Aplicaciones serveless en Amazon AWS

TRABAJO GRUPAL

Emanuel Amalfitano | Aplicaciones Distribuidas en Internet | 14-01-2022

¿Qué es un Serveless?

La computación sin servidor (o serverless para abreviar) es un modelo de ejecución en el que el proveedor en la nube (AWS, Azure o Google Cloud) es responsable de ejecutar un fragmento de código mediante la asignación dinámica de los recursos. Y cobrando solo por la cantidad de recursos utilizados para ejecutar el código. El código, generalmente, se ejecuta dentro de contenedores sin estado que pueden ser activados por una variedad de eventos que incluyen solicitudes HTTP, eventos de base de datos, servicios de colas, alertas de monitoreo, carga de archivos, eventos programados (trabajos cron), etc. El código que se envía a al proveedor en la nube para la ejecución es generalmente en forma de una función. Por lo tanto, serverless a veces se denomina “Funciones como servicio” o “FaaS”.

Ventajas de las aplicaciones serverless

1. **Elimina la administración de infraestructuras TI**
   Las aplicaciones Serverless ayudan a eliminar de raíz todo lo que tenga que ver con la administración del entorno informático, como aprovisionar de servidores a los distintos equipos de trabajo, la implementación de clústeres y parches, aplicar actualizaciones, además del mantenimiento de los sistemas operativos, entre otros.
2. **Reducen los costos a nivel organizativo**
   Permite centrar los recursos en otras áreas que pueden ayudar a darle más valor e innovación al producto final.
3. **Permite enfocar el trabajo en lo que importa**
   Permite centrarse en el desarrollo del proyecto gracias a que permite despreocuparse de la administración del servidor ni de posibles interrupciones.
4. **Contribuye con la escalabilidad del negocio**
   Como las apps serverless hacen uso de una arquitectura externa a una empresa, se puede aprovechar para beneficiarse solo de las funciones que se necesite para el desarrollo.

AWS Lambda

AWS Lambda (o Lambda para abreviar) es un servicio de computación sin servidor proporcionado por AWS.

Soporta los siguientes entornos de ejecución:

- Node.js: v8.10 y v6.10
- Java 8
- Python: 3.6 y 2.7
- .NET Core: 1.0.1 y 2.0
- Go 1.x
- Ruby 2.5
- Rust

Cada función corre en un contenedor con Amazon Linux AMI a 64-bit. Y el entorno de ejecución cuenta con:

- Memoria: 128MB - 3008MB, en incrementos de 64 MB
- Espacio en disco efímero: 512MB
- Duración máxima de ejecución: 900 segundos
- Tamaño del paquete comprimido: 50MB
- Tamaño del paquete descomprimido: 250MB

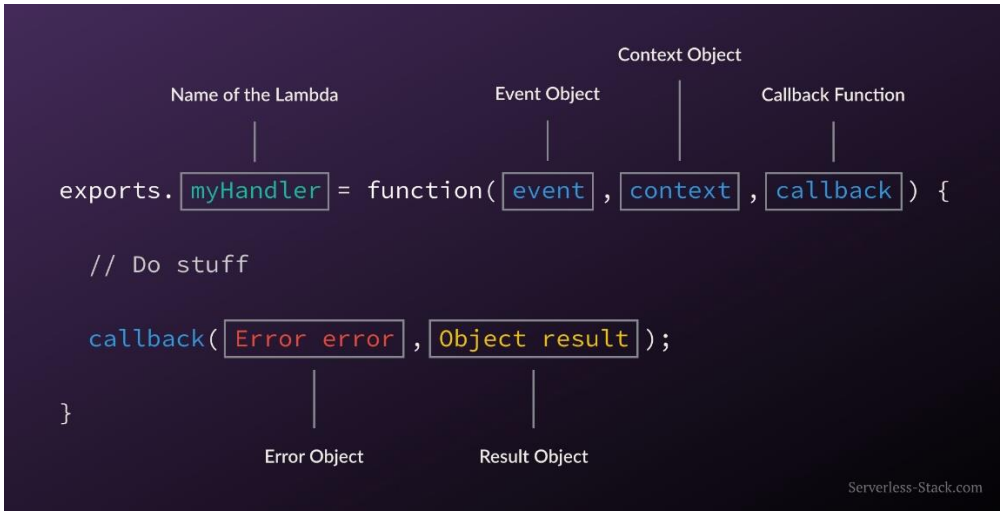
La CPU no se menciona como parte de las especificaciones del contenedor ya que se debe a que no se puede controlar la CPU directamente. A medida que aumenta la memoria, la CPU también aumenta.

El espacio en disco efímero está disponible en forma de directorio en /tmp. Sólo puedes usar este espacio para el almacenamiento temporal, ya que las invocaciones posteriores no tendrán acceso a este. Hablaremos un poco más sobre la naturaleza sin estado de las funciones Lambda más adelante.

La duración de ejecución significa que tu función Lambda puede ejecutarse durante un máximo de 900 segundos o 15 minutos. Esto significa que Lambda no está destinado a procesos de larga ejecución.

El tamaño de paquete se refiere a todo el código necesario para ejecutar tu función. Este incluye cualquier dependencia (directorio node_modules/ en el caso de Node.js) que tu función necesitará importar. Hay un límite de 250MB para el paquete descomprimido y 50MB una vez comprimido. Daremos un vistazo al proceso de empaquetado más adelante.

Funciones



myHandler es el nombre de nuestra función Lambda. El objeto event contiene toda la información sobre el evento que desencadenó este Lambda. En el caso de una solicitud HTTP, será la información específica sobre esta. El objeto context contiene información sobre el entorno donde se está ejecutando nuestra función Lambda. Después de hacer todo el trabajo dentro de nuestra función Lambda, simplemente llamamos a la función callback con los resultados (o el error) y AWS responderá a la solicitud HTTP con él.

Empaquetando funciones

Las funciones Lambda se deben empaquetar y enviar a AWS. Este suele ser un proceso que comprime la función y todas sus dependencias y la carga en un contenedor S3. Se le dice a AWS que deseas utilizar este paquete cuando se realice un evento específico.

Modelo de ejecución

El contenedor (y los recursos que usa) para ejecutar la función son administrados completamente por AWS. Se activa cuando se produce un evento y se apaga si no se está utilizando. Si se realizan solicitudes adicionales mientras se está sirviendo el evento original, se abre un nuevo contenedor para atender la solicitud. Esto significa que, si estamos experimentando un pico de uso, el proveedor de la nube simplemente crea varias instancias del contenedor con nuestra función para atender esas solicitudes.

Esto tiene algunas implicaciones interesantes. En primer lugar, las funciones son efectivamente sin estado. En segundo lugar, cada solicitud (o evento) es atendida por una sola instancia de una función Lambda. Esto significa que no vas a manejar solicitudes concurrentes en tu código. AWS abre un contenedor cada vez que hay una nueva solicitud. Hace algunas optimizaciones y se mantendrá en el contenedor durante unos minutos (5 a 15 minutos dependiendo de la carga) para que pueda responder a las solicitudes posteriores sin un arranque en frío.

El modelo de ejecución anterior hace que Lambda funcione efectivamente sin estado. Esto significa que cada vez que la función Lambda es desencadenada por un evento, se invoca en un entorno completamente nuevo. No tiene acceso al contexto de ejecución del evento anterior.

Sin embargo, debido a la optimización mencionada anteriormente, la función Lambda actual se invoca sólo una vez por instanciación de contenedor. Recordemos que nuestras funciones se ejecutan dentro de contenedores. Entonces, cuando se invoca una función por primera vez, todo el código en nuestra función es ejecutado y luego es invocada. Si el contenedor aún está disponible para solicitudes posteriores, se invocará la función y no el código que lo envuelve.

Por ejemplo, el método createNewDbConnection a continuación se llama una vez por instanciación de contenedor y no cada vez que se invoca la función Lambda. La función myHandler por otra parte se llama en cada invocación.

```
var dbConnection = createNewDbConnection();

exports.myHandler = function(event, context, callback) {
var result = dbConnection.makeQuery();
callback(null, result);
};
```

Este efecto de almacenamiento en caché de los contenedores también se aplica al directorio /tmp del que hablamos anteriormente. Está disponible siempre y cuando el contenedor esté en caché. Esta no es una forma muy confiable de hacer que nuestras funciones Lambda tengan estado. Esto se debe a que simplemente no controlamos el proceso subyacente mediante el cual Lambda es invocado o sus contenedores se almacenan en caché.

Precios

Las funciones Lambda se facturan solo por el tiempo que lleva ejecutar tu función. Y se calcula desde el momento en que comienza a ejecutarse hasta que retorna o termina. Se redondea a los 100ms más cercanos.

Hay que tener en cuenta que, mientras AWS pueda mantener el contenedor con la función Lambda después de que se haya completado, no se cobrará por esto.

La tarifa gratuita de Lambda incluye 1 millón de solicitudes gratuitas por mes y 400,000 GB-segundos de tiempo de cómputo por mes. Más allá de esto, cuesta $0.20 por 1 millón de solicitudes y $0.00001667 por cada GB-segundo. Los segundos en GB se basan en el consumo de memoria de la función Lambda.

Referencias

https://codster.io/blog/5-ventajas-de-las-aplicaciones-serverless-en-cloud-computing/

https://serverless-stack.com/chapters/es/what-is-serverless.html

https://serverless-stack.com/chapters/es/what-is-aws-lambda.html