Serverless

Kevin Barbón García UO263779

Thalía Cuetos Fernández UO264545

Introducción

Al desarrollar una aplicación, muchas veces es necesario disponer de un servidor en el que se corren las aplicaciones, están instaladas las bases de datos, etc. Esto conlleva preocuparnos de una serie de cuestiones que en equipos sin especialistas en infraestructura puede suponer un problema y quitar tiempo de desarrollo o mantenimiento.

Arquitectura serverless

Una alternativa posible es la arquitectura serverless, lo que significa sin servidor.

La arquitectura serverless ejecuta fragmentos de código en contenedores temporales proporcionados por un proveedor, lo que nos permite olvidarnos de los servidores. Estos fragmentos de código suelen ser en forma de funciones, por lo que este modelo se denomina "Funciones como servicio" (FaaS).

Otra forma de arquitectura serverless es utilizar servicios de terceros en nuestra aplicación, lo que se conoce como "Backend como servicio" (BaaS).

Modelo FaaS

Como se menciona antes, el modelo FaaS permite ejecutar funciones en contenedores que se crean en el momento en el que se produce un evento (eventos de bases de datos, solicitudes HTTP, eventos programados) que conlleva ejecutar una función y luego desaparecen.

Los contenedores son sin estado, es decir, no guardan datos y cada ejecución es independiente de las demás. Esto implica que no permite ejecutar funciones que utilicen un contexto anterior.

Los proveedores son los encargados de asignar contenedores y escalar el número necesario para cada invocación, lo que supone que no nos tenemos que preocupar si se produce un crecimiento de uso que no contemplábamos.

El coste es por tiempo de ejecución del código, por lo que se busca que las funciones tarden el menos tiempo posible y tengan un único propósito.

Modelo BaaS

Existen funcionalidades que son muy comunes en las aplicaciones web y suelen estar implementadas de forma similar. Los proveedores BaaS se encargan de implementar estas funcionalidades y permiten que otros desarrolladores las usen a través de una API.

Algunos servicios muy comunes son: autenticación, almacenamiento en la nube, servicios de analítica o notificaciones push.

Suelen seguir un modelo de pago freemium, lo que significa que permite de forma gratuita un determinado número de usuarios o llamadas a la API al mes y, si se excede ese límite, se paga una cuota por cada usuario o llamada de más. También existen cuotas fijas que permiten más usuario y llamadas mensuales.

Principales proveedores

- **AWS Lambda**: es el más conocido y usado del mercado. Lo usan empresas tan conocidas como Netflix.
- **Azure Functions**: desarrollado por Microsoft.
- **Google Cloud Functions**: lo puedes encontrar en la plataforma de Google y solo se puede usar con el código Javascript para ejecutarse en un entorno Node.js.

Ventajas

- **Costes operativos**: la infraestructura es compartida entre varias personas. No es necesario administrar directamente los servidores y las bases de datos, por lo que también disminuye el costo de mano de obra.
- **Costes de desarrollo**: esto está relacionado con el modelo BaaS. Al usar servicios que se encargan de realizar distintas tareas, podemos ahorrar el tiempo de desarrollo que supondría implementarlo desde 0. Como he dicho antes con la autenticación.
- **Costes de escalado**: relacionado con el modelo FaaS. El escalado es totalmente automático por parte de los proveedores y solo se paga por el tiempo de ejecución, al contrario que en otros modelos en los que se paga por servidores incluso cuando no se están utilizando.
- Por esto mismo, la **gestión operativa** es más sencilla, ya que nos permite olvidarnos de configurar y mantener el sistema en función de las solicitudes simultaneas que pensamos que podemos llegar a tener.

Desventajas

- **Alojamiento compartido:** Pueden existir varias instancias de software para varios clientes en la misma máquina y los proveedores de servicios hacen todo lo posible para que los clientes sientan que cada uno de ellos es el único que usa su sistema y, por lo general, los buenos proveedores de servicios hacen un gran trabajo al respecto, pero ninguna solución es perfecta y, a veces, pueden tener problemas de seguridad como que un cliente puede ver los datos de otro, o que un error en el software de un cliente cause un error en el software de otro cliente diferente o de rendimiento porque un cliente necesite mucho procesamiento, causando que el otro software reduzca su tiempo de ejecución.
- **Dependencias del proveedor:** Es muy probable que cualquier característica de serverless de un proveedor sea implementada de manera diferente por otro proveedor, por lo que, si queremos cambiar de proveedor, es muy probable que haya que actualizar las herramientas operativas implementación y monitoreo, además, habría que modificar el código, e incluso puede ser necesario cambiar el diseño o la arquitectura si hay diferencias en el comportamiento de las implementaciones de los proveedores.
- **Pérdida de optimizaciones del servidor:** Con una arquitectura BaaS completa, no es posible optimizar el diseño del servidor para el rendimiento del cliente, toda la lógica personalizada está en el cliente y los únicos servicios de backend son proporcionados por el proveedor, por lo que no se podrán abstraer aspectos dentro del servidor para que el cliente pueda realizar operaciones más rápidamente y usar menos energía.
- **Contenedores sin estado:** Las funciones FaaS tienen restricciones significativas cuando se trata del estado, la razón es que con FaaS normalmente no tenemos control sobre cuándo se inician y se detienen los contenedores de host para nuestras funciones.
- **Latencia de inicio:** Dado que las funciones se ejecutan en un contenedor que se activa bajo demanda para responder a un evento, existe una cierta latencia asociada a él, esta duración depende de la implementación del proveedor específico, en Amazon Web Services Lambda puede variar desde unos pocos cientos de milisegundos hasta unos pocos segundos.

- **Poco monitoreo:** El monitoreo es un área complicada debido a la naturaleza efímera de los contenedores, la mayoría de los proveedores de la nube proporcionan cierta cantidad de soporte de monitoreo. Estos datos dependen de lo que proporcione el proveedor que muchos casos, como es el de Amazon Web Services Lambda, son muy básicos.

¿Cuándo se puede usar?

Los casos de uso son tantos como se nos ocurran, pero se pueden citar, entre otros:

- **Backends para móviles**: las funciones permiten extender el backend de las aplicaciones móviles para atender y emitir los eventos necesarios.
- **Backends de aplicaciones web**: de igual modo que en el caso anterior, se puede complementar mediante funciones.
- **Procesamiento de datos**: recopilando datos y procesándolos en tiempo casi real, se puede aplicar a procesamiento de archivos, datos procedentes de sitios webs o imagen y vídeo.
- **Ejecutar tareas programadas**: ejecutar acciones de forma periódica o en un momento concreto.
- **IoT**: Un sistema formado por miles de dispositivos es un gran ejemplo de la capacidad para procesar, transformar y almacenar datos que pueden tener los servicios Serverless.

El futuro de serverless

A continuación, se discuten algunas áreas en las que el mundo serverless puede desarrollarse en los próximos años.

Este mundo todavía es bastante nuevo y por ello cuenta con algunos inconvenientes, así que durante los próximos meses y años se tratará de mitigar los inconvenientes y eliminar, o mejorar los inconvenientes de implementación.

La persistencia en el estado del servidor no repercute en un gran número de aplicaciones, pero sí que es un factor decisivo para muchas otras, así que es probable que en el futuro veamos diferentes arquitecturas de aplicaciones híbridas, serverless y con servidor, que se adopten para tenerla en cuenta. Por ejemplo, para aplicaciones que necesitan latencias bajas, un enfoque puede ser, un servidor tradicional que recopila todo el contexto necesario para procesar su estado y luego, realiza una solicitud contextualizada a una granja de funciones FaaS que no necesitan buscar datos externamente.

Ya existe una comunidad sin servidores de tamaño considerable con conferencias, reuniones en muchas ciudades y varios grupos en línea. Sin embargo, se espera que esto continúe creciendo y que cada vez se dé más a conocer este tipo de arquitectura.

Conclusiones

No es el enfoque correcto para todos los problemas, y pueden surgir problemas de monitoreo y depuración. Sin embargo, hay que tener en cuenta aspectos positivos de esta arquitectura, que incluyen costos operativos y de desarrollo reducidos y una administración operativa más sencilla.

Además, como beneficio más destacable, es el tiempo reducido en la creación de nuevos componentes de aplicaciones, en gran parte porque que es muy valioso poner el sistema frente a un usuario final lo antes posible para obtener comentarios tempranos y también, para reducir el tiempo para llevarlo al mercado.