[ENGLISH]

Hello, Cloud Gurus and welcome to this lesson,

which discusses Lambda performance tuning best practices.

We'll begin with a review

of how Lambda handles memory and CPU capacity.

Next, we'll take a look at the execution environment

and how we can optimize the static initialization phase

and then we'll finish off with some exam tips.

Now, when it comes to capacity allocation in Lambda,

we only have the ability to directly add more memory.

When you add more memory,

you automatically get more CPU capacity

but we cannot only add CPU capacity by itself.

So basically, we control CPU capacity

by configuring function memory.

By default, you get 128 megabytes,

which is only really suitable for very basic functions

and that is the bare minimum

but it's configurable up to around about 10 gigabytes.

And as are all of them, functions that import libraries

or interact with data that's loaded from other services

are definitely gonna need more than 128 megabytes.

So you'll need to test out your function and try

and work out how much memory is appropriate,

so how much memory it needs

in order to get the performance that you're looking for.

Adding memory is gonna improve the performance

of your function if it is either memory or CPU bound

because when we add more memory, we'll get more CPU as well.

And adding memory may also reduce the duration

that the function runs for.

Now, when we're considering performance best practices,

it's worth keeping in mind

how the Lambda execution environment operates.

So the first time your Lambda function is invoked

Lambda creates the execution environment

and it will also download the code that you've provided

to the execution environment,

and that's where the code is gonna run.

Next, it sets up the configuration,

including configuring memory, the runtime, for instance,

Node.js or Python,

and also setting up any other configuration.

The next step is static initialization,

and this is where it imports any dependencies.

It might run some static initialization code,

import libraries, or other dependencies.

Now, after the function is finished processing,

this execution environment is retained

for a short period of time.

And during this time, if the function happens

to be invoked again,

the execution environment is available to be reused.

So it doesn't need to go through all

of these downloading code configuration

and static initialization steps all over again.

But this execution environment

is only temporary, so you can't rely on it

to always be available

when you subsequently run your functions.

It's gonna be taken away at some point

but it's this static initialization phase

that's actually a large contributor to latency.

And if we could optimize this phase to reduce latency,

it's gonna help our functions to initialize faster,

which ultimately translates into better performance.

So if we want to optimize this static initialization phase,

there are actually three factors that we need to consider.

The first is the amount of code

that needs to run during this initialization phase.

The second is the function package size,

including any imported libraries,

dependencies and Lambda layers as well.

And the third is the performance of libraries

and any other services that require connections

to be set up.

For instance, if you're connecting to S3 or a database,

that's gonna take time and add latency as well.

But there's one thing that we can really easily implement

that's gonna make a big difference

and that is to be mindful

when we are importing libraries and SDKs

and where possible, we should avoid importing an entire SDK

if we don't need to.

So up here at the top of the screen,

we are importing the entire AWS SDK.

Instead, if we don't need everything, let's say

if we only needed DynamoDB, we could just import that.

And you might be surprised to find

that just that one small change alone

can make your function initialization complete up

to 125 milliseconds faster

than importing the entire AWS SDK.

Well, that's according to AWS anyway.

So for the exam, bear in mind that increasing the memory

on your function will also increase the CPU capacity

and it can reduce the duration that your function runs for.

Importing libraries and SDKs takes time

and can slow down the initialization phase of your function.

Therefore, only import what you need.

And that will help to avoid wasting time

importing an entire SDK if you don't need to.

So instead of doing this, importing the entire SDK,

just import what you need to.

So if you only need DynamoDB, then just import that.

Well, that is it for this lesson.

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,

que analiza las prácticas recomendadas de ajuste del rendimiento de Lambda.

Comenzaremos con una revisión.

de cómo Lambda maneja la memoria y la capacidad de la CPU.

A continuación, echaremos un vistazo al entorno de ejecución.

y cómo podemos optimizar la fase de inicialización estática

y luego terminaremos con algunos consejos para el examen.

Ahora, cuando se trata de la asignación de capacidad en Lambda,

solo tenemos la capacidad de agregar directamente más memoria.

Cuando agrega más memoria,

obtienes automáticamente más capacidad de CPU

pero no podemos solo agregar capacidad de CPU por sí mismo.

Básicamente, controlamos la capacidad de la CPU

mediante la configuración de la función de memoria.

Por defecto, obtienes 128 megabytes,

que solo es realmente adecuado para funciones muy básicas

y eso es lo minimo

pero es configurable hasta alrededor de 10 gigabytes.

Y como todas ellas, funciones que importan librerías

o interactuar con datos que se cargan desde otros servicios

definitivamente van a necesitar más de 128 megabytes.

Por lo tanto, deberá probar su función e intentar

y calcular cuánta memoria es apropiada,

Entonces, ¿cuánta memoria necesita?

para obtener el rendimiento que buscas.

Agregar memoria mejorará el rendimiento

de su función si está vinculada a la memoria o a la CPU

porque cuando agregamos más memoria, también obtendremos más CPU.

Y agregar memoria también puede reducir la duración

para el que se ejecuta la función.

Ahora, cuando estamos considerando las mejores prácticas de rendimiento,

vale la pena tenerlo en cuenta

cómo funciona el entorno de ejecución de Lambda.

Entonces, la primera vez que se invoca la función Lambda

Lambda crea el entorno de ejecución

y también descargará el código que has proporcionado

al entorno de ejecución,

y ahí es donde se ejecutará el código.

A continuación, establece la configuración,

incluyendo la configuración de la memoria, el tiempo de ejecución, por ejemplo,

Node.js o Python,

y también establecer cualquier otra configuración.

El siguiente paso es la inicialización estática,

y aquí es donde importa cualquier dependencia.

Podría ejecutar algún código de inicialización estático,

importar bibliotecas u otras dependencias.

Ahora, después de que la función haya terminado de procesarse,

este entorno de ejecución se conserva

por un corto periodo de tiempo.

Y durante este tiempo, si la función sucede

para ser invocado de nuevo,

el entorno de ejecución está disponible para ser reutilizado.

Por lo tanto, no es necesario pasar por todos

de estos configuración de código de descarga

y los pasos de inicialización estática de nuevo.

Pero este entorno de ejecución

es solo temporal, por lo que no puede confiar en él

para estar siempre disponible

cuando ejecute posteriormente sus funciones.

Se lo quitarán en algún momento.

pero es esta fase de inicialización estática

eso es en realidad un gran contribuyente a la latencia.

Y si pudiéramos optimizar esta fase para reducir la latencia,

ayudará a que nuestras funciones se inicialicen más rápido,

lo que finalmente se traduce en un mejor rendimiento.

Entonces, si queremos optimizar esta fase de inicialización estática,

en realidad hay tres factores que debemos considerar.

El primero es la cantidad de código.

que debe ejecutarse durante esta fase de inicialización.

El segundo es el tamaño del paquete de funciones,

incluidas las bibliotecas importadas,

dependencias y capas Lambda también.

Y el tercero es el rendimiento de las bibliotecas.

y cualquier otro servicio que requiera conexiones

que se creará.

Por ejemplo, si se está conectando a S3 o a una base de datos,

eso llevará tiempo y también agregará latencia.

Pero hay una cosa que podemos implementar fácilmente

eso va a hacer una gran diferencia

y eso es estar atento

cuando estamos importando bibliotecas y SDK

y donde sea posible, debemos evitar importar un SDK completo

si no es necesario.

Así que aquí arriba en la parte superior de la pantalla,

estamos importando todo el SDK de AWS.

En cambio, si no necesitamos todo, digamos

si solo necesitáramos DynamoDB, podríamos importarlo.

Y es posible que se sorprenda al encontrar

que solo ese pequeño cambio solo

puede hacer que la inicialización de su función se complete

a 125 milisegundos más rápido

que importar todo el SDK de AWS.

Bueno, eso es según AWS de todos modos.

Entonces, para el examen, tenga en cuenta que aumentar la memoria

en su función también aumentará la capacidad de la CPU

y puede reducir la duración de la ejecución de su función.

La importación de bibliotecas y SDK lleva tiempo

y puede ralentizar la fase de inicialización de su función.

Por lo tanto, importa solo lo que necesites.

Y eso ayudará a evitar perder el tiempo.

importar un SDK completo si no es necesario.

Entonces, en lugar de hacer esto, importar todo el SDK,

solo importa lo que necesites.

Entonces, si solo necesita DynamoDB, simplemente impórtelo.

Bueno, eso es todo por esta lección.

Para cualquier duda, por favor hágamelo saber.

De lo contrario, te veré en la próxima.

Gracias.