Serverless computing

Miguel Torio[1]

Universidad Católica "Nuestra Señora de la Asunción", Sede regional de Asunción,
Campus de Santa Librada, Asunción, Paraguay,
migueltorio@hotmail.com

Resumen Este documento es el resultado de un proceso de investigación sobre la computación sin servidores. Analizamos las características de este nuevo paradigma, la arquitectura tanto del ambiente de ejecución como de las posibles aplicaciones y por último exponemos las herramientas que nos ofrece el mercado así como los costos asociados.

Keywords: Serverless computing, Software as a service, Microservice, Amazon Lambda, Azure Functions, Google Cloud Functions

1. Introducción

El término *informática sin servidores* es contradictorio ya que técnicamente ninguna aplicación puede funcionar sin un servidor (o computadora que ejecute algún tipo de código).

Serverless computing o informática sin servidores, es una forma de *cloud computing* o informática en la nube en la que un proveedor en la nube, en forma dinámica, gestiona el alojamiento de los recursos computacionales. El precio de sus servicios se basa en la cantidad de recursos consumidos por una aplicación.[1]

Por lo tanto, los servidores siguen existiendo y lo único diferente es que lo hacen debajo de una capa de abstracción (invisible), de esta forma la tarea del manejo de la infraestructura queda relegada al proveedor del servicio y no al programador.[2]

Duarante el proceso de desarrollo de software un aspecto, no menos importante, es determinar la cantidad de recursos computacionales que serán requeridos (servidores, almacenamiento y base de datos) para hacer funcionar una aplicación. La informática sin servidores intenta simplificar este aspecto en una forma de *utility computing*, prestando servicios bajo demanda en vez de a través de una tarifa plana por el servicio.

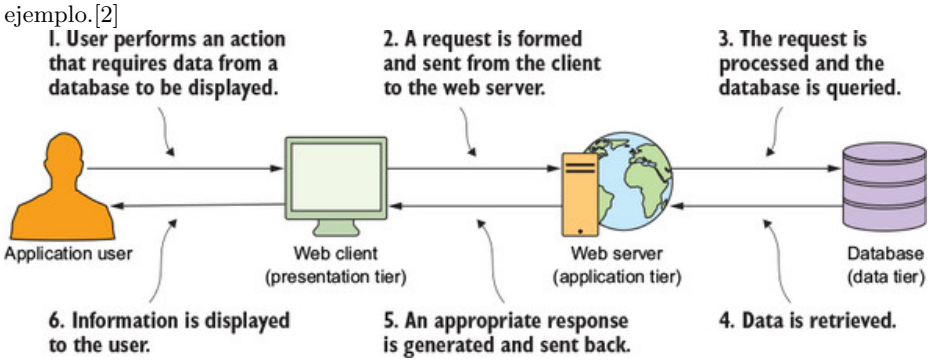
2. Principios de la arquitecturas sin servidor

A modo de introducción, tomaremos como ejemplo un aplicación moderna que consta de dos componentes principales: un *backend* que realiza diversas ta-

reas de computación del lado del servidor y un *frontend* que sirve a modo de interface para que los usuarios operen la aplicación a través de un navegador, un dispositivo móvil o un equipo de escritorio.

Cada vez que el *backend* recibe una petición, los datos pasan a través de varias capas de aplicación antes de realizar una operación propia de la lógica de negocios (como guardar información en una base de datos, enviar un correo electrónico o incluso interactuar con otro sistema). A medida que la complejidad de la aplicación aumenta, se deberán incluir otros elementos como: balanceadores de carga, cacheo, clustering, redundancia de datos etc. Estos elementos deben ejecutarse en servidores especializados, normalmente en *datacenters* con las implicaciones que esto conlleva como ser: la administración, el mantenimiento y las actualizaciones de seguridad entre otros.

Figura 1. Ejemplo de un request-response. En este ejemplo solo existe un servidor y una base de datos. La mayoría de los sistemas no son tan simples como en este ejemplo.[2]

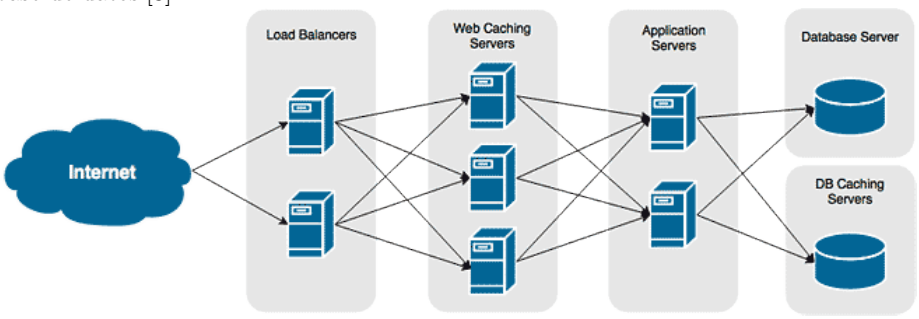


Aprovisionar, gestionar y actualizar servidores es una tarea compleja y necesaria en cualquier sistema. También es una distracción en lo que debería el objetivo principal .enfocarse en el problema de la lógica de negocios".

El paradigma sin servidor pretende evitar las preocupaciones de infraestructura, osea permitie al desarrollador centrarse principalmente en el código, que es el objetivo final detrás del proceso de desarrollo de software.

La arquitectura sin servidor permite ejecturar código sin necesidad de provisionar servidores, instalar software, desplegar contenedores o preocuparse por detalles de bajo nivel. El proveedor se ocupa del aprovisionamiento y la gestión de los servidores que ejecutan el código real y proporciona una infraestructura de computación de alta disponibilidad -incluyendo el aprovisionamiento de capacidad y escalado automatizado- detalles en los que el desarrollador no necesita

Figura 2. Un ejemplo de configuración típica para la mayoría de los sitios web de tamaño medio. Las solicitudes comienzan en el equilibrador de carga, que distribuye la carga entre los servidores de caché web HTTP. Si los servidores de caché no tienen una copia en caché de la solicitud, se envía a los servidores de aplicaciones para que se procesen. El servidor de aplicaciones, consulta los servidores de caché de base de datos para una copia en caché de las consultas de base de datos. Si existe un caché, se envía al servidor de aplicaciones, de lo contrario, la aplicación debe consultar el servidor de base de datos.[3]



pensar.

Entonces, las arquitecturas sin servidor se refieren a estos nuevos tipos de arquitectura de software que no dependen del acceso directo a un servidor para funcionar con el objetivo de construir aplicaciones poco acopladas, escalables y eficientes rápidamente.[3]

2.1. Arquitectura orientada a servicios y microservicios

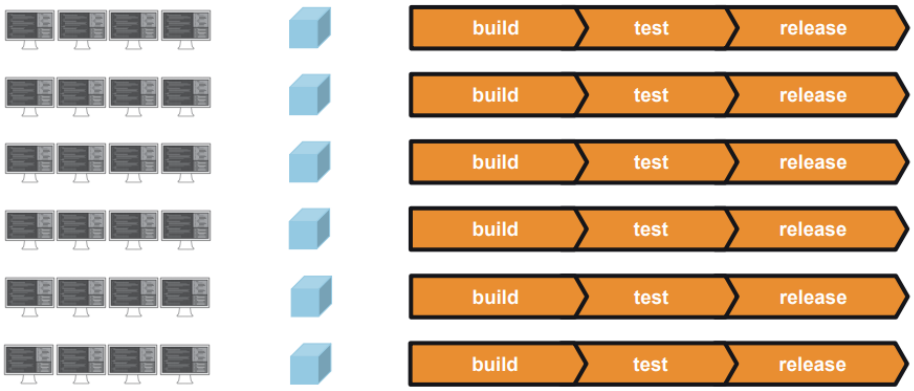
Las arquitecturas orientadas a servicios se conceptualizaron a partir de la idea de que un sistema puede estar compuesto de muchos servicios independientes (microservicios).

No se dicta el uso de ninguna tecnología en particular. En su lugar, se fomenta un enfoque arquitectónico en el que los desarrolladores crean servicios autónomos que se comunican a través del paso de mensajes y, a menudo, tienen un esquema que define cómo se crean o intercambian los mensajes.

Podemos pensar en los microservicios como módulos pequeños, autónomos y totalmente independientes construidos alrededor de un propósito o capacidad comercial particular.

Como cada microservicio es independiente al sistema en general por lo que podemos utilizar diferentes lenguajes de programación para programar diferentes microservicios aunque, sin un disciplina estricta, tener una mezcla de lenguajes puede conducir a la confusión a medida que el ciclo de vida del software vaya

Figura 3. Ciclo de desarrollo de un microservicio[4]



avanzando.

Como cada microservicio es independiente y no puede mantener la información de su estado, luego de la ejecución, entonces nuestras solicitudes deben tener toda la información necesaria para procesar la solicitud, para lo cual, normalmente se emplean bases de datos. Esta característica permite que si los microservicios están desacoplados correctamente, los equipos de desarrollo pueden trabajar y desplegar microservicios independientemente unos de otros. Por otra parte, la consistencia, el manejo de errores y transacciones pueden hacer laas cosas más difíciles (especialmente sin un plan bien pensado).

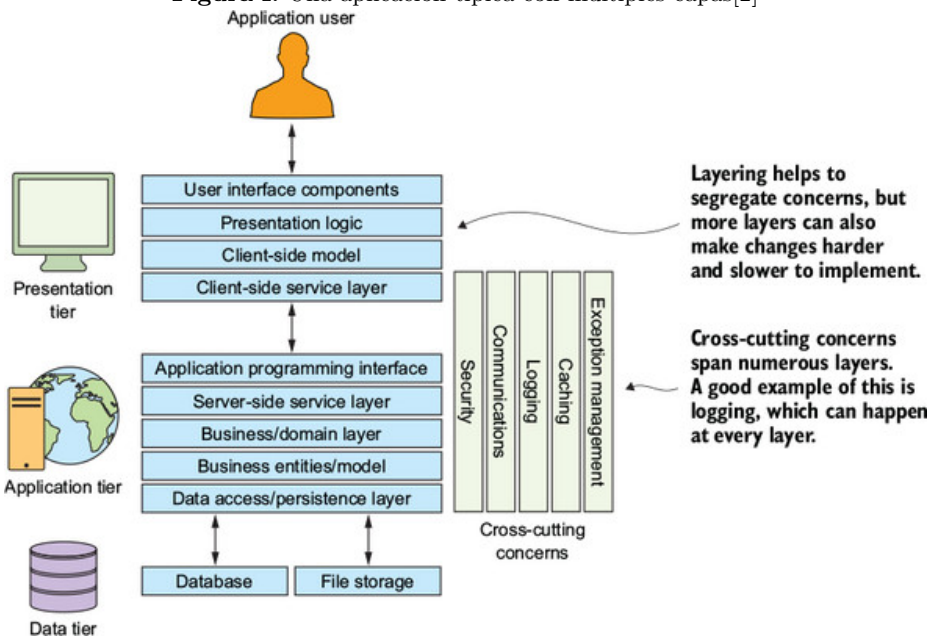
Se puede argumentar que la arquitectura sin servidor incorpora muchos principios de los microservicios. Después de todo, dependiendo del diseño del sistema, cada función de cálculo podría considerarse como su propio servicio autónomo.

Las arquitecturas sin servidor dan la libertad, al programador, de aplicar tantos principios de microservicio como crea necesario sin forzarlo por un solo camino.

2.2. Principios de diseño de software

Un principio básico del diseño de software, consiste en separarlo en niveles (presentación, datos, aplicación y lógica de negocios) y además dentro de cada nivel, en varias capas lógicas en donde cada una trata un aspecto particular de la funcionalidad. El diseño por capas permite desarrollar aplicaciones mas mantenibles.

Figura 4. Una aplicación típica con multiples capas[2]

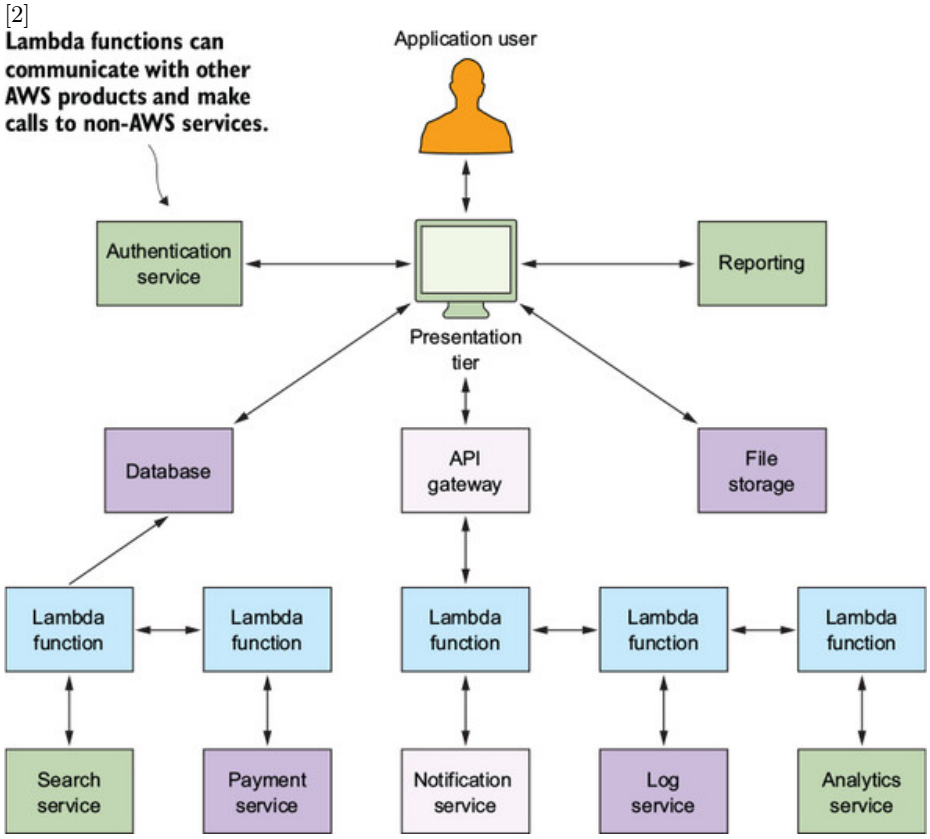


Una desventaja de tener demasiadas capas puede conducir a ineficiencias. Un pequeño cambio a menudo puede producir cascadas y hacer que el desarrollador tenga que modificar varias capas (incluso todas) en todo el sistema, lo que implica tiempo y costos adicionales tanto en la implementación y en las pruebas. Cuantas más capas tenga nuestro sistema, más complejo y poco manejable será el mismo a lo largo del tiempo.

Las arquitecturas sin servidor pueden ayudar con el problema de capas y de tener que actualizar demasiadas cosas. Hay espacio para que los desarrolladores eliminen o minimicen las capas separando el sistema en funciones y permitiendo que el front end se comunique de forma segura con los servicios e incluso con la base de datos directamente.

Aunque un enfoque sin servidor no resuelve todos los problemas, ni elimina las complejidades subyacentes del sistema. Pero cuando se implementa correctamente, puede proporcionar oportunidades para reducir, organizar y administrar la complejidad. Una arquitectura sin servidor bien planificada puede facilitar los cambios futuros, lo cual es un factor importante para cualquier aplicación a largo plazo.

Figura 5. Ejemplo de una arquitectura sin servidor donde no hay un único back end tradicional. El front end de la aplicación se comunica directamente con los servicios, la base de datos o las funciones de cálculo. Algunos servicios deben estar ocultos detrás de las funciones, donde pueden tener lugar medidas adicionales de seguridad y validación [2]



2.3. Principios de la arquitectura sin servidor

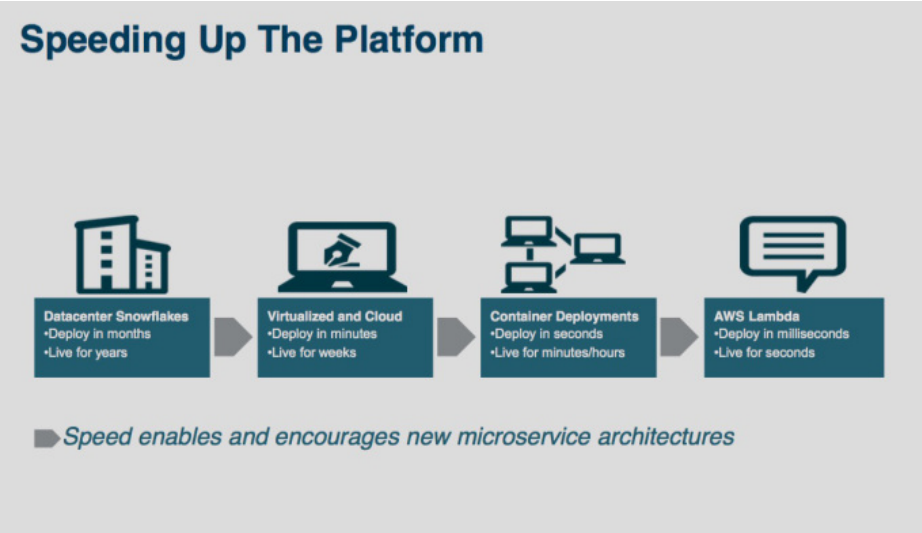
A lo largo de los años, la velocidad de despliegue y el tiempo de vida de las aplicaciones ha disminuido drásticamente. Con la implementación de las arquitecturas sin servidor, el tiempo se reduce a milisegundos.

El despliegue de una aplicación en arquitecturas sin servidor se puede resumir así: ***escribir código, definir los desencadenantes y luego el código se ejecutará cuando se reunan las condiciones requeridas.***

Los desencadenantes pueden ser acciones como un usuario cargando un archivo desde un teléfono inteligente o haciendo clic en el botón comprar de un sitio web, o podrían ser acciones de máquina a máquina sin que los seres humanos participen. La idea es que son flexibles por lo que casi cualquier cosa puede ser

un disparador.

Figura 6. Evolución del desarrollo [1]

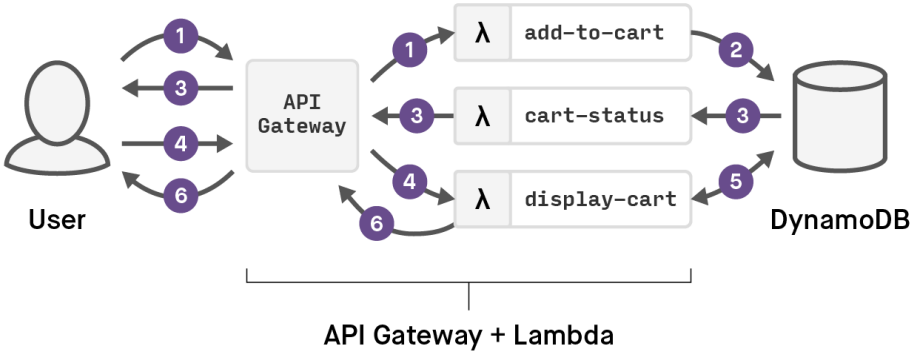


Ejecutar código sobre demanda Los desarrolladores pueden escribir funciones para realizar casi cualquier tarea común, como leer y escribir en una fuente de datos, llamar a otras funciones y realizar un cálculo. En casos más complejos, los desarrolladores pueden establecer pipelines [1] más elaborados y orquestar invocaciones de múltiples funciones. Puede haber escenarios en los que todavía se necesita un servidor para hacer algo. Estos casos, sin embargo, pueden ser muy pocos, y como desarrollador debe evitar ejecutar e interactuar con un servidor si es posible.

Funciones sin estado y de un solo propósito Las funciones sin estado y de un solo propósito, implican que la petición realizada al servidor contiene toda la información necesaria para su procesamiento y que además los recursos no sobrevivirán una vez que termine la petición.

[1] " Pipeline o tuberia, consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. Es común el uso de buffer de datos entre elementos consecutivos."

Figura 7. Ejemplo de funciones sin estado de un solo propósito [5]



Funciones impulsadas por eventos basadas en tuberias Las funciones basadas en tuberias son buenas candidatas para ser programadas en arquitecturas sin servidor.

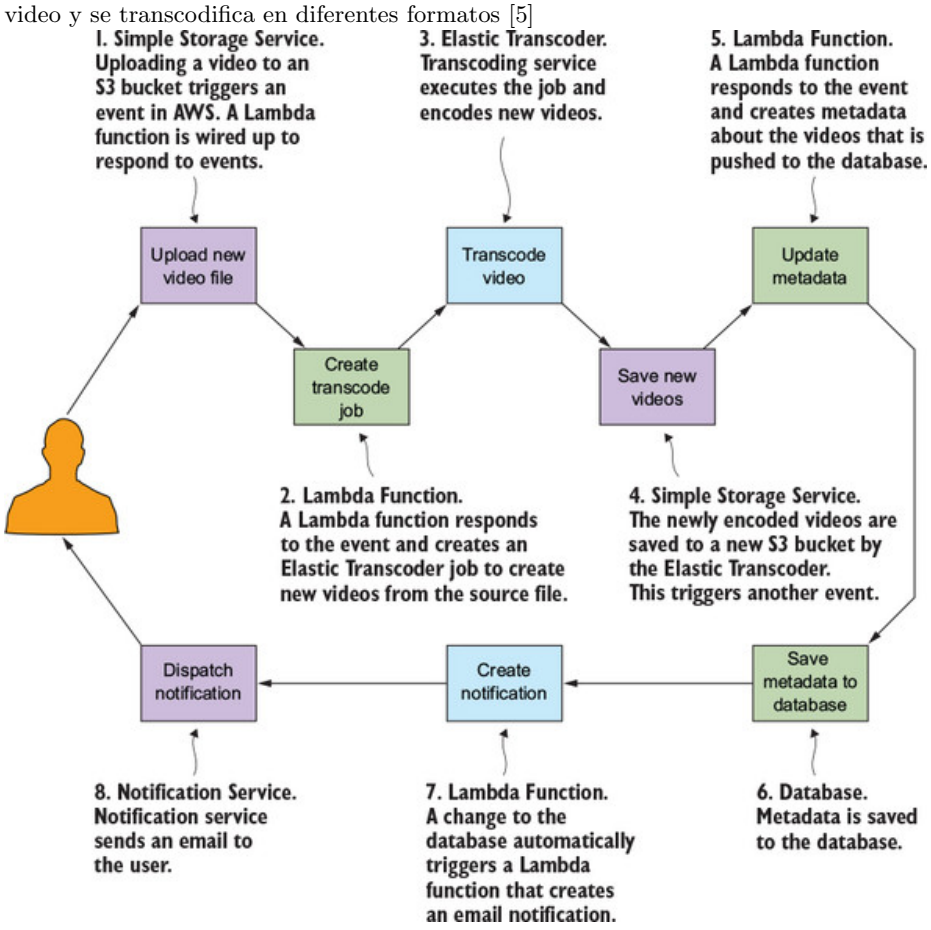
3. Ventajas y desventajas del modelo sin servidor

Segun Matt Wood (Gerente de estrategia de productos de Amazon Web Services) La informática sin servidor funciona mejor en dos tipos de escenarios. En un extremo del espectro, es posible que se tenga una situación en la que las acciones ocurren con poca frecuencia y no tiene mucho sentido pagar por los servidores que no se estarán utilizando la mayoría del tiempo, como el escenario semanal de análisis de fotografías tomadas por un drone.

En el otro extremo, se podría estar construyendo algo grande y complejo que necesita escalar rápidamente y tratar de desplegar la infraestructura sería un desafío. Supongamos que se tiene una red de sensores meteorológicos que le dan información y una vez que la información se recopila un número de cosas tienen que suceder. Se puede desencadenar un evento cada vez que el sensor envía datos, y programar la serie de acciones necesarias, teniendo en mente que esto es probable que ocurra (un evento) muy a menudo, medido en fracciones de segundos.

Si bien este modelo no es una solución mágica por ningún medio, es una nueva herramienta para los desarrolladores que podrían no necesitar una configuración de servidor más tradicional y les da opciones cuando diseñan el programa y deciden cómo implementarlo.

Figura 8. Ejemplo de una función basada por eventos, cuando un usuario sube un video y se transcodifica en diferentes formatos [5]



3.1. Ventajas

Abstracción de los recursos computacionales La primera ventaja del modelo sin servidor es que no hay no hay infraestructura para gestionar. Debido a que la máquina subyacente (física y virtual) se abstraen, las tareas tradicionales tales como como el aprovisionamiento de infraestructuras, la gestión de actualizaciones, entre otros, dejan de ser relevantes.

Por lo tanto, pasar a un modelo sin servidor puede permitir destinar más recursos a perfeccionar el código de nivel de aplicación.

Bajo demanda Con serverless, los usuarios sólo pagan por los recursos de computación que son consumido de manera útil, reduciendo el desperdicio y aumentando la eficiencia.

A diferencia de la mayoría de las ofertas de computación virtual que normalmente se facturan por hora (u otra fracción de tiempo), las plataformas sin servidor facturan a un nivel muy granular(por milisegundos). Esto significa que no hay perdida de dinero en recursos que están inactivos o subutilizados.

Con el cambio a un modelo de üso cero = costo cero", se evidencian otras ventajas como:

- El crecimiento del coste es lineal, lo que permite predecir los costos en forma directa
- El ritmo de la innovación puede ser mayor, ya que los experimentos pueden ser realizados a un costo extremadamente bajo
- Las aplicaciones de pruebas de carga se puede realizar sin la necesidad de mantener los entornos de tamaño de producción a un alto costo
- Los entornos de producción y no productivos pueden configurados y dimensionados de forma idéntica sin impactar el gasto, lo que ayuda a las pruebas representativas y reduce la probabilidad de problemas de producción

Computación literalmente sin límites Al utilizar la informática sin servidor no hay restricción en los recursos de cálculo, esto significa que las cargas de trabajo pueden cubrir amplios conjuntos de datos o servir cargas extremadamente altas. Otra ventaja es la facilidad para ejecutar múltiples invocaciones de funciones simultáneamente. Debido a que el proveedor de infraestructuras puede gestionar con mayor eficacia en el hardware subyacente, los precios disminuyen y los ahorros se transmiten al consumidor.

Rápida escalabilidad Consideremos como ejemplo un backend de un sitio web que ve un aumento de tráfico considerable inmediatamente después de anuncios promocionales de televisión.

Con las plataformas tradicionales que normalmente ofrecen una nueva instancia tiempo de servicio medido en horas o minutos, se debe acordar con el proveedor cada vez que se pretenda realizar un anuncio a fin de cumplir por adelantado con estas ráfagas de tráfico.

Con una plataforma implementada utilizando un modelo sin servidor con la capacidad de escalar instantáneamente de decenas a muchos miles de peticiones, el escalamiento preventivo se vuelve innecesario, y además permite que la plataforma pueda satisfacer el tráfico que, en algunos casos, puede ser impredecible, como por ejemplo, con noticias de última hora.

Seguridad En entornos autogestionados, el error del operador es la razón subyacente para comprometer una plataforma. Un cortafuegos mal administrado, vulnerabilidades de día cero o no reveladas que pueden ser explotadas por delincuentes o agentes de espionaje.
Los proveedores de informática sin servidores son responsables por la seguridad de la ejecución subyacente, incluyendo el entorno operativo debido a que dedican recursos significativos a esa tarea. Por este motivo, los desarrolladores pueden beneficiarse de la seguridad y son libres de concentrarse en la seguridad a nivel de aplicación.

Esto no quiere decir que la seguridad ya no sea una preocupación, por ejemplo la vulnerabilidad de la aplicación o el agujero de seguridad de cross-site-scripting todavía pueden ser un problema aunque en menor medida ya que la superficie de ataque disminuye.

3.2. Desventajas

La naturaleza stateless [2] Las funciones programadas en ambientes sin servidor son inherentemente *stateless*. Las arquitecturas sin servidor típicas almacenanan información en bases de datos SQL / NoSQL o colas de mensajes con datos a procesar leídos o escritos. La concurrencia puede introducir una complejidad significativa en una plataforma y se debe tener cuidado de manejar esto con controles apropiados, como la lógica transaccional, el bloqueo y el uso de plataformas que den apoyo a las operaciones atómicas.

Tiempo de ejecución limitado Normalmente las funciones tienen un tiempo límite de ejecución por ejemplo: 5 minutos. Algunos tipos de aplicaciones desarrolladas para servidores tradicionales podrían no ser aptas para los ambientes

[2] "En informática, un protocolo sin estado *en inglés stateless protocol* es un protocolo de comunicaciones que trata cada petición como una transacción independiente que no tiene relación con cualquier solicitud anterior, de modo que la comunicación se compone de pares independientes de solicitud y respuesta"

serverless con excepción de aquellas que sean adaptadas específicamente para este tipo de arquitectura (batch[3]).

Límitaciones en los ambientes de ejecución Los proveedores de informática sin servidor, estan limitados a un conjunto de lenguajes de programación y ambientes de ejecución motivo por el cual el programador debe adecuarse a aquello que se encuentre diponible.

Tamaño de los ejecutables Así mismo como en el punto anterior, los proveedores pueden imponer límites en el tamaño del código fuente entregado (incluyendo dependencias y librerias adicionales).

Tiempo de arranque en frio Normalmente algunas funciones programadas en arquitecturas sin servidor experimentan un retraso adicional (de hasta un número de segundos de un dígito) en las primera ejecución periódica. Una técnica para evitar que esta condición afecte la experiencia del usuario es utilizar alguna función que ejecute periodicamente la funcion a modo de "ping".

Otras desventajas podrían ser los propios límites de seguridad impuestos por los proveedores tales como: límites en la ejecución concurrente de funciones debido a que todos los usuarios comparten un mismo espacio de computación etc.

4. Implementación

En comparación con un desarrollo convencional cambia un poco el método de trabajo, ya que el código es dividido en funciones y cada función responderá a un evento.

Por ejemplo, una función puede contener el código que inserte en nuestra base de datos y el evento que la ponga en marcha será la llamada a una url con una petición de tipo POST[4]. De esta forma, nuestro código se encuentra más aislado, logrando así que el tiempo de ejecución sea menor y reduciendo directamente los costes.[6]

[3] Se conoce como sistema por lotes (en inglés batch processing), o modo batch, a la ejecución de un programa sin el control o supervisión directa del usuario (que se denomina procesamiento interactivo). Este tipo de programas se caracterizan porque su ejecución no precisa ningún tipo de interacción con el usuario.

Generalmente, este tipo de ejecución se utiliza en tareas repetitivas sobre grandes conjuntos de información, ya que sería tedioso y propenso a errores realizarlo manualmente. Un ejemplo sería el renderizado de los fotogramas de una película.

[4] Es un método de peticiones HTTP. Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.

4.1. Proveedores de informática sin servidor

Amazon Web Services AWS Lambda es un servicio de informática (un sub producto de Amazon Web Services) sin servidores que ejecuta el código como respuesta a eventos y administra automáticamente los recursos informáticos subyacentes. Puede usar AWS Lambda para ampliar otros productos de AWS con lógica personalizada o puede crear sus propios servicios back-end para que administren la seguridad, el rendimiento y el escalado de AWS. AWS Lambda puede ejecutar código automáticamente en respuesta a varios eventos, como solicitudes HTTP a través de Amazon API Gateway, modificaciones a objetos en buckets de Amazon S3, actualizaciones de tablas en Amazon DynamoDB y transiciones de estado en AWS Step Functions.

Lambda ejecuta el código en una infraestructura informática de alta disponibilidad y ejecuta la administración integral de los recursos informáticos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, la implementación de parches de seguridad y código, así como la monitorización y los registros. Lo único que tiene que hacer es proporcionar el código.[7]

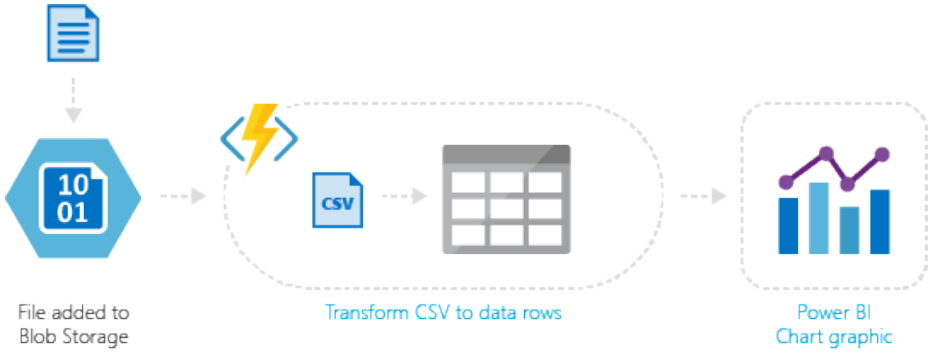
Azure functions Es la alternativa propuesta por Azure, permite crear funciones en el lenguaje que uno prefiera, como JavaScript, C sharp, y F sharp, y con opciones de scripting como Python, PHP, Bash, Batch y PowerShell. Permite escribir código en una interfaz basada en web, o bien cargar código precompilado. Permite interactividad con otros productos de Azure. [8]

Ejemplos de lo que se puede hacer con Azure Functions:

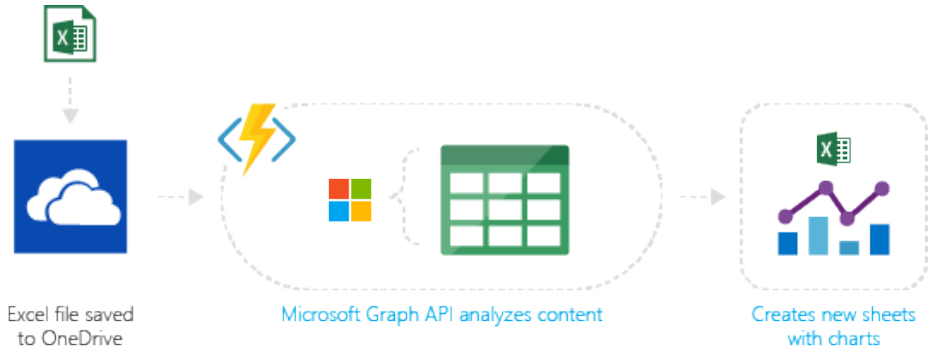
- **Procesamiento basado en temporizador** Azure Functions admite un evento basado en un temporizador mediante la sintaxis de trabajos CRON. Por ejemplo, escriba código que se ejecute cada 15 minutos y limpie una tabla de base de datos según una lógica de negocios personalizada.

- **Procesamiento de eventos de servicio de Azure** Azure Functions permite desencadenar un evento en función de una actividad en un servicio de Azure. Por ejemplo, ejecute código sin servidor que lea archivos de registro de prueba recién detectados en un contenedor de Azure Blob Storage y

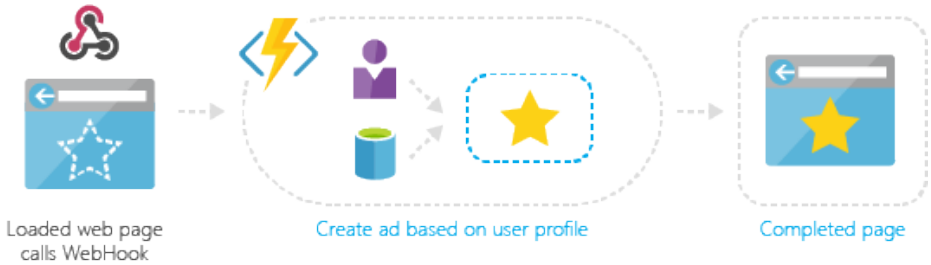
transfórmelo en una fila de una tabla de Azure SQL Database.



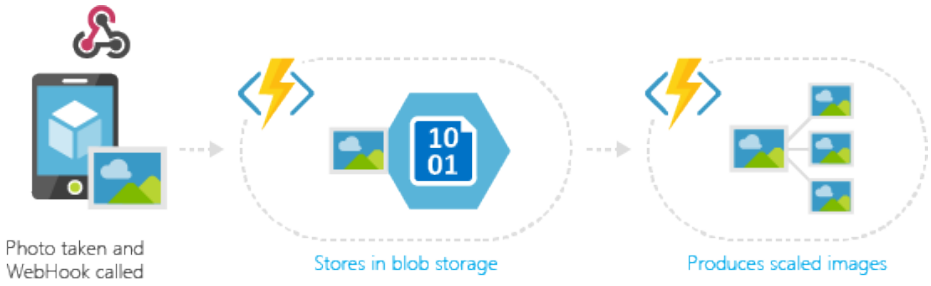
- **Procesamiento de eventos SaaS** Azure Functions admite desencadenadores en función de la actividad en un servicio SaaS. Por ejemplo, guarde un archivo en OneDrive, que desencadena una función que usa la API Graph de Microsoft para modificar la hoja de cálculo y crea gráficos adicionales y datos calculados.



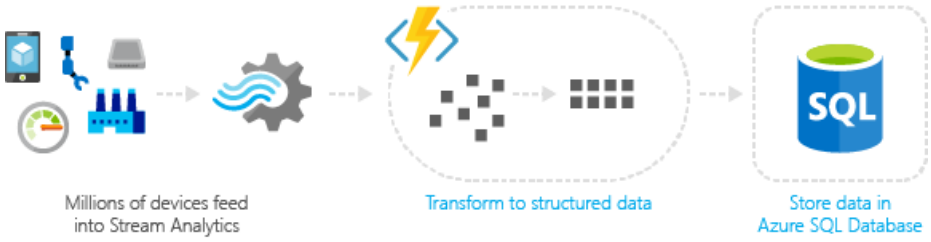
- **Arquitecturas de aplicaciones web sin servidor** Azure Functions puede mejorar una aplicación de una sola página. La aplicación llama a las funciones con la URL de WebHook, guarda los datos de usuario y decide qué datos se muestran. O bien, realice personalizaciones sencillas, como cambiar el destino de anuncios mediante una llamada a una función y pasarle información del perfil del usuario.



- **Back-end móviles sin servidor** Un back-end móvil puede ser un conjunto de API HTTP a las que se llama desde un cliente móvil con la URL de WebHook. Por ejemplo, una aplicación móvil puede capturar una imagen y luego llamar a una función de Azure para obtener un token de acceso para cargar en Blob Storage. La carga del blob desencadena una segunda función de Azure, con la que se ajusta el tamaño de la imagen al dispositivo móvil.



- **Procesamiento de datos en tiempo real** Por ejemplo, los dispositivos Internet de las cosas (IoT) envían mensajes a Azure Stream Analytics, que luego llama a una función de Azure para transformar el mensaje. Esta función procesa los datos y crea otro registro en una instancia de Azure SQL Database.



- **Mensajería de bots en tiempo real** Utilice Azure Functions para personalizar el comportamiento de un bot con la utilización de un webhook. Por ejemplo, cree una función de Azure que procese un mensaje con Cortana Analytics y llamar a esta función con Microsoft Bot Framework.

Cloud Functions Google pone a disposición su herramienta *Cloud Functions* de informática sin servidor y estos son algunos de los casos prácticos propuestos[9]:

- **Backend para móviles** Accede a Firebase, la plataforma para móviles de Google pensada para desarrolladores de aplicaciones, y extiende tu backend para móviles con Cloud Functions. Escucha y responde a eventos desde Firebase Analytics, Realtime Database, Authentication y Storage.
- **APIs y microservicios** Crea aplicaciones a partir de bits de lógica ligeros de bajo acoplamiento que se compilan y escalan de forma rápida y automática. Asimismo, puedes configurar tus funciones para que se basen en eventos o se invoquen directamente mediante HTTP(S).
- **Procesamiento de datos/ETL** Escucha y responde a eventos de Cloud Storage (por ejemplo, cuando se crea, modifica o retira un archivo). Procesa imágenes, realiza transcodificaciones de vídeo, valida o transforma datos e invoca cualquier tipo de servicio en Internet desde tu función de Cloud Functions.
- **Webhooks** Mediante un sencillo activador de HTTP, puedes responder a eventos que se generen desde sistemas de terceros, como GitHub, Slack, Stripe o cualquier sitio capaz de enviar solicitudes HTTP(S).
- **Internet de las Cosas** Imagina contar con un sistema en el que decenas o cientos de miles de dispositivos emitieran datos a Cloud Pub/Sub mientras lanzan, de forma automática, funciones de Cloud Functions para procesar, transformar y almacenar datos. Con Cloud Functions puedes hacerlo sin necesidad de servidores.

4.2. Precios y condiciones

El precio de la informática sin servidor es variable dependiendo del número de solicitudes de funciones, del tiempo de ejecución y de la condiciones de ejcución.

- **Solicitud** se cuenta una solicitud cada vez que la función es invocada y comienza a ejecutarse como respuesta a un evento o a una llamada de invocación
- **Computación** es el segundo costo involucrado en el precio de la ejecución de la función. A fin de calcular el costo de computación se toman en cuenta el tiempo de ejecución (generalmente en incremento de 100 milisegundos) y la cantidad de memoria utilizada.

A continuación elaboramos una tabla con los precios de los tres proveedores mas importantes de informática sin servidor. Asumimos un millon de solicitudes, con un tiempo de duración promedio de 1000 ms y consumiendo 512MB de memoria.

Proveedor	Costo de la solicitud	Costo de la computación	Total
AWS Lambda	0.20	8.34	8.54
Azure Functions	0.20	8.00	8.20
Google Cloud Functions	0.40	9.25	9.65

5. Conclusión

La informática sin servidores constituye un cambio en el paradigma tradicional al cual estamos acostumbrados. Si bien flexibiliza algunas tareas y nos proporciona recursos a precios relativamente bajos (en comparación con la informática tradicional de servidores dedicados), desarrollar aplicaciones implica nuevos desafios en el propio diseño del software.

Referencias

1. Ron Miller, Aws Lambda makes serverless applications a Reality, https://techcrunch.com/2015/11/24/aws-lamda-makes-serverless-applications-a-reality/, 24 de noviembre de 2015
2. Peter Sbarski Serverless Architectures on AWS Manning Publications Abril de 2017
3. Shane Rainville, Architect Web Infrastructures , http://www.serverlab.ca/tutorials/linux/web-servers-linux/architect-web-infrastructures-on-ubuntu-14-part-1/,
4. Tomasz Stachlewski - AWS Solutions Architect, Serverless Computing, https://www.sita.aero/globalassets/docs/events/2016-sita-innovation-day/amazon-tomasz-stachlewski.pdf,
5. AWS Lambda and the Evolution of the Cloud, Josh Stella, https://blog.fugue.co/2016-01-31-aws-lambda-and-the-evolution-of-the-cloud.html,
6. ¿ QUÉ ES ARQUITECTURA SIN SERVIDORES O SERVERLESS?, Antonio Fernández, https://antoniofernandez.com/serverless/,
7. Detalles del producto, AWS Lambda, https://aws.amazon.com/es/lambda/details/,
8. Functions, Microsoft Azure, https://azure.microsoft.com/es-es/services/functions/,
9. Cloud Functions, Google Cloud Platform, https://cloud.google.com/functions/?hl=es,