[ENGLISH]

Hello, Cloud Gurus, and welcome to this lesson,

which is going to cover the different storage patterns

that are available with Lambda.

And first of all,

we'll consider the stateless nature of Lambda

and what that actually means when we need to persist data.

Next, we'll take a look at the options available

and when we should use each option,

including /tmp,

options for storing Lambda libraries,

Lambda layers, S3, EFS,

and my exam tips as well.

Now, as you know, Lambda functions are stateless.

And that means that you can't permanently store any data

within the function itself.

For instance, think about an application that needs

to store session data or customer data, et cetera.

Lambda is also ephemeral,

which means that the functions only run

for a short period of time.

So Lambda should not be used

for applications that need to run

for longer than 15 minutes.

For instance, think about a database application

or a web server that needs to be up and running 24/7.

So if you would like to persist any data,

then the function needs to interact

either with a database or a data store

in order to persist the data.

For instance, save it to S3, EFS,

or DynamoDB, for example.

Now, there are a few different options for storing data,

and it all depends on your use case.

And you will need to understand the differences

between each option and when to use each one.

And there are options that come native within Lambda.

For instance, /temp, Lambda layers,

or even storing libraries within a deployment package.

And there's also some external storage options as well.

So we've got S3, EFS, and any other storage option

that you can think of.

For instance, RDS or DynamoDB.

And we're gonna look at these options

in a little bit more detail

so that you can understand when to use each one,

beginning with /tmp.

So this is temporary storage.

It's provided within the execution environment

of the Lambda function.

By default, you get 512 megs,

configurable up to 10 gigabytes.

It behaves like a cached file system,

and data can actually be accessed

by multiple invocations of your function

that are sharing the same execution environment.

And that's gonna allow you to optimize performance

for multiple invocations of the same function.

However, with /tmp, the data is not persistent,

and it's only gonna be available

for the lifetime of the execution environment.

So it's not a place to be restoring permanent data.

Instead, it's more like a temporary storage space,

where data can be stored while the function is executing.

And you should assume that the data is gonna be lost

as soon as your execution is completed.

So what options do we have for more persistent storage?

What if we need to include libraries within our function?

Well, additional libraries that are needed

by the function can be included

in your Lambda deployment package.

In other words, the ZIP file that contains your code.

And this is gonna be fine for small libraries.

However, when we do that,

it increases the size of the deployment package.

And that has a knock-on effect

because a larger deployment package

is actually gonna slow down your deployment,

and it's gonna make everything run a little bit slower

because each time you deploy the function,

it's just gonna take a little bit longer.

So if you want to keep your deployment package small

and have the best performance for your function,

then consider using Lambda layers.

And this is actually a best practice

if you want to add libraries and SDKs

that can be referenced by multiple functions.

And also if you have large dependencies.

For instance, if you want

to import image manipulation libraries, graphics libraries,

or even a particular version

of the AWS SDK.

So if you have a large dependency,

then you are definitely gonna get better performance

by using a Lambda layer.

And deployment is gonna be faster

because the ZIP file containing your code is smaller

because your dependencies are included

in the layer instead of being within the deployment package.

However, using a Lambda layer is not going to be dynamic.

And what I mean by that is that

if you want to change the version of the library

that you included, then you can't update it dynamically.

You can't go into the layer and start updating it.

Instead, you'll need to create a new layer,

and then reference that from within your function.

So now let's move on

to the persistent storage options available

outside of Lambda.

So first of all, we've got S3.

Now, remember that S3 is object storage only.

So it allows you to store and retrieve objects,

and it's not a file system.

So that means that you cannot append data

to the objects that are stored in S3.

So you can't directly open and start writing data

to the objects you've stored in S3.

Instead, if you want to change the data,

you will need to upload a completely new object.

So that introduces a constraint on

how you can operate with S3.

It might work for some applications,

but other applications are just not gonna be able

to work like that.

And that brings us to EFS, or Elastic File System,

which is basically a shared file system.

It's based on NFS, or the Network File System, protocol,

and it acts just like a file system.

So data is persistent, and it can be dynamically updated.

For instance, you can open up a file

and start writing to it and modifying it.

The EFS file system needs to be mounted by the function

when the execution environment is created.

And then after the file system is mounted,

it can be shared across invocations of your function.

And when you create an EFS file system,

it's within a VPC.

So in order to use EFS,

your Lambda function must be associated

with the same VPC as your EFS file system.

So for the exam,

just be aware that there are a few different options

when it comes to storing data with Lambda.

Some are native to Lambda, and some are external.

So firstly, we have /tmp, which is for temporary data.

You can store up to 10 gigabytes.

You get dynamic read and writes,

and the data can be shared by other invocations

of your function within the same execution environment,

but it is still temporary.

Next, we have Lambda layers.

And these are great for libraries and SDKs.

You can store up to 50 megabytes zipped

or 250 megabytes unzipped.

However, that limit applies to all the files

that you upload, including the deployment package

and custom runtimes.

If you need to make an update,

you'll need to create a new layer

and reference it within your function.

And the layer can be shared across execution environments.

Then for the external storage options, there's S3,

which is, of course, persistent data.

There's no size limit because it's elastic.

However, it only allows you to store and retrieve objects.

You can't open up a file and start modifying it.

If you need to change something,

you need to upload a brand-new version.

And S3 can be shared across execution environments.

And then, finally, we have EFS,

which behaves just like a regular file system.

It's persistent data.

It's also elastic, so no size limit.

And this is a great option

if you need dynamic read and writes

because you can open files and append them and modify them.

And this one is also shared across execution environments.

So that is it for this lesson.

Any questions, please let me know.

Otherwise, please join me for the next one. Thank you.

[SPANISH]

Hola, Cloud Gurus, y bienvenidos a esta lección.

que va a cubrir los diferentes patrones de almacenamiento

que están disponibles con Lambda.

Y en primer lugar,

consideraremos la naturaleza sin estado de Lambda

y lo que eso significa realmente cuando necesitamos conservar los datos.

A continuación, veremos las opciones disponibles.

y cuando debemos usar cada opción,

incluyendo /tmp,

opciones para almacenar bibliotecas Lambda,

Capas lambda, S3, EFS,

y mis consejos de examen también.

Ahora, como sabe, las funciones de Lambda no tienen estado.

Y eso significa que no puede almacenar permanentemente ningún dato

dentro de la función misma.

Por ejemplo, piense en una aplicación que necesita

para almacenar datos de sesión o datos de clientes, etcétera.

Lambda es también efímero,

lo que significa que las funciones solo se ejecutan

por un corto periodo de tiempo.

Entonces Lambda no debe usarse

para aplicaciones que necesitan ejecutarse

durante más de 15 minutos.

Por ejemplo, piense en una aplicación de base de datos

o un servidor web que necesita estar en funcionamiento las 24 horas del día, los 7 días de la semana.

Entonces, si desea conservar algún dato,

entonces la función necesita interactuar

ya sea con una base de datos o un almacén de datos

para persistir los datos.

Por ejemplo, guárdelo en S3, EFS,

o DynamoDB, por ejemplo.

Ahora, hay algunas opciones diferentes para almacenar datos,

y todo depende de su caso de uso.

Y tendrás que entender las diferencias.

entre cada opción y cuándo usar cada una.

Y hay opciones que vienen de forma nativa dentro de Lambda.

Por ejemplo, /temp, capas Lambda,

o incluso almacenar bibliotecas dentro de un paquete de implementación.

Y también hay algunas opciones de almacenamiento externo.

Así que tenemos S3, EFS y cualquier otra opción de almacenamiento

que puedas pensar.

Por ejemplo, RDS o DynamoDB.

Y vamos a ver estas opciones

con un poco mas de detalle

para que puedas entender cuándo usar cada uno,

comenzando con /tmp.

Así que esto es almacenamiento temporal.

Se proporciona dentro del entorno de ejecución.

de la función Lambda.

Por defecto, obtienes 512 megas,

Configurable hasta 10 gigas.

Se comporta como un sistema de archivos en caché,

y se puede acceder a los datos

por múltiples invocaciones de su función

que comparten el mismo entorno de ejecución.

Y eso te permitirá optimizar el rendimiento.

para múltiples invocaciones de la misma función.

Sin embargo, con /tmp, los datos no son persistentes,

y solo estará disponible

durante la vida útil del entorno de ejecución.

Por lo tanto, no es un lugar para restaurar datos permanentes.

En cambio, es más como un espacio de almacenamiento temporal,

donde se pueden almacenar datos mientras se ejecuta la función.

Y debe asumir que los datos se perderán

tan pronto como se complete su ejecución.

Entonces, ¿qué opciones tenemos para un almacenamiento más persistente?

¿Qué pasa si necesitamos incluir bibliotecas dentro de nuestra función?

Bueno, bibliotecas adicionales que se necesitan

por la función puede ser incluido

en su paquete de implementación de Lambda.

En otras palabras, el archivo ZIP que contiene su código.

Y esto va a estar bien para bibliotecas pequeñas.

Sin embargo, cuando hacemos eso,

aumenta el tamaño del paquete de implementación.

Y eso tiene un efecto dominó

porque un paquete de implementación más grande

en realidad va a ralentizar su despliegue,

y hará que todo funcione un poco más lento

porque cada vez que implementas la función,

solo tomará un poco más de tiempo.

Entonces, si desea mantener su paquete de implementación pequeño

y tener el mejor rendimiento para su función,

luego considere usar capas Lambda.

Y esto es en realidad una mejor práctica

si desea agregar bibliotecas y SDK

que puede ser referenciado por múltiples funciones.

Y también si tienes grandes dependencias.

Por ejemplo, si quieres

para importar bibliotecas de manipulación de imágenes, bibliotecas de gráficos,

o incluso una versión particular

del SDK de AWS.

Así que si tienes una gran dependencia,

entonces definitivamente obtendrás un mejor rendimiento

mediante el uso de una capa Lambda.

Y el despliegue será más rápido

porque el archivo ZIP que contiene tu código es más pequeño

porque sus dependencias están incluidas

en la capa en lugar de estar dentro del paquete de implementación.

Sin embargo, usar una capa Lambda no será dinámico.

Y lo que quiero decir con eso es que

si desea cambiar la versión de la biblioteca

que incluiste, entonces no puedes actualizarlo dinámicamente.

No puede entrar en la capa y comenzar a actualizarla.

En su lugar, deberá crear una nueva capa,

y luego haga referencia a eso desde dentro de su función.

Así que ahora sigamos adelante

a las opciones de almacenamiento persistente disponibles

fuera de Lambda.

En primer lugar, tenemos S3.

Ahora, recuerde que S3 es solo almacenamiento de objetos.

Por lo tanto, le permite almacenar y recuperar objetos,

y no es un sistema de archivos.

Eso significa que no puede agregar datos

a los objetos que se almacenan en S3.

Por lo tanto, no puede abrir y comenzar a escribir datos directamente

a los objetos que ha almacenado en S3.

En cambio, si desea cambiar los datos,

deberá cargar un objeto completamente nuevo.

Entonces eso introduce una restricción en

cómo se puede operar con S3.

Podría funcionar para algunas aplicaciones,

pero otras aplicaciones simplemente no podrán

para trabajar así.

Y eso nos lleva a EFS, o Elastic File System,

que es básicamente un sistema de archivos compartido.

Se basa en NFS, o el sistema de archivos de red, protocolo,

y actúa como un sistema de archivos.

Por lo tanto, los datos son persistentes y se pueden actualizar dinámicamente.

Por ejemplo, puede abrir un archivo

y empezar a escribir en él y modificarlo.

El sistema de archivos EFS debe ser montado por la función

cuando se crea el entorno de ejecución.

Y luego, después de montar el sistema de archivos,

se puede compartir entre las invocaciones de su función.

Y cuando crea un sistema de archivos EFS,

está dentro de una VPC.

Entonces, para usar EFS,

su función Lambda debe estar asociada

con la misma VPC que su sistema de archivos EFS.

Así que para el examen,

solo tenga en cuenta que hay algunas opciones diferentes

cuando se trata de almacenar datos con Lambda.

Algunos son nativos de Lambda y otros son externos.

En primer lugar, tenemos /tmp, que es para datos temporales.

Puede almacenar hasta 10 gigabytes.

Obtiene lecturas y escrituras dinámicas,

y los datos pueden ser compartidos por otras invocaciones

de su función dentro del mismo entorno de ejecución,

pero sigue siendo temporal.

A continuación, tenemos capas Lambda.

Y estos son excelentes para bibliotecas y SDK.

Puede almacenar hasta 50 megas comprimidos

o 250 megas descomprimidos.

Sin embargo, ese límite se aplica a todos los archivos.

que cargue, incluido el paquete de implementación

y tiempos de ejecución personalizados.

Si necesita hacer una actualización,

tendrás que crear una nueva capa

y referenciarlo dentro de su función.

Y la capa se puede compartir entre entornos de ejecución.

Luego, para las opciones de almacenamiento externo, está S3,

que es, por supuesto, datos persistentes.

No hay límite de tamaño porque es elástico.

Sin embargo, solo le permite almacenar y recuperar objetos.

No puede abrir un archivo y comenzar a modificarlo.

Si necesitas cambiar algo,

necesita cargar una nueva versión.

Y S3 se puede compartir entre entornos de ejecución.

Y luego, finalmente, tenemos EFS,

que se comporta como un sistema de archivos regular.

Son datos persistentes.

También es elástico, por lo que no hay límite de tamaño.

Y esta es una gran opción.

si necesita lectura y escritura dinámicas

porque puede abrir archivos, agregarlos y modificarlos.

Y este también se comparte entre entornos de ejecución.

Eso es todo por esta lección.

Para cualquier duda, por favor hágamelo saber.

De lo contrario, por favor únase a mí para el próximo. Gracias.