SERVERLESS

¿Qué es?

Serverless es un modelo o servicio de ejecución en la nube autogestionado en el que el proveedor de la nube ejecuta el servidor y lo administra asignando y adaptando los recursos de éste según las necesidades del usuario o de la empresa.

Con serverless los usuarios pueden concentrarse en la actividad de la empresa sin tener que preocuparse por nada ya que no habría ningún servidor, ni ningún so ningún software que administrar

Serverless = BaaS +FaaS

Serverless tiene dos tipos principales de servicios: el Baas y el Faas

Baas cuyas siglas significan Backend as service, es un modelo de computación basado en la nube que ofrece varias funciones de back-end que son esenciales para los servicios de backend de cada aplicación, estas funciones permiten desarrollar un backend de la mejor manera posible. Como ventaja de Baas es que los desarrolladores que lo han utilizado solo han necesitado mantener la interfaz.

Faas cuyas siglas significan Function as Service, es una plataforma que permite ejecutar funciones independientes (fragmentos de código) en la nube. Se usa para el procesamiento de datos en tiempo real con lo que ayuda a la generación y a la actualización de código. El objetivo de Faas está destinado a administrar y regular los microservicios de manera óptima.

Entonces la diferencia entre Baas y Faas es que Baas se encarga de administrar y manejar la funcionalidad de backend mientras que Faas se utiliza para administrar microservicios de una manera más efectiva.

Ventajas de serverless

Paga por lo que usas: Se cobra por el número total de solicitudes en todas sus funciones y, por lo general, habrá un nivel gratuito. Esto trae grandes beneficios desde el punto de vista de análisis de costos

Elasticidad: Toda la arquitectura se escala hacia arriba y hacia abajo automáticamente al aprovisionar o desaprovisionar recursos de manera autónoma.

Productividad: Tiene una gran productividad en cuanto a velocidad de desarrollo al principio cuando hay pocas funciones.

Pruebas unitarias: Es fácil probar con pruebas unitarias ya que las funciones deben seguir el principio de responsabilidad única.

Modelo basado en eventos: Serverless es perfecto para una arquitectura basada en eventos.

Microservicios: Facilita el desarrollo de un patrón de arquitectura de microservicios.

Inmutabilidades: Es inmutable

Aislamiento y seguridad: Cada función solo puede acceder a su propio espacio.

Crea más fácilmente nuevos entornos

Alta disponibilidad y es muy tolerante a fallos

Desventajas de serverless

- Curva de aprendizaje más larga para el formato de "código" sin funciones .
  A pesar de usar "low-code" (programación mediante interfaz visual), aún se requiere algo de código para definir las asignaciones para estas integraciones y algunos lenguajes de asignación son bastante difíciles de entender si no los has utilizado antes además de tener características limitadas.
- Problemas en el sistema de APIs de terceros
  Usar APIs de terceros siempre trae consigo varios problemas. Algunos de estos incluyen el control de dichos proveedores, los problemas de mantenimiento, el bloqueo de proveedores y consideraciones relacionadas con la seguridad, entre otros.
- La seguridad
  La tecnología sin servidor a veces se considera erróneamente más segura que las arquitecturas tradicionales.
  Si bien esto es cierto hasta cierto punto porque el proveedor de la nube se ocupa de las vulnerabilidades del sistema operativo, la superficie de ataque total es significativamente mayor, ya que hay muchos más componentes en la aplicación en comparación con las arquitecturas tradicionales y cada componente es un punto de entrada a la aplicación sin servidor.
- Reutilización
  Los lenguajes de mapeo no admiten funciones como importar/incluir o la capacidad de definir funciones de biblioteca y, por lo tanto, dificultan o imposibilitan la reutilización.
- Difícil/imposible de probar (en forma aislada) .
  Probar Step Functions ya es difícil en sí mismo, por lo que Step Functions que se integra directamente a DynamoDB puede ser aún más difícil de probar. Si, en cambio, invocara una función Lambda desde su máquina de estado, podría probar fácilmente la función Lambda de forma aislada.
- Hay que buscar un servicio que soporte los lenguajes de programación que has escogido
- Gran dependencia del proveedor (“efecto Lock-in”)
  Dependencia del proveedor FaaS que, a largo plazo, puede obstaculizar el cambio, al tener que modificar los lenguajes de programación y las funciones utilizadas para que se adecuen a los requisitos del nuevo prestador del servicio Serverless.
- Las pruebas de integración son difíciles. Una vez que llega a un cierto nivel de funciones, las pruebas de integración se vuelven un desafío. Esto sucede cuando no se tiene el control de todos los demás servicios que rodean la solución y se necesita

comenzar a tomar decisiones sobre la creación de stubs de otros servicios y también si el proveedor ofrece algunas capacidades para esto o si necesita construirlo usted mismo.

- La depuración puede ser bastante desafiante. Es difícil replicar el entorno de la nube en su máquina local y los proveedores de la nube no proporcionan opciones para depurar
- Microservicios. El uso de funciones facilita el desarrollo de un patrón de arquitectura de microservicios, ya que es más fácil desarrollar componentes discretos e implementables por separado. Por supuesto, por otro lado, al incrementar la cantidad de componentes básicos, aumenta el mantenimiento, la complejidad, la latencia de la red y el monitoreo. Este artículo Módulos frente a microservicios describe la complejidad operativa de los microservicios.

Cold start

Uno de los principales problemas de Serverless es el cold start. Esto se puede definir como el tiempo de inicialización de un contenedor de función antes de su ejecución. Como bien hemos hablado en las ventajas, con Serverless pagas únicamente por aquellos recursos que vas a utilizar, de manera que, si alguna función lleva tiempo sin ser utilizada, se considera que entra en un "sleeping state", ya que la infraestructura que hay detrás de ella lleva en desuso cierto tiempo. Todo esto provoca que la primera ejecución realizada sobre una función "dormida" siempre lleve más tiempo que el resto, lo que conlleva largos tiempos de espera para el cliente que realiza peticiones. Una vez realizada esa primera ejecución, ese código pasa a ser considerado caliente y se ejecutará inmediatamente mientras su contenedor esté vivo (normalmente AWS Lambda suele eliminarlos a los 30-45 minutos de desuso).

Ahora bien, ¿es completamente inevitable un cold start en mi aplicación? La respuesta es sí, pero existen algunas prácticas que ayudan a prevenir y mitigar este problema.

1. **Evitar funciones excesivamente grandes**
   Debes intentar separar las funciones amplias en varias más pequeñas. De esta manera reducirás el tiempo de preparación para su ejecución.

2. **Reducir el uso de variables globales y el tamaño del archivo en general**
   Debes minimizar el uso de variables en tu aplicación borrando todas aquellas que sean innecesarias. El impacto en rendimiento de esta práctica solo es visible a gran escala, es decir, no notarás ningún cambio por eliminar tres variables. Además, también es recomendable reducir el numero de espacios en blanco para que nuestro archivo ocupe lo menos posible.

3. **Ajustar la memoria dedicada**
   Cuando subes tu código a la AWS Lambda también puedes ajustar cuánta memoria deseas dedicar a cada función, así que un aumento de memoria en aquellas funciones más complejas puede ayudar a reducir los tiempos de espera.

4. **Mantener calientes nuestras funciones**

Consiste en enviar invocaciones periódicamente a las funciones para simular una actividad constante. Aunque esto puede sonar como la mejor opción, no es tan fácil, ya que bombardear nuestra función con muchas invocaciones vacías aumentará los costes debido a que “pagamos por los recursos utilizados”. Por ello, antes de utilizar esta medida se debe hacer un estudio en profundidad de que funciones tienen mayor tendencia a enfriarse y cada cuánto ocurre, con el fin de minimizar los gastos.