[ENGLISH]

Hello Cloud Gurus and welcome to this lesson

on handling Lambda event lifecycle and errors.

In this lesson, we'll cover the invocation lifecycle,

Lambda retries,

dead-letter queues,

Lambda destinations,

and my exam tips.

Now, when invoking a function,

you can invoke it either synchronously or asynchronously.

With synchronous invocation,

Lambda is gonna run the function, wait for a response,

and then it's gonna return the response.

That means that the service that is calling the function

will know if the function completed successfully or not.

And an example of this is API gateway, invoking a function

and then returning the error code to the caller.

Whereas with asynchronous invocation,

there is no acknowledgement at all, to let you know

that the invocation has successfully processed.

And this means that the service calling the function

is not going to be notified if the function

failed to complete successfully.

And an example of this is S3 invoking a function

when an object has been created.

It's not gonna wait to get notified, and it's not

gonna receive a notification that the function

successfully processed whatever it was gonna do afterwards

which might lead you to ask this question:

With asynchronous invocation,

how do we handle errors when they occur?

Well, first of all, we have Lambda retries.

So by default, if a function returns an error

then Lambda will automatically perform two retries.

Common errors includes something going wrong

in the function code,

or the function timing out before processing could complete.

And the approach that Lambda uses, is that it's gonna wait

one minute before the first retry.

If that fails, it's gonna wait two minutes

before the second retry.

So that is what you get by default.

But in addition to that,

you can also configure dead-letter queues.

And these are used to save failed invocations

for further processing.

They're associated with a particular version of a function

and they can even be configured to become an event source

for another function, allowing you to reprocess events.

But the one thing to remember about dead-letter queues

is that they handle failures only.

Now, when creating a dead-letter queue

we can use either SQS, which holds the failed events

in the queue until they're retrieved, or we can use SNS

which we can use to send notifications

about failed events to one or multiple destinations.

And in this example, SNS is used to fan

out notifications to multiple destinations,

so it's sending emails, triggering Lambda,

and also sending notifications to an HTTP endpoint.

But there's also another option

which is Lambda destinations.

And we can use Lambda destinations to optionally

configure Lambda to send invocation records

to another service.

If invocation is successful,

we can send a record to one destination when the invocation

was successfully processed.

For instance, we can send a record

to an SQS queue if successful.

And then on invocation failure, we could send a record

to another different destination when the invocation

and all of the retry attempts have failed.

For instance, on failure,

we could send the invocation record to SNS

and have it send an email

to the relevant team for investigation.

And here is an example of the kind of information

that gets included in the invocation record.

So it's all the details of the invocation,

like the timestamp, the request ID, function ARN,

request payload, and most importantly, the response payload.

And that's where we're gonna find the error message.

And this invocation failed to complete successfully.

Now, Lambda supports a few different destinations.

For example, you can use SQS to send a message

containing the error message

to an SQS queue for a developer to review.

Alternatively, you could use SNS to send a notification

to the support team using email or SMS.

You could trigger another Lambda function to

start an automatic error-handling process,

or a successful response could be used to trigger

an EventBridge event so that successful invocations attract.

And these are just a few different examples of how

you could use Lambda destinations.

So for the exam, be aware that if a function

returns an error, Lambda will automatically perform

two retries before giving up.

We can use dead-letter queues to save failed invocations

for further processing.

SNS or SQS are supported, and we can use SNS to fan

out notifications to multiple different destinations.

But Lambda destinations are a little bit more versatile.

The great thing about them is that they include

the invocation record so you can see

what's actually happened and you've got the opportunity

to configure one destination for sending

successful invocation records

and another one for sending unsuccessful invocations.

And it supports SQS, SNS, Lambda and EventBridge.

So that is all for this lesson.

If you have any questions, please let me know.

Otherwise, I will see you in the next one.

Thank you.

[SPANISH]

Hola Cloud Gurus y bienvenidos a esta lección.

sobre el manejo del ciclo de vida y los errores de los eventos de Lambda.

En esta lección, cubriremos el ciclo de vida de la invocación,

Lambda reintenta,

colas de mensajes fallidos,

destinos lambda,

y mis consejos para el examen.

Ahora, al invocar una función,

puede invocarlo de forma síncrona o asíncrona.

Con invocación sincrónica,

Lambda ejecutará la función, esperará una respuesta,

y luego devolverá la respuesta.

Eso significa que el servicio que está llamando a la función

sabrá si la función se completó con éxito o no.

Y un ejemplo de esto es la puerta de enlace API, invocando una función

y luego devolver el código de error a la persona que llama.

Mientras que con la invocación asíncrona,

no hay reconocimiento en absoluto, para hacerle saber

que la invocación ha procesado con éxito.

Y esto significa que el servicio que llama a la función

no va a ser notificado si la función

no se pudo completar con éxito.

Y un ejemplo de esto es S3 invocando una función

cuando se ha creado un objeto.

No va a esperar para recibir una notificación, y no es

va a recibir una notificación de que la función

procesó con éxito lo que fuera que iba a hacer después

lo que podría llevarte a hacerte esta pregunta:

Con invocación asíncrona,

¿Cómo manejamos los errores cuando ocurren?

Bueno, antes que nada, tenemos reintentos de Lambda.

Por defecto, si una función devuelve un error

luego, Lambda realizará automáticamente dos reintentos.

Los errores comunes incluyen algo que va mal

en el código de función,

o la función agotó el tiempo de espera antes de que pudiera completarse el procesamiento.

Y el enfoque que usa Lambda es que va a esperar

un minuto antes del primer reintento.

Si eso falla, esperará dos minutos.

antes del segundo reintento.

Así que eso es lo que obtienes por defecto.

Pero además de eso,

también puede configurar colas de mensajes fallidos.

Y estos se utilizan para guardar invocaciones fallidas.

para su posterior procesamiento.

Están asociados con una versión particular de una función.

e incluso se pueden configurar para convertirse en una fuente de eventos

para otra función, permitiéndole reprocesar eventos.

Pero lo único que debe recordar acerca de las colas de mensajes fallidos

es que solo manejan fallas.

Ahora, al crear una cola de mensajes fallidos

podemos usar SQS, que contiene los eventos fallidos

en la cola hasta que se recuperen, o podemos usar SNS

que podemos usar para enviar notificaciones

sobre eventos fallidos a uno o varios destinos.

Y en este ejemplo, SNS se usa para fan

enviar notificaciones a múltiples destinos,

por lo que está enviando correos electrónicos, activando Lambda,

y también enviar notificaciones a un punto final HTTP.

Pero también hay otra opción.

que es destinos Lambda.

Y podemos usar destinos Lambda para opcionalmente

configurar Lambda para enviar registros de invocación

a otro servicio.

Si la invocación tiene éxito,

podemos enviar un registro a un destino cuando la invocación

fue procesado con éxito.

Por ejemplo, podemos enviar un registro

a una cola SQS si tiene éxito.

Y luego, si falla la invocación, podríamos enviar un registro

a otro destino diferente cuando la invocación

y todos los reintentos han fallado.

Por ejemplo, en caso de falla,

podríamos enviar el registro de invocación a SNS

y haz que envíe un correo electrónico

al equipo correspondiente para su investigación.

Y aquí hay un ejemplo del tipo de información

que se incluye en el registro de invocación.

Así que son todos los detalles de la invocación,

como la marca de tiempo, el ID de la solicitud, el ARN de la función,

solicitud de carga útil y, lo que es más importante, la carga útil de respuesta.

Y ahí es donde vamos a encontrar el mensaje de error.

Y esta invocación no pudo completarse con éxito.

Ahora, Lambda admite algunos destinos diferentes.

Por ejemplo, puede usar SQS para enviar un mensaje

que contiene el mensaje de error

a una cola de SQS para que un desarrollador la revise.

Alternativamente, puede usar SNS para enviar una notificación

al equipo de soporte mediante correo electrónico o SMS.

Podría activar otra función de Lambda para

iniciar un proceso automático de gestión de errores,

o una respuesta exitosa podría usarse para desencadenar

un evento EventBridge para que las invocaciones exitosas atraigan.

Y estos son solo algunos ejemplos diferentes de cómo

podría usar destinos Lambda.

Entonces, para el examen, tenga en cuenta que si una función

devuelve un error, Lambda realizará automáticamente

dos reintentos antes de darse por vencido.

Podemos usar colas de mensajes fallidos para guardar invocaciones fallidas

para su posterior procesamiento.

SNS o SQS son compatibles, y podemos usar SNS para fan

enviar notificaciones a múltiples destinos diferentes.

Pero los destinos Lambda son un poco más versátiles.

Lo mejor de ellos es que incluyen

el registro de invocación para que puedas ver

lo que realmente sucedió y tienes la oportunidad

para configurar un destino para el envío

registros de invocaciones exitosas

y otro por enviar invocaciones fallidas.

Y es compatible con SQS, SNS, Lambda y EventBridge.

Así que eso es todo por esta lección.

Si tiene alguna pregunta, por favor hágamelo saber.

De lo contrario, te veré en la próxima.

Gracias.