Paradigma Serverless: función como servicio

DOI: 10.29236/sistemas.n168a7

Resumen
Las funciones como servicio (FaaS) nacen como un modelo de computación en la nube en el cual los desarrolladores de software solo se preocupan por el diseño y ejecución de pequeños fragmentos de código (es decir, las funciones) delegando a un tercero los aspectos concernientes al aprovisionamiento, operación y mantenimiento de la infraestructura de hardware y software necesarias para la ejecución de la aplicación. En este artículo se presenta una introducción al modelo FaaS; se explican sus ventajas y desventajas y se esbozan los desafíos a futuro.

Palabras claves
Cloud Computing, FaaS, Serverless

Introducción

Para que una aplicación (por ejemplo, una aplicación web) pueda funcionar se requiere el aprovisionamiento, configuración y administración de varios componentes (ver Figura 1), entre los que podríamos destacar el hardware, la capa de virtualización, el sistema operativo, el sistema de *runtime* (e.g., Java, Node o .NET Core) y la aplicación propiamente dicha. Hasta hace algunos años la gestión de toda esa infraestructura era responsabilidad de la empresa encargada de la aplicación. Este modelo, conocido como *on-premise*, es por tanto, costoso y complejo de administrar.

En la década del 2000, Amazon presentó Amazon Web Service (AWS), un nuevo modelo de servicios de computación denominado infraestructura como servicio (IaaS) (Mathew y Varia, 2014). En este modelo, AWS se encarga de proveer al usuario las capas de hardware y virtualización. Esto hace que los costos de operación se reduzcan ya que la empresa dueña de la aplicación no se tendrá que ocupar por el mantenimiento de

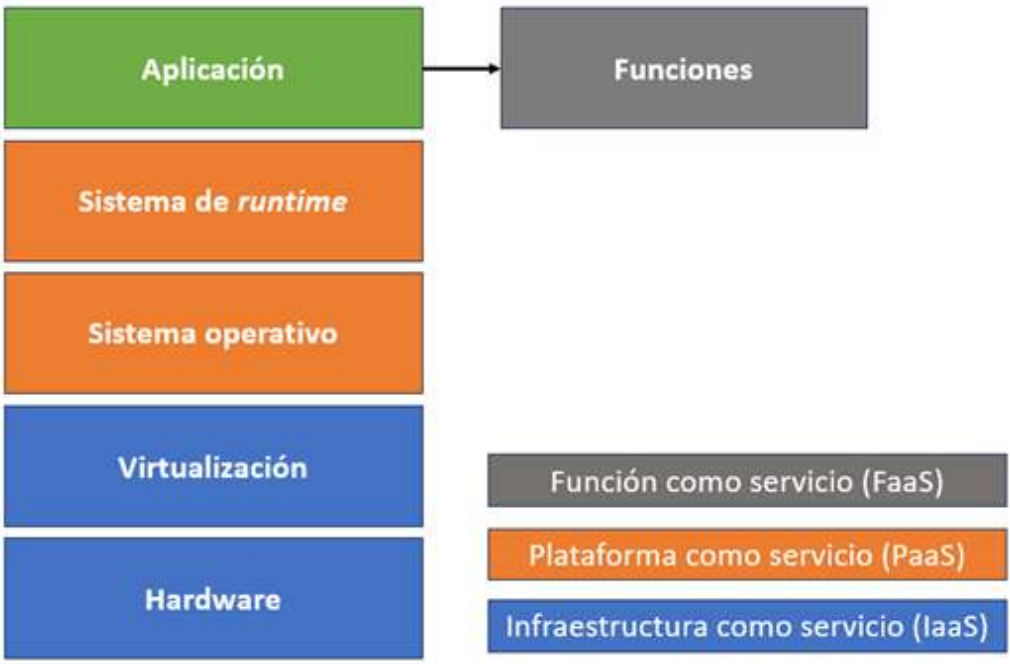

Figura 1.
Componentes requeridos para la ejecución de una aplicación web

esas capas; además el aprovisionamiento del hardware que antes podría tomar meses mientras se hacían los procesos de compra, instalación y configuración, ahora solo toma unas cuantas horas o incluso, minutos.

El siguiente modelo que emerge es el denominado plataforma como servicio (PaaS) (Keller y Rexford, 2010). En este modelo se incluyen, además del hardware y la virtualización, las capas correspondientes al sistema operativo y de *runtime*. De este modo la preocupación de la empresa radica ahora solo en la administración de la aplicación.

En el 2014, hook.io[1] presentó un nuevo modelo de computación conocido como función como servicio (FaaS). Este modelo pretende abstraer la complejidad de una aplicación para que la empresa pueda enfocarse únicamente en sus componentes principales que en este caso son las funciones.

Cómo funciona

Para entender cómo funciona este modelo pensemos, por ejemplo, en una aplicación que se encarga de registrar la actividad física de un deportista. Esta aplicación dependerá de varias funciones. Una de ellas podría ser la que consulta el plan del usuario (i.e., getUserPlan) para saber el tipo de plan que tiene contratado y si está vigente o no. Esta función se conectará con la base de datos de planes. Otra función, por su parte, permitirá consultar el detalle de la actividad de un usuario para un día en específico (i.e., getActivity). Esta función se conecta con la base de datos de registro de actividad.

En una vista simplificada de una posible arquitectura (ver Figura 2), la aplicación se conecta por ejemplo a un API Gateway que identifica

[1] https://hook.io-mps.mto.zing.vn/

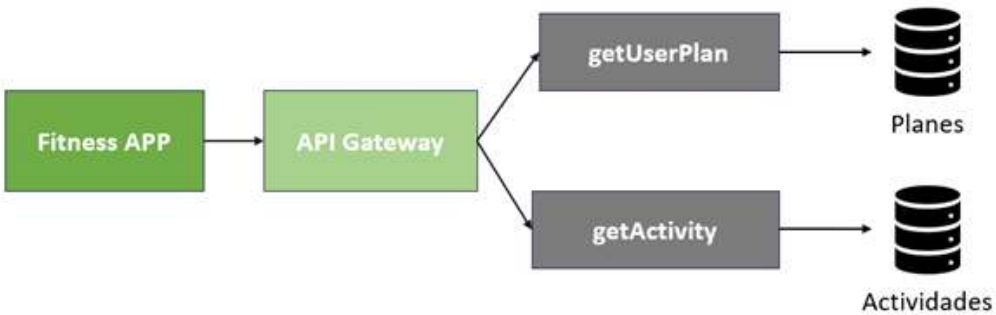

Figura 2.
Arquitectura de una aplicación construida bajo el modelo FaaS

el tipo de petición y la redirige a la función como servicio requerida. De este modo, una petición del tipo GET /api/userplans será redirigida a la función getUserPlan, mientras que la petición GET /api/activities será redirigida a la función getActivity.

Lo anterior denota que una función como servicio es una pequeña porción de código que se ejecuta en la nube en contenedores sin estado (*stateless containers*). En otras palabras, estas funciones ejecutan la tarea para la cual están destinadas, no almacenan ningún estado y se apagan luego de su ejecución.

Las funciones como servicio se invocan y comienzan a ejecutarse mediante activadores, también conocidos como *triggers*. Algunos activadores pueden ser intervalos de tiempo predefinidos o una petición HTTP. De igual forma, las funciones tienen la capacidad de interactuar con una amplia gama de servicios, por ejemplo, de almacenamiento, autenticación, seguridad, entre otros. Algunas plataformas permiten configurar esos activadores de forma declarativa sin escribir ningún código.

En resumen, una función se ejecuta cada vez que es invocada por un activador. La función se ejecuta durante un intervalo de tiempo particular y entra en un estado inactivo. Luego, se activa y se ejecuta cada vez que un disparador la invoca nuevamente.

Ventajas y desventajas de las FaaS

En una función como servicio podemos resaltar cuatro ventajas. La primera es que solo se paga por lo que usa; de este modo se evita el pago por los denominados tiempos muertos o *idle time*, que es el tiempo en el que la función no se está usando. Por su naturaleza el tiempo de ejecución de una función se puede medir en milisegundos, y ese será el periodo por el cual el proveedor del servicio cobra. La segunda es que los desarrolladores solo deben enfocarse en el fragmento de código de interés, dejando de lado temas relacionados con la infraestructura subyacente. La tercera ventaja es que el escalado se hace de forma automática: si la demanda de la función es mayor (o menor), la infraestructura que da soporte se ajusta por sí misma dependiendo de las reglas definidas. La cuarta radica en que son servicios con alta disponibilidad. Esto implica que una función como servicio se puede desplegar en diferentes zonas geográficas.

A pesar de las ventajas mencionadas, el modelo de función como servicio presenta una desventaja fundamental. Recordemos que las funciones se ejecutan cuando se activan y pasan a un estado inactivo cuando no realizan ningún trabajo. Cuando una función está inactiva, pasará algún tiempo antes de que entre en acción cada vez que se active. Esto se debe a que la infraestructura subyacente tardará

algún tiempo en alistarse (también denominado como el proceso de calentamiento o *warm up*) y comenzar a ejecutar el código. Este fenómeno se conoce como el problema de inicio en frío (o *cold start*) que debe tenerse en cuenta al diseñar soluciones basadas en FaaS (Gias y Casale, 2020).

El futuro

Una vez se ha diseñado una solución basada en funciones como servicios viene el proceso de despliegue en la plataforma seleccionada, entre las que están AWS Lambda,Google Cloud Functions, Microsoft Azure Functions, IBM/ Apache's OpenWhisk u Oracle Cloud Fn. No obstante, sin embargo, el despliegue de la solución para cada plataforma puede distar bastante de ser trivial.

En este escenario lo ideal es aplicar un principio ampliamente usado en la ingeniería de software que consiste en la incorporación de una nueva capa de abstracción que opere sobre las plataformas existentes.

Conclusiones / Reflexiones finales

Un modelo de computación basado en FaaS supone unas claras ventajas por encima de los modelos *on-premise*. De este modo, la empresa se encargará de solo de administrar los pequeños fragmentos de código que constituyen una aplicación. Esto permite ahorros significativos y reduce la complejidad de la administración de la infraestructura tecnológica.

A pesar de las ventajas, también existen condicionantes que se deben tener en cuenta. Por eso antes de implementar una solución basada en una función como servicio, conviene una reflexión sobre los siguientes aspectos:

- Se debe decidir que las funciones como servicio es la mejor aproximación para el escenario en el cual las quiere implementar.
- Hay un mercado amplio de proveedores. Por eso se hace clave elegir proveedores existentes el mejor plan de hosting y revisar que la facturación se ajuste al presupuesto disponible.
- Una solución FaaS implica que existan retrasos en el inicio de la ejecución de una función (cold start); por esto se debe mitigar su impacto en la solución.
- Revise aquellas funciones que impliquen largos periodos de duración en su ejecución; esto tendrá una correlación directa con el valor de la facturación.
- Facilite la integración y comunicación entre otros servicios (particularmente con los que ofrece el mismo proveedor de las soluciones de Faas).
- Identifique y gestione los posibles cuellos de botella.
- Asegúrese de que su solución sea tolerante a fallos pero también facilite el monitoreo eficiente y la depuración de las fallas.

- Incorpore prácticas de DevOps y aporte un enfoque de infraestructura como código (IaC).

Referencias

Gias, A. U., & Casale, G. (2020). Cocoa: Cold start aware capacity planning for function-as-a-service platforms. En *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* 1-8. IEEE.Doi: 10.1109/MASCOTS50786.2020.9285966

Keller, E., & Rexford, J. (2010). The "Platform as a Service" Model for Networking. *INM/WREN*. 10, 95-108. https://www.researchgate.net/publication/228824473_The_Platform_as_a_Service_Model_for_Networking

Mathew, S., & Varia, J. (2014). Overview of amazon web services. *Amazon Whitepapers*, 105, 1-22. https://media.amazonwebservices.com/AWS_Overview.pdf

José Bocanegra. *Ingeniero de sistemas de la Universidad Distrital Francisco José de Caldas. DEA en Tecnología e Ingeniería de Software de la Universidad de Sevilla; doctor en Tecnología de la Universitat de Girona y doctor en Ingeniería de la Pontificia Universidad Javeriana. Actualmente se desempeña como profesor asistente del Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes. Sus áreas de interés se centran en la Ingeniería de Software Dirigida por Modelos y los Lenguajes Específicos de Dominio.*