# SERVERLESS TRADE-OFFS

ALEJANDRO ÁLVAREZ SOLÍS UO232627

MARCOS CARADUJE MARTINEZ UO270285

GERMÁN DÍAZ GONZÁLEZ UO258472

## Introducción. ¿En qué consiste el serverless y el funcionales?

El serverless consiste principalmente, en delegar una parte del despliegue y el funcionamiento de nuestro producto en un tercero, que nos ofrece un servicio (porque sí, serverless y functionless son servicios), en este caso, un servidor.

Esto se traduce en que el desarrollador no tiene que gastar tiempo, esfuerzo y recursos en crear y mantener un servidor, puesto que lo que hacemos es pagar a una empresa que nos proporcione ese servicio, y que sea esa empresa la que se preocupa de tener a disposición el servidor, para que nosotros podamos gastar nuestros recursos en preocuparnos de otras cosas de más relevancia y que vayan a ser más importantes como el lenguaje en el que vamos a desarrollar nuestra aplicación, el modelo de datos, la arquitectura sobre la que vamos a trabajar, etc.

Para llevarlo a un ejemplo práctico, a ningún arquitecto de software se le ocurriría gastar tiempo de desarrollo y por tanto, dinero, en desarrollas un sistema gestor de bases de datos independientes, en lugar de eso, la decisión más lógica sería que es lo que a ti te viene más a cuenta, comprar una licencia de Oracle (producto) sobre el que apoyarte para crear tu base de datos, o contratar el servicio de MongoDB (servicio) para que 'hostee' tu base de datos. De esta manera, estamos centrando nuestras decisiones de arquitectura en algo más concreto y práctico que nos ayudara a ser más eficientes durante el desarrollo de la aplicación.

Debido a esto, y a que este modelo de arquitectura está en auge, la principales empresas del sector de la "nube", Google, Amazon y Microsoft, están ofreciendo servicios para las arquitecturas serverless a precios cada vez más competitivo, por el crecimiento de este modelo.



## ¿Qué es serverless?

En la ingeniería todo está en constante evolución, y dicha evolución, va poco a poco transformando los productos en servicios. Hace unos años comprabas un producto, lo descargabas, lo instalabas y ya era tuyo para siempre, actualmente cada vez más empresas han derivado a ofrecer servicios por los que los clientes pagamos durante el tiempo que los necesitamos.

Serverless es precisamente esto, pagar por unos recursos computacionales sin tener que preocuparnos de la infraestructura. Pagamos por el tiempo de ejecución de nuestro código.

Faas o Function as a Service nos permite crear aplicaciones con una alta escalabilidad, concurrencia y alta disponibilidad. Pero también nos acarrea una serie de problemas que debemos afrontar.

## Ventajas de serverless

- Al utilizar serverless los ingenieros de software pueden tomar decisiones de arquitectura a bajo coste, es decir, si se toma una decisión de utilizar un servicio y más adelante nos damos cuenta de que no ese servicio no nos sirve, podemos fácilmente sustituirlo por otro. Como esa decisión es fácil de deshacer, tiene un bajo coste.
- Como se mencionó anteriormente una de las ventajas de serverless es que pagas exactamente por lo que usas. Tiempo de ejecución, número de usuarios, etc.
- Escalabilidad y elasticidad de la arquitectura. La cantidad de recursos que pagas y utilizas va totalmente ligada a la demanda actual de tu sistema.
- Productividad. Este tipo de arquitectura permite a los desarrolladores centrarse en lo verdaderamente importante, en la parte que resuelve los problemas del cliente final, en los aspectos por los que el cliente los va a contratar y que los hacen distinguirse del resto de empresas. Los aspectos que no se ven de la arquitectura son los costes para desarrollar el servicio y son los que se contratan.
- Facilidad de testing. Cada función es más fácil de probar ya que solo realiza una única acción.
- Un alto nivel de abstracción que nos permite centrarnos en el dominio de negocio.
- La utilización de funciones en el desarrollo facilita la implementación de un patrón de arquitectura de microservicios. Se utilizan componentes únicos e implementables por separado. Aunque esto es una ventaja también conlleva un aumento en la complejidad del mantenimiento.
- El aislamiento y la seguridad de las funciones es una ventaja ya que cada función accede solamente a su entorno.
- Variedad en los equipos de desarrollo. Un mismo proyecto puede tener varios lenguajes de programación.
- Cada petición está totalmente aislada de las demás. Esto añade flexibilidad ya que no tenemos que preocuparnos del estado de las peticiones.
- Alta disponibilidad y tolerancia a fallos. Dependemos de servicios con una disponibilidad entorno al 99.95%.

## Desventajas de serverless



- Complejidad añadida, al tener tantos servicios a elegir, debemos conocer cuáles son, que hacen, en que nos pueden ser útiles y como implementarlos. Lo cual nos delega en nuevos problemas y nuevas responsabilidades.
- Riesgo de cometer un error y que repercuta en elevados gastos económicos. La optimización del código es más importante que nunca.
- Mayor complejidad en los test de integración, ya que no estas en control de todos los servicios.
- Dificultad para debuggear. Ya que replicar el funcionamiento del servidor en tu máquina local puede ser muy dificil.
- La observación, monitorización, logs, etc. conllevan un mayor esfuerzo.
- El problema del arranque frio (Cold Start) y la alta latencia. Cuando pasa un periodo de tiempo en el que una función Lambda no es ejecutada, esta se pone en un estado de espera para ahorrar recursos, lo que produce que, en su siguiente llamada, tarde un tiempo extra en ejecutarse.

- Como pagamos por tiempo de ejecución, tamaño del programa, etc. Tendemos a comprimir todo lo que podamos el código, por tanto, podemos estar creando código imposible de entender incluso por nosotros mismos. Así mismo el propio lenguaje en el que se desarrolla también juega un papel importante.

## ¿Qué es el concepto functionless dentro de la arquitectura serverless?

Una integración functionless es una integración directa entre dos componentes de Amazon Web Services (AWS) que configuramos en vez de utilizar código diseñado por los desarrolladores en funciones Lambda.

Uno de los ejemplos más claros que tenemos es la API Gateway de AWS (Amazon Web Services), donde las peticiones de la API se mapean directamente en los servicios (como por ejemplo DynamoDB).

## Trade-offs

Se podría pensar en functionless como tratar de llevar serverless al extremo de la eficiencia y el rendimiento. Como en todo tipo de enfoques, tenemos unos beneficios y unas desventajas llamados trade-offs.

Para explicarlos un poco en detalle, se separarán los trade-offs de los que se hablará en estas dos categorías.

**Beneficios:**

- **Latencia más baja**: Cuando usamos una función lambda, podemos encontrarnos con algo de latencia, como se explicó anteriormente con las "cold start". Este problema desaparece en functionless ya que la latencia es casi nula al no haber ninguna ejecución intermedia.
- **Mejor mantenibilidad**: Los problemas de mantenibilidad desaparecen casi totalmente al no haber ningún código ni dependencia que tenga que mantener el equipo de desarrollo. Todo está actualizado y mantenido por AWS.
- **Gratis**: Con las funciones lambda tenemos que pagar por ejecución. La mayoría de los functionless no tienen coste, solo se paga por los servicios que quieres conectar, como por ejemplo la base de datos.
- **Mayor escalabilidad**: Aunque las funciones lambda son muy escalables, podemos encontrarnos con límites de ejecuciones concurrentes. Functionless no tiene este límite, por lo que si tenemos un entorno en el que las ejecuciones sean excesivas, puede ser una mejor opción en cuanto a escalabilidad.
- **Mejor seguridad**: Al no tener código hecho por nosotros, hay menos riesgo potencial de tener algún fallo en la seguridad del sistema.
- **Menos puntos de ruptura**: Como en cualquier mecanismo que haya múltiples piezas trabajando de forma simultánea y sincronizada, cuantas menos piezas tenga, menos puntos de rotura hay.

Como se puede observar, la mayoría de los beneficios de usar functionless vienen de la casi total ausencia de código y de lo muy mantenible que es.

**Inconvenientes:**

- **Curva de aprendizaje más pronunciada**: Aunque no tenga funciones lambda si que necesitan pequeñas partes de código para mapear todo el sistema. Estos lenguajes de mapeo pueden ser un poco confusos si no se está acostumbrado a ellos.
- **Malas herramientas**: Las herramientas que hay para estos lenguajes de mapeo son por lo general bastante pobres, pudiendo hacer que el desarrollo sea más lento.
- **Sin componibilidad**: Estos lenguajes no soportan imports ni includes, por lo que reusar partes es prácticamente imposible.
- **Riesgo de integridad de datos**: Las conexiones directas a las bases de datos pueden llegar a dar problemas. Los diseños de tabla única requieren muchas claves y atributos de indexado para poder ser usadas. Esto puede hacer que surjan problemas si por ejemplo se usa un formato incorrecto al introducir un atributo de indexado.
- **Difíciles de testar**: Estos sistemas son casi imposibles de probar ya que, al tener la integración hecha directamente, no se pueden probar las partes aisladas para comprobar su funcionamiento por separado.

Al igual que en el caso de los beneficios, casi todos los inconvenientes se enfocan también en una sola dirección, lo incomodo y poco intuitivo que es el uso de los lenguajes requeridos para este tipo de integración.

## ¿Es mejor usar hacer nuestro sistema functionless?

Según para que se quiera usar podemos tomar la decisión de seguir este enfoque. Tenemos dos planteamientos principales que son los que nos pueden ayudar a tomar esa decisión:

- Si queremos un sistema que solo transporte datos sin hacer ningún tipo de transformación de la información, functionless es el mejor enfoque.
- Si queremos ejecutar lógica importante donde los bugs pueden ser difícilmente detectables, serverless es mejor opción.



Por último, habría que plantearse como va a evolucionar el sistema en el futuro. Debemos estudiar bien cuál sería el impacto para nuestro sistema si queremos pasar en el futuro de un enfoque functionless a uno serverless y el coste y complejidad que esto supondría.

En este enlace se hace un pequeño estudio donde se compara el rendimiento de ambos enfoques: *"Estudio del rendimiento de ambos enfoques"*.

## Bibliografía y sitios consultados

*https://pablo-iorio.medium.com/serverless-architectures-i-iii-design-and-technical-trade-offs-8ca5d637f98e*

*https://medium.com/ssense-tech/the-trade-offs-with-serverless-functions-71ea860d446d*

*https://serverlessfirst.com/functionless-integration-trade-offs/*

*https://www.stackery.io/blog/future-of-serverless/*

*https://medium.com/lego-engineering/dont-wait-for-functionless-write-less-functions-instead-8f2c331cd651*

*https://dev.to/kumo/functionless-can-we-do-better-without-lambda-4koa*

*https://martinfowler.com/articles/serverless.html#drawbacks*

*https://martinfowler.com/articles/serverless.html*