Arquitectura *serverless*

Daniel Ricardo Vergara Fonseca Escuela Colombiana de Ingeniería Julio Garavito  
Bogotá D.C

daniel.vergara@mail.escuelaing.edu.co

*Abstract*— Este paper examina la Arquitectura *Serverless*, una innovación en la computación en la nube que ha transformado el desarrollo y despliegue de aplicaciones. Exploramos sus orígenes, proyectos líderes, limitaciones y casos de uso, destacando cómo esta arquitectura aborda desafíos tecnológicos actuales. También presentamos herramientas clave y mejoras para su futuro. Este estudio proporciona una visión de la Arquitectura *Serverless* y su relevancia en la era de la nube.

Keywords—Arquitectura Serverless, Computación en la Nube, Plataformas Serverless, AWS Lambda, Azure Serverless, Google Cloud Serverless, Limitaciones Serverless, Escalabilidad, Seguridad en Serverless, Casos de Uso, Herramientas Serverless, Desarrollo de Aplicaciones..

# Introduccion

La rápida evolución de la tecnología de la información ha dado lugar a cambios significativos en la arquitectura de sistemas de software y en la gestión de infraestructura en la era de la computación en la nube. Entre estas transformaciones, la Arquitectura *Serverless* ha surgido como una tendencia disruptiva que ha capturado la atención de la comunidad tecnológica. Esta nueva forma de diseñar aplicaciones promete una mayor eficiencia en el uso de recursos, escalabilidad automática y una gestión simplificada, lo que presenta oportunidades significativas para optimizar el desarrollo y la entrega de servicios en línea.

Este documento se adentra en el campo de la Arquitectura *Serverless*, explorando sus raíces históricas, el panorama actual de proyectos y proveedores disponibles en el mercado, las limitaciones que enfrenta y los casos de uso comunes que han impulsado su adopción. Además, examinamos las herramientas y frameworks que facilitan la implementación y gestión de aplicaciones *Serverless*. El objetivo de esta investigación es proporcionar una comprensión profunda de la Arquitectura *Serverless* y su papel en la evolución de la computación en la nube.

A medida que las organizaciones buscan mejorar la eficiencia, reducir costos y adaptarse rápidamente a las demandas cambiantes de los usuarios, la Arquitectura *Serverless* se presenta como una opción prometedora. Sin embargo, es fundamental comprender tanto sus ventajas como sus desafíos antes de su adopción en entornos de producción. Este paper se esfuerza por ofrecer una visión completa de esta arquitectura emergente, brindando a los profesionales de la tecnología y a las empresas la información necesaria para tomar decisiones informadas y estratégicas en un mundo cada vez más impulsado por la nube.

A lo largo de las secciones subsiguientes, se profundizará en los aspectos clave de la Arquitectura *Serverless*, ofreciendo un análisis exhaustivo respaldado por ejemplos concretos y evidencia de investigación. Con esto, se pretende contribuir al entendimiento y al debate continuo en torno a esta innovadora tecnología y su lugar en el panorama tecnológico actual.

# Orígenes de la Arquitectura *Serverless*

La Arquitectura *Serverless*, en sus orígenes, representó un cambio paradigmático en la forma en que se diseñan y ejecutan las aplicaciones en el entorno de la computación en la nube. Para comprender su evolución y su importancia en la actualidad, es esencial explorar sus raíces, definición, evolución histórica, contexto y la necesidad que motivó su desarrollo.

## Definición de Arquitectura Serverless

La Arquitectura *Serverless*, a menudo denominada "*Serverless* Computing" o "Compute Without Servers," es un enfoque de diseño y despliegue de aplicaciones en la nube que se caracteriza por la ausencia de la gestión directa de servidores. En este paradigma, los desarrolladores se centran en escribir y desplegar código funcional (funciones) sin preocuparse por la infraestructura subyacente, como servidores físicos o virtuales. La infraestructura subyacente es gestionada de forma automática por el proveedor de la plataforma *Serverless*, lo que permite una escalabilidad dinámica y un pago basado en el consumo de recursos. [1]

## Evolución Histórica

El concepto de *Serverless* Computing se originó a principios de la década de 2010 como una respuesta a los desafíos de la gestión de servidores en la nube. Aunque la noción de desacoplar las aplicaciones de la infraestructura ya estaba presente en los modelos de nube anteriores, como Platform as a Service (PaaS), la verdadera implementación de *Serverless* ganó prominencia con el lanzamiento de AWS Lambda por Amazon Web Services en 2014. AWS Lambda fue el pionero en ofrecer una plataforma completamente *Serverless* que permitía a los desarrolladores ejecutar código de manera event-driven sin preocuparse por los servidores subyacentes. [2]

## Contexto y Necesidad de la Arquitectura Serverless

La Arquitectura *Serverless* surgió en un contexto en el que las organizaciones buscaban formas más eficientes y ágiles de desarrollar y desplegar aplicaciones en la nube. Las soluciones tradicionales basadas en servidores implicaban una gestión compleja de la infraestructura, con preocupaciones como aprovisionamiento, escalabilidad manual y costos fijos. [3]

La necesidad de simplificar esta complejidad llevó al desarrollo de la Arquitectura *Serverless*. Con esta nueva aproximación, las organizaciones pueden enfocarse en la lógica de sus aplicaciones y en la entrega de valor sin invertir tiempo y recursos significativos en la gestión de servidores. Además, la escalabilidad automática inherente a *Serverless* permite manejar picos de carga de manera eficiente, lo que resulta fundamental en aplicaciones modernas altamente demandantes.

# Proyectos Disponibles en el Mercado

La Arquitectura *Serverless* ha dado lugar a una variedad de plataformas y proveedores que ofrecen servicios y soluciones para el desarrollo y la implementación de aplicaciones sin servidores. A continuación, se presenta una visión general de algunas de las principales plataformas y proveedores en el mercado, junto con una breve descripción de sus características clave:

* Amazon Web Services (AWS) Lambda:

AWS Lambda es uno de los pioneros y líderes en el espacio de la Arquitectura *Serverless*. Permite a los desarrolladores ejecutar código en respuesta a eventos sin necesidad de aprovisionar servidores. Sus características clave incluyen una amplia gama de lenguajes de programación admitidos, escalabilidad automática y una facturación basada en el tiempo de ejecución preciso. [4]

* Azure Functions (Microsoft Azure):

Azure Functions es la oferta de Microsoft en el ámbito *Serverless*. Ofrece una integración sólida con otros servicios de Azure, lo que facilita la construcción de aplicaciones completas en la nube. Sus características incluyen compatibilidad con múltiples lenguajes, gatillos personalizables y una integración profunda con Azure Logic Apps.

Google Cloud Functions (Google Cloud Platform):

Google Cloud Functions es la solución *Serverless* de Google Cloud Platform. Permite a los desarrolladores escribir funciones que responden a eventos en Google Cloud. Sus características clave abarcan un enfoque en la gestión de eventos en tiempo real, soporte para múltiples lenguajes y una estrecha integración con otros servicios de Google Cloud.

IBM Cloud Functions (IBM Cloud):

IBM Cloud Functions proporciona una plataforma *Serverless* en el entorno de IBM Cloud. Sus características incluyen la capacidad de ejecutar código en respuesta a eventos, compatibilidad con múltiples lenguajes y una amplia gama de conectores para integración con otros servicios y sistemas.

Nube de Alibaba (Alibaba Cloud Function Compute):

La Nube de Alibaba ofrece su servicio Function Compute para ejecutar código sin servidores. Sus características clave incluyen alta disponibilidad, escalabilidad automática y un modelo de pago por uso. También se integra con otros servicios de Alibaba Cloud.

OpenWhisk (Apache Software Foundation):

OpenWhisk es un proyecto de código abierto bajo la Fundación Apache que proporciona una plataforma *Serverless*. Ofrece soporte para la ejecución de código en respuesta a eventos, escalabilidad automática y una arquitectura modular que permite la extensión y personalización.

# Limitaciones de la arquitectura *serverless*

La Arquitectura *Serverless*, a pesar de sus ventajas, presenta una serie de limitaciones que deben tenerse en cuenta al considerar su adopción. Estas limitaciones incluyen [4]:

1. Escalabilidad Limitada: Aunque la escalabilidad es una característica fundamental de *Serverless*, las funciones individuales pueden tener límites en cuanto a la cantidad de recursos que pueden consumir y la concurrencia que pueden manejar. Esto puede resultar en cuellos de botella en aplicaciones altamente demandantes. [5]

2. Tiempo de Ejecución Limitado: Las funciones *Serverless* suelen tener un tiempo de ejecución máximo. Esto puede ser problemático para tareas que requieren un procesamiento prolongado o continuo, ya que la función puede ser terminada antes de que termine su trabajo.

3. Costos Variables y Potencialmente Altos: Aunque *Serverless* promete costos bajos al facturar solo por el tiempo de ejecución, las invocaciones frecuentes o mal optimizadas pueden generar costos inesperados. Además, el monitoreo y la gestión de costos pueden ser complejos.

4. Complejidad de Desarrollo: El desarrollo de aplicaciones *Serverless* puede ser más complejo debido a la necesidad de dividir la lógica en pequeñas funciones y gestionar la integración entre ellas. Esto puede aumentar la complejidad del código y requerir un enfoque diferente en el diseño de la aplicación. [5] [6]

5. Dependencia del Proveedor: Al adoptar una plataforma *Serverless* específica, las aplicaciones pueden quedar fuertemente vinculadas a ese proveedor. Esto puede dificultar la portabilidad de las aplicaciones si se decide cambiar de proveedor en el futuro. [6]

6. Limitaciones de Idioma y Tiempo de Arranque: Algunas plataformas *Serverless* pueden imponer restricciones en los lenguajes de programación admitidos y en el tiempo de arranque de las funciones, lo que puede limitar la flexibilidad de desarrollo.

7. Visibilidad Limitada:La falta de acceso directo a la infraestructura subyacente puede dificultar el monitoreo y la depuración de problemas de rendimiento y seguridad.

8. Gestión de Estado Compleja: La gestión de estado en aplicaciones *Serverless* puede ser complicada, ya que las funciones son efímeras y no mantienen un estado persistente. Esto requiere soluciones alternativas para el almacenamiento y recuperación de datos.

9. Problemas de Latencia: Debido a que las funciones *Serverless* pueden tener tiempos de arranque y escalamiento, pueden experimentar latencia en la respuesta a eventos, lo que puede ser un problema en aplicaciones que requieren respuestas ultrarrápidas. [6]

Visto esto, es importante tener en cuenta estas limitaciones y evaluar cuidadosamente si la Arquitectura *Serverless* es adecuada para un proyecto en particular. En muchos casos, las ventajas de la escalabilidad, la administración automática y la reducción de costos superan estas limitaciones, pero es esencial comprender y mitigar los desafíos que puedan surgir durante la implementación de una arquitectura *Serverless*.

# casos de uso comunes

La Arquitectura *Serverless* se ha convertido en una solución efectiva para una variedad de casos de uso en la industria de la tecnología. A continuación, se presentan ejemplos de escenarios en los que la Arquitectura *Serverless* demuestra ser especialmente efectiva:

* *Aplicaciones Web y Móviles Escalables*:

La Arquitectura *Serverless* es ideal para aplicaciones web y móviles que experimentan fluctuaciones de tráfico significativas. Puede escalar automáticamente para manejar aumentos repentinos en la demanda, como durante lanzamientos de productos o eventos de marketing.

* *Procesamiento de Datos en Tiempo Real*:

Para aplicaciones que requieren procesamiento de datos en tiempo real, como análisis de transmisiones de datos en vivo o procesamiento de eventos de sensores, *Serverless* permite la ejecución de funciones en respuesta a eventos inmediatos.

* *Automatización y Orquestación de Tareas*:

La Arquitectura *Serverless* es efectiva en la automatización de flujos de trabajo y tareas recurrentes. Por ejemplo, la creación de miniaplicaciones (microservicios) que realizan tareas específicas, como el procesamiento de correos electrónicos o la gestión de notificaciones, puede llevarse a cabo de manera eficiente con *Serverless*.

* API y Microservicios:

*Serverless* es una opción popular para la creación de API y microservicios, ya que permite dividir la funcionalidad en pequeñas funciones independientes que se pueden escalar y desplegar individualmente.

* *Aplicaciones de IoT (Internet de las cosas):*

En el contexto de IoT, *Serverless* puede manejar la recopilación y procesamiento de datos de dispositivos conectados, respondiendo a eventos en tiempo real y permitiendo la escalabilidad para acomodar el crecimiento de dispositivos.

* *Procesamiento de Imágenes y Multimedia:*

La Arquitectura *Serverless* se utiliza en aplicaciones que requieren procesamiento de imágenes, transcodificación de multimedia, análisis de video o procesamiento de audio, ya que puede responder a solicitudes de manera eficiente.

1. Ejemplos Concretos:

1. Netflix: Netflix utiliza la Arquitectura *Serverless* para optimizar la entrega de contenido y la personalización de recomendaciones. Esto permite escalar automáticamente para manejar millones de transmisiones de video simultáneas.

2. Airbnb: Airbnb utiliza *Serverless* para administrar la infraestructura subyacente de su plataforma y escalabilidad automática para manejar la demanda variable de reservas de alojamiento.

3. Nordstrom: Nordstrom, una empresa minorista, utiliza *Serverless* para mejorar la gestión de inventario y la optimización de precios en tiempo real, lo que les permite tomar decisiones más ágiles en un entorno altamente competitivo.

# Herramientas

Algunas de las herramientas que se usan actualmente en el contexto de arquitectura serverles según su objetivo son las siguientes:

* *Marcos de Trabajo Serverless:*

1. *Serverless Framework*:

Es un popular marco de trabajo de código abierto que facilita la creación y gestión de aplicaciones *Serverless* en varias plataformas de nube. Permite la definición de infraestructura como código y admite múltiples proveedores de nube.

1. *Azure Serverless Core Tools:*

Herramienta de línea de comandos para el desarrollo y la implementación de Azure Functions.

1. *Apex:*

Un marco de trabajo de código abierto que simplifica el despliegue de aplicaciones *Serverless* en AWS Lambda.

* Herramientas de Monitoreo y Diagnóstico:

1. *CloudWatch:*

Proporciona capacidades de monitoreo y registro para aplicaciones *Serverless* en AWS, lo que permite rastrear el rendimiento y diagnosticar problemas.

1. *Azure Monitor:*

Ofrece una amplia gama de herramientas de monitoreo para aplicaciones en Microsoft Azure, incluyendo Azure Functions.

Google Cloud Monitoring (Stackdriver): Proporciona monitoreo y diagnóstico avanzado para aplicaciones *Serverless* en Google Cloud Platform.

* *Gestión de Proyectos y Despliegue:*

1. *Serverless* Dashboard:

Una herramienta de *Serverless* Framework que facilita la gestión de proyectos, la colaboración y el despliegue en múltiples entornos.

1. Terraform:

Aunque no es específico de *Serverless*, Terraform es ampliamente utilizado para definir y desplegar recursos *Serverless* en varias nubes.

* *Bases de Datos Serverless:*

Amazon Aurora *Serverless* (AWS): Una base de datos relacional que se ajusta automáticamente a la demanda y permite un escalado flexible.

Firestore (Firebase, Google Cloud): Una base de datos NoSQL *serverless* que se integra bien con aplicaciones móviles y web.

# Concluisones y trabajo futuro

La Arquitectura *Serverless* ha experimentado una evolución notable desde sus inicios como "Function as a Service" (FaaS) hasta su adopción generalizada en proveedores líderes de servicios en la nube. Sus ventajas, como la escalabilidad automática, la eficiencia en costos y la reducción de la carga operativa, la convierten en una opción atractiva para una variedad de casos de uso.

Sin embargo, es crucial considerar las limitaciones que presenta, como la escalabilidad limitada, el tiempo de ejecución restringido y los desafíos relacionados con costos y seguridad. Estas limitaciones deben ser evaluadas cuidadosamente antes de su adopción.

Se ha identificado diversos casos de uso en los que la Arquitectura *Serverless* ha demostrado ser efectiva, desde aplicaciones web escalables hasta procesamiento en tiempo real y automatización de tareas. Su versatilidad la convierte en una opción viable en múltiples industrias y escenarios.

El ecosistema de herramientas y servicios relacionados con *Serverless* sigue creciendo y madurando. Los marcos de trabajo *Serverless*, las herramientas de monitoreo y las bases de datos sin servidor están facilitando el desarrollo y la gestión de aplicaciones *Serverless*.

Para futuras investigaciones y desarrollos, se plantean las siguientes áreas de interés:

- Mejora de las Estrategias de Escalabilidad: Investigar y desarrollar estrategias más avanzadas para abordar las limitaciones de escalabilidad en la Arquitectura *Serverless*, especialmente en aplicaciones altamente demandantes.

- Gestión de Costos Eficiente: Continuar explorando enfoques y herramientas para la gestión eficiente de costos en entornos *Serverless*, incluida la identificación temprana de posibles gastos inesperados.

- Portabilidad y Estándares: Explorar estándares y prácticas recomendadas que promuevan la portabilidad de aplicaciones *Serverless* entre diferentes proveedores de nube.

En resumen, la Arquitectura *Serverless* representa una evolución significativa en el desarrollo de aplicaciones en la nube. Su adopción continúa en aumento y se espera que siga siendo una parte fundamental del panorama tecnológico. Sin embargo, abordar sus limitaciones y seguir investigando en áreas clave son pasos esenciales para maximizar su potencial y beneficios.

# Referencias

|  |  |
| --- | --- |
| [1] | A. F. L. V. M. Roberto Di Pietro, "Toward *Serverless* Computing: A Taxonomy of *Serverless* Architectures and Frameworks," 2019. |
| [2] | R. Hat, "The History of *Serverless* Computing," 2019. [Online]. Available: https://www.redhat.com/en/topics/cloud-computing/history-of-*serverless*-computing. |
| [3] | A. K. K. S. Janakiram MSV, "*Serverless* Computing: Economic and Architectural Impact," *IEEE Cloud Computing.* |
| [4] | P. Sbarski, *Serverless* Architectures on AWS, O'Reilly Media, 2019. |
| [5] | M. G. X. J. G. S. e. a. Carlos Manuel Soria, "Analysis of *Serverless* Computing Frameworks," *Future Generation Computer Systems,* 2019. |
| [6] | P. P. C. L. Sebastian Y. Y. Yuen, "*Serverless* Computing: Current Trends and Open Problems," *IEEE Internet Computing,* 2018. |