Sofía Suárez Fernández UO270149
Iván Valle Soto UO270762
Rafael Muñiz Reguera UO271728

Serverless tardes-offs

Contexto

¿Qué es Serverless?

Las arquitecturas Serverless utilizan los recursos informáticos sin gestionar la infraestructura subyacente. No quiere decir que los servidores no formen parte de la ecuación.

Suelen hacer uso extensivo de servicios terceros para ejecutar tareas que no son específicas de la aplicación. Pueden usar una familia de terceros de un mismo proveedor (Firebase) o servicios de proveedores distintos (Azure Media Services para streaming y OneSignal para push notifications).

En estas aplicaciones, el comportamiento se realiza a partir de servicios, el control del flujo se hace del lado del cliente y el contenido se genera dinámicamente en esa parte de la aplicación (las aplicaciones móviles usan frameworks de interfaz de usuario para generar contenido dinámico a partir de información que obtienen a partir de llamadas a APIs remotas).

El ciclo de vida de este tipo de programas o aplicaciones suele ser bastante corto. Se activa cuando recibe una petición, la procesa, y termina.

Características principales

Propuestas por Mike Roberts para definir el criterio de aceptación de una arquitectura Serverless:

- No requiere administración de hosts de servidor o procesos de servidor: los servidores están abstraídos. Todavía siguen ahí, detrás de escena, pero no necesitamos tenerlos en cuenta.
- Escalado automático y aprovisionamiento automático en función de la carga de solicitudes.
- Los precios se basan en el uso según la oferta del proveedor.
- El tamaño y número de servidores no definen el rendimiento de la aplicación.
- Alta disponibilidad (capacidad de seguir funcionando incluso cuando un componente falla) y tolerancia a fallos, por defecto.

Serverless = FaaS + BaaS

Este tipo de arquitecturas Serverless combinan las Funciones como Servicio (FaaS) y el Back-end como Servicio (BaaS).

La Función como Servicio (FaaS) podría definirse como un tipo de servicio que permite a los desarrolladores diseñar, ejecutar y gestionar aplicaciones como funciones sin tener que ocuparse del mantenimiento de la infraestructura. Este modelo de ejecución está basado en eventos y se ejecuta en contenedores sin estado. Las propias funciones gestionan el estado y la lógica de los servidores haciendo uso de los servicios de un proveedor. Algunos ejemplos son: IBM Cloud Functions, AWS Lambda de Amazon, Google Cloud Functions y Microsoft Azure Functions y OpenFaaS como open source.

El Back-end como Servicio (BaaS) quiere decir que los componentes lógicos y de datos altamente escalables de nuestra aplicación son alojados en los proveedores para gestionar las necesidades de bases de datos, plataformas de mensajería, manejo de usuarios y más.

Sofía Suárez Fernández UO270149
Iván Valle Soto UO270762
Rafael Muñiz Reguera UO271728

Ventajas

-Reducción de costes: Tener usar una arquitectura sin servidor nos repercute en una rebaja del precio del despliegue debido a distintos factores. Los proveedores de estos servicios (AWS, Azure) ofrecen tarifas muy reducidas o incluso gratuitas según las complejidad y tamaño de nuestro sistema, su sistema de cobro se calcula en base a las peticiones que realizamos a su servicio (Invocaciones Lambda) además, nos ahorramos los gastos directos de mantenimiento e inversión de tener estos sistemas en nuestra empresa. Otro punto a tener en cuenta es que debido a la alta competitividad de los proveedores (Microsoft, Amazon, Google) mantienen por hacerse con la cuota de mercado, hace que los precios se mantengan en rangos muy asequibles además de estables.

-Escalabilidad: Escalar nuestro sistema es muy sencillo ya que el consumo de recursos se realiza bajo demanda (algo posible debido al enorme tamaño de procesamiento que tienen disponible los proveedores de estos servicios) además que las propias funciones Lambda por sí mismas ofrecen una gran escalabilidad. Un aspecto a tener en cuenta es que si la aplicación escala y hace uso de más clusters esto repercutirá en el precio.

-Seguridad: Esta arquitectura es muy segura por dos motivos principales:

> -La seguridad en la nube es mucho mayor que en local, los proveedores invierten grandes cantidades en hacer que sus servicios sean lo más seguros posibles y su mantenimiento es continuo.
>
> -Las funciones lambda acceden exclusivamente a su docker asignado siendo este posiblemente efímero y con una duración de su vida útil determinada y después de estas serán desechados.

-Comodidad: Esta forma de trabajar es muy cómoda ya que nosotros simplemente accedemos a múltiples servicios que ya están desarrollados por otros, es posible automatizar la creación de nuevos componentes, las funciones lambda pueden ejecutarse en múltiples lenguajes de manera sencilla y también actualizar las dependencias de las funciones de forma automática según surjan nuevas versiones de las mismas.

Desventajas

-Complejidad: Aunque algunas partes del sistema se vuelvan más sencillas desde el punto de vista del programador gracias a las funciones, la complejidad se vuelve mayor por tener que administrar las distintas funciones, servicios, API's y demás.

-Lock-in: Cada proveedor de funciones tiene una forma diferente de crearlas dentro de la infraestructura. Esto produce que si se desea crear código que sea una abstracción de proveedores sea imposible y que siempre que se desee cambiar de proveedor sea necesario cambiar todas las funciones haciendo que se tenga una gran dependencia del proveedor.

-Las pruebas de integración: Probar funciones desde un servidor se vuelve más complejo que probar funciones aisladas desde la propia máquina. Además, cuando existe un numero de funciones en la

aplicación y es necesario una gestión de estas funciones, es complejo testear pues no se tiene el control de todos los servicios que pueden envolver la funcionalidad a probar. Algunos proveedores pueden dar unas pequeñas y sencillas soluciones a este problema pero que no consigue resolver el problema en completo.

-La depuración y monitorización: Localizar errores se hace más complicado pues lograr la agregación de registros y el seguimiento del sistema distribuido es más complejo en el servidor que en la maquina local.

-Cold start: Es el tiempo que necesita el contenedor de funciones antes de una ejecución. Se produce cuando la función transcurre un tiempo sin usarse y se encuentra en un “estado de suspensión”. Esto se hace para así abaratar costes y que solo se active cuando se vaya a utilizar. Una vez se utilice la función, pero no se produzca un cold start entonces se ejecutará normal y su tiempo será menor.

-Pérdida de control: Tanto plataformas FaaS como BaaS son desarrolladas y operadas por un tercero. Por esta razón el control de la administración del software sobre el código que se está ejecutando se cede al tercero. Esta administración que se cede referencia a la configuración, el rendimiento, la solución de problemas y la seguridad.

-Herramientas pobres: Las herramientas de mapeo de las funciones son más pobres que aquellas del propio lenguaje completo. Esto produce un ciclo interno de desarrollo más lento. El ciclo interno de desarrollo es el proceso iterativo de escribir, crear y depurar código por un desarrollador hasta que este lo comparte.

Bibliografía

https://www.nubersia.com/es/blog/serverless-es/tecnologia-serverless-y-sus-ventajas/

https://www.aplyca.com/es/blog/limitaciones-de-serverless

https://pablo-iorio.medium.com/serverless-architectures-i-iii-design-and-technical-trade-offs-8ca5d637f98e

https://medium.com/ssense-tech/the-trade-offs-with-serverless-functions-71ea860d446d

https://sg.com.mx/revista/52/un-vistazo-la-arquitectura-serverless

https://blog.symphonia.io/posts/2017-06-22_defining-serverless-part-1

https://www.redhat.com/es/topics/cloud-native-apps/what-is-faas#:~:text=La%20funci%C3%B3n%20como%20servicio%20(FaaS)%20es%20un%20tipo%20de%20servicio,mantenimiento%20de%20su%20propia%20infraestructura.

https://www.ionos.es/digitalguide/servidores/know-how/serverless-computing/