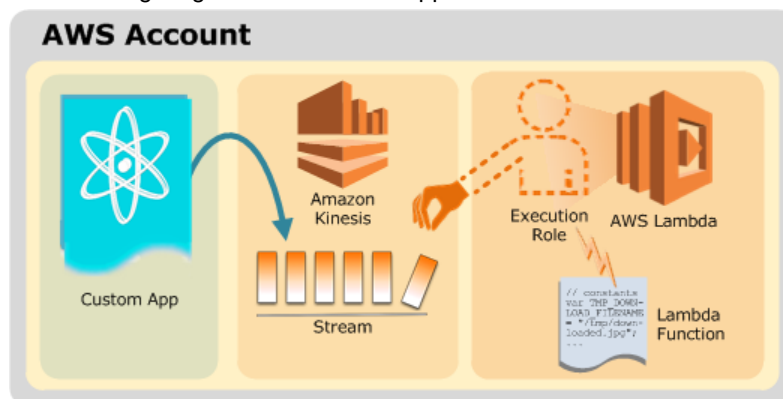
2. AWS Lambda polls the stream and when it detects updates to the stream, it will invoke your Lambda function by passing in event data it found in the stream.
3. Your function executes and creates logs in Amazon CloudWatch. The *adminuser* can also verify the logs reported in the Amazon CloudWatch console.

AWS Lambda Walkthrough 4: Processing Events from an Amazon Kinesis Stream (Using the AWS CLI)

In this walkthrough, you create a Lambda function to consume events from an Amazon Kinesis stream. The Lambda function is simple: it reads incoming event data and logs some of the information to Amazon CloudWatch.

This is an example of the "pull" model (see [AWS Lambda: How it Works \(p. 3\)](#)) where AWS Lambda polls the Amazon Kinesis stream and invokes your Lambda function when it detects new data on the stream. That is, in the pull model, AWS Lambda both invokes and executes the Lambda function.

The following diagram illustrates the application workflow:



In this walkthrough, you will do the following:

- Create a Lambda function to process Amazon Kinesis events
- Invoke a Lambda function manually using sample Amazon Kinesis event data
- Create an Amazon Kinesis stream
- Add an event source in AWS Lambda associating the stream and your Lambda function.

As soon as you add the event source, AWS Lambda starts polling the stream.

- Test the setup

You will then add a sample event record to the Amazon Kinesis stream and verify AWS Lambda executed your Lambda function on your behalf.

Important

Both the Lambda function and the Amazon Kinesis stream must be in the same AWS region. This exercise assumes the `us-west-2` (Oregon) region.

Note

In this walkthrough, you use the AWS Command Line Interface to perform AWS Lambda operations such as the create and invoke functions. This example is similar to the getting started example (see [Getting Started 3: Handling Amazon Kinesis Events \(Using the AWS Lambda Console\)](#) (p. 25)). The difference is that the getting started exercise provides a console-based experience. The console does many things for you, simplifying your experience. When you use the CLI, you get the raw experience of making the API calls, which can help you familiarize yourself with the AWS Lambda operations. In addition to creating and invoking Lambda function, you will explore other Lambda APIs.

Next Step

[Step 1: Preparing for the Walkthrough](#) (p. 60)

Step 1: Preparing for the Walkthrough

In this section, you do the following:

- If you don't have an administrator user in your account, create one. For instructions, see [Setup an AWS Account and Create an Administrator User](#) (p. 13).
- Set up the AWS CLI. For instructions, see [Step 1: Prepare for the Walkthrough](#) (p. 31).

You will use AWS CLI to perform the AWS Lambda activities, such as create a Lambda function, initially invoke it manually, and add an event source.

Next Step

[Step 2: Create a Lambda Function and Invoke it Manually Using Sample Event Data](#) (p. 60)

Step 2: Create a Lambda Function and Invoke it Manually Using Sample Event Data

Topics

- [Step 2.1: Create a Lambda Function](#) (p. 61)
- [Step 2.2: Invoke Your Lambda Function Manually](#) (p. 63)
- [Next Step](#) (p. 63)

In this section, you create a Lambda function and manually invoke it by passing sample Amazon Kinesis event.

Step 2.1: Create a Lambda Function

To create a Lambda function, you need to first create the following:

- A deployment package (a .zip file) containing your code and dependencies.

After you upload the deployment package to AWS Lambda, we refer it as your Lambda function. The code for this exercise is a Node.js example, and there are no dependencies.

- An IAM role (execution role).

At the time you create your Lambda function, you specify an IAM role that AWS Lambda can assume to execute the function on your behalf.

You must grant this execution role the necessary permissions. For example, AWS Lambda will need permission for Amazon Kinesis actions so it can poll the stream and read records from the stream. In the pull model, you must also grant AWS Lambda permission to invoke your Lambda function. The example Lambda function writes some of the event data to Amazon CloudWatch so your function will need permissions for the necessary Amazon CloudWatch actions.

For more information, see [Execution Permissions \(p. 7\)](#).

You provide both the deployment package and the IAM role when you create a Lambda function. You can also specify other configuration information such as the function name, memory size, runtime environment (nodejs) to use, and the handler. For more information about these parameters, see [CreateFunction \(p. 103\)](#).

After creating the Lambda function, you will invoke it using sample Amazon Kinesis event data.

Step 2.1.1: Create a Lambda Function Deployment Package

Follow the instructions to create AWS Lambda function deployment package.

1. Open a text editor, and copy the following code.

```
console.log('Loading function');

exports.handler = function(event, context) {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(function(record) {
    // Kinesis data is base64 encoded so decode here
    payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');

    console.log('Decoded payload:', payload);
  });
  context.succeed();
};
```

2. Save the file as `ProcessKinesisRecords.js`.
3. Zip the `ProcessKinesisRecords.js` file as `ProcessKinesisRecords.zip`.

Step 2.1.2: Create an IAM Role (execution role)

In this section, you create an IAM role using the following predefined role type and access policy:

- AWS service role of the "AWS Lambda" type. This role grants AWS Lambda permission to assume the role.

- "AWSLambdaKinesisExecutionRole" access policy that you attach to the role.

For more information about IAM roles, go to [Roles \(Delegation and Federation\)](#) in *Using IAM*. Use the following procedure to create the IAM role.

To create an IAM role (*executionrole*)

1. Sign in to the AWS Management Console.
2. Create an IAM role.

For instructions on creating the role, go to [Creating a Role for an AWS Service \(AWS Management Console\)](#) in *Using IAM*. As you follow the steps to create a role, note the following:

- In **Select Role Type**, click **AWS Service Roles**, and then select **AWS Lambda**. This will grant AWS Lambda service permission to assume the role.
- In **Attach Policy** select `AWSLambdaKinesisExecutionRole`.

3. Write down the role ARN. You will need it in the next step when you create your Lambda function.

Step 2.1.3: Create a Lambda Function

Execute the following Lambda CLI `create-function` command to create a Lambda function. You provide the deployment package and IAM role ARN as parameters.

```
$ aws lambda create-function \
--region us-west-2 \
--function-name ProcessKinesisRecords \
--zip-file fileb://file-path/ProcessKinesisRecords.zip \
--role execution-role-arn \
--handler ProcessKinesisRecords.handler \
--runtime nodejs \
--profile adminuser
```

For more information, see [CreateFunction \(p. 103\)](#). AWS Lambda creates the function and returns function configuration information as shown in the following example:

```
{
  "FunctionName": "ProcessKinesisRecords",
  "CodeSize": 412,
  "MemorySize": 128,
  "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:ProcessKinesisRecords",
  "Handler": "ProcessKinesisRecords.handler",
  "Role": "arn:aws:iam::account-id:role/kinesis-lambda-role",
  "Timeout": 3,
  "LastModified": "2015-04-02T01:20:42.355+0000",
  "Runtime": "nodejs",
  "Description": ""
}
```

Step 2.2: Invoke Your Lambda Function Manually

In this section, you invoke your Lambda function manually using the `invoke` CLI command.

Save the following JSON in a file, `input.txt`.

```
{
  "Records": [
    {
      "kinesis": {
        "partitionKey": "partitionKey-3",
        "kinesisSchemaVersion": "1.0",
        "data": "SGVsbG8sIHRobXMgaXMgYSB0ZXN0IDEyMy4=",
        "sequenceNumber":
"49545115243490985018280067714973144582180062593244200961"
      },
      "eventSource": "aws:kinesis",
      "eventID": "shardId-
000000000000:49545115243490985018280067714973144582180062593244200961",
      "invokeIdentityArn": "arn:aws:iam::059493405231:role/testLEBRole",
      "eventVersion": "1.0",
      "eventName": "aws:kinesis:record",
      "eventSourceARN": "arn:aws:kinesis:us-west-2:35667example:stream/ex
amplestream",
      "awsRegion": "us-west-2"
    }
  ]
}
```

Execute the following `invoke` command.

```
$ aws lambda invoke \
--invocation-type Event \
--function-name ProcessKinesisRecords \
--region us-west-2 \
--payload file://file-path/input.txt \
--profile adminuser
outputfile.txt
```

Note that if you request synchronous execution ("RequestResponse" as the invocation type), function returns the string message (message in the `context.succeed()` in the code) in the response body. In the preceding example it will be saved in `outputfile.txt`.

```
"Hello World"
```

Note

The Amazon Kinesis stream and Lambda function must be in the same AWS account. Cross-account scenario.

Next Step

[Step 3: Add an AWS Lambda Event Source and Test](#) (p. 64)

Step 3: Add an AWS Lambda Event Source and Test

Topics

- [Step 3.1: Create an Amazon Kinesis Stream \(p. 64\)](#)
- [Step 3.2: Add an Event Source in AWS Lambda \(p. 64\)](#)
- [Step 3.3: Test the Setup \(p. 65\)](#)

In this section, you create an Amazon Kinesis stream and add an event source in AWS Lambda to associate the stream with your Lambda function. After you create an event source, AWS Lambda will start polling the stream. You will then test the setup by adding events to the stream and verify that AWS Lambda executed your Lambda function on your behalf.

Step 3.1: Create an Amazon Kinesis Stream

Use the following Amazon Kinesis `create-stream` CLI command to create a stream.

```
$ aws kinesis create-stream \  
--stream-name examplestream \  
--shard-count 1 \  
--region us-west-2 \  
--profile adminuser
```

Run the following Amazon Kinesis `describe-stream` CLI command to get the stream ARN.

```
$ aws kinesis describe-stream \  
--stream-name examplestream \  
--region us-west-2 \  
--profile adminuser
```

You need the stream ARN in the next step to associate the stream with your Lambda function. The stream is of the form:

```
arn:aws:kinesis:aws-region:account-id:stream/stream-name
```

Step 3.2: Add an Event Source in AWS Lambda

Run the following AWS CLI `add-event-source` command. After the command executes, note down the UUID. You'll need this UUID to refer to the event source in any commands, for example, when deleting the event source.

```
$ aws lambda create-event-source-mapping \  
--region us-west-2 \  
--function-name ProcessKinesisRecords \  
--event-source kinesis-stream-arn \  
--batch-size 100 \  
--starting-position TRIM_HORIZON \  
--profile adminuser
```

Note

This creates a mapping between the specified Amazon Kinesis stream and the Lambda function. You can associate an Amazon Kinesis stream with only one Lambda function. If you associate another function with the same stream, it replaces the previous mapping.

You can get a list of event source mappings.

```
$ aws lambda list-event-source-mappings \  
--region us-west-2 \  
--function-name ProcessKinesisRecords \  
--event-source kinesis-stream-arn \  
--debug
```

In the response, you can verify the status value is "enabled".

Step 3.3: Test the Setup

You are all done! Now *adminuser* can test the setup as follows:

1. Using the following AWS CLI command, add event records to your Amazon Kinesis stream. The `--data` value is a base64-encoded value of the "Hello, this is a test." string. You can run the same command more than once to add multiple records to the stream.

```
$ aws kinesis put-record \  
--stream-name examplestream \  
--data "This is a test. final" \  
--partition-key shardId-00000000000000000000 \  
--region us-west-2 \  
--profile adminuser
```

2. AWS Lambda polls the stream and when it detects updates to the stream, it will invoke your Lambda function by passing in event data it found in the stream.

AWS Lambda assumes the execution role to poll the stream. You have granted the role permissions for necessary Amazon Kinesis actions, and therefore AWS Lambda can poll the stream and read events from the stream.

3. Your function executes and adds logs to the log group that corresponds to the Lambda function in Amazon CloudWatch.

The *adminuser* can also verify the logs reported in the Amazon CloudWatch console. Make sure you are checking for logs in the same AWS region where you created the Lambda function.

AWS Lambda Walkthrough 5: Handling AWS CloudTrail Events (Using the AWS CLI)

Scenario

Suppose you have turned on AWS CloudTrail for your AWS account to maintain records (logs) of AWS API calls made on your account. As API calls are made in your account, CloudTrail writes logs to an Amazon S3 bucket you configured. You want Amazon S3 to publish the "object created" events to AWS Lambda and invoke your Lambda function, as CloudTrail creates log objects.

When Amazon S3 invokes your Lambda function, it will pass an S3 event identifying, among other things, the bucket name and the new object key that CloudTrail created. So your Lambda function can read the log object, and it will know the API calls that were reported in the log.

You want your Lambda function to notify you via email if the log reports a specific API was called. Each CloudTrail is a JSON object with one or more event records. Each record, among other things, provides `eventSource` and `eventName`.

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        ...
      },
      "eventTime": "2014-12-16T19:17:43Z",
      "eventSource": "sns.amazonaws.com",
      "eventName": "CreateTopic",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "72.21.198.64",
      ...
    },
    {
      ...
    },
    ...
  ]
}
```

Your Lambda function will parse the log for records with specific `eventSource` ("sns.amazonaws.com") and `eventName` ("CreateTopic"). If found, it will publish the event to your Amazon SNS topic, which you will configure to send email.

Implementation Summary

Upon completing this walkthrough, you will have Amazon S3, AWS Lambda, Amazon SNS, and AWS Identity and Access Management (IAM) resources in your account:

Note

The walkthrough assumes you create these resources in the `us-west-2` (Oregon) region.

In Lambda:

- A Lambda function – The function first reads incoming S3 event data so it knows where the CloudTrail log object is created. It will then read that object and process as explained in the preceding section.
- An access policy – In the Lambda function's access policy, you will add a permission to allow Amazon S3 to invoke the Lambda function.

In IAM:

- Administrator user – Called *adminuser*.

You use the *adminuser* credentials to perform steps in this walkthrough.

- An IAM role (*executionrole*) – When you create this role, you will trust AWS Lambda to assume this role. You will also grant sufficient permissions that your Lambda function needs—for example, permissions for Amazon S3 actions to read objects from a bucket, permission for Amazon SNS actions to publish events, and permissions for CloudWatch actions to write logs.

In Amazon S3:

- A bucket – We refer to the bucket as *examplebucket*. When you turn the trail on in the CloudTrail console, you specify this bucket for CloudTrail to save the logs.
- Configure notification on *examplebucket*.

You will add notification configuration to your bucket to request Amazon S3 to publish object-created events to Lambda, by invoking your Lambda function. For more information about the Amazon S3 notification feature, go to [Setting Up Notification of Bucket Events](#).

- Sample CloudTrail log object, `ExampleCloudTrailLog.json`, in *examplebucket* bucket.

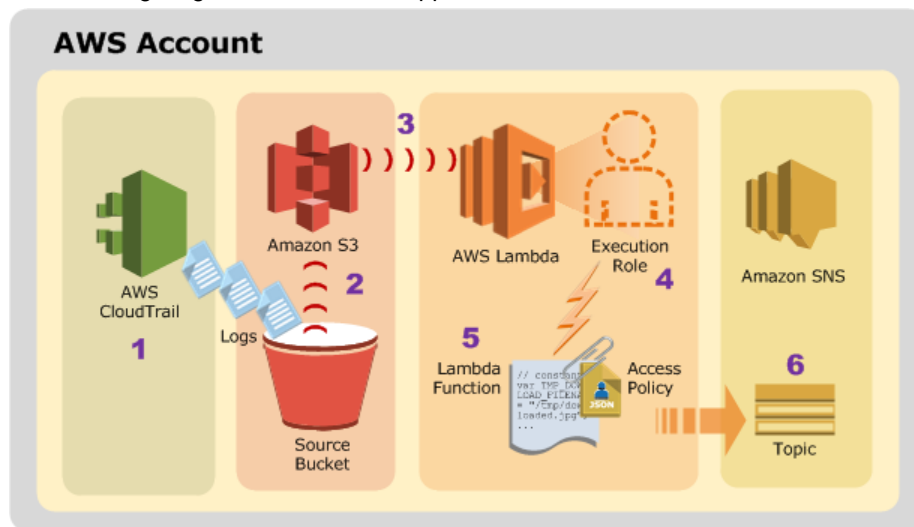
You will use a sample Amazon S3 event that will identify *examplebucket* and this object as a CloudTrail object. You will use this Amazon S3 object to first manually invoke your Lambda function. Your Lambda function will then read the sample CloudTrail log object and send you email notifications via an SNS topic.

In Amazon SNS

- An SNS topic.

You subscribe to this topic by specifying email as the protocol.

The following diagram illustrates the application flow:



1. AWS CloudTrail saves logs to an S3 bucket.
2. Amazon S3 detects the object-created event.
3. Amazon S3 publishes the `s3:ObjectCreated:*` event to AWS Lambda by invoking the Lambda function, per the bucket notification configuration.

Because the Lambda function's access policy includes a permission for Amazon S3 to invoke the function, Amazon S3 will be able to invoke the function.

4. Lambda assumes the execution role and executes the function.
5. The Lambda function first reads the Amazon S3 event it receives as a parameter and determines where the CloudTrail object is. It then reads the CloudTrail object and processes log records in it.
6. If the log includes a record with specific `eventType` and `eventSource` values, it publishes the event to your Amazon SNS topic.

In this walkthrough, you subscribe to the SNS topic using the email protocol, so you will get email notifications.

Now you are ready to try the steps.

First Walkthrough Step

[Step 1: Preparing for the Walkthrough](#) (p. 68)

Step 1: Preparing for the Walkthrough

In this section, you do the following (the example assumes you are creating your AWS resources—S3 bucket, SNS topic, and Lambda function—in the `us-west-2` (Oregon) region.):

- Create an administrator user, *adminuser*. For instructions, see [Setup an AWS Account and Create an Administrator User](#) (p. 13). If you already have an administrator user, you can skip this step.
- Set up the AWS CLI. If you followed the walkthroughs in order, you already have the AWS CLI setup. If not, follow the instructions provided in walkthrough 1. For more information, see [Step 1: Prepare for the Walkthrough](#) (p. 31).
- Turn CloudTrail on. For instructions, see the following section.
- Create an SNS topic. For instructions, see the following section.

Step 1.1: Turn on CloudTrail

In the AWS CloudTrail console, turn on the trail in your account by specifying *examplebucket* in the `us-west-2` (Oregon) region for CloudTrail to save logs. When configuring the trail, do not enable SNS notification.

For instructions, go to [Creating and Updating Your Trail](#) in the *AWS CloudTrail User Guide*.

Step 1.2: Create an SNS Topic and Subscribe to the Topic

Follow the procedure to create an SNS topic in the `us-west-2` (Oregon) region and subscribe to it by providing an email address as the endpoint.

To create and subscribe to a topic

1. Create an SNS topic.

For instructions, go to [Create a Topic](#) in the *Amazon Simple Notification Service Developer Guide*.

2. Subscribe to the topic by providing an email address as the endpoint.

For instructions, go to [Subscribe to a Topic](#) in the *Amazon Simple Notification Service Developer Guide*.

3. Note down the topic ARN. You will need the value in the following sections.

Next Step

[Step 2: Create and Invoke a Lambda Function](#) (p. 69)

Step 2: Create and Invoke a Lambda Function

In this section, the *adminuser* creates a function in AWS Lambda and invokes it manually, using the AWS CLI command, using sample Amazon S3 event data. For instructions, see the following sections:

[Step 2.1: Create a Lambda Function Deployment Package \(p. 69\)](#)

[Step 2.2: Create an IAM Role \(execution role\) \(p. 71\)](#)

[Step 2.3: Create Lambda Function \(p. 72\)](#)

[Step 2.4: Invoke Your Lambda Function Manually \(p. 73\)](#)

After you complete the steps, you can add a notification configuration on the *examplebucket* and test the end-to-end experience. For instructions, see the following section.

[Step 3: Configure Amazon S3 to Publish Events \(p. 76\).](#)

Step 2.1: Create a Lambda Function Deployment Package

The code example is created using Node.js. You first creates a folder to save the following:

- An example JavaScript function
- Dependencies

After you complete the steps, you will have the following folder structure:

```
CloudTrailEventProcessing.js
/node_modules/async
```

You will then zip the folder content; the .zip file is the Lambda function deployment package.

To create the Lambda function deployment package

1. Install the Node.js platform. For more information, go to nodejs.org.
2. Create a folder (examplefolder). After creating the folder, create a subfolder (node_modules) in it.
3. Open a command prompt, navigate to the examplefolder, and install the following libraries using the npm command, which is part of Node.js.

- async (Async utility module)

```
npm install async
```

4. Open a text editor, and copy the following code.

```
var aws = require('aws-sdk');
var zlib = require('zlib');
var async = require('async');

var EVENT_SOURCE_TO_TRACK = /sns.amazonaws.com/;
var EVENT_NAME_TO_TRACK = /CreateTopic/;
var DEFAULT_SNS_REGION = 'us-west-2';
var SNS_TOPIC_ARN = 'SNS Topic ARN';
```

```
var s3 = new aws.S3();
var sns = new aws.SNS({
  apiVersion: '2010-03-31',
  region: DEFAULT_SNS_REGION
});

exports.handler = function(event, context) {
  var srcBucket = event.Records[0].s3.bucket.name;
  var srcKey = event.Records[0].s3.object.key;

  async.waterfall([
    function fetchLogFromS3(next){
      console.log('Fetching compressed log from S3...');
      s3.getObject({
        Bucket: srcBucket,
        Key: srcKey
      },
      next);
    },
    function uncompressLog(response, next){
      console.log("Uncompressing log...");
      zlib.gunzip(response.Body, next);
    },
    function publishNotifications(jsonBuffer, next) {
      console.log('Filtering log...');
      var json = jsonBuffer.toString();
      console.log('CloudTrail JSON from S3:', json);
      var records;
      try {
        records = JSON.parse(json);
      } catch (err) {
        next('Unable to parse CloudTrail JSON: ' + err);
        return;
      }
      var matchingRecords = records
        .Records
        .filter(function(record) {
          return record.eventSource.match(EVENT_SOURCE_TO_TRACK)
            && record.eventName.match(EVENT_NAME_TO_TRACK);
        });

      console.log('Publishing ' + matchingRecords.length + ' notification(s) in parallel...');
      async.each(
        matchingRecords,
        function(record, publishComplete) {
          console.log('Publishing notification: ', record);
          sns.publish({
            Message:
              'Alert... SNS topic created: \n TopicARN=' +
              record.responseElements.topicArn + '\n\n' +
              JSON.stringify(record),
            TopicArn: SNS_TOPIC_ARN
          }, publishComplete);
        },
        next
      );
    }
  ]);
};
```

```
    }  
  ], function (err) {  
    if (err) {  
      console.error('Failed to publish notifications: ', err);  
    } else {  
      console.log('Successfully published all notifications.');    }  
    context.done(err);  
  });  
};
```

5. Update the code by providing your SNS topic ARN.
6. Save the file as `CloudTrailEventProcessing.js` in `examplefolder`.
7. Zip the folder content as `CloudTrailEventProcessing.zip`.

Note

You zip the folder content, not the folder itself.

This is your Lambda function deployment package.

Next Step

[Step 2.2: Create an IAM Role \(execution role\) \(p. 71\)](#)

Step 2.2: Create an IAM Role (execution role)

Now the *adminuser* is ready to create a Lambda function by uploading the deployment package to Lambda that you created in the previous step. But at the time of creating the function, *adminuser* will need to specify an IAM role that Lambda can assume. This role will have permissions the Lambda function needs to access your AWS resources. So let's first create an IAM role, *executionrole*. For more information about the *executionrole*, see [Execution Permissions \(p. 7\)](#).

Each IAM role has two policies. For the *executionrole*, you will use the following two policies:

- Trust policy specifying that AWS Lambda account principal who can assume the role.
- Access policy defining permissions for this role.

The role will have permission for Amazon S3 actions to perform operations on the *examplebucket*, CloudWatch actions so the function can write application logs to CloudWatch, and Amazon SNS actions so the function can publish events to your SNS topic.

To create an IAM role (*executionrole*)

1. Sign in to the AWS Management Console using the *adminuser* credentials.
2. Create a managed policy that you will attach to the IAM role.
 - a. In the navigation pane of the IAM console, click **Policies**, and then click **Create Policy**.
 - b. Next to **Copy an AWS Managed Policy**, click **Select**.
 - c. Next to **AWSLambdaExecute**, click **Select**.
 - d. Copy the following policy into the **Policy Document** replacing the existing policy, and then update the policy with the ARN of the Amazon SNS topic you created.

Note the policy name because you will use it in the next step.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:*"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": "your sns topic ARN"
    }
  ]
}
```

3. Create an IAM role, the `executionrole`, and attach the policy you just created to the role.

For instructions on creating the role, go to [Creating a Role for an AWS Service \(AWS Management Console\)](#) in *Using IAM*. Note the following:

- In **Select Role Type**, click **AWS Service Roles**, and then select **AWS Lambda**.
- In **Attach Policy**, select the policy you created in the previous step.

Next Step

[Step 2.3: Create Lambda Function \(p. 72\)](#)

Step 2.3: Create Lambda Function

Now you are ready to upload the deployment package to create Lambda function. You will also test invoke the function. At this point, you have not configured notification on your S3 bucket, so Amazon S3 will not invoke your function. But you can test your Lambda function by manually invoking it, which requires the following:

- Sample S3 event data identify bucket name and an object key.
- Sample CloudTrail log object in the S3 bucket.

You want this sample log to include a record that has the `eventType` value set to `"ec2.amazonaws.com"` and the `eventName` value set to `"CreateSecurityGroup"`. The Lambda function looks for event records with these values.

Follow the steps to test invoke your Lambda function.

1. Create a Lambda function.

At the command prompt, run the following Lambda CLI `create-function` command using the `adminuser` profile. You will need to update the command by providing the .zip file path and the "executionrole" ARN.

```
$ aws lambda create-function \
--region us-west-2 \
--function-name CloudTrailEventProcessing \
--zip-file fileb://file-path/CloudTrailEventProcessing.zip \
--role execution-role-arn \
--handler CloudTrailEventProcessing.handler \
--runtime nodejs \
--profile adminuser \
--timeout 10 \
--memory-size 1024
```

2. Note down the Lambda function ARN. You will need this ARN when you configure a bucket notification in the next section.

Next Step

[Step 2.4: Invoke Your Lambda Function Manually \(p. 73\)](#)

Step 2.4: Invoke Your Lambda Function Manually

In this section, you invoke your Lambda function manually using sample Amazon S3 event data that you will pass to the Lambda function. When Lambda function executes, it will read the S3 object (a sample CloudTrail log) from the bucket identified in the S3 event data, and publish an event to your SNS topic if the sample CloudTrail log reports use of a specific API. For this walkthrough, the API is the SNS API used to create a topic. That is, the CloudTrail log reports a record identifying "sns.amazonaws.com" as the `eventSource`, and "CreateTopic" as the `eventName`.

1. **Prepare to invoke the Lambda function manually**

- a. Save the following JSON (an example S3 event) in a file, `input.txt`.

You will provide this sample event when you invoke your Lambda function. For more information about the S3 event structure, go to [Event Message Structure](#) in the *Amazon Simple Storage Service Developer Guide*.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-west-2",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AIDAJDPLRKLG7UEXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      }
    }
  ]
}
```

```
    },
    "responseElements": {
      "x-amz-request-id": "C3D13FE58DE4C810",
      "x-amz-id-2": "FMMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JR
WeUWerMUE5JgHvANOjpD"
    },
    "s3": {
      "s3SchemaVersion": "1.0",
      "configurationId": "testConfigRule",
      "bucket": {
        "name": "your bucket name",
        "ownerIdentity": {
          "principalId": "A3NL1KOZZKExample"
        },
        "arn": "arn:aws:s3:::mybucket"
      },
      "object": {
        "key": "ExampleCloudTrailLog.json.gz",
        "size": 1024,
        "eTag": "d41d8cd98f00b204e9800998ecf8427e",
        "versionId": "096fKKXTRTtl3on89fVO.nfljtsv6qko"
      }
    }
  }
}
```

b. **Upload a sample CloudTrail log to your *examplebucket*.**

- i. Save the following sample CloudTrail log to a file (ExampleCloudTrailLog.json).

Note that one of events in this log has "sns.amazonaws.com" as the `eventSource` and "CreateTopic" as the `eventName`. Your Lambda function reads the logs and if it finds event of this type, it publishes the event to the Amazon SNS topic you created and you will receive one email when you invoke the Lambda function manually.

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "Root",
        "principalId": "account-id",
        "arn": "arn:aws:iam::account-id:root",
        "accountId": "account-id",
        "accessKeyId": "access-key-id",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-01-24T22:41:54Z"
          }
        }
      },
      "eventTime": "2015-01-24T23:26:50Z",
      "eventSource": "sns.amazonaws.com",
      "eventName": "CreateTopic",
      "awsRegion": "us-west-2",

```

```
        "sourceIPAddress": "205.251.233.176",
        "userAgent": "console.amazonaws.com",
        "requestParameters": {
            "name": "dropmeplease"
        },
        "responseElements": {
            "topicArn": "arn:aws:sns:us-west-2:account-id:exampletopic"
        },
        "requestID": "3fdb7834-9079-557e-8ef2-350abc03536b",
        "eventID": "17b46459-dada-4278-b8e2-5a4ca9ffa9c",
        "eventType": "AwsApiCall",
        "recipientAccountId": "account-id"
    },
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "Root",
            "principalId": "account-id",
            "arn": "arn:aws:iam::account-id:root",
            "accountId": "account-id",
            "accessKeyId": "access-key-id",
            "sessionContext": {
                "attributes": {
                    "mfaAuthenticated": "false",
                    "creationDate": "2015-01-24T22:41:54Z"
                }
            }
        },
        "eventTime": "2015-01-24T23:27:02Z",
        "eventSource": "sns.amazonaws.com",
        "eventName": "GetTopicAttributes",
        "awsRegion": "us-west-2",
        "sourceIPAddress": "205.251.233.176",
        "userAgent": "console.amazonaws.com",
        "requestParameters": {
            "topicArn": "arn:aws:sns:us-west-2:account-id:exampletopic"
        },
        "responseElements": null,
        "requestID": "4a0388f7-a0af-5df9-9587-c5c98c29cbe",
        "eventID": "ec5bb073-8fa1-4d45-b03c-f07b9fc9ea18",
        "eventType": "AwsApiCall",
        "recipientAccountId": "account-id"
    }
  ]
}
```

- ii. Run the `gzip` command to create `.gz` file from the preceding source file.

```
$ gzip ExampleCloudTrailLog.json
```

This creates `ExampleCloudTrailLog.json.gz` file.

- iii. Upload the `ExampleCloudTrailLog.json.gz` file to your bucket that you specified in the CloudTrail configuration.

This object is specified in the sample Amazon S3 event data that we use in manually invoking the Lambda function.

2. Invoke the Lambda function.

Execute the following CLI command to invoke the function manually using *adminuser* profile.

```
$ aws lambda invoke-async \
  --function-name CloudTrailEventProcessing \
  --region us-west-2 \
  --invoke-args /filepath/input.txt \
  --debug \
  --profile adminuser
```

Because your example log object has an event record showing the SNS API to call to create a topic, the Lambda function will post that event to your SNS topic, and you should get an email notification.

You can monitor the activity of your Lambda function by using CloudWatch metrics and logs. For more information about CloudWatch monitoring, see [Troubleshooting and Monitoring AWS Lambda Functions with Amazon CloudWatch \(p. 79\)](#).

Next Step

[Step 3: Configure Amazon S3 to Publish Events \(p. 76\)](#)

Step 3: Configure Amazon S3 to Publish Events

In this section, you add the remaining configuration so Amazon S3 can publish object-created events to AWS Lambda and invoke your Lambda function. You will do the following:

- Add permission to the Lambda function's access policy to allow Amazon S3 to invoke the function.
- Add notification configuration to your source bucket. In the notification configuration, you provide:
 - The event type for which you want Amazon S3 to publish events. For this exercise, you will specify the `s3:ObjectCreated:*` event type.
 - Lambda function to invoke.

Add permission to the Lambda Function's Access Policy

1. Run the following Lambda CLI `add-permission` command to grant Amazon S3 service principal ("s3.amazonaws.com") permission for the `lambda:InvokeFunction` action. Note that the permission is granted with the following conditions:
 - Amazon S3 can invoke the function only if an object-created event is detected on a specific bucket.
 - The bucket is owned by a specific AWS account. If a bucket owner deletes a bucket, some other AWS account can create a bucket with the same name. This condition ensures that only a specific AWS account can invoke your Lambda function.

```
$ aws lambda add-permission \
  --function-name CloudTrailEventProcessing \
  --region us-west-2 \
```

```
--statement-id Id-1 \  
--action "lambda:InvokeFunction" \  
--principal s3.amazonaws.com \  
--source-arn arn:aws:s3:::examplebucket \  
--source-account examplebucket-owner-account-id \  
--profile adminuser
```

2. Verify the access policy of your function by calling the CLI `get-policy` command.

```
$ lambda get-policy \  
--function-name function-name \  
--profile adminuser
```

Step 3.2: Add Notification Configuration to Your Bucket

Add notification configuration on the `examplebucket` to request Amazon S3 to publish object-created events to Lambda. In the configuration, you specify the following:

- Event type — For this exercise, these can be any event types that create objects.
- Lambda function ARN – This is your Lambda function that Amazon S3 will invoke. The ARN is of the following form:

```
arn:aws:lambda:aws-region:account-id:function:function-name
```

For example, the function `CloudTrailEventProcessing` created in `us-west-2` region will have this ARN:

```
arn:aws:lambda:us-west-2:account-id:function:CloudTrailEventProcessing
```

For instructions on adding notification configuration to a bucket, go to [Enabling Event Notifications](#) in the *Amazon Simple Storage Service Console User Guide*.

Step 3.3: Test the Setup

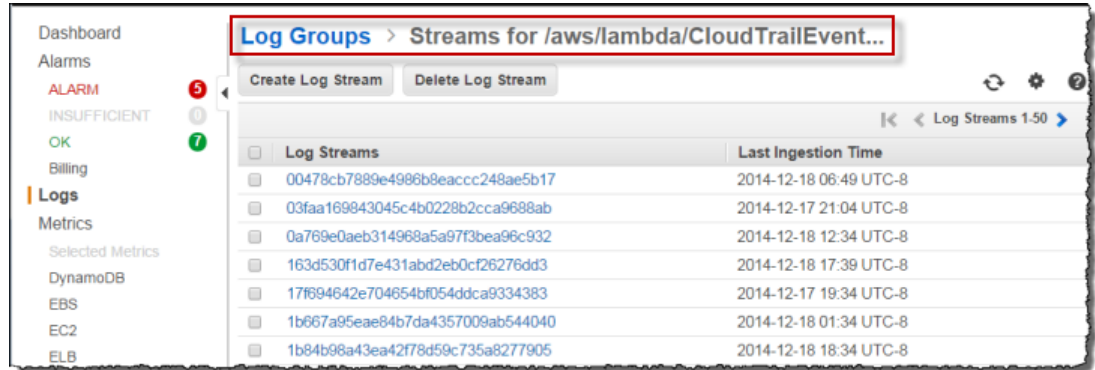
You are all done! You can now test the setup as follows:

1. Perform some action in your AWS account. For example, add another topic in the Amazon SNS console.
2. You should get an email notification about this event. You also notice the following:
 - AWS CloudTrail creates a log object in your bucket.
 - If you open the log object (.gz file), the log shows the `CreateTopic` SNS event.
 - For each object AWS CloudTrail creates, Amazon S3 invokes your Lambda function by passing in the log object as event data.
 - Lambda executes your function. The function parses the log, and it finds a `CreateTopic` SNS event, sends you an email notification.

You can monitor the activity of your Lambda function by using CloudWatch metrics and logs. For more information about CloudWatch monitoring, see [Troubleshooting and Monitoring AWS Lambda Functions with Amazon CloudWatch](#) (p. 79).

AWS Lambda Developer Guide

Step 3: Configure Amazon S3 to Publish Events



Troubleshooting and Monitoring AWS Lambda Functions with Amazon CloudWatch

AWS Lambda automatically monitors Lambda functions on your behalf, reporting metrics through Amazon CloudWatch. To help you monitor your code as it executes, Lambda automatically tracks the number of requests, the latency per request, and the number of requests resulting in an error and publishes the associated CloudWatch metrics. You can leverage these metrics to set CloudWatch custom alarms. For more information about CloudWatch, see the [Amazon CloudWatch Developer Guide](#).

You can view request rates and error rates for each of your Lambda functions by using the AWS Lambda console, the CloudWatch console, and other Amazon Web Services (AWS) resources. The following topics describe Lambda CloudWatch metrics and how to access them.

- [Accessing Amazon CloudWatch Metrics for AWS Lambda \(p. 81\)](#)
- [AWS Lambda Metrics \(p. 83\)](#)

You can insert logging statements into your code to help you validate if your code is working as expected. Lambda automatically integrates with Amazon CloudWatch Logs and pushes all logs from your code to a CloudWatch Logs group associated with a Lambda function (Lambda/<function name>). To learn more about log groups and accessing them through the CloudWatch console, see the [Monitoring System, Application, and Custom Log Files](#) in the *Amazon CloudWatch Developer Guide*.

The following topic describes how to access CloudWatch log entries.

- [Accessing Amazon CloudWatch Logs for AWS Lambda \(p. 82\)](#)

Note

If your Lambda function code is executing but you don't see any log data being generated after several minutes, this could mean your *execution role* for the Lambda function did not grant permissions to write log data to CloudWatch Logs. For information about how to make sure that you have set up the *execution role* correctly to give these permissions, see [Execution Permissions \(p. 7\)](#).

AWS Lambda Troubleshooting Scenarios

This section describes examples of how to monitor and troubleshoot your Lambda functions using the logging and monitoring capabilities of CloudWatch.

Troubleshooting Scenario 1: Lambda function not working as expected

In this scenario, you have just finished [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\)](#) (p. 38). However, the Lambda function you created to upload a thumbnail image to Amazon S3 when you create an S3 object is not working as expected. When you upload objects to Amazon S3, you see that the thumbnail images are not being uploaded. You can troubleshoot this issue in the following ways.

To determine why your Lambda function is not working as expected

1. Check if your code is working correctly. An increased error rate would indicate that it is not.

You can test your code locally as you would any other Node.js function, or you can test it within Lambda by using the "test invoke" capability on the console, or you can use the AWS CLI `asyncInvoke` command. Each time the code is executed in response to an event, it writes a log entry into the log group associated with a Lambda function, which is `Lambda/<function name>`.

The following are some errors that might show up in the logs:

- If you see a stack trace in your log, there is probably an error in your code. Review your code and debug the error the stack trace refers to.
- If you see a "permissions denied" error in the log, the IAM role you have provided as an *execution role* may not have the necessary permissions. Check if the IAM role has all the necessary permissions to access any AWS resources your code references. To ensure that you have correctly set up the *execution role*, see [Execution Permissions](#) (p. 7).
- If you see a "timeout exceeded" error in the log, your timeout setting exceeds the run time of your function code. This may be because the timeout is too low, or the code is taking too long to execute.
- If you see a "memory exceeded" error in the log, your memory setting is too low. Set it to a higher value, up to 1024 MB. Changing the memory setting can change how you are charged for duration. For information about pricing, see [AWS Lambda](#).

2. Check if your Lambda function is receiving requests.

Even if your function code is working as expected and responding correctly to test invokes, the function may not be receiving requests from Amazon S3. If Amazon S3 is able to invoke the function, you should see an increase in your CloudWatch requests metric. If you do not see an increase in your CloudWatch requests, check the access policy associated with the function.

Troubleshooting Scenario 2: Increased latency in Lambda function execution

In this scenario, you have just finished [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\)](#) (p. 38). However, the Lambda function you created to upload a thumbnail image to Amazon S3 when you create an S3 object is not working as expected. When you upload objects to Amazon S3, you can see that the thumbnail images are being uploaded, but your code is taking much longer to execute than expected. You can troubleshoot this issue in a couple of different ways. For example, you could

monitor the "latency" CloudWatch metric for the Lambda function to see if the latency is increasing. Or you could see an increase in the "errors" CloudWatch metric for the Lambda function, which might be due to timeout errors.

To determine why there is increased latency in the execution of a Lambda function

1. Test your code with different memory settings.

If your code is taking too long to execute, it could be that it does not have enough compute resources to execute its logic. Try increasing the memory allocated to your function and testing the code again, using the Lambda console test invoke functionality. You can see the memory used, code execution time, and memory allocated in the function log entries. Changing the memory setting can change how you are charged for duration. For information about pricing, see [AWS Lambda](#).

2. See where the execution bottleneck is using logs.

You can test your code locally as you would any other Node.js function, or you can test it within Lambda using the "test invoke" capability on the Lambda console, or the `asyncInvoke` command by using AWS CLI. Each time the code is executed in response to an event, it writes a log entry into the log group associated with a Lambda function, which is named (Lambda/<function name>). Add logging statements around various parts of your code, such as callouts to other services, to see how much time is being spent in executing different parts of your code.

Accessing Amazon CloudWatch Metrics for AWS Lambda

AWS Lambda automatically monitors functions on your behalf, reporting metrics through Amazon CloudWatch. These metrics include total requests, latency, and error rates. For more information about Lambda metrics, see [AWS Lambda Metrics \(p. 83\)](#). For more information about CloudWatch, see the [Amazon CloudWatch Developer Guide](#).

You can monitor metrics for Lambda and view logs by using the Lambda console, the CloudWatch console, the AWS CLI, or the CloudWatch API. The following procedures show you how to access metrics using these different methods.

To access metrics using the Lambda console

1. Sign in to the AWS Management Console and open the Lambda console.
2. If you have not created a Lambda function before, go to [Getting Started with AWS Lambda \(p. 16\)](#).
3. On the **Lambda: Function List** page, click the radio button or the triangle to the left of a function name to expand the function details.



A graphical representation of the metrics for the Lambda function are shown.

4. Click **CloudWatch logs** next to a metric name to view the log for the metric.

To access metrics using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. From the navigation bar, select a region.
3. In the navigation pane, click **Metrics**.
4. In the **CloudWatch Metrics by Category** pane, select **Lambda Metrics**.
5. (Optional) In the graph pane, select a statistic and a time period, and then create a CloudWatch alarm using these settings.

To access metrics using the AWS CLI

Use the [list-metrics](#) and [get-metric-statistics](#) commands.

To access metrics using the CloudWatch CLI

Use the [mon-list-metrics](#) and [mon-get-stats](#) commands.

To access metrics using the CloudWatch API

Use the [ListMetrics](#) and [GetMetricStatistics](#) operations.

Accessing Amazon CloudWatch Logs for AWS Lambda

AWS Lambda automatically monitors Lambda functions on your behalf, reporting metrics through Amazon CloudWatch. To help you troubleshoot failures in a function, Lambda logs all requests handled by your function and also automatically stores logs generated by your code through Amazon CloudWatch Logs.

You can insert logging statements into your code to help you validate if your code is working as expected. Lambda automatically integrates with CloudWatch Logs and pushes all logs from your code to a

CloudWatch Logs group associated with a Lambda function, which is named Lambda/<function name>. To learn more about log groups and accessing them through the CloudWatch console, see the [Monitoring System, Application, and Custom Log Files](#) in the *Amazon CloudWatch Developer Guide*.

You can view logs for Lambda by using the Lambda console, the CloudWatch console, the AWS CLI, or the CloudWatch API. The following procedure show you how to view the logs by using the Lambda console.

To view logs using the Lambda console

1. Sign in to the AWS Management Console and open the Lambda console.
2. If you have not created a Lambda function before, go to [Getting Started with AWS Lambda \(p. 16\)](#).
3. On the **Lambda: Function List** page, click the radio button or the triangle to the left of a function name to expand the function details.



A graphical representation of the metrics for the Lambda function are shown.

4. Click **logs** next to a metric name to view the log for the metric.

For more information on accessing CloudWatch Logs, see the following guides:

- [Amazon CloudWatch Developer Guide](#)
- [Amazon CloudWatch Logs API Reference](#)
- [CloudWatch Logs section of the AWS CLI Reference](#)

AWS Lambda Metrics

This topic describes the AWS Lambda namespace, metrics, and dimensions. AWS Lambda automatically monitors functions on your behalf, reporting metrics through Amazon CloudWatch (CloudWatch). These metrics include total invocations, errors, duration, and throttles.

CloudWatch is basically a metrics repository. A metric is the fundamental concept in CloudWatch and represents a time-ordered set of data points. You or AWS products publish metric data points into CloudWatch and you retrieve statistics about those data points as an ordered set of time-series data.

Metrics are uniquely defined by a name, a namespace, and one or more dimensions. Each data point has a time stamp, and (optionally) a unit of measure. When you request statistics, the returned data

stream is identified by namespace, metric name, and dimension. For more information about CloudWatch, see the [Amazon CloudWatch Developer Guide](#).

Metrics

The AWS Lambda namespace for CloudWatch is **AWS/Lambda**.

The following metrics are available from AWS Lambda.

Metric	Description
Invocations	<p>Measures the number of times a function is invoked in response to an event or invocation API call. This replaces the deprecated RequestCount metric. This includes successful and failed invocations, but does not include throttled attempts. This equals the billed requests for the function. Note that AWS Lambda only sends these metrics to CloudWatch if they have a nonzero value.</p> <p>Units: Count</p>
Errors	<p>Measures the number of invocations that failed due to errors in the function (response code 4XX). This replaces the deprecated ErrorCount metric. Failed invocations may trigger a retry attempt that succeeds. This includes:</p> <ul style="list-style-type: none">• Handled exceptions (e.g., context.fail(error))• Unhandled exceptions causing the code to exit• Out of memory exceptions• Timeouts• Permissions errors <p>This does not include invocations that fail due to invocation rates exceeding default concurrent limits (error code 429) or failures due to internal service errors (error code 500).</p> <p>Units: Count</p>
Duration	<p>Measures the elapsed wall clock time from when the function code starts executing as a result of an invocation to when it stops executing. This replaces the deprecated Latency metric. The maximum data point value possible is the function timeout configuration. The billed duration will be rounded up to the nearest 100 millisecond. Note that AWS Lambda only sends these metrics to CloudWatch if they have a nonzero value.</p> <p>Units: Milliseconds</p>
Throttles	<p>Measures the number of Lambda function invocation attempts that were throttled due to invocation rates exceeding the customer's concurrent limits (error code 429). Failed invocations may trigger a retry attempt that succeeds.</p> <p>Units: Count</p>

Dimensions

You can use the dimensions in the following table to refine the metrics returned for your Lambda functions.

Dimension	Description
FunctionName	Filters the data you request for a Lambda function.

Logging AWS Lambda API Calls By Using AWS CloudTrail

AWS Lambda is integrated with AWS CloudTrail, a service that captures API calls made by or on behalf of AWS Lambda in your AWS account and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls made from the AWS Lambda console or from the AWS Lambda API. Using the information collected by CloudTrail, you can determine what request was made to AWS Lambda, the source IP address from which the request was made, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

AWS Lambda Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to AWS Lambda actions are tracked in log files. AWS Lambda records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

The following actions are supported:

- [AddPermission](#) (p. 96)
- [CreateEventSourceMapping](#) (p. 99)
- [CreateFunction](#) (p. 103)
 - The `ZipFile` parameter is omitted from the CloudTrail logs for `CreateFunction`.
- [DeleteEventSourceMapping](#) (p. 108)
- [DeleteFunction](#) (p. 110)
- [GetEventSourceMapping](#) (p. 112)
- [GetFunction](#) (p. 114)
- [GetFunctionConfiguration](#) (p. 116)
- [GetPolicy](#) (p. 119)
- [ListEventSourceMappings](#) (p. 126)
- [ListFunctions](#) (p. 128)
- [RemovePermission](#) (p. 130)
- [UpdateEventSourceMapping](#) (p. 132)

- [UpdateFunctionCode](#) (p. 135)
 - The `ZipFile` parameter is omitted from the CloudTrail logs for `UpdateFunctionCode`.
- [UpdateFunctionConfiguration](#) (p. 138)

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the **userIdentity** field in the [CloudTrail Event Reference](#).

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

You can choose to have CloudTrail publish Amazon SNS notifications when new log files are delivered if you want to take quick action upon log file delivery. For more information, see [Configuring Amazon SNS Notifications](#).

You can also aggregate AWS Lambda log files from multiple AWS regions and multiple AWS accounts into a single S3 bucket. For more information, see [Aggregating CloudTrail Log Files to a Single Amazon S3 Bucket](#).

Understanding AWS Lambda Log File Entries

CloudTrail log files contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they are not an ordered stack trace of the public API calls.

The following example shows CloudTrail log entries for the `GetFunction` and `DeleteFunction` actions.

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::999999999999:user/myUserName",
        "accountId": "999999999999",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:03:36Z",
      "eventSource": "lambda.amazonaws.com",
      "eventName": "GetFunction",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "Python-httpplib2/0.8 (gzip)",
      "errorCode": "AccessDenied",
      "errorMessage": "User: arn:aws:iam::999999999999:user/myUserName is not authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-west-2:999999999999:function:other-acct-function",
      "requestParameters": null,
      "responseElements": null,
    }
  ]
}
```

```
"requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
"eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
"eventType": "AwsApiCall",
"recipientAccountId": "999999999999"
},
{
  "eventVersion": "1.03",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "A1B2C3D4E5F6G7EXAMPLE",
    "arn": "arn:aws:iam::999999999999:user/myUserName",
    "accountId": "999999999999",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "myUserName"
  },
  "eventTime": "2015-03-18T19:04:42Z",
  "eventSource": "lambda.amazonaws.com",
  "eventName": "DeleteFunction",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "Python-httpplib2/0.8 (gzip)",
  "requestParameters": {
    "functionName": "basic-node-task"
  },
  "responseElements": null,
  "requestID": "a2198ecc-cda1-11e4-aaa2-e356da31e4ff",
  "eventID": "20b84ce5-730f-482e-b2b2-e8fcc87ceb22",
  "eventType": "AwsApiCall",
  "recipientAccountId": "999999999999"
}
]
```

Best Practices for Working with AWS Lambda Functions

The following are recommended best practices for using AWS Lambda.

- Write your Lambda function code in a stateless style, and ensure there is no affinity between your code and the underlying compute infrastructure.
- Lower costs and improve performance by minimizing the use of 'startup' code not directly related to processing the current event.
- Use the standard (default) version of libraries to minimize deployment and startup overhead.
- Use the built-in CloudWatch monitoring of your Lambda functions to view and optimize request latencies.
- Delete old Lambda functions that you are no longer using.

AWS Lambda Limits

Resources such as local disk space and maximum file descriptors are handled by fixed limits that apply to all AWS Lambda functions regardless of their memory allocation. Functions that exceed a given limit will fail with an “exceeded limits” exception.

The following is a list of runtime resource limits for a Lambda function.

Resource	Limit
Ephemeral disk capacity (“/tmp” space)	512 MB
Number of file descriptors	1,024
Number of processes and threads (combined total)	1,024
Concurrent requests per account	100
Requests per second per account	1,000
Maximum execution duration per request	60 seconds

AWS Lambda also has the following input limits.

Input	Limit
Compressed function .zip file	50 MB
Uncompressed function .zip file	250 MB
Invoke (p. 121) request body JSON	6 MB
Inactive code storage retention period	90 days
Code storage limit	1.5 GB

Increasing AWS Lambda Limits

You can request a limit increase for concurrent requests and requests per second for your account. The other AWS Lambda limits cannot be changed at this time.

To request a limit increase for concurrent requests and requests per second

1. Go to [Support Center](#).
2. Click **Create case**.
3. Fill out the case form regarding a service limit increase for limit type Lambda.

Appendix: API Updates

From Preview to General Availability

As part of making the service generally available, the AWS Lambda API and programming model have been updated to address customer feedback. If you are a customer who has been using Lambda before April 9, 2015, the following is a useful reference to understand what changes need to be made in order to use the new API and permissions model.

Task	Previous	New	Migration guidance for existing users
Permissions			
Setting invoke permissions for Amazon S3	Set the “invocationrole” parameter for function properties using <code>CreateFunction</code> .	Create permissions policy using <code>AddPermission</code> API for a given function. <code>Invocationrole</code> is deprecated.	If you use the Amazon S3 console, anytime you create a new notification rule, editing an existing rule, or deleting a rule, the console will look at all the existing rules on the same bucket and convert any existing Lambda rules as needed (remove invocation roles and automatically add permissions). If you use the Amazon S3 API for editing or updating the event source, you will need to first set permissions using <code>Lambda AddPermissions</code> , and then create the notification role.

Task	Previous	New	Migration guidance for existing users
Setting invoke permissions for Amazon Kinesis	Set the “invocationrole” parameter for function properties using <code>AddEventSource</code> API.	Add appropriate permissions to the function execution role: "kinesis:GetRecords", "kinesis:GetShardIterator", "kinesis:DescribeStream", "kinesis:ListStreams"	<ul style="list-style-type: none"> • <u>New event source mapping with a new function</u> - When you create a new function, you must ensure the execution role has the necessary permissions to read from your event source. If you use the AWS Lambda console, you can use the 1-Click role creation option to automatically create a role with the necessary permissions for a given code template. You can customize the permissions on the execution role by using the IAM console (Roles -> Click on desired role -> scroll to inline policies -> edit policy). • <u>New event source mapping with an existing function</u> - Before setting up a new event source on an existing function, you need to update your execution role with stream read permissions. You can customize the permissions on the execution role by using the IAM console (Roles -> Click on desired role -> scroll to inline policies -> edit policy). If you are using the AWS Lambda console, you can update your execution role with the additional policy for reading and writing from Streams. • <u>Existing event source mapping with an existing function</u> - Existing event source mappings will continue to use the invocation role they were configured with. However, if you modify the event source through the Lambda CLI, you will need to remove the invocation role and update the execution role.
Managing functions			
Invoking a function in “Event” mode (asynchronous response)	Use <code>InvokeAsync</code> API	Use <code>Invoke</code> API with “event” <code>InvocationType</code>	Update any references to <code>InvokeAsync</code> to use <code>Invoke</code> . <code>InvokeAsync</code> is deprecated.
Creating functions programmatically	Use <code>UploadFunction</code> API	Use <code>CreateFunction</code> API	Update any references to <code>UploadFunction</code> to use <code>CreateFunction</code>
Updating function code programmatically	Use <code>UploadFunction</code> API	Use <code>UpdateFunctionCode</code> API	Update any references to <code>UploadFunction</code> to use <code>UpdateFunctionCode</code>
Managing event sources			

Task	Previous	New	Migration guidance for existing users
Creating event sources (stream to function mappings) programmatically	Use <code>AddEventSource</code> API	Use <code>AddEventSourceMapping</code>	Update any references to <code>AddEventSource</code> to use <code>AddEventSourceMapping</code> when used to create an event source
Updating event sources (stream to function mappings) programmatically	Use <code>AddEventSource</code> API	Use <code>UpdateEventSourceMapping</code>	Update any references to <code>AddEventSource</code> to use <code>UpdateEventSourceMapping</code> when used to update an event source
Retrieve single event source	<code>GetEventSource</code>	<code>GetEventSourceMapping</code>	Update any references to <code>GetEventSource</code> to use <code>GetEventSourceMapping</code> when used to update an event source
List all event sources	<code>ListEventSource</code>	<code>ListEventSourceMapping</code>	Update any references to <code>GetEventSource</code> to use <code>GetEventSourceMapping</code> when used to update an event source
Delete an event source	<code>RemoveEventSource</code>	<code>DeleteEventSourceMapping</code>	Update any references to <code>RemoveEventSource</code> to use <code>DeleteEventSourceMapping</code> when used to delete an event source
Programming model			
Specifying successful execution	<code>context.done(null, "successmessage")</code>	<code>context.succeed(result)</code>	<code>context.done</code> will continue to work; however, it is recommended you use the new methods.
Specifying an error in execution	<code>context.done(errorObject, "failuremessage")</code>	<code>context.fail(error)</code>	<code>context.done</code> will continue to work; however, it is recommended you use the new methods.

API Reference

This section contains the AWS Lambda API Reference documentation. When making the API calls, you will need to authenticate your request by providing a signature. AWS Lambda supports signature version 4. For more information, go to [Signature Version 4 Signing Process](#) in the *Amazon Web Services General Reference*.

For an overview of the service, see [What Is AWS Lambda? \(p. 1\)](#). For information about how the service works, see [AWS Lambda: How it Works \(p. 3\)](#).

You can use the AWS CLI to explore the AWS Lambda API. For examples, see [AWS Lambda Walkthroughs \(p. 30\)](#).

Actions

The following actions are supported:

- [AddPermission \(p. 96\)](#)
- [CreateEventSourceMapping \(p. 99\)](#)
- [CreateFunction \(p. 103\)](#)
- [DeleteEventSourceMapping \(p. 108\)](#)
- [DeleteFunction \(p. 110\)](#)
- [GetEventSourceMapping \(p. 112\)](#)
- [GetFunction \(p. 114\)](#)
- [GetFunctionConfiguration \(p. 116\)](#)
- [GetPolicy \(p. 119\)](#)
- [Invoke \(p. 121\)](#)
- [InvokeAsync \(p. 124\)](#)
- [ListEventSourceMappings \(p. 126\)](#)
- [ListFunctions \(p. 128\)](#)
- [RemovePermission \(p. 130\)](#)
- [UpdateEventSourceMapping \(p. 132\)](#)
- [UpdateFunctionCode \(p. 135\)](#)
- [UpdateFunctionConfiguration \(p. 138\)](#)

AddPermission

Adds a permission to the access policy associated with the specified AWS Lambda function. In a "push event" model, the access policy attached to the Lambda function grants Amazon S3 or a user application permission for the Lambda `lambda:Invoke` action. For information about the push model, see [AWS Lambda: How it Works](#). Each Lambda function has one access policy associated with it. You can use the `AddPermission` API to add a permission to the policy. You have one access policy but it can have multiple permission statements.

This operation requires permission for the `lambda:AddPermission` action.

Request Syntax

```
POST /2015-03-31/functions/FunctionName/versions/HEAD/policy HTTP/1.1
Content-type: application/json

{
  "Action": "string",
  "Principal": "string",
  "SourceAccount": "string",
  "SourceArn": "string",
  "StatementId": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

Name of the Lambda function whose access policy you are updating by adding a new permission.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

(arn:aws:lambda:)?([a-z]{2}-[a-z]+-d{1}:)?(d{12}:)?(function:)?([a-zA-Z0-9-_]+)

Request Body

The request requires the following data in JSON format.

Action

The AWS Lambda action you want to allow in this statement. Each Lambda action is a string starting with "lambda:" followed by the API name (see [Actions \(p. 95\)](#)). For example, "lambda:CreateFunction". You can use wildcard ("lambda:*") to grant permission for all AWS Lambda actions.

Type: String

Pattern: (lambda:[*]|lambda:[a-zA-Z]+|[*])

Required: Yes

The principal who is getting this permission. It can be Amazon S3 service Principal ("s3.amazonaws.com") if you want Amazon S3 to invoke the function, an AWS account ID if you are granting cross-account permission, or any valid AWS service principal such as "sns.amazonaws.com". For example, you might want to allow a custom application in another AWS account to push events to AWS Lambda by invoking your function.

Pattern: . *

SourceAccount

Type: String

Required: No

This is optional; however, when granting Amazon S3 permission to invoke your function, you should specify this field with the bucket Amazon Resource Name (ARN) as its value. This ensures that only events generated from the specified bucket can invoke the function.

If you add a permission for the Amazon S3 principal without providing the source ARN, any AWS account that creates a mapping to your function ARN can send events to invoke your Lambda function from Amazon S3.

Pattern: `arn:aws:([a-zA-Z0-9\-.]+):([a-z]{2}-[a-z]+-\d{1})?:([a-z]{12})?:(.*)`

StatementId

Type: String

Pattern: ([a-zA-Z0-9-]+)

Required: Yes

```
HTTP/1.1 201
Content-type: application/json

{
```

```
}  "Statement": "string"
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

Statement

The permission statement you specified in the request. The response returns the same as a string using "\" as an escape character in the JSON.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PolicyLengthExceededException

Lambda function access policy is limited to 20 KB.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

CreateEventSourceMapping

Identifies a stream as an event source for a Lambda function. It can be either an Amazon Kinesis stream or an Amazon DynamoDB stream. AWS Lambda invokes the specified function when records are posted to the stream.

This is the pull model, where AWS Lambda invokes the function. For more information, go to [AWS Lambda: How it Works](#) in the *AWS Lambda Developer Guide*.

This association between an Amazon Kinesis stream and a Lambda function is called the event source mapping. You provide the configuration information (for example, which stream to read from and which Lambda function to invoke) for the event source mapping in the request body.

Each event source, such as an Amazon Kinesis or a DynamoDB stream, can be associated with multiple AWS Lambda function. A given Lambda function can be associated with multiple AWS event sources.

This operation requires permission for the `lambda:CreateEventSourceMapping` action.

Request Syntax

```
POST /2015-03-31/event-source-mappings/ HTTP/1.1
Content-type: application/json

{
  "BatchSize": number,
  "Enabled": boolean,
  "EventSourceArn": "string",
  "FunctionName": "string",
  "StartingPosition": "string"
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request requires the following data in JSON format.

BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records. The default is 100 records.

Type: Number

Valid range: Minimum value of 1. Maximum value of 10000.

Required: No

Enabled

Indicates whether AWS Lambda should begin polling the event source.

Type: Boolean

Required: No

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis or the Amazon DynamoDB stream that is the event source. Any record added to this stream could cause AWS Lambda to invoke your Lambda function, it depends on the `BatchSize`. AWS Lambda POSTs the Amazon Kinesis event, containing records, to your Lambda function as JSON.

Type: String

Pattern: `arn:aws:([a-zA-Z0-9_-])+:([a-z]{2}-[a-z]+-\d{1})?:(\d{12})?:(.*)`

Required: Yes

FunctionName

The Lambda function to invoke when AWS Lambda detects an event on the stream.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

`(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}):?(\d{12}):?(function:)?([a-zA-Z0-9-_]+)`

Required: Yes

StartingPosition

The position in the stream where AWS Lambda should start reading. For more information, go to [ShardIteratorType](#) in the *Amazon Kinesis API Reference*.

Type: String

Valid Values: `TRIM_HORIZON` | `LATEST`

Required: Yes

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "BatchSize": number,
  "EventSourceArn": "string",
  "FunctionArn": "string",
  "LastModified": number,
  "LastProcessingResult": "string",
  "State": "string",
  "StateTransitionReason": "string",
  "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Number

Valid range: Minimum value of 1. Maximum value of 10000.

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: `arn:aws:kinesis:[a-z0-9-]+:[a-z]{2}-[a-z]+-\d{1}}?:(\d{12}})?:(.*)`

FunctionArn

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern:

`arn:aws:lambda:[a-z]{2}-[a-z]+-\d{1}}:\d{12}}:function:[a-z0-9-]+(\/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}})?`

LastModified

The UTC time string indicating the last time the event mapping was updated.

Type: DateTime

LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

State

The state of the event source mapping. It can be "Creating", "Enabled", "Disabled", "Enabling", "Disabling", "Updating", or "Deleting".

Type: String

StateTransitionReason

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

UUID

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

CreateFunction

Creates a new Lambda function. The function metadata is created from the request parameters, and the code for the function is provided by a .zip file in the request body. If the function name already exists, the operation will fail. Note that the function name is case-sensitive.

This operation requires permission for the `lambda:CreateFunction` action.

Request Syntax

```
POST /2015-03-31/functions HTTP/1.1
Content-type: application/json

{
  "Code": {
    "ZipFile": blob
  },
  "Description": "string",
  "FunctionName": "string",
  "Handler": "string",
  "MemorySize": number,
  "Role": "string",
  "Runtime": "string",
  "Timeout": number
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request requires the following data in JSON format.

Code

A structure that includes ZipFile.

Type: [FunctionCode](#) (p. 143) object

Required: Yes

Description

A short, user-defined function description. Lambda does not use this value. Assign a meaningful description as you see fit.

Type: String

Length constraints: Minimum length of 0. Maximum length of 256.

Required: No

FunctionName

The name you want to assign to the function you are uploading. You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length. The function names appear in the console and are returned in the

[ListFunctions \(p. 128\)](#) API. Function names are used to specify functions to other AWS Lambda APIs, such as [Invoke \(p. 121\)](#).

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_.]+)

Required: Yes

Handler

The function within your code that Lambda calls to begin execution. For Node.js, it is the *module-name.export* value in your function.

Type: String

Length constraints: Minimum length of 0. Maximum length of 128.

Pattern: [a-zA-Z0-9-_.]+

Required: Yes

MemorySize

The amount of memory, in MB, your Lambda function is given. Lambda uses this memory size to infer the amount of CPU and memory allocated to your function. Your function use-case determines your CPU and memory requirements. For example, a database operation might need less memory compared to an image processing function. The default value is 128 MB. The value must be a multiple of 64 MB.

Type: Number

Valid range: Minimum value of 128. Maximum value of 1024.

Required: No

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources. For more information, see [AWS Lambda: How it Works](#)

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z0-9+=,.\@_-_/]+

Required: Yes

Runtime

The runtime environment for the Lambda function you are uploading. Currently, Lambda supports only "nodejs" as the runtime.

Type: String

Valid Values: nodejs

Required: Yes

Timeout

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Number

Valid range: Minimum value of 1. Maximum value of 60.

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
  "CodeSize": number,
  "Description": "string",
  "FunctionArn": "string",
  "FunctionName": "string",
  "Handler": "string",
  "LastModified": "string",
  "MemorySize": number,
  "Role": "string",
  "Runtime": "string",
  "Timeout": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

CodeSize

The size, in bytes, of the function .zip file you uploaded.

Type: Long

Description

The user-provided description.

Type: String

Length constraints: Minimum length of 0. Maximum length of 256.

FunctionArn

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern:

`arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9_-]+(\d{8}-[0-9-f]{4}-[0-9-f]{4}-[0-9-f]{4}-[0-9-f]{12})?`

FunctionName

The name of the function.

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

`(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9_-]+)`

Handler

The function Lambda calls to begin executing your function.

Type: String

Length constraints: Minimum length of 0. Maximum length of 128.

Pattern: [a-zA-Z0-9.\/_-]+

LastModified

The timestamp of the last time you updated the function.

Type: String

MemorySize

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Number

Valid range: Minimum value of 128. Maximum value of 1024.

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.\@_-/_]+

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs

Timeout

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Number

Valid range: Minimum value of 1. Maximum value of 60.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

DeleteEventSourceMapping

Removes an event source mapping. This means AWS Lambda will no longer invoke the function for events in the associated source.

This operation requires permission for the `lambda:DeleteEventSourceMapping` action.

Request Syntax

```
DELETE /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

UUID

The event source mapping ID.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "BatchSize": number,
  "EventSourceArn": "string",
  "FunctionArn": "string",
  "LastModified": number,
  "LastProcessingResult": "string",
  "State": "string",
  "StateTransitionReason": "string",
  "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Number

Valid range: Minimum value of 1. Maximum value of 10000.

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Pattern: `arn:aws:([a-zA-Z0-9-])+:([a-z]{2}-[a-z]+-\d{1})?:(\d{12})?:(.*)`

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Pattern:

```
an:as:lat:ba:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9_]+(√[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?
```

The UTC time string indicating the last time the event mapping was updated.

LastProcessingResult

Type: String

The state of the event source mapping. It can be "Creating", "Enabled", "Disabled", "Enabling", "Disabling", "Updating", or "Deleting".

StateTransitionReason

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

UUID

Type: String

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

HTTP Status Code: 429

DeleteFunction

Deletes the specified Lambda function code and configuration.

When you delete a function the associated access policy is also deleted. You will need to delete the event source mappings explicitly.

This operation requires permission for the `lambda:DeleteFunction` action.

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

The Lambda function to delete.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

GetEventSourceMapping

Returns configuration information for the specified event source mapping (see [CreateEventSourceMapping](#) (p. 99)).

This operation requires permission for the `lambda:GetEventSourceMapping` action.

Request Syntax

```
GET /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

UUID

The AWS Lambda assigned ID of the event source mapping.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "BatchSize": number,
  "EventSourceArn": "string",
  "FunctionArn": "string",
  "LastModified": number,
  "LastProcessingResult": "string",
  "State": "string",
  "StateTransitionReason": "string",
  "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Number

Valid range: Minimum value of 1. Maximum value of 10000.

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: `arn:aws:([a-zA-Z0-9-])+:([a-z]{2}-[a-z]+-\d{1})?:(\d{12})?:(.*)`

FunctionArn

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern:

`arn:aws:lambda:[a-z]{2}-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9_-]+(\/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?`

LastModified

The UTC time string indicating the last time the event mapping was updated.

Type: DateTime

LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

State

The state of the event source mapping. It can be "Creating", "Enabled", "Disabled", "Enabling", "Disabling", "Updating", or "Deleting".

Type: String

StateTransitionReason

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

UUID

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

GetFunction

Returns the configuration information of the Lambda function and a presigned URL link to the .zip file you uploaded with [CreateFunction](#) (p. 103) so you can download the .zip file. Note that the URL is valid for up to 10 minutes. The configuration information is the same information you provided as parameters when uploading the function.

This operation requires permission for the `lambda:GetFunction` action.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/versions/HEAD HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

The Lambda function name.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

```
(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\+] )
```

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Code": {
    "Location": "string",
    "RepositoryType": "string"
  },
  "Configuration": {
    "CodeSize": number,
    "Description": "string",
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "LastModified": "string",
    "MemorySize": number,
    "Role": "string",
```

```
    "Runtime": "string",  
    "Timeout": number  
  }  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Code

The object for the Lambda function location.

Type: [FunctionCodeLocation](#) (p. 143) object

Configuration

A complex type that describes function metadata.

Type: [FunctionConfiguration](#) (p. 144) object

Errors

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

GetFunctionConfiguration

Returns the configuration information of the Lambda function. This the same information you provided as parameters when uploading the function by using [CreateFunction](#) (p. 103).

This operation requires permission for the `lambda:GetFunctionConfiguration` operation.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/versions/HEAD/configuration HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

The name of the Lambda function for which you want to retrieve the configuration information.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

```
(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)
```

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "CodeSize": number,
  "Description": "string",
  "FunctionArn": "string",
  "FunctionName": "string",
  "Handler": "string",
  "LastModified": "string",
  "MemorySize": number,
  "Role": "string",
  "Runtime": "string",
  "Timeout": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeSize

The size, in bytes, of the function .zip file you uploaded.

Type: Long

Description

The user-provided description.

Type: String

Length constraints: Minimum length of 0. Maximum length of 256.

FunctionArn

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern:

`arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9_]+(\/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?`

FunctionName

The name of the function.

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

`(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9_]+)`

Handler

The function Lambda calls to begin executing your function.

Type: String

Length constraints: Minimum length of 0. Maximum length of 128.

Pattern: `[a-zA-Z0-9.\/_-]+`

LastModified

The timestamp of the last time you updated the function.

Type: String

MemorySize

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Number

Valid range: Minimum value of 128. Maximum value of 1024.

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z0-9+=,.\@_/_]+`

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: `nodejs`

Timeout

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Number

Valid range: Minimum value of 1. Maximum value of 60.

Errors

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

GetPolicy

Returns the access policy, containing a list of permissions granted via the `AddPermission` API, associated with the specified bucket.

You need permission for the `lambda:GetPolicy` action.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/versions/HEAD/policy HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

Function name whose access policy you want to retrieve.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

```
(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\+]*)
```

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Policy": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Policy

The access policy associated with the specified function. The response returns the same as a string using "\" as an escape character in the JSON.

Type: String

Errors

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

Invoke

Invokes a specified Lambda function.

This operation requires permission for the `lambda:InvokeFunction` action.

Request Syntax

```
POST /2015-03-31/functions/FunctionName/invocations HTTP/1.1
X-Amz-Client-Context: ClientContext
X-Amz-Invocation-Type: InvocationType
X-Amz-Log-Type: LogType

Payload
```

URI Request Parameters

The request requires the following URI parameters.

ClientContext

Using the `ClientContext` you can pass client-specific information to the Lambda function you are invoking. You can then process the client information in your Lambda function as you choose through the context variable. For an example of a `ClientContext` JSON, go to [PutEvents](#) in the *Amazon Mobile Analytics API Reference and User Guide*.

The `ClientContext` JSON must be base64-encoded.

FunctionName

The Lambda function name.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

```
(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-]+)
```

InvocationType

By default, the `Invoke` API assumes "RequestResponse" invocation type. You can optionally request asynchronous execution by specifying "Event" as the `InvocationType`. You can also use this parameter to request AWS Lambda to not execute the function but do some verification, such as if the caller is authorized to invoke the function and if the inputs are valid. You request this by specifying "DryRun" as the `InvocationType`. This is useful in a cross-account scenario when you want to verify access to a function without running it.

Valid Values: `Event` | `RequestResponse` | `DryRun`

LogType

You can set this optional parameter to "Tail" in the request only if you specify the `InvocationType` parameter with value "RequestResponse". In this case, AWS Lambda returns the base64-encoded last 4 KB of log data produced by your Lambda function in the `x-amz-log-results` header.

Valid Values: `None` | `Tail`

Request Body

The request requires the following as the HTTP body.

Payload

JSON that you want to provide to your Lambda function as input.

Response Syntax

```
HTTP/1.1 StatusCode  
X-Amz-Function-Error: FunctionError  
X-Amz-Log-Result: LogResult
```

Payload

Response Elements

Status Code

The HTTP status code will be in the 200 range for successful request. For the "RequestResponse" invocation type this status code will be 200. For the "Event" invocation type this status code will be 202. For the "DryRun" invocation type the status code will be 204.

The response returns the following HTTP headers.

FunctionError

Indicates whether an error occurred while executing the Lambda function. If an error occurred this field will have one of two values; `Handled` or `Unhandled`. `Handled` errors are errors that are reported by the function while the `Unhandled` errors are those detected and reported by AWS Lambda. `Unhandled` errors include out of memory errors and function timeouts. For information about how to report an `Handled` error, see [Programming Model](#).

LogResult

It is the base64-encoded logs for the Lambda function invocation. This is present only if the invocation type is "RequestResponse" and the logs were requested.

The response returns the following as the HTTP body.

Payload

It is the JSON representation of the object returned by the Lambda function. This is present only if the invocation type is "RequestResponse".

In the event of a function error this field contains a message describing the error. For the `Handled` errors the Lambda function will report this message. For `Unhandled` errors AWS Lambda reports the message.

Errors

InvalidRequestContentException

The request body could not be parsed as JSON.

HTTP Status Code: 400

RequestTooLargeException

HTTP Status Code: 413

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

UnsupportedMediaTypeException

HTTP Status Code: 415

InvokeAsync

Important

This API is deprecated. We recommend you use `Invoke` API (see [Invoke](#) (p. 121)).

Submits an invocation request to AWS Lambda. Upon receiving the request, Lambda executes the specified function asynchronously. To see the logs generated by the Lambda function execution, see the CloudWatch logs console.

This operation requires permission for the `lambda:InvokeFunction` action.

Request Syntax

```
POST /2014-11-13/functions/FunctionName/invoke-async/ HTTP/1.1
InvokeArgs
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

The Lambda function name.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_.]+)

Request Body

The request requires the following as the HTTP body.

InvokeArgs

JSON that you want to provide to your Lambda function as input.

Response Syntax

```
HTTP/1.1 Status
```

Response Elements

Status

It will be 202 upon success.

Errors

InvalidRequestContentException

The request body could not be parsed as JSON.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

Examples

Invoke a Lambda function

The following example uses a POST request to invoke a Lambda function.

Sample Request

```
POST /2014-11-13/functions/helloworld/invoke-async/ HTTP/1.1
[input json]
```

Sample Response

```
HTTP/1.1 202 Accepted

x-amzn-requestid: f037bc5c-5a08-11e4-b02e-af446c3f9d0d
content-length: 0
connection: keep-alive
date: Wed, 22 Oct 2014 16:31:55 GMT
content-type: application/json
```

ListEventSourceMappings

Returns a list of event source mappings you created using the `CreateEventSourceMapping` (see [CreateEventSourceMapping](#) (p. 99)), where you identify a stream as an event source. This list does not include Amazon S3 event sources.

For each mapping, the API returns configuration information. You can optionally specify filters to retrieve specific event source mappings.

This operation requires permission for the `lambda:ListEventSourceMappings` action.

Request Syntax

```
GET /2015-03-31/event-source-mappings/?Marker=Marker&MaxItems=MaxItems&FunctionName=FunctionName&EventSourceArn=EventSourceArn HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis stream.

Pattern: `arn:aws:([a-zA-Z0-9-])+:([a-z]{2}-[a-z]+-\d{1})?:(\d{12})?:(.*)`

FunctionName

The name of the Lambda function.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

`(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1})?:?(\d{12})?:?(function:)?([a-zA-Z0-9-]+)`

Marker

Optional string. An opaque pagination token returned from a previous `ListEventSourceMappings` operation. If present, specifies to continue the list from where the returning call left off.

MaxItems

Optional integer. Specifies the maximum number of event sources to return in response. This value must be greater than 0.

Valid range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "EventSourceMappings": [
    {
      "BatchSize": number,
      "EventSourceArn": "string",
      "FunctionArn": "string",
      "LastModified": number,
      "LastProcessingResult": "string",
      "State": "string",
      "StateTransitionReason": "string",
      "UUID": "string"
    }
  ],
  "NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EventSourceMappings

An array of `EventSourceMappingConfiguration` objects.

Type: array of [EventSourceMappingConfiguration](#) (p. 142) objects

NextMarker

A string, present if there are more event source mappings.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

ListFunctions

Returns a list of your Lambda functions. For each function, the response includes the function configuration information. You must use [GetFunction](#) (p. 114) to retrieve the code for your function.

This operation requires permission for the `lambda:ListFunctions` action.

Request Syntax

```
GET /2015-03-31/functions/?Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

Marker

Optional string. An opaque pagination token returned from a previous `ListFunctions` operation. If present, indicates where to continue the listing.

MaxItems

Optional integer. Specifies the maximum number of AWS Lambda functions to return in response. This parameter value must be greater than 0.

Valid range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Functions": [
    {
      "CodeSize": number,
      "Description": "string",
      "FunctionArn": "string",
      "FunctionName": "string",
      "Handler": "string",
      "LastModified": "string",
      "MemorySize": number,
      "Role": "string",
      "Runtime": "string",
      "Timeout": number
    }
  ],
  "NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Functions

A list of Lambda functions.

Type: array of [FunctionConfiguration \(p. 144\)](#) objects

NextMarker

A string, present if there are more functions.

Type: String

Errors

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

RemovePermission

You can remove individual permissions from an access policy associated with a Lambda function by providing a Statement ID.

Note that removal of a permission will cause an active event source to lose permission to the function.

You need permission for the `lambda:RemovePermission` action.

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName/versions/HEAD/policy/StatementId HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

Lambda function whose access policy you want to remove a permission from.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-]+)

StatementId

Statement ID of the permission to remove.

Length constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

UpdateEventSourceMapping

You can update an event source mapping. This is useful if you want to change the parameters of the existing mapping without losing your position in the stream. You can change which function will receive the stream records, but to change the stream itself, you must create a new mapping.

This operation requires permission for the `lambda:UpdateEventSourceMapping` action.

Request Syntax

```
PUT /2015-03-31/event-source-mappings/UUID HTTP/1.1
Content-type: application/json

{
  "BatchSize": number,
  "Enabled": boolean,
  "FunctionName": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

UUID

The event source mapping identifier.

Request Body

The request requires the following data in JSON format.

BatchSize

The maximum number of stream records that can be sent to your Lambda function for a single invocation.

Type: Number

Valid range: Minimum value of 1. Maximum value of 10000.

Required: No

Enabled

Specifies whether AWS Lambda should actively poll the stream or not. If disabled, AWS Lambda will not poll the stream.

Type: Boolean

Required: No

FunctionName

The Lambda function to which you want the stream records sent.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\+])

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "BatchSize": number,
  "EventSourceArn": "string",
  "FunctionArn": "string",
  "LastModified": number,
  "LastProcessingResult": "string",
  "State": "string",
  "StateTransitionReason": "string",
  "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Number

Valid range: Minimum value of 1. Maximum value of 10000.

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: arn:aws:([a-zA-Z0-9-])+:([a-z]{2}-[a-z]+-\d{1}):?(\d{12}):?(\.*)

FunctionArn

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern:

arn:aws:lambda:([a-z]{2}-[a-z]+-\d{1}):\d{12}:function:[a-zA-Z0-9-](\d{8}-\d{4}-\d{4}-\d{4}-\d{4}-\d{12})?

LastModified

The UTC time string indicating the last time the event mapping was updated.

Type: DateTime

LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

State

The state of the event source mapping. It can be "Creating", "Enabled", "Disabled", "Enabling", "Disabling", "Updating", or "Deleting".

Type: String

StateTransitionReason

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

UUID

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

UpdateFunctionCode

Updates the code for the specified Lambda function. This operation must only be used on an existing Lambda function and cannot be used to update the function configuration.

This operation requires permission for the `lambda:UpdateFunctionCode` action.

Request Syntax

```
PUT /2015-03-31/functions/FunctionName/versions/HEAD/code HTTP/1.1
Content-type: application/json

{
  "ZipFile": blob
}
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

The existing Lambda function name whose code you want to replace.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\+])

Request Body

The request requires the following data in JSON format.

ZipFile

Based64-encoded .zip file containing your packaged source code.

Type: Blob

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "CodeSize": number,
  "Description": "string",
```

```
{
  "FunctionArn": "string",
  "FunctionName": "string",
  "Handler": "string",
  "LastModified": "string",
  "MemorySize": number,
  "Role": "string",
  "Runtime": "string",
  "Timeout": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeSize

The size, in bytes, of the function .zip file you uploaded.

Type: Long

Description

The user-provided description.

Type: String

Length constraints: Minimum length of 0. Maximum length of 256.

FunctionArn

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern:

`arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9_]+(\d{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?`

FunctionName

The name of the function.

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

`(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9_]+)`

Handler

The function Lambda calls to begin executing your function.

Type: String

Length constraints: Minimum length of 0. Maximum length of 128.

Pattern: `[a-zA-Z0-9._\-_]+`

LastModified

The timestamp of the last time you updated the function.

Type: String

MemorySize

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Number

Valid range: Minimum value of 128. Maximum value of 1024.

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.\@\-_/\]+`

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: `nodejs`

Timeout

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Number

Valid range: Minimum value of 1. Maximum value of 60.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

UpdateFunctionConfiguration

Updates the configuration parameters for the specified Lambda function by using the values provided in the request. You provide only the parameters you want to change. This operation must only be used on an existing Lambda function and cannot be used to update the function's code.

This operation requires permission for the `lambda:UpdateFunctionConfiguration` action.

Request Syntax

```
PUT /2015-03-31/functions/FunctionName/versions/HEAD/configuration HTTP/1.1
Content-type: application/json
```

```
{
  "Description": "string",
  "Handler": "string",
  "MemorySize": number,
  "Role": "string",
  "Timeout": number
}
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName

The name of the Lambda function.

You can specify an unqualified function name (for example, "Thumbnail") or you can specify Amazon Resource Name (ARN) of the function (for example, "arn:aws:lambda:us-west-2:account-id:function:ThumbNail"). AWS Lambda also allows you to specify only the account ID qualifier (for example, "account-id:Thumbnail"). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

```
(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)
```

Request Body

The request requires the following data in JSON format.

Description

A short user-defined function description. AWS Lambda does not use this value. Assign a meaningful description as you see fit.

Type: String

Length constraints: Minimum length of 0. Maximum length of 256.

Required: No

Handler

The function that Lambda calls to begin executing your function. For Node.js, it is the *module-name.export* value in your function.

Type: String

Length constraints: Minimum length of 0. Maximum length of 128.

Pattern: [a-zA-Z0-9.\/_-]+

Required: No

MemorySize

The amount of memory, in MB, your Lambda function is given. AWS Lambda uses this memory size to infer the amount of CPU allocated to your function. Your function use-case determines your CPU and memory requirements. For example, a database operation might need less memory compared to an image processing function. The default value is 128 MB. The value must be a multiple of 64 MB.

Type: Number

Valid range: Minimum value of 128. Maximum value of 1024.

Required: No

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda will assume when it executes your function.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.\@_-/+]

Required: No

Timeout

The function execution time at which AWS Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Number

Valid range: Minimum value of 1. Maximum value of 60.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "CodeSize": number,
  "Description": "string",
  "FunctionArn": "string",
  "FunctionName": "string",
  "Handler": "string",
  "LastModified": "string",
  "MemorySize": number,
  "Role": "string",
  "Runtime": "string",
  "Timeout": number
}
```


Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeSize

The size, in bytes, of the function .zip file you uploaded.

Type: Long

Description

The user-provided description.

Type: String

Length constraints: Minimum length of 0. Maximum length of 256.

FunctionArn

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern:

`arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9_]+(\/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?`

FunctionName

The name of the function.

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

`(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9_]+)`

Handler

The function Lambda calls to begin executing your function.

Type: String

Length constraints: Minimum length of 0. Maximum length of 128.

Pattern: `[a-zA-Z0-9.\/_-]+`

LastModified

The timestamp of the last time you updated the function.

Type: String

MemorySize

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Number

Valid range: Minimum value of 128. Maximum value of 1024.

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z0-9+=,.\@_/_]+`

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: `nodejs`

Timeout

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Number

Valid range: Minimum value of 1. Maximum value of 60.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

HTTP Status Code: 429

Data Types

The AWS Lambda API contains several data types that various actions use. This section describes each data type in detail.

Note

The order of each element in the response is not guaranteed. Applications should not assume a particular order.

The following data types are supported:

- [EventSourceMappingConfiguration](#) (p. 142)
- [FunctionCode](#) (p. 143)
- [FunctionCodeLocation](#) (p. 143)
- [FunctionConfiguration](#) (p. 144)

EventSourceMappingConfiguration

Description

Describes mapping between an Amazon Kinesis stream and a Lambda function.

Contents

BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Number

Valid range: Minimum value of 1. Maximum value of 10000.

Required: No

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: `arn:aws:kinesis:([a-zA-Z0-9\-])+:([a-z]{2}-[a-z]+-\d{1})?:(\d{12})?:(.*)`

Required: No

FunctionArn

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern:

`arn:aws:lambda:([a-z]{2}-[a-z]+-\d{1}):\d{12}:function:([a-zA-Z0-9_-]+(\d{8}|\d{9}|\d{10}|\d{11}|\d{12}))?`

Required: No

LastModified

The UTC time string indicating the last time the event mapping was updated.

Type: DateTime

Required: No

LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

Required: No

State

The state of the event source mapping. It can be "Creating", "Enabled", "Disabled", "Enabling", "Disabling", "Updating", or "Deleting".

Type: String

Required: No

StateTransitionReason

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

Required: No

UUID

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Required: No

FunctionCode

Description

The code for the Lambda function.

Contents

ZipFile

A base64-encoded .zip file containing your packaged source code. For more information about creating a .zip file, go to [Execution Permissions](#) in the *AWS Lambda Developer Guide*.

Type: Blob

Required: No

FunctionCodeLocation

Description

The object for the Lambda function location.

Contents

Location

The presigned URL you can use to download the function's .zip file that you previously uploaded. The URL is valid for up to 10 minutes.

Type: String

Required: No

RepositoryType

The repository from which you can download the function.

Type: String

Required: No

FunctionConfiguration

Description

A complex type that describes function metadata.

Contents

CodeSize

The size, in bytes, of the function .zip file you uploaded.

Type: Long

Required: No

Description

The user-provided description.

Type: String

Length constraints: Minimum length of 0. Maximum length of 256.

Required: No

FunctionArn

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern:

`arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9_]+(\|[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?`

Required: No

FunctionName

The name of the function.

Type: String

Length constraints: Minimum length of 1. Maximum length of 111.

Pattern:

`(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9_]+)`

Required: No

Handler

The function Lambda calls to begin executing your function.

Type: String

Length constraints: Minimum length of 0. Maximum length of 128.

Pattern: `[a-zA-Z0-9.\/_-]+`

Required: No

LastModified

The timestamp of the last time you updated the function.

Type: String

Required: No

MemorySize

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Number

Valid range: Minimum value of 128. Maximum value of 1024.

Required: No

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.\@\-_/]+`

Required: No

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: `node.js`

Required: No

Timeout

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Number

Valid range: Minimum value of 1. Maximum value of 60.

Required: No

Document History

The following table describes the important changes to the *AWS Lambda Developer Guide*.

Relevant Dates to this History:

- **Current product version:** 2014-11-13
- **Last documentation update:** April 9, 2015

Change	Description	Date
AWS Lambda now generally available with added support for mobile backends	<p>AWS Lambda is now generally available for production use. The release also introduces new features that make it easier to build mobile, tablet, and Internet of Things (IoT) backends using AWS Lambda that scale automatically without provisioning or managing infrastructure. AWS Lambda now supports both real-time (synchronous) and asynchronous events. Additional features include easier event source configuration and management. The permission model and the programming model have been simplified by the introduction of resource policies for your Lambda functions.</p> <p>The documentation has been updated accordingly. For information, see the following topics:</p> <p>AWS Lambda: How it Works (p. 3)</p> <p>Getting Started with AWS Lambda (p. 16)</p> <p>http://aws.amazon.com/lambda/whatsnew</p>	In this release
Preview release	Preview release of the <i>AWS Lambda Developer Guide</i> .	November 13, 2014

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.