ReTEC

Revista Electrónica de Tecnología, Educación y Ciencia
ISSN: 2953-5654
http://retec.unsa.edu.ar
Universidad Nacional de Salta

Buenas Prácticas en el Desarrollo y la Programación de funciones Serverless

Nelson Rodríguez, María Murazzo, Martín Gómez

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales, Universidad Nacional de San Juan, San Juan, Argentina

nelson@iinfo.unsj.edu.ar, maritemurazzo@gmail.com, martinsj0811@gmail.com

Revista Electrónica de Tecnología, Educación y Ciencia,

Volumen 1, Número 1, pág. 15-22, jun, 2023. ISSN: 2953-5654

Disponible en http://retec.unsa.edu.ar/

Buenas Prácticas en el Desarrollo y la Programación de funciones Serverless

Nelson Rodríguez, María Murazzo, Martín Gómez

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales, Universidad Nacional de San Juan, San Juan, Argentina

nelson@iinfo.unsj.edu.ar, maritemurazzo@gmail.com, martinsj0811@gmail.com

Resumen. Serverless Computing es una arquitectura reciente para Cloud Computing que presenta diferencias con el Cloud tradicional al no reservar recursos de hardware. Presenta ventajas considerables para los usuarios, sin embargo, aún existen varias limitaciones o desventajas que no están totalmente resueltas. Si bien el desarrollo serverless presenta condiciones comunes o muy similares a la construcción de aplicaciones en otro tipo de plataformas o entornos, de las cuales se pueden obtener buenas prácticas de tipo "genéricas", también serverless presenta aspectos propios y distintivos como son: está basada en ejecución de funciones, la ejecución es conducida por eventos y las funciones no mantienen estado, 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 en especial aquellas que tiene que ver con el desarrollo, los lenguajes de programación y la implementación de aplicaciones, realizando una serie de pruebas y análisis de los resultados, 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 bastante aceptada y adoptada con varios años de utilización por parte de la industria, la academia y los usuarios en general. Según los expertos de Berkeley, "la computación en la nube se refiere tanto a las aplicaciones entregadas como servicios a través de Internet como al hardware y el software de sistemas en los centros de datos que proporcionan esos servicios. Los servicios en sí mismos se han referido durante mucho tiempo como Software como Servicio (SaaS). El hardware y el software del centro de datos es lo que llamaremos una nube" [1].

A partir de la publicación de un artículo de Google en 2003, la computación en la nube fue evolucionando desde un 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 pocos 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.

El trabajo de Johannes Manner et al [12], presenta puntos de referencia económicos y también orientados al rendimiento. Compara las plataformas AWS y Azure, y los lenguajes Java y Javascript. Analizar los factores que influyen en la duración percibida del arranque en frío mediante la realización de un banco de pruebas en AWS Lambda y Microsoft Azure Functions con 49 500 ejecuciones de funciones Cloud. Su aporte más importante es con referencia a aspectos económicos y no hace propuestas de mejoras para el arranque en frío, sino que compara el comportamiento de las plataformas.

En el trabajo de David Bermbach et al [5], presenta tres enfoques (ingenuo, el extendido y la aproximación global), que reducen el número de arranques en frío durante el tratamiento del servicio FaaS como una caja negra, implementado como parte de un middleware coreográfico liviano, utilizando composición de funciones y por lo tanto, el aprovisionamiento de nuevos contenedores antes de que el proceso de aplicación invoque la función respectiva. Las pruebas las realizan sobre AWS Lambda y OpenWhisk, con un solo lenguaje de programación Node.js.

En la publicación, de Priscilla Benedetti et al [4], se explora la conveniencia de los modos de arranque en caliente y en frío para implementar aplicaciones de IoT, teniendo en cuenta un banco de pruebas de bajos recursos comparable a un nodo en el extremo (Edge). Modelando la implementación y el análisis experimental de una plataforma serverless que incluye elementos de servicio de IoT típicos. Presenta un estudio de rendimiento en términos de consumo de recursos y latencia y para realizar las pruebas utiliza OpenFaaS, un frameworks FaaS de código abierto que permite probar una implementación de arranque en frío con una configuración precisa del tiempo de inactividad gracias a su flexibilidad.

En la publicación [7], los autores describen los principales problemas que afectan al rendimiento de las plataformas serverless y presentan algunos resultados experimentales. Esta investigación hace uso de funciones disponibles comercialmente para tres estudios de casos específicos, siendo: el entrenamiento de modelos basado en aprendizaje automático, las predicciones en vivo y la clasificación de entradas de procesamiento por lotes. Los experimentos presentados son interesantes, pero no se analiza en profundidad el problema del arranque en frío.

En el trabajo [13] se describen seis malas prácticas que han sido identificadas, y propone soluciones para tratar de superarlas. Las mismas son: las llamadas asincrónicas (que pueden incrementar la complejidad y requiere un canal de respuesta alternativo), las funciones que llaman a otras funciones (que causa una depuración compleja, y puede llevar un costo extra), el código compartido entre funciones (se podrían interrumpir las funciones serverless existentes que dependen del código compartido si éste cambia), el uso de demasiadas librerías (dado que se aumenta el espacio destinado a las mismas), adopción de demasiadas tecnologías como bibliotecas, frameworks, lenguajes (incrementa la complejidad del mantenimiento y aumenta los requisitos de conocimientos de los integrantes del proyecto) y demasiadas funciones que no son reusadas (causa menor mantenibilidad y menor comprensión del sistema).

En la publicación de Jiang, Choon Lee y Zomaya [11], aparecen resultados interesantes, a pesar de que dicho trabajo se enfoca en la ejecución de worksflow científico. Demuestran que FaaS ofrece un entorno de ejecución ideal para flujo de trabajo aplicado a las ciencias, con su mecanismo dinámico de asignación de recursos y un modelo de precios conveniente y además que AWS Lambda ofrece un entorno de ejecución ideal para aplicaciones de flujo de trabajo científicas con restricciones de precedencia complejas.

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].

4 Desarrollo de la pruebas

4.1 Ejecución múltiple

Las funciones serverless no suelen implementarse de forma aislada. En su lugar, se desencadenan para realizar una tarea en respuesta a una acción o un evento.

La mayoría de los proveedores de FaaS tienen arranques en frío de 1 a 3 segundos y esto afecta a ciertos tipos de aplicaciones donde esta latencia tendrá un impacto dramático. El arranque en frío varía según el proveedor de la nube y los lenguajes de programación.

Se analizó cuál es la estrategia más adecuada para la ejecución múltiple de funciones y además cómo impacta el arranque en frio cuando se desea ejecutar múltiples funciones en AWS Lambda y si existen estrategias que puedan minimizar su impacto.

Para realizar las pruebas se utilizó el servicio AWS Lambda.

Se tuvieron en cuenta los siguientes escenarios:

a) Se realizaron las pruebas utilizando una función que utiliza hilos de ejecución escrita en Python, en la cual por cada hilo se invoca una sola función con 20, 50 y 100 invocaciones.
b) Se procedió a realizar las pruebas de ejecución múltiple con la función Lambda Invoke perteneciente a Lambda, la cual permite que una función Lambda llame a otra función Lambda, Se realizó con 20, 50 y 100 invocaciones.

La Función Invoke, invoca una función de Lambda. Puede invocar una función de forma sincrónica (y esperar la respuesta) o de forma asincrónica.

Cabe aclarar que antes de poder ejecutar este tipo de funciones, se deben de definir un conjunto de permisos en los roles de ejecución que permita este tipo de invocación.

Se trató de encontrar si existía algún impacto del arranque en frío y por otro lado si la funcionalidad provista por Lambda (en este caso invoke) mejora la performance de la ejecución de instrucciones.

Los resultados más relevantes se obtuvieron con la ejecución de 100 funciones. A continuación se muestran los mismos:

Para 100 funciones, con 100 invocaciones de la función el resultado obtenido es 11594.42 ms, casi duplicando el tiempo obtenido para 50 ejecuciones.

Como conclusión se puede afirmar que el tiempo de ejecución es proporcional a la cantidad de ejecuciones de la función, es decir, se puede equiparar a un orden de ejecución lineal, y no se observa impacto del arranque en frío. Los resultados se muestran en la Figura 1.

Figura 1

Luego se realizaron las pruebas usando AWS Invoke. Se probó para 100 funciones, en este caso los resultados no fueron como se esperaba, ya que hubo una notable mejoría respecto del tiempo de ejecución de las anteriores pruebas, con un tiempo de ejecución de 2338,7 ms. La ejecución se muestra en la Figura 2.

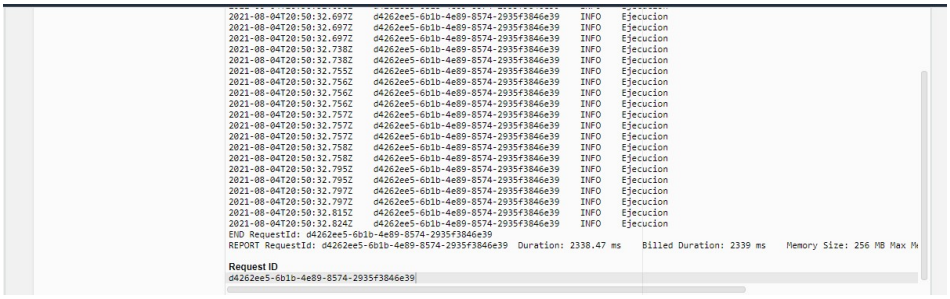

Figura 2

4.2 Optimización de código y dependencias

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 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 serverless 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.

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 no se encuentra optimizado para obtener una inicialización rápida. Es importante destacar que luego de la primera ejecución, las tres implementaciones disminuyen 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.

Tabla 1

		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

Por otro lado, se realizaron inicializaciones de operaciones CRUD en diversos lenguajes de programación y se obtuvo la tabla 2, que se muestra a continuación:

Tabla 2

Lenguaje	**Create**	**Read**	**Update**	**Delete**
JavaScript (NodeJS)	418,232	431,04	413,586	442,15
Python	384,902	400,43	361,624	411,2
Java	1623,528	1630,52	1526,12	1577,302

5 Conclusiones y Futuros trabajos

El objetivo del presente trabajo es analizar las buenas prácticas de desarrollo y programación Serverless. Las mismas impactan directa o indirectamente probando mejoras notables. En particular se probó la optimización del código y las dependencias de las funciones y se pudo demostrar que se mejora la performance. También se trató de probar el impacto del arranque en frío, sin embargo cuando se realizaron pruebas de forma repetida, AWS demostró que provee recursos que minimizan este problema o al menos en estas pruebas no pudo apreciarse.

En cuanto a las pruebas de ejecución múltiple, se pudo comprobar que Lambda Invoke es una buena alternativa a medida que se escala con la cantidad de funciones, ya que reduce la complejidad temporal lineal que se ve en la implementación con hilos. En una publicación de 2021[18] se pudo comprobar además 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 en este trabajo. También resulta una práctica conveniente utilizar los Frameworks disponibles como Serverless Frameworks, dado que los mismos no solo facilitan la programación, sino también tienen optimizada la ejecución de las funciones. En cuanto a los lenguajes de programación no existen diferencias importantes entre Go, NodeJS y Python, pero Java si presenta un tiempo de inicialización considerable. 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 estrictamente al desarrollo o la programación de funciones.

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. EMR: Global Serverless Computing Market Outlook
11. https://www.expertmarketresearch.com/reports/serverless-computing-market (2021).
12. Investopedia: Best Practices. Best Practices Definition (investopedia.com)
13. Cambridge Bussiness Dictionary. BEST PRACTICE | meaning in the Cambridge English Dictionary
14. What are Software Engineering Best Practices?. Software Engineering Best Practices. Principles of Programming & Development | LITSLINK Blog
15. Best practices for working with AWS Lambda functions. https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html
16. Serverless Cloud Function Best Practice. Product Documentation. Tencent Cloud
17. Yan CuiTop 10 Serverless best practices. https://www.datree.io/resources/serverless-best-practices
18. Mukherjee R. et al. Static Analysis for AWS Best Practices in Python Code. https://arxiv.org/abs/2205.04432 (2022)
19. Rodríguez N.,Atencio H. et al. Análisis de ejecución múltiple de Funciones Serverless en AWS. XXVII CACIC.(2021).