Optimización del código y las dependencias de las funciones Serverless para mejorar el rendimiento de las aplicaciones

Nelson Rodríguez, Martín Gómez, María Murazzo, Ana Laura Molina, Lorena Parra

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales, Universidad Nacional de San Juan, San Juan, Argentina
nelson@iinfo.unsj.edu.ar, martinsj0811@gmail.com, maritemurazzo@g,mail.com, almm95@gmail.com, lorenaparra152@yahoo.com.ar,

Resumen. Serverless Computing es una reciente arquitectura para Cloud Computing que presenta ventajas considerables para los usuarios. Sin embargo, debido a su reciente aparición, muchas de sus limitaciones o desventajas no están totalmente resueltas. Si bien el desarrollo serverless presenta condiciones comunes al desarrollo para otro tipo de plataformas o entornos, de las cuales se pueden obtener buenas prácticas de tipo "genéricas", también serverless presenta aspectos propios debido a que son funciones distribuidas ejecutándose en una plataforma Cloud, cuya ejecución es conducida por eventos, sin estado y sin responsabilidades operativas por parte del usuario, entre otras. Se debe considerar además el hecho de que determinadas prácticas pueden reducir costos como aquellas que conducen a minimizar el arranque en frío, otras apuntan a aspectos de la seguridad o a la gestión del BackEnd. En el presente trabajo se realizó un análisis de las buenas prácticas para serverless y se trabajó en especial en aquellas que impactan en la performance, en particular en el uso de recursos provistos por la plataforma y en optimizar las dependencias de las funciones, realizando una serie de pruebas y análisis en diversos lenguajes de programación, que permiten emitir conclusiones sobre el impacto que causan estas buenas prácticas en la mejora de los tiempos de ejecución.

Keywords: Serverless Computing, FaaS, Best Practices, Cloud Computing

1 Introducción

Cloud Computing es un modelo de provisión de servicios y una arquitectura con varios años de utilización por parte de la industria y la academia [1].

A partir de la publicación de un artículo de Google en 2003, la computación en la nube ha evolucionado del sistema de TI interno al servicio público y ya está arraigada en el diseño de plataformas de motores de búsqueda [2].

Siguiendo la evolución observada en la historia de la contenerización, los servicios en la nube se han adaptado para ofrecer contenedores de mejor ajuste que requieren menos tiempo para cargar (arranque) y proporcionar mayor automatización en el manejo (orquestación) de contenedores en nombre del cliente [3]. La Computación Serverless promete lograr la automatización completa en la gestión de contenedores.

El modelo de computación serverless es impulsado por eventos en el que los recursos informáticos se proporcionan como servicios escalables. En el modelo tradicional se cobra un costo fijo y recurrente por los recursos informáticos del servidor, independientemente de la cantidad de trabajo realizado por el servidor. Sin embargo, la implementación Serverless ha superado esta deficiencia, ya que se paga solo por el uso del servicio y no se cobra por el tiempo de inactividad.

En este paradigma emergente, las aplicaciones de software se descomponen en múltiples funciones independientes sin estado [4] [5]. Las funciones solo se ejecutan en respuesta a acciones desencadenantes (como interacciones de usuario, eventos de mensajería o cambios en la base de datos), y se pueden escalar de forma independiente y pueden ser efímeras (pueden durar una invocación) y están completamente administrados por el proveedor de Cloud.

Los principales proveedores de nube han propuesto diferentes plataformas informáticas sin servidor como AWS Lambda, Microsoft Azure Functions, Google Functions, IBM Cloud Functions, Cloudflare Worker, Alibaba Function Compute. Dichas plataformas facilitan y permiten que los desarrolladores se centren más en la lógica de negocios, sin la sobrecarga de escalar y aprovisionar la infraestructura [8].

Castro et al [6], ofrecen una definición basada en las características: “La informática serverless se puede definir por su nombre, que es pensar (o preocuparse) menos por los servidores. Los desarrolladores no necesitan preocuparse por los detalles de bajo nivel de administración y escalado de servidores, y solo pagan cuando procesan solicitudes o eventos”. Luego la define como: “La informática serverless es una plataforma que oculta el uso del servidor a los desarrolladores y ejecuta código que escala bajo demanda automáticamente y facturado solo por el tiempo que se ejecuta el código”.

En la mayoría de los casos, se pueden escribir funciones en el lenguaje que el programador considere más adecuado (Node.js, Python, Go, Java y más) y utilizar herramientas de contenedor y serverless, como AWS SAM o la CLI de Docker, para compilar, probar e implementar las funciones.

Por otro lado, especialistas de Expert Market Research pronostican que el mercado global de computación serverless crecerá en el período de pronóstico de 2022-2027 a una tasa compuesta anual del 22.2% [10].

Un modelo basado en funciones es adecuado para ráfagas, uso de CPU intensivo, cargas de trabajo granulares. Actualmente, los casos de uso de FaaS varían ampliamente, incluido el procesamiento de datos, el procesamiento de flujo, la computación de borde (IoT) y la computación científica [3].

Serverless cubre una amplia gama de tecnologías, que se pueden agrupar en dos categorías: Backend-as-a-Service (BaaS) y Functions-as-a-Service (FaaS).

Backend-as-a-Service permite reemplazar los componentes del lado del servidor con servicios listos para usar.Algunos ejemplos son los sistemas de autenticación remota, la administración de bases de datos, el almacenamiento en el cloud.

Existen diversos desafíos, oportunidades y problemas a resolver, entre ellos la experiencia del desarrollador [7], Interoperabilidad, testing, composición de funciones, seguridad, administración del ciclo de vida, administración de requerimientos no funcionales, performance, optimización del overhead, ingeniería para costo-performance, entre otros [6].

2 Trabajos relacionados

Existen solo dos trabajos que analicen las buenas prácticas aplicadas a Serverless Computing. Si bien existen variadas publicaciones de la industria, algunas de ellas no tienen la rigurosidad necesaria y otras presentan conclusiones parciales.

La publicación de Greg Wilson et at [9], trata solo sobre las mejores prácticas en computación científica. En el resumen indica que: Los científicos pasan cada vez más tiempo construyendo y usando software. Sin embargo, a la mayoría de los científicos nunca se les enseña cómo hacer esto de manera eficiente. Como resultado, muchos desconocen las herramientas y prácticas que les permitirían escribir código más confiable y mantenible con menos esfuerzo.

El trabajo de Rajdeep Mukherjee et al [10], presenta un framework de fuerza industrial para el análisis estático preciso de aplicaciones Python que utilizan los servicios en la nube de AWS. No trata sobre Serverless y se enfoca más en las características de Python.

3 Mejores Prácticas

Las mejores prácticas son un conjunto de directrices, ética o ideas que representan el curso de acción más eficiente o prudente en una situación empresarial determinada.

Las mejores prácticas pueden ser establecidas por autoridades, como reguladores, organizaciones autorreguladoras (SRO) u otros órganos de gobierno, o pueden ser decretadas internamente por el equipo directivo de una empresa. [11]

Según Diccionario Cambridge, Mejores prácticas es un método de trabajo, o conjunto de métodos de trabajo, que se acepta oficialmente como el mejor para usar en un negocio o industria en particular, la aplicación del término puede ser:

La compañía identificó las mejores prácticas que han llevado a un desarrollo de productos más exitoso.

Muchos empleadores quieren adoptar las mejores prácticas en la gestión de su gente, pero no saben qué es esto.

La empresa lleva adelante una política/programa de mejores prácticas [12]

Las buenas prácticas se pueden clasificar en: a) para el desarrollo de software en general, b) para el desarrollo serverless en general c) específicas para algún frameworks d) específicas para alguna plataforma Serverless como AWS o Cloud Function.

Debido a que el desarrollo serverless comparte algunas características comunes a todo desarrollo, se debe aplicar (aunque en efecto la mayoría de las veces aplican) los principios comunes al desarrollo como: KISS (mantener su código lo más simple posible), DRY (que significa "no repetir", y su idea subyacente es que tienes que evitar y reducir las repeticiones y la redundancia reemplazándolas con abstracciones o utilizando la normalización de datos), SOLID (representado por 5 prácticas que son: principio de responsabilidad única, Principio abierto-cerrado (OCP), Principio de sustitución de Liskov , Principio de segregación de interfaz, Principio de inversión de dependencia) [13].

En cuanto a las de desarrollo Serverless se puede mencionar a: diseñar las aplicaciones pensando en escalado automático, aplicar estrategias para minimizar el arranque en frío como reducir en lo posible el número de dependencias de las funciones o utilizar WebPack que optimiza el arranque en frío.

Específicas para AWS se puede mencionar: aprovechar la reutilización del entorno de ejecución para mejorar el rendimiento de su función utilizar una directiva keep-alive para mantener conexiones persistentes y minimizar el tamaño del paquete de implementación según sus necesidades de tiempo de ejecución [14].

Específicas para Cloud Function: escribir código de función en un estilo sin estado para asegurarse de que el código no se someterá a ningún mantenimiento de estado, crear instancias de cualquier objeto que pueda reutilizarse y se sugiere utilizar servicios de administración de código externo (como Git) para fines de administración de versiones y auditorías del código principal [15].

Específicas para Serverless Framework: implementar almacenamiento en caché y reducir los tiempos de inicialización [16].

Se ha probado una de estas mejores prácticas que se indican a continuación.

4 Desarrollo de las pruebas

Una de las mejores prácticas que se decidió probar es la que establece que se debe optimizar el código y las dependencias de las funciones sin servidor para mejorar el rendimiento de las aplicaciones. Por ejemplo, las funciones escritas en un lenguaje interpretado como Node.js y Python tienen tiempos de invocación inicial significativamente más rápidos que las funciones escritas en un lenguaje compilado como Go.

Para realizar esto se crearon dos funciones sin servidor utilizando el servicio Lambda de la plataforma Amazon Web Services. Las mismas resuelven un mismo problema de integración numérica de la misma manera, aunque codificadas en distintos lenguajes, una de ellas codificada en el lenguaje interpretado Python y la otra en el lenguaje compilado Go. Además se realizó una tercera prueba utilizando una función codificada en lenguaje Java.

A continuación, en la Figura 1 se muestra el código de la función en Python.

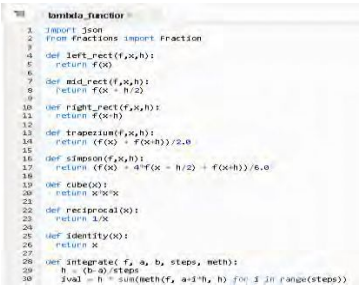

Figura 1

Los códigos utilizados fueron extraídos del sitio web rosettacode.org, el cual contiene algoritmos que son desarrollados en distintos lenguajes de programación a fin de realizar distintas comparaciones entre estos.

En las siguientes imágenes se muestran los resultados de las ejecuciones

Función en Go en su primera prueba

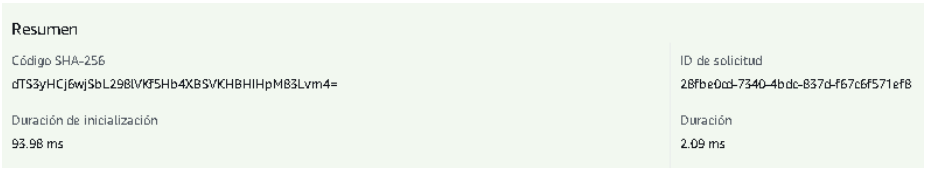

Figura 2

Función en Python en su primera prueba

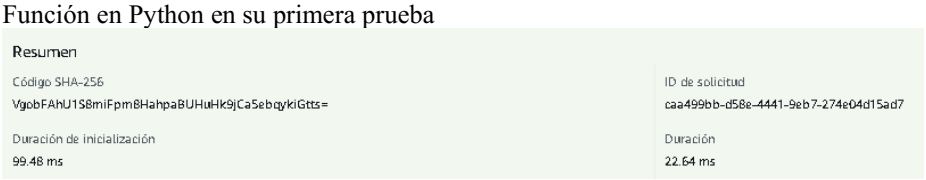

Figura 3

Función en Java en su primera prueba

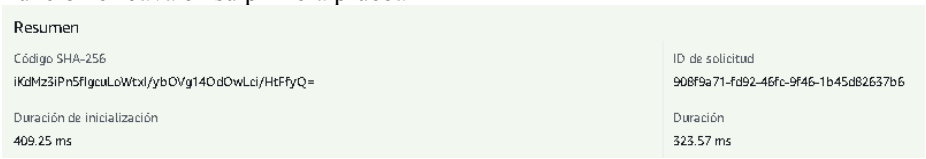

Figura 4

5 Resultados obtenidos

Al ejecutar las funciones se obtuvieron tiempos de inicialización similares entre la función desarrollada en Python y la desarrollada en Go, pero la gran diferencia se obtuvo en la duración de la ejecución, donde la función desarrollada en Go supera en 10 veces a la función desarrollada en Python. En el caso particular de Java, se observa que su inicialización y duración de primera ejecución supera enormemente a las otras dos implementaciones, esto se debe a que Java aún no se encuentra optimizado completamente para obtener una inicialización rápida. Es importante destacar que luego de la primera ejecución, las tres implementaciones disminuyen notablemente su duración de ejecución, debido a que luego del arranque en frío, la función Lambda queda inicializada en espera de otra ejecución y también utiliza una caché propia para ejecutarse rápidamente.

Los resultados de las observaciones anteriores se resumen en la Tabla 1, donde se puede observar el tiempo de inicialización de la función y la duración de su ejecución para cada uno de los lenguajes analizados.

		Python	Go	Java
Primera prueba	Tiempo de inicialización	99.48ms	93.98ms	409.25ms
	Duración primera ejecución	22.64ms	2.09ms	323.57ms
	Duración segunda ejecución	13.43ms	2.31ms	2.36ms
Segunda prueba	Tiempo de inicialización	101.95ms	72.02ms	419.35ms
	Duración primer ejecución	20.13ms	1.73ms	350.08ms

Tabla 1

6 Conclusiones y Futuros trabajos

El objetivo del presente trabajo es analizar las buenas prácticas de desarrollo Serverless que impacten directa o indirectamente en la performance, en particular se probó la optimización del código y las dependencias de las funciones para mejorar la performance. En una publicación de 2021[18] se pudo probar las ventajas de usar algunos recursos provistos por la plataforma Serverless como es Step Function en AWS, pero por razones de espacio no se muestran estos gráficos. También es ventajoso usar los Frameworks disponibles como Serverless Frameworks, Restaría probar en ambos casos, cuanto es la mejora y bajo qué condiciones. Por otro lado no se ha llegado a evaluar todas las prácticas que consideramos que van a impactar en la performance y creemos como la mayoría están indicadas por la industria, que es necesario hacerlo cuidadosamente porque muchas de ellas pueden presentar mejoras pero solo bajo determinadas situaciones. Existen otras buenas prácticas, que por el momento no han sido analizadas dado que tratan de resolver problemas no vinculados a la performance.

Referencias

1.M. Armbrust, et al., Above the clouds: A Berkeley view of cloud computing, Tech. Rep. No. UCB/EECS-2009-28, 2009.

2.S. Ghemawat, H. Gobioff and S. T. Leung, The Google file system, *SOSP*, 2003.
3.E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uță and A. Iosup, "Serverless is More: From PaaS to Present Cloud Computing," in IEEE Internet Computing, vol. 22, no. 5, pp. 8-17, Sep./Oct. 2018, doi: 10.1109/MIC.2018.053681358.
4.Adzic, G., Chatley, R.: Serverless computing: economic and architectural impact. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 884-889). ACM.(2917)
5.AWS Lambda: aws.amazon.com/es/lambda/
6.Castro p., Ishakian v., Muthusamy v., Slominski a.: The rise of serverless computing. In: Communications of the ACM | Dec. 2019 | VOL. 62 | NO. 12 (2019)..
7.Rodríguez N., Atencio H. et al: Interoperabilidad de funciones en el Modelo de Programación de Serverless Computing. IV CICCSI. Universidad Champagnat (2020).
8.Bermbach D., Karakaya A., Buchholz S.: Using Application Knowledge to Reduce Cold Starts in FaaS Services. In: SAC '20, March 30-April 3, 2020, Brno, Czech Republic (2020).
9.Greg Wilson et al. Best Practices for Scientific Computing. https://doi.org/10.1371/journal.pbio.1001745 (2014)
10.10. EMR: Global Serverless Computing Market Outlook https://www.expertmarketresearch.com/reports/serverless-computing-market (2021).
11. Investopedia: Best Practices. Best Practices Definition (investopedia.com)
12.Cambridge Bussiness Dictionary. BEST PRACTICE | meaning in the Cambridge English Dictionary
13.What are Software Engineering Best Practices?. Software Engineering Best Practices. Principles of Programming & Development | LITSLINK Blog
14.Best practices for working with AWS Lambda functions. https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html
15.Serverless Cloud Function Best Practice. Product Documentation. Tencent Cloud
16.Yan CuiTop 10 Serverless best practices. https://www.datree.io/resources/serverless-best-practices
17.Mukherjee R. et al. Static Analysis for AWS Best Practices in Python Code. https://arxiv.org/abs/2205.04432 (2022)
18.Rodríguez N.,Atencio H. et al. Análisis de ejecución múltiple de Funciones Serverless en AWS. XXVII CACIC.(2021).