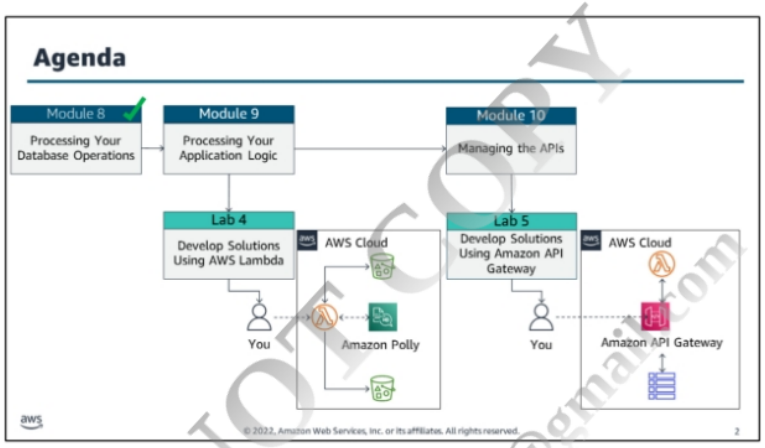
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification

Module 9: Processing Your Application Logic



Module 9 – Compare the feature sets and use cases for available AWS compute solutions. Build an AWS Lambda function to store data from a web application in Amazon DynamoDB.

Lab 4 – Build an AWS Lambda function to store data from a web application in DynamoDB.

Module 10 – Explore the methods available for API Gateway to connect AWS resources.

Lab 5 – Use API Gateway to connect Lambda and DynamoDB.

Module objectives

By the end of this module, you will be able to:

- Explore how AWS Lambda works
- Develop an AWS Lambda function using SDKs
- Configure triggers and permissions for Lambda functions
- Test, deploy, and monitor Lambda functions

You have worked with the following core services:

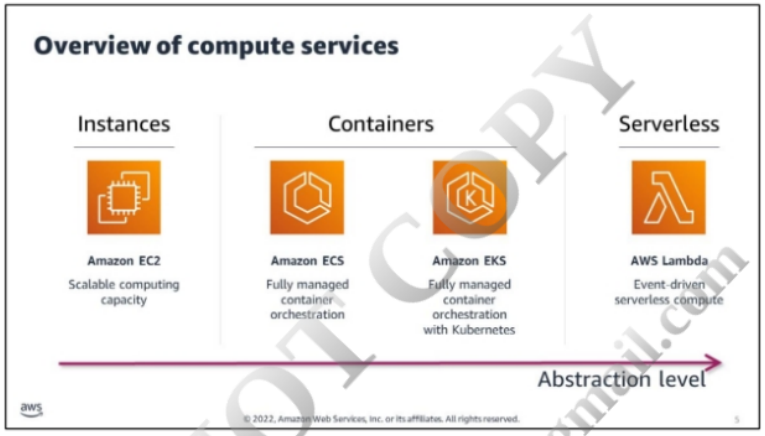
- Amazon Simple Storage Service (Amazon S3) for storing the data and hosting the website
- Amazon DynamoDB to store notes based on the userID

Now, you will use AWS Lambda to process the application logic.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Compute services

Module 9: Processing Your Application Logic



This lesson explores the options available to manage less when it comes to computing.

Types of computing services
AWS provides three types of computing:

- **Instances** – This is the traditional way of thinking about infrastructure. Instances provide you with complete control of your computing resources. It allows access and customization at deep levels.
- **Containers** – One of the most crucial application types in the cloud. The container virtualization systems are smaller, portable, and easier to manage than complete virtual management systems.
- **Serverless** – You do not have to provision servers to run the backend code or worry about the infrastructure model for applications.

Amazon EC2
With Amazon Elastic Compute Cloud (Amazon EC2), you control instances of physical servers. This means that you might need load balancers to distribute the traffic to your instances. Depending on traffic and demand, you may also need to start or stop EC2 instances.

Amazon ECS and Amazon EKS
Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS) are container orchestration services managed by AWS. A container is a software package

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

that includes your application's code, configurations, and dependencies. Containers abstract the configuration from your application, which developers can use to create predictable environments that are isolated from other applications. On containers, you can deploy any code, from a complete application to a microservice. You can run Amazon ECS and Amazon EKS clusters on EC2 instances. A cluster is a logical unit grouping of tasks and services that use one or more containers. Amazon ECS and Amazon EKS coordinate these clusters and manage the states of the containers. The services ensure that containers have the resources they need. If one fails, the failure is detected, and another container can replace it.

AWS Lambda
AWS Lambda is an event-driven compute service. An event is a signal that a system's state has changed. The event invokes a Lambda function to run. Lambda runs on demand. You don't have to worry about the infrastructure the function runs on, the load balancing, or the scaling aspects of the function. With Lambda, you focus on the code and not the underlying infrastructure. As a result, you can quickly deliver and update your application. You write code for business logic, and AWS manages the infrastructure provisioning or configuration.

You can use any of the AWS compute services to run your code, so choose the one that best fits your needs. This course uses AWS Lambda.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.

Comparing compute services

Categories	EC2	ECS /EKS	Lambda
Abstraction	Hardware	Operating system	Runtime
Packaging	Amazon Machine Image (AMI)	Container	Function code
Pricing model	Infrastructure consumption-based	Infrastructure consumption-based	Pay per request
Scalability and concurrency	Full control over configuration and scaling	You control	Implicit scaling

Pricing model
The pricing model for Lambda means that you pay per invocation, run duration, and memory used.

Scalability and concurrency
For Amazon EC2, you can provision more instances and increase the size of AMI.

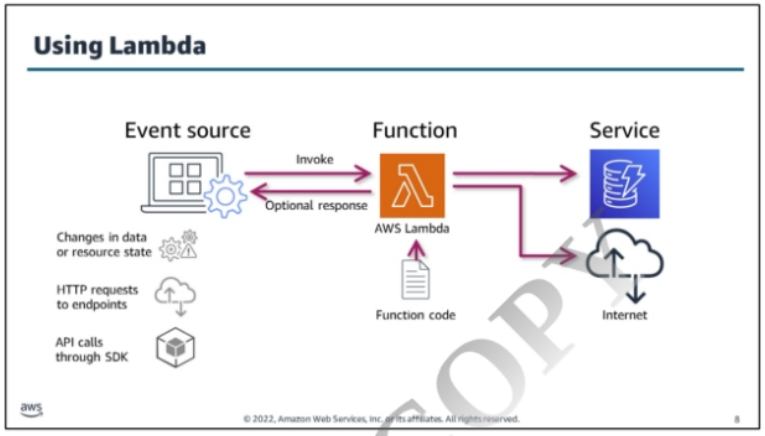
For Amazon EKS and Amazon EFS, you can do the following:
- Increase instance size
- Add more instances
- Add more replicas
- Configure load balancing
- Configure auto scaling

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

How AWS Lambda works

Module 9: Processing Your Application Logic



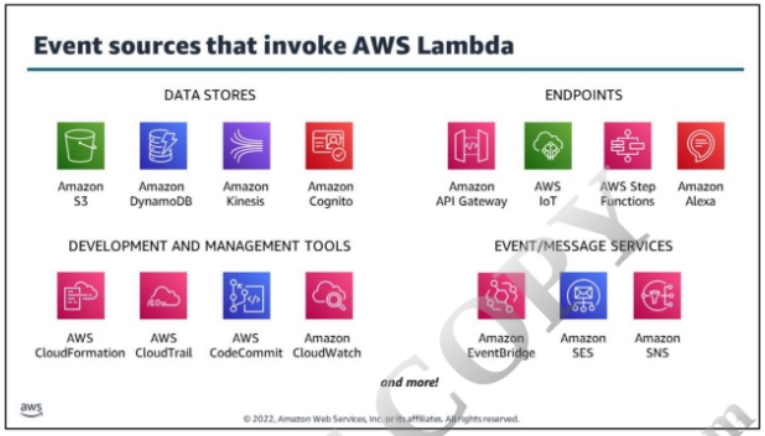
As data flows through your application, Lambda can run code in response to observable events. Lambda runs when it is triggered by an event. Because Lambda can connect to many AWS services, it can run your code in response to the following event sources:

- **Events**, such as a new message in a log file, or changes to data in an Amazon S3 bucket or an Amazon DynamoDB table
- **HTTP requests** using Amazon API Gateway
- **API calls** made using AWS Software Development kits (AWS SDK)

With AWS Lambda, you can run code for virtually any type of application or backend service. The code that AWS Lambda runs is a Lambda function. Lambda functions are stateless. By being stateless, Lambda can quickly scale to the rate of incoming events. Although the AWS Lambda programming model is stateless, your code can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.

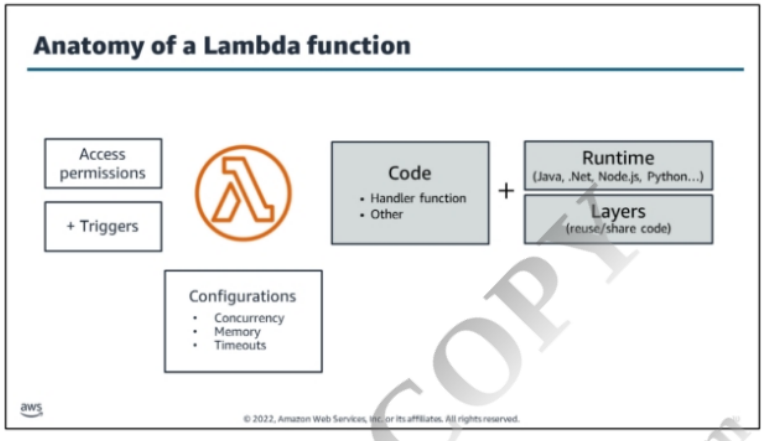
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic



Event sources that invoke AWS Lambda include but are not limited to:

- **DATA STORES**
  - Amazon S3
  - Amazon DynamoDB
  - Amazon Kinesis
  - Amazon Cognito
- **ENDPOINTS**
  - Amazon API Gateway
  - AWS IoT
  - AWS Step Functions
  - Amazon Alexa
- **DEVELOPMENT AND MANAGEMENT TOOLS**
  - AWS CloudFormation
  - AWS CloudTrail
  - AWS CodeCommit
  - Amazon CloudWatch
- **EVENT/MESSAGE SERVICES**
  - Amazon EventBridge
  - Amazon SES
  - Amazon SNS



A Lambda function is a resource that you can invoke to run your code in Lambda.

Prepare your code. The Lambda function code has a handler function and other code. The handler function runs each time an event occurs.

Your code may use runtimes. Lambda supports runtimes for language-specific environments (Node.js, Java, Python, .NET Core, Go, Ruby), or you can create your own custom runtime. A runtime is responsible for the following:

1. Running the function's setup code
2. Reading the handler name from an environment variable
3. Reading invocation events from the Lambda runtime API
4. Passing the event data to the function handler
5. Posting the response from the handler back to Lambda

Your code can also use layers that provide shared code or custom runtimes.

The Lambda function requires permissions that define what it can interact with.

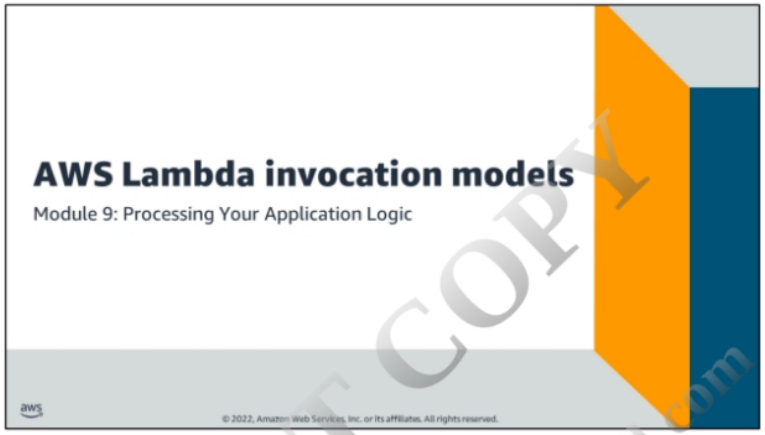
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

A Lambda function must also set up trigger events that specify events or event sources it can interact with. A trigger can be characterized by the following:

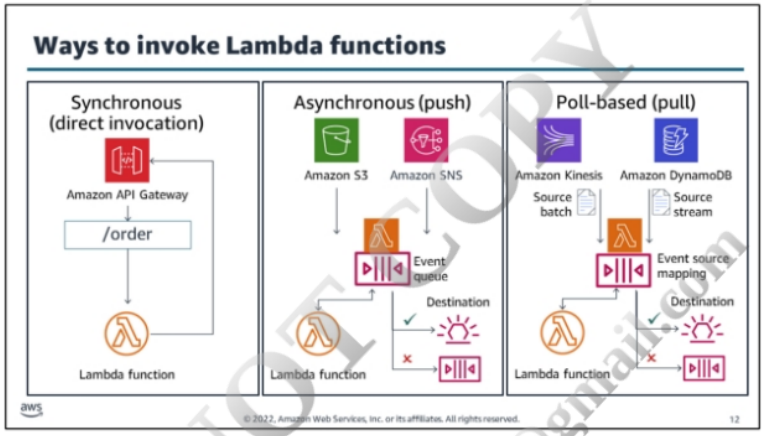
- An AWS service that is configured to invoke a function
- An event source-mapping resource in Lambda that reads items from a stream or queue and invokes a function

You can configure your Lambda runtime environment. You can configure concurrency, memory allocation, timeouts and more.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic



Synchronous invocation (direct invocation)
When you invoke a function synchronously, Lambda runs the function and waits for a response. The function runs immediately when it is invoked. When the function completes its run, Lambda returns the response from the function's code with additional data. Examples of data include status code and the version of the function that was run. This invocation model does not have build-in retries. You can perform a synchronous invocation from the AWS Management Console, AWS Command Line Interface (AWS CLI), or from the AWS SDKs. Also, many AWS services can invoke Lambda functions synchronously, including Amazon Cognito, Amazon API Gateway, and more.

Asynchronous invocation (Push)
When you invoke a function asynchronously, Lambda sends the event to a queue. A separate process reads events from the queue and runs your function. When the event is added to the queue, Lambda returns one of the following:

- A success response without additional information
- An error that indicates that it cannot add it to the event queue

Lambda manages the function's asynchronous event queue and makes several attempts to retry on errors. The number of retries and wait between retries is configurable.

You can send an invocation record, which contains details about the request and response, to a destination service, such as the following:

- AWS Lambda
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon EventBridge

Successful and unsuccessful events can have separate destinations. For example, successful invocation records can be sent to an EventBridge event bus, and unsuccessful records can be sent to another event queue.

A best practice for asynchronous invocations is to create and use dead-letter queues to better handle message failures.

Poll-based invocation (pull)

You can configure an AWS Lambda resource to read from a streaming-based or queuing-based event source and invoke your Lambda function. Examples of event sources include the following:

- Amazon DynamoDB
- Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka
- Self-managed Apache Kafka
- Amazon SQS

This is known as event source mapping. It is used to process items from streams or queues in a service that does not normally invoke Lambda functions directly.
As an example, consider DynamoDB. Stream-based invocation changes the traditional way of processing data in batches. Now data may be processed in real-time. DynamoDB Streams capture item-level modifications in real-time and store this information in logs for up to 24 hours. Lambda polls the service, retrieves matching events, and invokes your function. Permissions, event structure, settings, and polling behavior vary by event source.

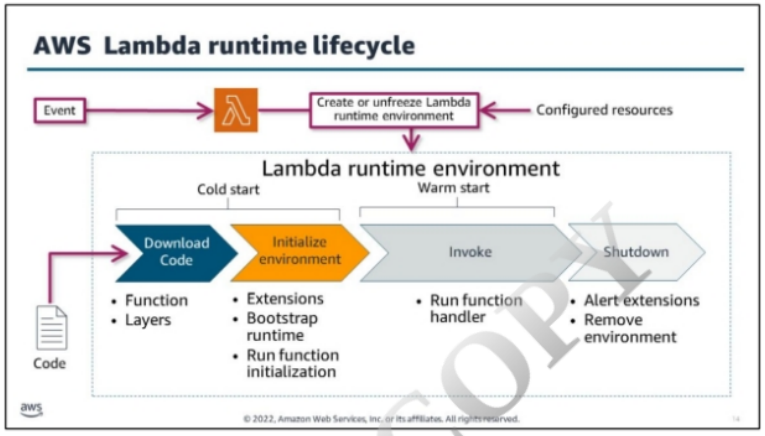
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

Function invocation and retries

Invocation	Retries
Synchronous	No retries (Service may retry based on error)
Asynchronous	Built-in retries (2x)
Poll-based	Depends on event source

DO NOT COPY sameersheik68@gmail.com

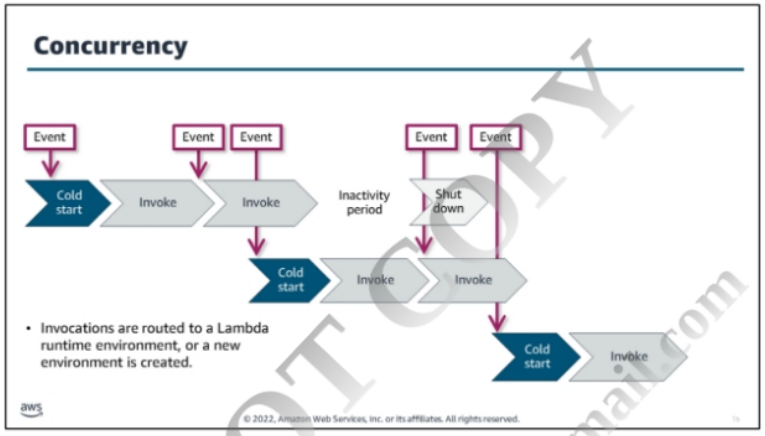


To understand how a Lambda function works, consider the following:

- **Trigger –** Events that invoke the call. You can set up your code to initiate from other AWS services automatically or call it directly from any web or mobile app.
- **Code –** Upload your code as a Lambda function, or directly code in the Lambda's code editor. The code can be in any of the supported languages or use custom runtimes and native libraries. AWS Lambda stores code in Amazon S3 and encrypts it at rest.
- **Configuration –** When you create the function, specify configuration information such as permissions, resources, credentials, and environment variables.
- **Extensions –** Integrate Lambda with existing tools, such as tools used for monitoring, observability, security, and governance.
- **Lambda runtime environment –** Lambda invokes your function in a secure and isolated container created according to resources defined by the configuration. Lambda then loads into memory the code that is to be run. This Lambda runtime environment manages the resources required to run your function and supports the lifecycle of the function's runtime. The process of creating and mounting the environment is called a *cold start*. After the environment is set up, Lambda retains the environment and handles requests. The invocation of a Lambda function after the environment exists (with its layers, initialization, and runtime setup) is known as a *warm start*.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

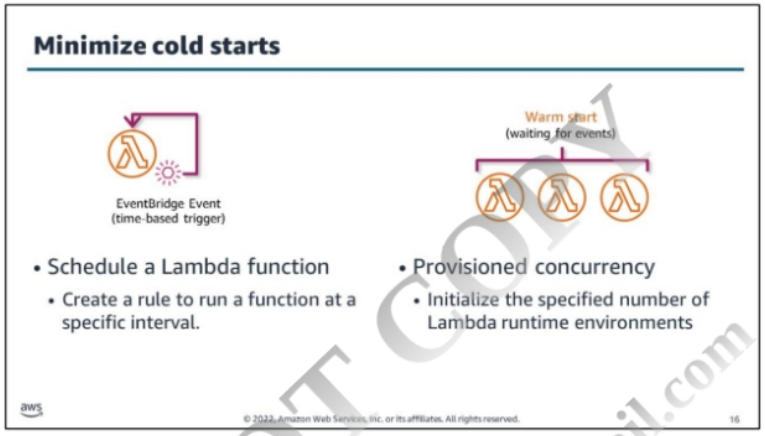
AWS Training and Certification Module 9: Processing Your Application Logic



New invocation requests are processed with a Lambda runtime environment that is available. If a runtime is not available or if an environment is busy, a new runtime is created to handle the request. This new runtime does not share resources with other Lambda runtime environments.

A runtime may not be available if it has shut down because an event has not come in for some time.

All requests are limited to 15 minutes (900 seconds) of running time. The default timeout is 3 seconds but can be set to any value between 1 and 900 seconds. Duration is calculated from the time your code begins running until it returns or otherwise terminates, rounded up to the nearest millisecond.



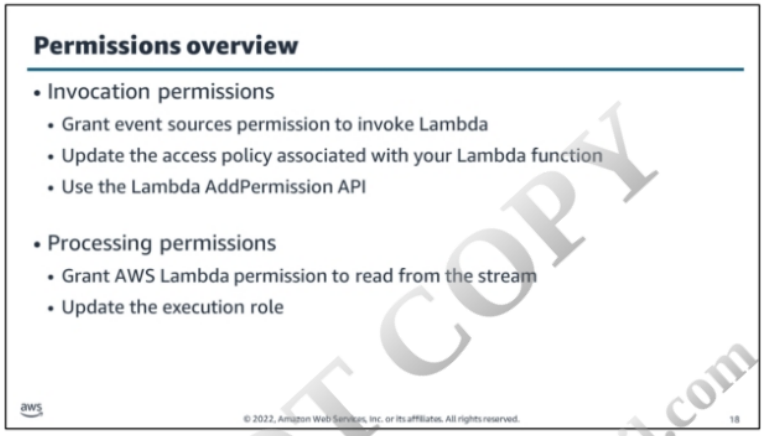
To avoid a cold start, you can set triggers for your Lambda function.
Also, you can use the Provision Concurrency feature in Lambda to prepare multiple Lambda runtime environments to process events.

For more information on scheduling a Lambda function, see (https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-run-lambda-schedule.html)

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

DO NOT COPY
sameersheik68@gmail.com

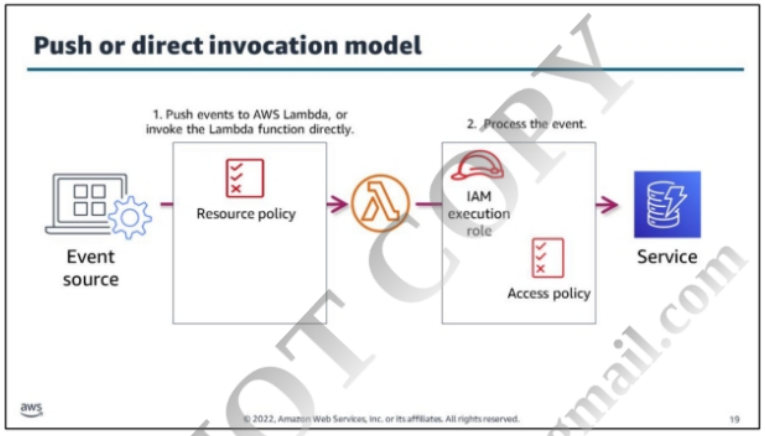


Types of permissions related to Lambda functions:

- **Invocation permissions –** Permissions that the event source needs to communicate with your Lambda function. Depending on the invocation model (push or pull model), you can grant these permissions using the execution role or resource policies (the access policy associated with your Lambda function).
- **Processing permissions –** Permissions that your Lambda function needs to access other AWS resources in your account. You grant these permissions by creating an AWS Identity and Access Management (IAM) role, known as an *execution role*.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic



- Resource policy - gives other accounts and AWS services permission to use your Lambda resources.
- IAM execution role - Grants the function permission to access AWS services and resources.
- Access policy - Grants the needed resource permissions.

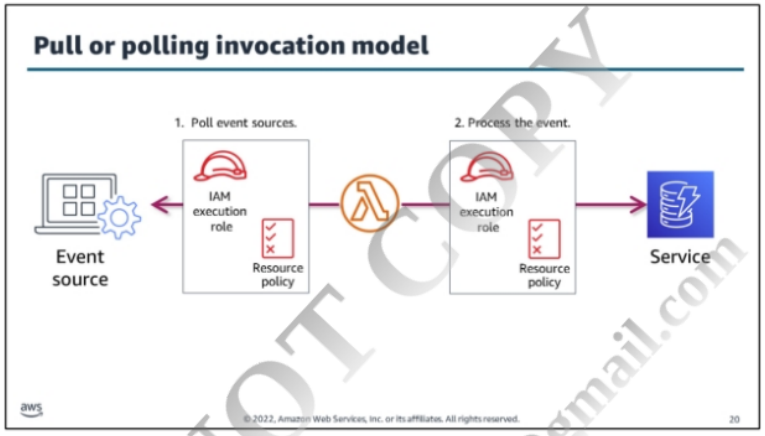
The entity that invokes your Lambda function must have permission to do so. The event source (such as Amazon S3 or a user-defined application) invokes your Lambda function by publishing events.

- You grant these event sources permissions to invoke your Lambda function by updating the access policy associated with your Lambda function.
- AWS Lambda provides the `AddPermission` API for this purpose.

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

For more information, see the following in the *AWS Lambda Developer Guide*:

- "AddPermission" (http://docs.aws.amazon.com/lambda/latest/dg/API_AddPermission.html)
- "AWS Lambda permissions" (http://docs.aws.amazon.com/lambda/latest/dg/intro-permission-model.html)

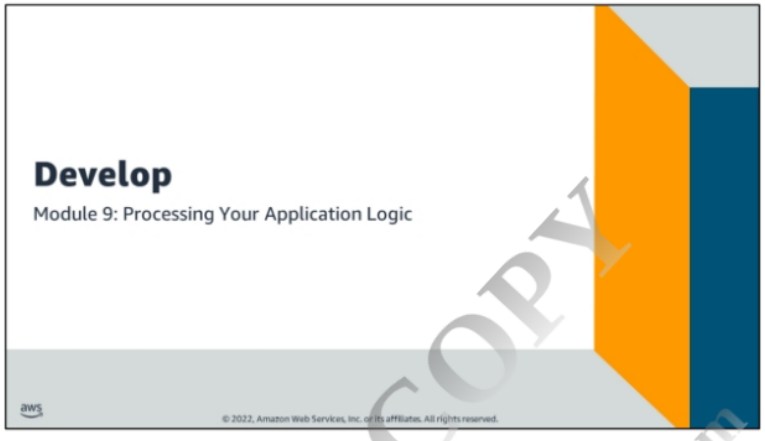


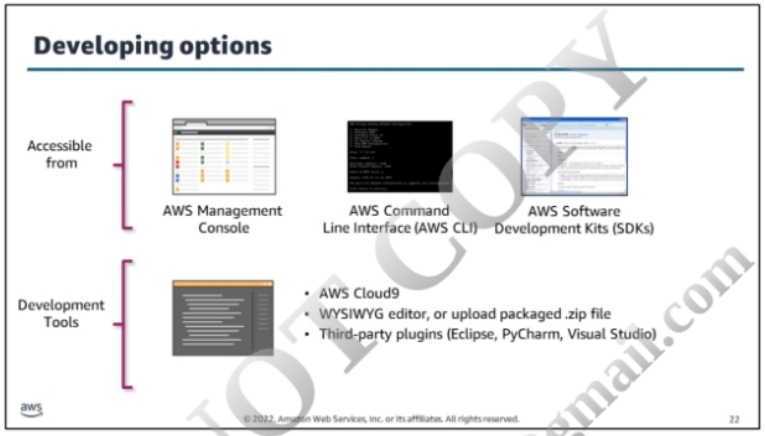
In the *pull* event model, AWS Lambda polls the event source and invokes your Lambda function when it detects an event. This model applies when AWS Lambda is used with streaming event sources such as Amazon Kinesis and Amazon DynamoDB Streams. For example, AWS Lambda polls your Kinesis data stream or DynamoDB stream and invokes your Lambda function when it detects new records on the stream. You must grant AWS Lambda permission to read from the stream. You do this by updating the IAM execution role associated with your Lambda function.

In this model, AWS Lambda manages the event source mapping. That is, it provides an API for you to create event source mappings that associate your Lambda function with a specific event source.

For more information, see "CreateEventSourceMapping" in the *AWS Lambda Developer Guide* (http://docs.aws.amazon.com/lambda/latest/dg/API_CreateEventSourceMapping.html).

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



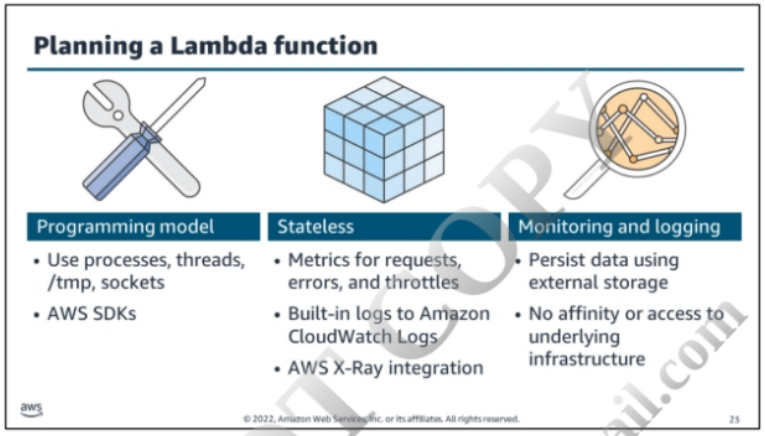


For more information on development tools, see:

- "Using Lambda with the Toolkit for Eclipse" in the *AWS Toolkit for Eclipse User Guide* (http://docs.aws.amazon.com/AWSToolkitEclipse/latest/GettingStartedGuide/lambda.html)
- AWS Lambda Developer Guide: AWS Toolkit for Visual Studio (https://docs.aws.amazon.com/lambda/latest/dg/csharp-package-toolkit.html)
- AWS Toolkit for PyCharm in the Jet Brains Marketplace (https://plugins.jetbrains.com/plugin/11349-aws-toolkit)

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic



Programming model

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

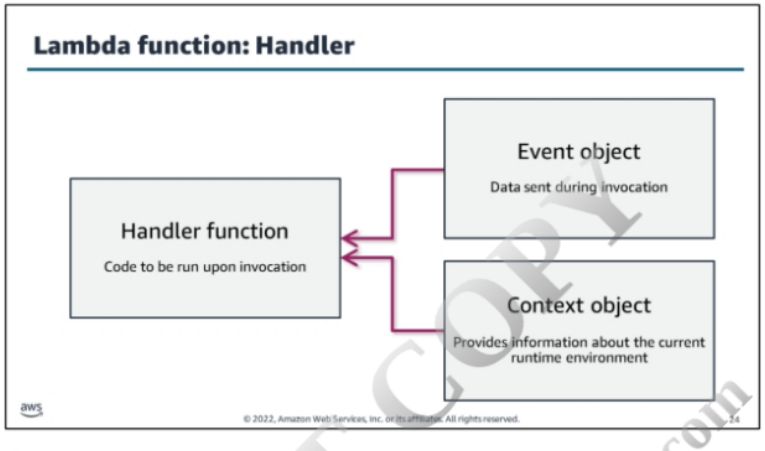
AWS SDKs provide language-specific APIs and manage many of the connection details. The runtimes for Python and Node.js have the SDKs built in, so you do not have to bundle those SDKs with your code.

Stateless

Keep your Lambda functions stateless. By doing this, AWS Lambda rapidly launches as many copies of the function as needed to scale to the rate of incoming events. Although the AWS Lambda programming model is stateless, your code can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.

Monitoring and logging best practices

- **Use AWS Lambda metrics and CloudWatch alarms** instead of creating or updating a metric from within your Lambda function code. It's a much more efficient way to track the health of your Lambda functions. You can catch issues early in the development process. For instance, you can configure an alarm based on the expected duration of your Lambda function runtime to address any bottlenecks or latencies attributable to your function code.
- **Use your logging library and AWS Lambda metrics and dimensions** to catch application errors (for example, `ERR`, `ERROR`, `WARNING`).



Handler
When a Lambda function is invoked, the handler code runs. The handler is a specific code method (Java, C#) or function (Node.js, Python) that you've created and included in your package. You specify the handler when creating a Lambda function. Each language that Lambda supports has its own requirements for how a function handler can be defined and referenced within the package.

A handler receives two arguments:

- **Event object**
  When your Lambda function is invoked in one of the supported languages, one of the parameters provided to your handler function is an event object. The event differs in structure and contents, depending on which event source created it. The contents of the event parameter include all of the data and metadata your Lambda function needs to drive its logic. For example, an event that the API Gateway creates will contain details related to the HTTPS request that was made by the API client. Details include path, query string, request body. By contrast, an event created by Amazon S3 when a new object is created includes details about the bucket and the new object.
- **Context object (optional)**
  Your Lambda function is also provided with a context object. The context object allows your function code to interact with the Lambda runtime environment. The contents and structure of the context object vary, based on the language runtime your Lambda function is using.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

However, at a minimum, it will contain the following elements:

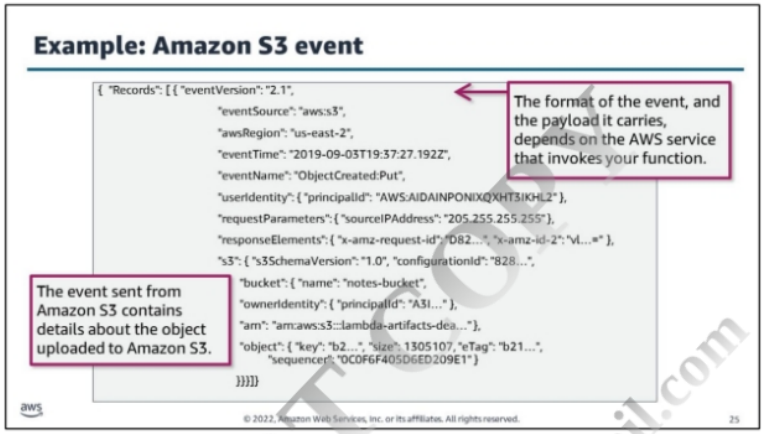
- **AWS RequestId** – Used to track specific invocations of a Lambda function (important for error reporting or when contacting AWS Support).
- **Timeout** – The amount of time in milliseconds that remain before your function timeout occurs. (Lambda functions can run a maximum of 900 seconds as of this publishing, but you can configure a shorter timeout.)
- **Logging** – Each language runtime provides the ability to stream log statements to CloudWatch Logs. The context object contains information about which CloudWatch Logs stream your log statements will be sent to.

Best practices:

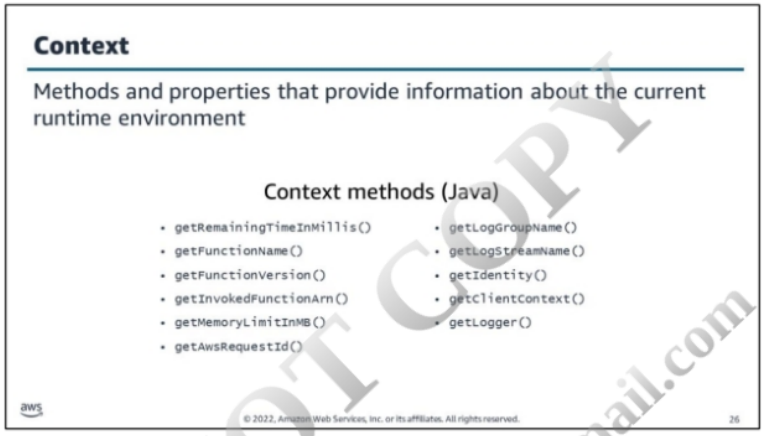
- **Separate the Lambda handler (entry point) from your core logic**
  You can make a more unit-testable function.

- **Avoid using recursive code**
  Avoid using recursive code in your Lambda function, wherein the function automatically calls itself until some arbitrary criteria is met. This could lead to unintended volume of function invocations and escalated costs. If you do accidentally do so, set the function concurrent runtime limit to 0 immediately to throttle all invocations to the function while you update the code.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Python
Context methods
- `get_remaining_time_in_millis` – Returns the number of milliseconds left before the lambda-run time limit is reached (default = 3 seconds).

Context properties
- `function_name` – The name of the Lambda function.
- `function_version` – The version of the function.
- `invoked_function_arn` – The Amazon Resource Name (ARN) that's used to invoke the function. Indicates if the invoker specified a version number or alias.
- `memory_limit_in_mb` – The amount of memory that's allocated for the function.
- `aws_request_id` – The identifier of the invocation request.
- `log_group_name` – The log group for the function.
- `log_stream_name` – The log stream for the function instance.

Java:
Context methods
- `getRemainingTimeInMillis()` – Returns the number of milliseconds left before the lambda-run time limit is reached (default = 15 minutes).
- `getFunctionName()` – Returns the name of the Lambda function.
- `getFunctionVersion()` – Returns the version of the function.

- `getInvokedFunctionArn()` – Returns the Amazon Resource Name (ARN) that's used to invoke the function. Indicates if the invoker specified a version number or alias.
- `getMemoryLimitInMB()` – Returns the amount of memory that's allocated for the function.
- `getAwsRequestId()` – Returns the identifier of the invocation request.
- `getLogGroupName()` – Returns the log group for the function.
- `getLogStreamName()` – Returns the log stream for the function instance.
- `getIdentity()` – (mobile apps) Returns information about the Amazon Cognito identity that authorized the request.
- `getClientContext()` – (mobile apps) Returns the client context that's provided to Lambda by the client application.
- `getLogger()` – Returns the logger object for the function.

.NET (C#)

Context properties

- `FunctionName` – The name of the Lambda function.
- `FunctionVersion` – The version of the function.
- `InvokedFunctionArn` – The Amazon Resource Name (ARN) that's used to invoke the function. Indicates if the invoker specified a version number or alias.
- `MemoryLimitInMB` – The amount of memory that's allocated for the function.
- `AwsRequestId` – The identifier of the invocation request.
- `LogGroupName` – The log group for the function.
- `LogStreamName` – The log stream for the function instance.
- `RemainingTime (TimeSpan)` – Returns the number of milliseconds left before the lambda-run time limit is reached (default = 15 minutes).
- Identity – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
- `ClientContext` – (mobile apps) Client context that's provided to Lambda by the client application.
- `Logger` – The logger object for the function.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

Example: Function handler (Python)

```
def handler_name(event, context):
        ...
return some_value
```

Example

```
def my_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'],
                                    event['last_name'])
    return {
        'message' : message
        }
```


Handler is a function that calls to start your AWS Lambda function. AWS Lambda passes any event data to the handler function as the first parameter. Your handler processes the event data and may invoke any other functions or methods in your code.

This example has one function called `my_handler`. The function returns a message that contains data from the event it received as input.

Example: Function handler (Java)

```
MyOutput output handlerName(MyEvent event, Context context)
{
    ...
}
```

Example

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Hello implements RequestHandler<Integer, String>{
    public String myHandler(int myCount, Context context) {
        return String.valueOf(myCount);
    }
}
```

 28

In this example, input is of type `Integer`, and output is of type `String`. If you package this code and dependencies and create your Lambda function, you specify `example.Hello::myHandler` (*package.class::method-reference*) as the handler.

In the example Java code, the first handler parameter is the input to the handler (`myHandler`), which can be any of the following:

- Event data (published by an event source, such as Amazon S3)
- Custom input you provide, such as an Integer object (as in this example) or any custom data object.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

Example: Function handler (C#)

```
myOutput HandlerName(MyEvent event, ILambdaContext context)
{
    ...
}
```

Example

```
using System.IO;
namespace Example
{
    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            // function logic
        }
    }
}
```

 29

When you create a Lambda function, you specify a handler that AWS Lambda can invoke when the service runs the function for you. You define a Lambda function handler as an instance or static method in a class. If you want access to the Lambda context object, it is available by defining a method parameter of type `ILambdaContext`. This parameter is an interface that you can use to access information about the current run, such as the following:

- Name of the current function
- Memory limit
- Runtime remaining
- Logging

In the example C# code, the first handler parameter is the input to the handler (MyHandler). The input can be event data (published by an event source, such as Amazon S3). It can also be custom input that you provide, such as a stream (as in this example), or any custom data object. The output is of type `Stream`.

Example: Lambda function (Python)

```
# Initializations outside of handler to make a more unit-testable function
dynamoDBResource = boto3.resource('dynamodb')
pollyClient = boto3.client('polly')
s3Client = boto3.client('s3')

def lambda_handler(event, context):
    # Extract the user parameters from the event and environment
    userId = event['userId']
    noteId = event['noteId']
    voiceId = event['voiceId']
    mp3Bucket = os.environ['MP3_BUCKET_NAME']
    ddbTable = os.environ['Notes_Table']
    # Get the note text from the database
    text = getNote(dynamoDBResource, ddbTable, userId, noteId)
    # Save an MP3 file locally with the output from polly
    filePath = createMP3File(pollyClient, text, voiceId, noteId)
    # Host the file on S3 that is accessed by a presigned url
    signedURL = hostFileOnS3(s3Client, filePath, mp3Bucket, userId, noteId)
    return signedURL
```

Pull info from event.

Pull info from environment variables.

Separate the Lambda handler from your core logic.

 10

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Environment variables

- Use
  - Adjust the behavior of a Lambda function without updating its code
- Storage
  - Stored in a function's version-specific configuration and are made available to the function
  - Encrypted at rest

```
Terminal                                                              X
>>aws lambda update-function-configuration --function-name my-function \
  --environment "Variables={BUCKET=my-bucket,KEY=file.txt}"
```

 11

Use environment variables to pass operational parameters to your function.
For example, if you are writing to an S3 bucket, instead of hardcoding the bucket name that you are writing to, configure the bucket name as an environment variable.

Using Lambda API

- CreateFunction
  - Create a Lambda function through a deployment package
  - Package type can be a ZIP or a container image
- UpdateFunctionCode
  - Update the code of the Lambda function
  - Code must be unpublished
- UpdateFunctionConfiguration
  - Modify the version-specific settings of the Lambda function
- Invoke
  - Invokes a Lambda function synchronously or asynchronously

 32

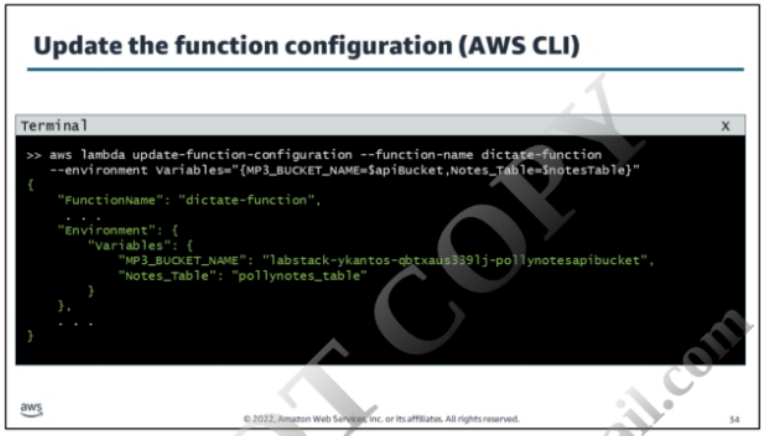
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Create a Lambda function (AWS CLI)

```
Terminal                                                                    X
>>  aws lambda create-function --function-name dictate-function --handler app.lambda_handler
--runtime python3.8 --role arn:aws:iam::563926481938:role/lambdaPollyRole --environment
Variables={TABLE_NAME=$notesTable} --zip-file fileb://dictate-function.zip
{    "LastUpdateStatus": "Successful",
     "FunctionName": "dictateABC",
     "LastModified": "2021-08-18T20:46:28.341+0000",
     "RevisionId": "a76cada1-ee56-4b0b-924e-d2c38fbdf9ab",
     "MemorySize": 128,
     "Environment": { "Variables": { "TABLE_NAME": "Notes" } },
     "State": "Active",
     "Version": "$LATEST",
     "Role": "arn:aws:iam::563926481938:role/lambdaPollyRole",
     "Timeout": 3,
     "Handler": "app.lambda_handler",
     "Runtime": "python3.8",
     "TracingConfig": { "Mode": "PassThrough" },
     "CodeSha256": "Oow63x1j1nCVbbfY2rJGTjAAclBrufctZxzjfXtOMOU=",
     "Description": "",
     "CodeSize": 1273,
     "FunctionArn": "arn:aws:lambda:us-west-2:563926481938:function:dictateABC",
     "PackageType": "Zip"}
```

 33

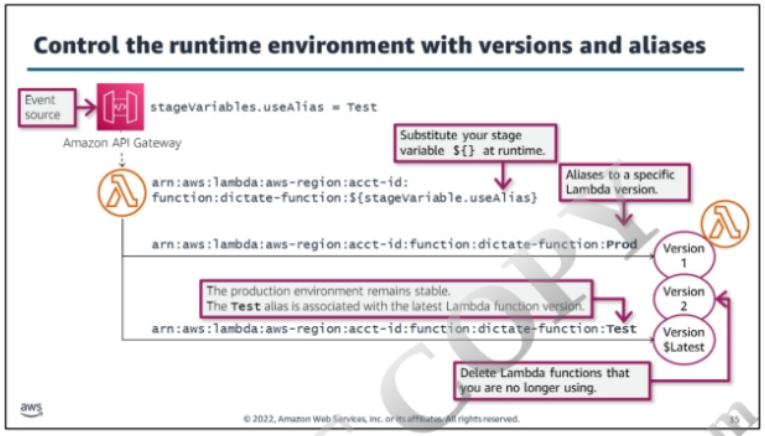
This example creates a Lambda function for python3.8 from a .zip file and specifies the handler function. The command uses the IAM role: `lambdaPollyRole.` An environment variable is set.



With `update-function-configuration`, you can change the configuration details of the AWS Lambda function, such as memory, timeout, environment variables, and more.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic



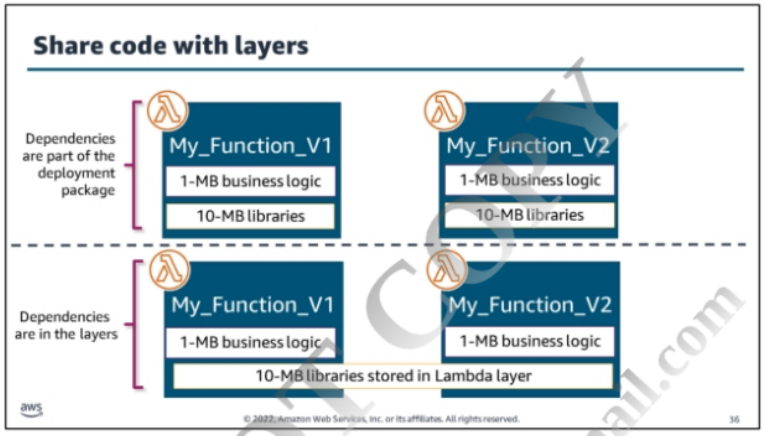
Lambda functions are versioned. When you publish a new version, the version number is incremented, and the earlier version is stored. You can publish one or more versions of your Lambda function. Each version has a unique ARN.

You can use aliases to invoke specific versions of the function. A Lambda alias is like a pointer to a specific function version. Aliases can be updated to point to other versions.

In the example shown on the slide, the lambda functions has version 1, version 2, and $latest. Each version has its unique ARN. Services need to use the correct version of the lambda function. This is where aliases can help. A Lambda alias can point to a function version that is configured with environment variables, and so avoiding hard-coding specific versions into your application. For example, the test team can use `arn:aws:lambda:aws-region:acct-id:function:dictate-function:Test`, where Text is an alias associated with a specific lambda version. And the production team can use `arn:aws:lambda:aws-region:acct-id:function:dictate-function:Prod`, where Prod is an alias associated with a specific lambda version.

AWS CLI command:

- Publish new version of a Lambda function:
  `aws lambda publish-version --function-name my-function`
- Create alias for the lambda function:
  `aws lambda create-alias --function-name my-function --name alias-name --function-version version-number --description " "`



A layer is a .zip file archive that contains code that is beyond the business logic, such as libraries, custom runtimes, and other dependencies. With layers, you can share such code across different versions of the same function or across multiple functions in a workload. The functions continue to have the same access to the dependencies but use less storage. Layers help keep your function deployment package small. That means that during a cold start, your deployment package can download faster.

A deployed layer is an immutable version. Each time a new layer is published, the version number increments. When you include a layer in a function, you specify the layer version that you want to use.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

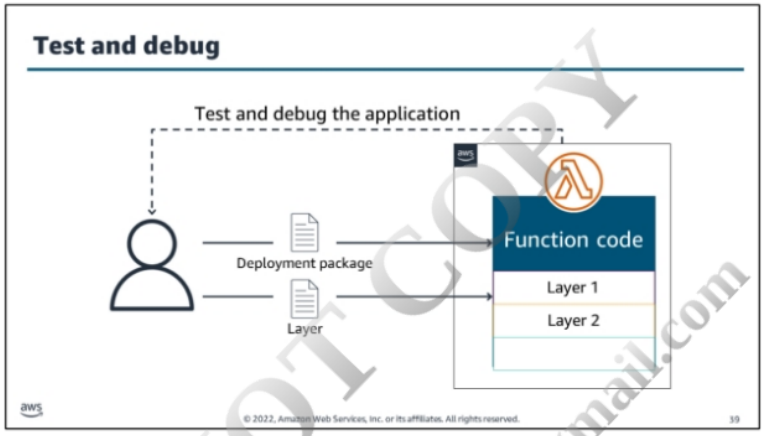
AWS Training and Certification Module 9: Processing Your Application Logic

Your libraries, dependencies to your code, or custom runtimes can be zipped as part of your build-and-deploy process of your Lambda function. However, this means that you cannot easily share those libraries, dependencies, or runtimes with other functions, other AWS accounts, or a third party. It also means that you cannot easily make changes and have those changes propagate to all the functions that use these items.

You can create one or more layers to package shared dependencies. You can include up to five layers per function, which count towards the standard Lambda deployment size limits.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification
Module 9: Processing Your Application Logic



Upload your deployment package. Separate shared dependencies and upload them as a layer. This way, you can test your function code more effectively.

Testing objectives

- Performance test your Lambda function for memory
- Load test your Lambda function for timeouts
- Understand Lambda limits

 40

Performance testing your Lambda function is a crucial part in making sure that you choose the optimal memory size configuration. Any increase in memory size initiates an equivalent increase in CPU available to your function. The memory usage for your function is determined per invoke and can be viewed in CloudWatch Logs.

Load test your Lambda function to determine an optimal timeout value. It is important to analyze how long your function runs. By doing this, you can better determine any problems with a dependency service that might increase the concurrency of the function beyond what you expect. This is especially important when your Lambda function makes network calls to resources that may not handle Lambda's scaling.

Understand Lambda limits. For more information, see "Lambda quotas" in the *AWS Lambda Developer Guide* (https://docs.aws.amazon.com/lambda/latest/dg/limits.html).

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Testing a Lambda function

AWS Management Console | AWS Command Line Interface (AWS CLI) | AWS Software Development Kits (AWS SDKs) | AWS Serverless Application Model (AWS SAM)

 41

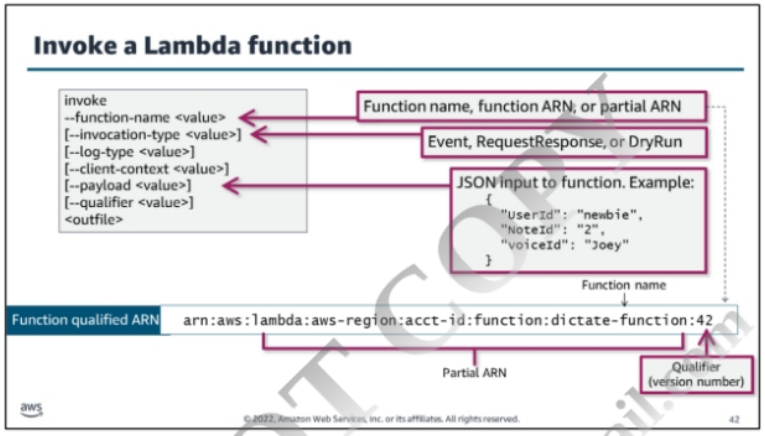
You can test Lambda functions in several ways:

AWS Management Console
- Event templates
- Custom payload
- Use CloudWatch Logs: log group and a log stream

AWS CLI
- Invoke the function or a version of it.
- Monitor through CloudWatch Logs.

AWS Software Development Kits
- AWS Toolkits with your IDE
- Run functions locally or remotely. Running locally requires configuring the runtime environment.

AWS Serverless Application Model (AWS SAM)
- AWS SAM with AWS toolkits and debuggers to test and debug locally.
- Consistently provision resources in multiple environments.



When you invoke a Lambda function, you provide some information. At a minimum, you need the function name, the function's ARN, or the function's partial ARN. If you are invoking a specific version number, qualify the ARN with the version prefix. Other parameters you can invoke the function with include:

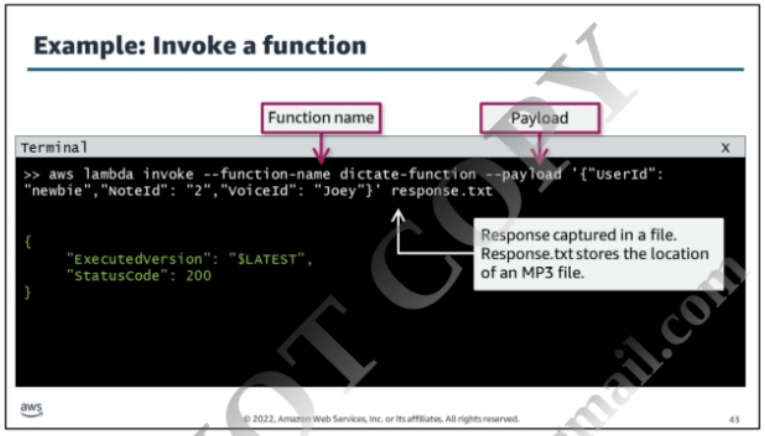
- **`invocation-type`** – Specifies how your function should run.
  - **`Event`**: Asynchronous
  - **`RequestResponse`**: (default invocation) Synchronous
  - **`DryRun`**: Do not run the function but perform some checks, such as checking if the caller has permissions and if the inputs are valid.
- **`log-type`** – If the invocation type is `RequestResponse`, set to Tail to include the execution log in the response. This populates the `x-amz-log-result` header with log data from your function.
- **`client-context`** – In a JSON format, pass client-specific information that you can then process through the context variable.
- **`Payload`** – A JSON input to the function, for example: `{"UserId": "newbie", "NoteId": "2", "voiceId": "Joey"}`.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

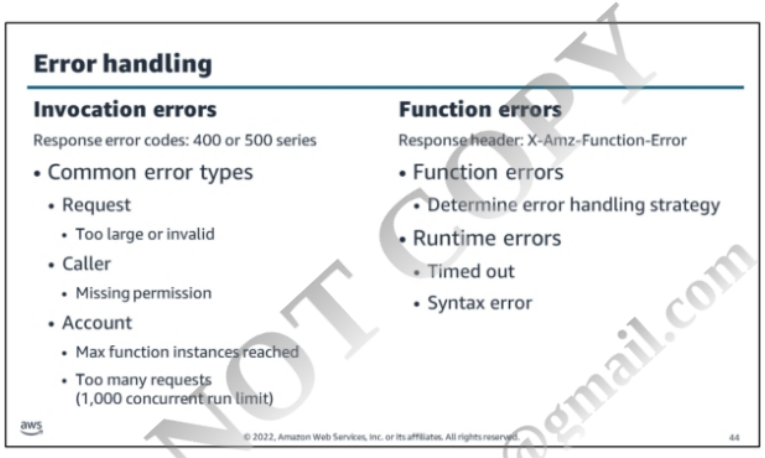
- **`Qualifier`** – Invoke a specific version of the Lambda function using the version number, or alias. To invoke the most current version of the Lambda function, qualify the functions ARN with $LATEST, or keep the ARN unqualified. For example, in this function qualified ARN, `arn:aws:lambda:aws-region:acct-id:function:dictate-function:42`, dictate-function is the function name, and 42 is version number qualifier.
- **`Outfile`** – File name where the output content will be saved.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

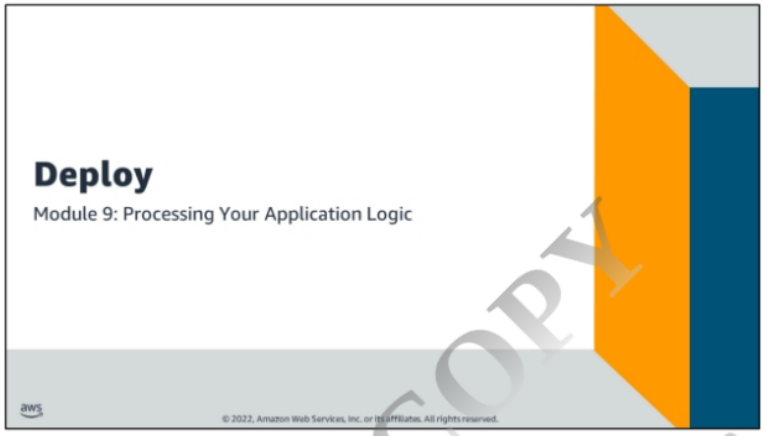
AWS Training and Certification Module 9: Processing Your Application Logic



Invoking a function can cause two types of errors:
- *Invocation errors* are caused when an invocation request is rejected before your function receives it.
- *Function errors* are caused by the function code or the runtime.

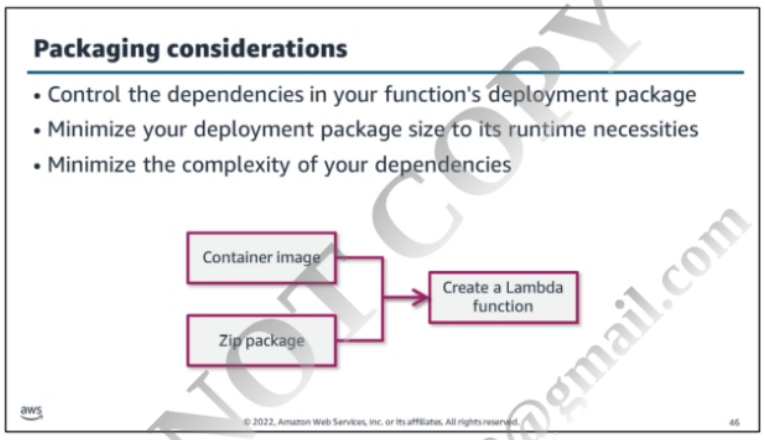
To avoid errors because of the account's 1,000 concurrent run limit and to minimize throttling, you can reserve part of that concurrency for your Lambda.

If your system depends on retries, make sure that your code is idempotent. This means that it can be retried multiple times without causing duplicate transactions or other unwanted side effects.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification
Module 9: Processing Your Application Logic



Lambda supports two types of deployment packages: container images and .zip file archives.

Control the dependencies in your function's deployment package
The AWS Lambda runtime environment contains a number of libraries, such as the AWS SDK for the Node.js and Python runtimes. For more information, see "Lambda runtimes" in the *AWS Lambda Developer Guide* (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

To enable the latest set of features and security updates, Lambda will periodically update these libraries. These updates may introduce subtle changes to the behavior of your Lambda function. To have full control of the dependencies that your function uses, package all of your dependencies with your deployment package.

Minimize your deployment package size to its runtime necessities
Minimizing your deployment package size reduces the amount of time that it takes for your deployment package to be downloaded and unpacked ahead of invocation. For functions authored in Java or .NET Core, avoid uploading the entire AWS SDK library as part of your deployment package. Instead, selectively depend on the modules that pick up components of the SDK that you need. Examples are DynamoDB, Amazon S3 SDK modules, and Lambda core libraries.

Minimize the complexity of your dependencies
Aim for simpler frameworks that load quickly on Execution Context startup. For example, use simpler Java dependency injection (IoC) frameworks such as Dagger or Guice, over more complex ones, such as Spring Framework.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

Deploy .zip file archives

Deploy .zip per language	Node.js, Python, Ruby	Java	.NET Core
What	ZIP archive consisting of your code and any dependencies	Zip file with all code and dependencies or standalone .jar	Zip file with all code or dependencies or a standalone .dll
How	Use npm , pip, or other build/packaging tool to install libraries.	Use Maven, Eclipse IDE plugins or other build/packaging tool	Use Nuget, Visual Studio plugins, or other build/packaging tool
Where	All dependencies must be at root level.	Compiled class and resource files at root level, required jars in /lib directory	All assemblies (.dll) at root level

Node.js, Ruby, and Python
To create a Lambda function, you first create a Lambda function deployment package, a .zip archive file that consists of your code and any dependencies. Also set the appropriate security permissions for the zip package.

Zip the directory *content* contained in the directory, not the directory itself. The contents of the .zip file are available as the current working directory of the Lambda function. For example: */project-dir/codefile.py/lib/YourLibraries*. In this case, you zip the content contained in */project-dir*.

Java
You decide whether your deployment package is a .zip file or a standalone jar. You can use any build and packaging tool that you are familiar with to create a deployment package.

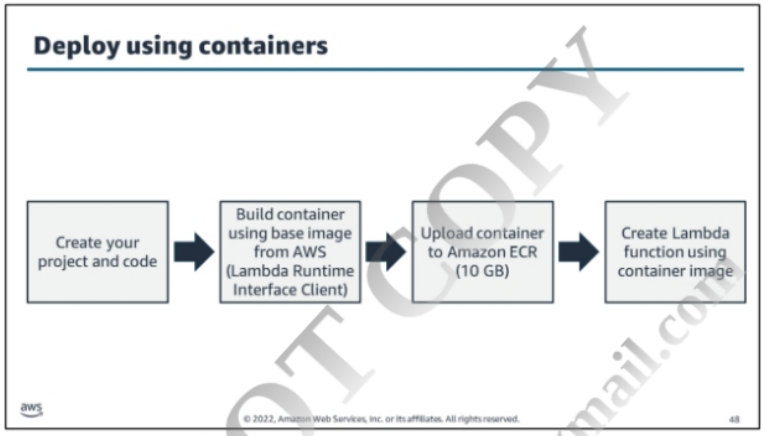
C#
A .NET Core Lambda deployment package is a .zip file of your function's compiled assembly along with all of its assembly dependencies. The package also contains the following:
- *proj*.deps.json file, which signals to the .NET Core runtime all of your function's dependencies
- *proj*.runtimeconfig.json file, which is used to configure the .NET Core runtime

The .NET `publish` command can create a folder with all of these files. However, the *proj*.runtimeconfig.json is not included by default because a Lambda project is typically configured to be a class library. To force the *proj*.runtimeconfig.json to be written as part of the publish process, pass in the following command line argument to the `publish` command:

```
/p:GenerateRuntimeConfigurationFiles=true
```


Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic



To create your container image, you can use any development tool that supports one of the following container image manifest formats:

- Docker image manifest V2
- Schema 2 (used with Docker version 1.10 and newer)
- OCI specifications (v1.0.0 and later)

For example, you can use the Docker CLI to build, test, and deploy your container images. Base images are provided for all supported Lambda runtimes, and even custom runtimes such as images based on Alpine or Debian Linux.

To work with Lambda, these images must implement the Lambda Runtime API. To make it easier to build your own base images, AWS provides Lambda Runtime Interface Clients implementing the Runtime API for all supported runtimes. These implementations are available through native package managers so that you can easily pick them up in your images. The implementations are shared with the community using an open source license.

Container images do not use layers. Instead, container images package the needed runtime, libraries, and other dependencies into the container image being built.

Lambda function limits

Resource	Quotas (as of Sept 2021)
Memory allocation	128 MB to 10,240 MB
Max runtime (Time out)	15 minutes
Burst concurrency	500–3000 (varies by Region)
Invocation payloads	Synchronous: 6 MB Asynchronous: 256 KB
Deployment package	Zip: 50 MB; unzipped: 250 MB; from console: 3MB Container image: 10 GB
/tmp directory storage	10 GB

Your functions' *concurrency* is the number of instances that serve requests at a given time. A sudden increase in the number of instances needed to fulfill the requested number of running functions is referred to as a *burst*.

For more information, see "Lambda quotas" in the *AWS Lambda Developer Guide* (https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html).

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification

Module 9: Processing Your Application Logic

Demo

Module 9: Processing Your Application Logic

DO NOT COPY
sameersheik68@gmail.com

Create a Lambda function with Cloud9

- Overview of the Lambda environment
- Using global/scope constraints – Environment variables and parameters
- Deploying versions, aliases, and Lambda layers
- Using Cloud9, SDK toolkits
- Monitoring using CloudWatch

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

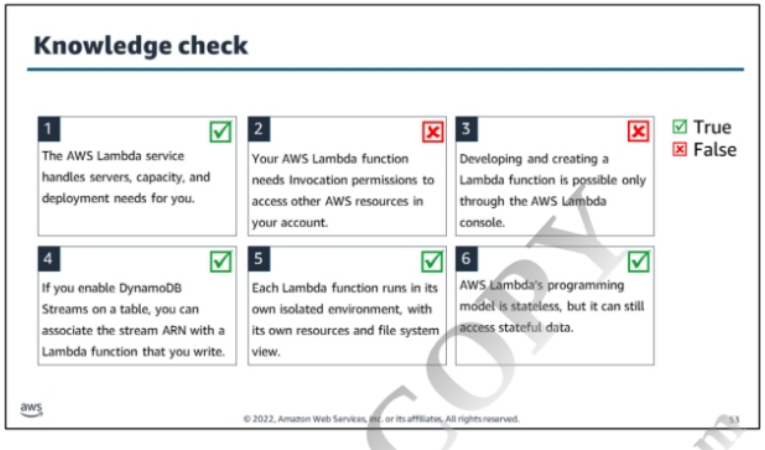
AWS Training and Certification Module 9: Processing Your Application Logic

Check your knowledge

Module 9: Processing Your Application Logic

DO NOT COPY
sameersheik68@gmail.com

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.

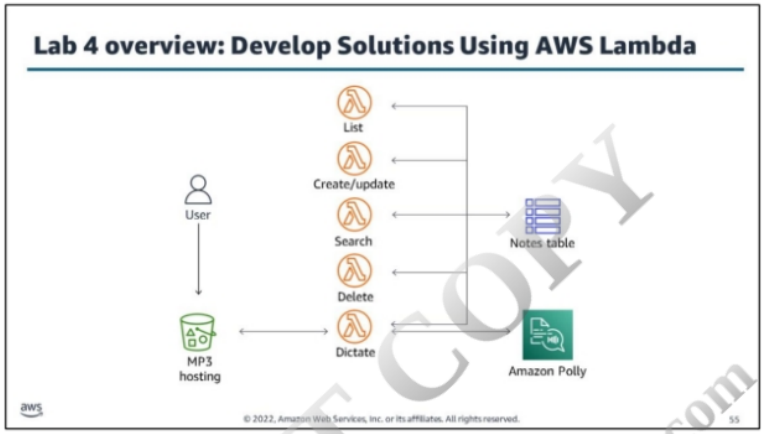


1. (True)
2. (False) Your AWS Lambda function needs **processing** permissions to access other AWS resources in your account.
3. (False) In addition to the AWS Lambda console, you can use the AWS CLI. You can also interact with your custom code using AWS SDKs. Further, both Eclipse and Visual Studio toolkits support the development of Lambda functions.
4. (True)
5. (True)
6. (True) Although AWS Lambda's programming model is stateless, your code can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 9: Processing Your Application Logic

DO NOT COPY
sameersheik68@gmail.com



Lab objectives

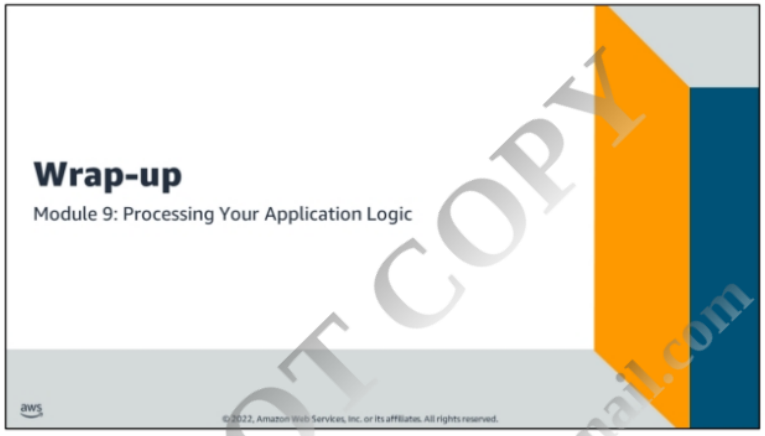
After completing this lab, you will be able to:

- Create AWS Lambda functions and interact programmatically by using the AWS SDK and the AWS CLI.
- Configure AWS Lambda functions to use environment variables and integrate with other services.
- Generate Amazon S3 presigned URLs by using the AWS SDK and verify access to bucket objects.
- Deploy the Lambda functions with .zip file archives, and test as needed.
- Invoke Lambda functions with multiple invocation options from the AWS Management Console and AWS CLI.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification

Module 9: Processing Your Application Logic



Module summary

You are now able to:

- Explore how AWS Lambda works
- Develop an AWS Lambda function using SDKs
- Configure triggers and permissions for Lambda functions
- Test, deploy, and monitor Lambda functions

 57

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification

Module 9: Processing Your Application Logic

Thank you

Corrections, feedback, or other questions?
Contact us at https://support.aws.amazon.com/#/contacts/aws-training.
All trademarks are the property of their owners.

 58