

**APLICACIONES DISTRIBUIDAS EN  
INTERNET**

---

# **APLICACIONES SERVERLESS EN AMAZON AWS**

---

Noelia Quiles Guilló  
Raquel Ortega Pérez  
Janira Elías Jiménez

# ÍNDICE

---

<b>Introducción</b>	<b>1</b>
AWS Lambda	1
APIGateway	2
SNS	2
SQS	2
EventBridge	3
Step Functions	3
<b>Aspectos Fundamentales</b>	<b>4</b>
1. Diseño impulsado por eventos	4
¿Qué es?	4
API síncrona	5
2. Disparadores y fuentes de eventos	6
3. Arquitecturas de referencia	7
Microservicios RESTful	7
Procesamiento de imágenes	8
Procesamiento de transmisiones	9
Aplicación web	10
4. Implementación	11
Todos de una vez	11
Azul/verde	11
Canary/lineal	11
<b>Tutorial conseguir “Hola Mundo”</b>	<b>12</b>
<b>Bibliografía</b>	<b>18</b>

# Introducción

A la hora de desplegar una aplicación, una de las operaciones más costosas es administrar la infraestructura como el aprovisionamiento de servidores, los parches, etc. Estos últimos días ha surgido una nueva moda en la industria del software y la programación llamada **Serverless Applications** o Aplicaciones sin servidor.

Como definición de las aplicaciones sin servidor podemos decir que son aplicaciones que, a pesar de que siguen existiendo servidores que ejecutan el código, las tareas relacionadas con el aprovisionamiento y la **administración de la infraestructura** son **invisibles** para el programador.

Esto permite que el equipo pueda centrarse más en la innovación y en la lógica de negocio, además, aumenta la **productividad** y permite comercializar los productos más rápido.

Una aplicación Serverless debe seguir estos 4 principios para que se considere de este tipo:

- **Pago por uso:** Estas aplicaciones no tienen un pago fijo, se paga según la cantidad de operaciones que se realice con ellas lo que permite abaratar los gastos si no se hace un gran uso.
- **Sin gestión de infraestructura:** El desarrollador no se tendrá que hacer cargo del mantenimiento. No hay que manejar sistemas operativos, no hay preocupación por las redes.
- **Escalable:** Las aplicaciones de este tipo permiten un margen de movilidad mucho mayor que las máquinas virtuales a la hora de escalar, de hecho, se podría considerar ilimitado.
- **Alta disponibilidad:** Normalmente tienes que elegir las zonas de disponibilidad, en cambio, en estas aplicaciones no tenemos control en qué centro de datos se está guardando la información porque permite acceder a ellos desde cualquier lugar prácticamente. Las Serverless permiten no tener que enfocarte en estos aspectos y poder enfocarte en las partes del proyecto que aporten valor a la aplicación.

A continuación, explicaremos los distintos servicios a partir de los cuales se crean las aplicaciones sin servidor. Estas pertenecen a 4 categorías diferentes que son informática, a la cual pertenece AWS Lambda, Proxy de la API a la cual pertenece API Gateway, Mensajería e Integración a la cual pertenecen SNS, SQS y Eventbridge y por último existe la categoría Organización a la cual pertenecen las Step Functions.

## AWS Lambda

Lambda es el servicio estrella Serverless que permite ejecutar código en la nube de forma escalable. Se encarga de ejecutar las funciones o las aplicaciones que el desarrollador le indique. Cabe destacar que Lambda por sí mismo no tiene un almacenamiento propiamente dicho, para suplir esto tiene un pequeño espacio temporal y sin estado.

A diferencia del mundo tradicional, en el cual un servidor es el encargado de ejecutar las aplicaciones, en las aplicaciones Serverless, el servicio Lambda es el que ejecuta las funciones siguiendo estos pasos:

1. Crea un servidor virtual que despliega solamente las funciones necesarias para ejecutar la acción que el cliente solicite, ahorrando así recursos.
2. Cuando se hace una petición, la función solicitada es ejecutada.
3. Por último, se inhabilita ese pequeño servidor hasta que se realice otra petición.

Siguiendo el principio de Pago por uso que hemos indicado anteriormente, en Lambda se paga cuando se ejecuta una de esas funciones que gestiona. Una buena práctica a tener en cuenta cuando se está trabajando con este servicio es hacer paquetes de archivos pequeños con poca dependencia entre ellos que permita un aprovechamiento de los recursos. Por último, otra ventaja que tiene es que se puede usar con la mayoría de lenguajes modernos, como puede ser ASP.NET.

## APIGateway

Este servicio permite aplicar seguridad a las funciones o aplicaciones ya que se puede ver quién está entrando al servicio y, además, se puede administrar distintos tipos de clientes para que cada uno pueda acceder solamente a los servicios que el desarrollador le permita.

Un ejemplo de uso podría ser el siguiente: Si ya se tiene un servicio API creado para una página web y de repente se quiere trasladar a móvil puedes utilizar API Gateway para configurar el sistema para que solamente pueda acceder a los servicios que la aplicación móvil va a usar. Este servicio, además, permite sacar métricas de uso de los servicios..

## SNS

Es el primer gran servicio que permitió a los desarrolladores enviar correos electrónicos, SMSs, etc. directamente sin hacer uso de un sistema aparte. Este permite enviar un mismo mensaje a diferentes puntos, siendo 12.5 millones el número máximo de puntos a los que puede enviar información, lo que quiere decir que permite que hasta aplicaciones con gran número de usuarios puedan hacer uso de este servicio. Como desventaja de este servicio, la monitorización es bastante escasa así que para este tipo de información habría que hacer uso de otras herramientas.

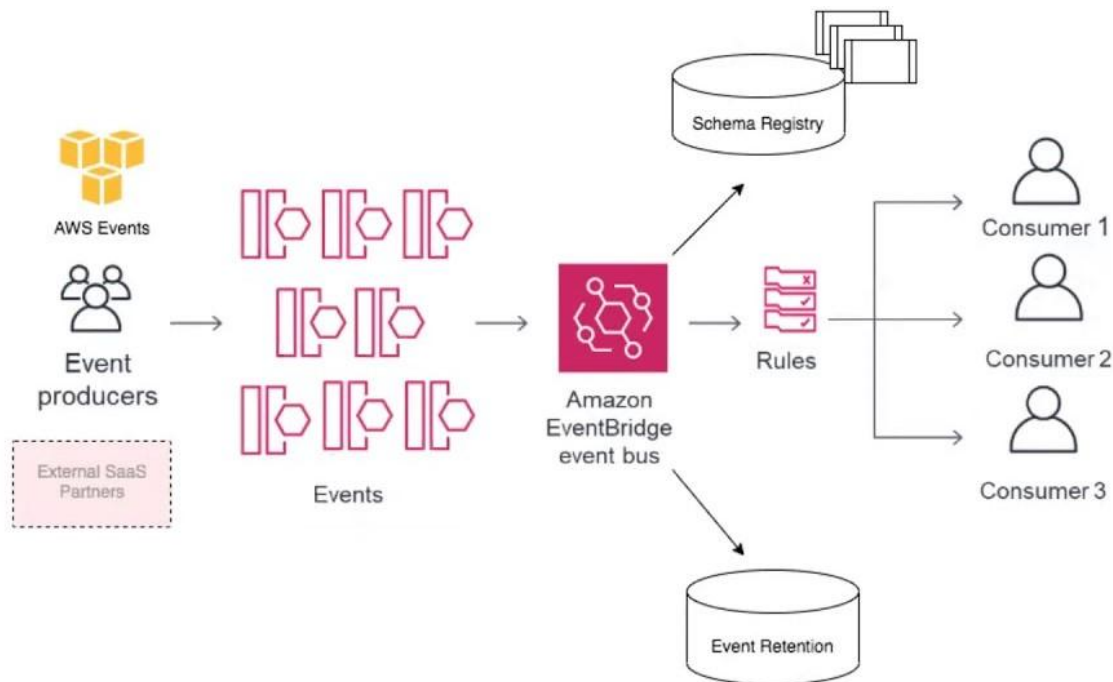
## SQS

Este servicio se encarga de realizar una cola de mensajería en la cual el emisor y el receptor pueden emitir y recibir mensajes. La ventajas que proporciona este servicio es que, además de que el desarrollador se puede olvidar totalmente de la administración de la infraestructura por ser Serverless, también permite enviar hasta 10 mensajes al mismo tiempo y los puede mantener almacenados hasta 14 días. Asimismo, se puede configurar la prioridad de los mensajes, de esta forma el servicio sabe en qué orden se tienen que mandar o si por defecto quiere utilizar un sistema FIFO.



## EventBridge

Es el último servicio de mensajería Serverless que ha sido lanzado. Este servicio permite que varios microservicios se registren en un bus de eventos. Cuando alguno de ellos emite uno, se analiza las reglas que hay en ese bus, las cuales indican a qué receptores debe ir dirigido ese evento. Este tipo de servicios, por sus características es muy escalable.



## Step Functions

Este servicio es recomendado cuando la aplicación en la que se utiliza tiene un flujo de trabajo, es decir, cuando una operación tiene unos pasos a seguir. Un ejemplo de esto es en el registro de un usuario en el cual se ingresa el DNI, el correo electrónico, y la tarjeta bancaria. Durante el proceso de registro se debe verificar que los tres datos introducidos son válidos. Se podría hacer con distintas funciones independientes o se puede hacer con colas pero esto no permite saber en qué estado está el proceso, en cambio, las Steps Function pueden indicar si ha ocurrido un error en alguno de los pasos y mostrar en qué estado se encuentra. Este servicio permite gestionar cualquier tipo de procesos tanto de funciones y contenedores como humanos.

Se recomienda usar este servicio dentro de un mismo microservicio y no conectar varios para que no existan dependencias. Además, las Step Functions permiten hacer ejecuciones en paralelo, en el ejemplo que hemos puesto, este servicio permitiría hacer varias validaciones al mismo tiempo lo que permitiría ahorrar tiempo. También se pueden incluir condicionales.

# Aspectos Fundamentales

## 1. Diseño impulsado por eventos

### ¿Qué es?

Antes de comenzar a explicar los diseños impulsados por eventos, deberemos aclarar qué es un evento y por qué se utilizan. Un evento es la representación de un **cambio** o una **actualización** en el sistema. Por ejemplo, un evento puede representar cosas muy distintas:

- Añadir un artículo al carrito de la compra.
- Un sensor que detecta movimiento y avisa al servidor del acontecimiento.

Ambos ejemplos son distintos pero con algo en común: se ha producido un cambio. Pues bien, el evento es el encargado de **transportar la información** necesaria para que el cambio sea notificado correctamente y se actúe en consecuencia (si es necesario).

En el caso del carrito de la compra, el evento puede contener la información del artículo que debe añadirse, con su precio y cantidad. Ahora que sabemos esto, ¿Cómo funciona un **diseño** que se basa en eventos?.

La arquitectura basada en eventos es una arquitectura de software que sigue unos patrones y diseños concretos para poder diseñar aplicaciones. Toda la arquitectura se centra en producir, comunicar, detectar y procesar eventos. También se centra en la permanencia de dichos eventos en el sistema. En resumen, la arquitectura entera se construye para **responder** a dichos eventos de la manera más rápida e **instantánea** posible.

Muchas de las aplicaciones modernas están basadas en eventos, puesto que la necesidad de las empresas de utilizar los datos de los clientes de forma inmediata queda cubierta por completo cuando se utiliza este diseño.

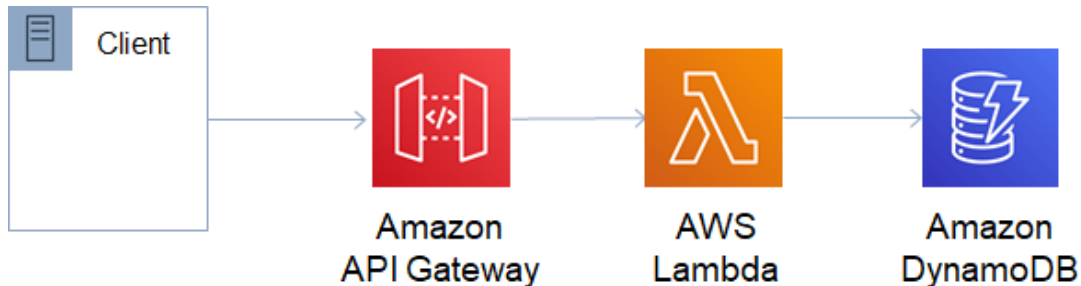
Encontraremos 3 componentes clave: productores de eventos, enrutadores de eventos u consumidores de eventos. Utilizaremos un ejemplo para ilustrar que es cada apartado:

- **Productor de evento:** Donde se **crea** o se **publica** el evento que desencadenará una serie de acciones. Por ejemplo, un cliente de una tienda digital hace un pedido. Esto desencadenará que se cree el evento “Pago” y que se envíe al enrutador de eventos.
- **Enrutador de eventos:** Es el encargado de permitir que el evento **esté disponible** en todo momento para los posibles clientes, y de filtrarlos y enviarlos a los consumidores de eventos. En este caso, permite al cliente suscribirse al evento “Pago” y lo envía a los consumidores necesarios.
- **Consumidores de eventos:** Son los encargados de **procesar** los eventos que lleguen de los distintos proveedores. Si seguimos con el ejemplo anterior, un consumidor de eventos puede ser un servicio “Inventario” que se encargue de recibir el evento, procesar su información y ver si el artículo del pedido está disponible para su compra. Es importante destacar que un consumidor de eventos puede **lanzar nuevos eventos** también, por ejemplo, “Inventario” podría lanzar un servicio “Mensaje” que se encargará de avisar al usuario de que su pedido está siendo procesado.

Esta forma de organizarse resulta muy útil, puesto que permite **desacoplar** los servicios del productor de los del consumidor, pudiendo así escalarse, actualizarse e implementarse de manera totalmente independiente.

## API síncrona

Explicar cómo funciona una API asíncrona nos servirá para entender mejor por qué nos es útil utilizar arquitecturas impulsadas por eventos.



En esta imagen podemos observar los distintos pasos cuando se produce una llamada a la API asíncrona:

- Primero, los **clientes** utilizan los microservicios para hacer llamadas **API HTTP**. Esta API es, en resumen, un servicio que recibe llamadas basadas en métodos HTTP. Son muy útiles dado que **cualquier dispositivo** capaz de utilizar HTTP (es decir, casi todos los dispositivos actuales) podrá **consumir** una API REST y hacer este tipo de llamadas.
- A continuación encontramos **Amazon API Gateway**. Aquí se alojan las solicitudes RESTful HTTP y es donde se **responderán** las peticiones de los clientes.
- Después, **AWS Lambda**, tal y como hemos explicado anteriormente, contiene la **lógica comercial** encargada de procesar las distintas llamadas de API que le lleguen.
- Finalmente, **Amazon DynamoDB** es una base de datos de **almacenamiento persistente** utilizada por AWS Lambda para almacenar toda la información necesaria de las llamadas.

El funcionamiento de todo este conjunto es el siguiente: Se producirán llamadas síncronas que llegarán a la API Gateway. Esta esperará una respuesta por parte de AWS Lambda casi inmediata (con un tiempo de espera de 30 segundos). Si Lambda no responde, se deberá controlar el error e informar al cliente de que se debe reintentar la operación.

De esta manera, los errores cuando una petición no es respondida son devueltos al cliente y no llegan a los componentes posteriores (Lambda o DynamoDB), los estamos desacoplando y a la vez consiguiendo un sistema más sólido y resistente a fallos, además de ser altamente escalable.

## 2. Disparadores y fuentes de eventos

AWS Lambda contiene diversas funciones que serán ejecutadas y disparadas por medio de eventos. Cuando se recibe un evento, Lambda ejecutará un código en respuesta y además podrá también generar y lanzar sus propios eventos si es necesario.

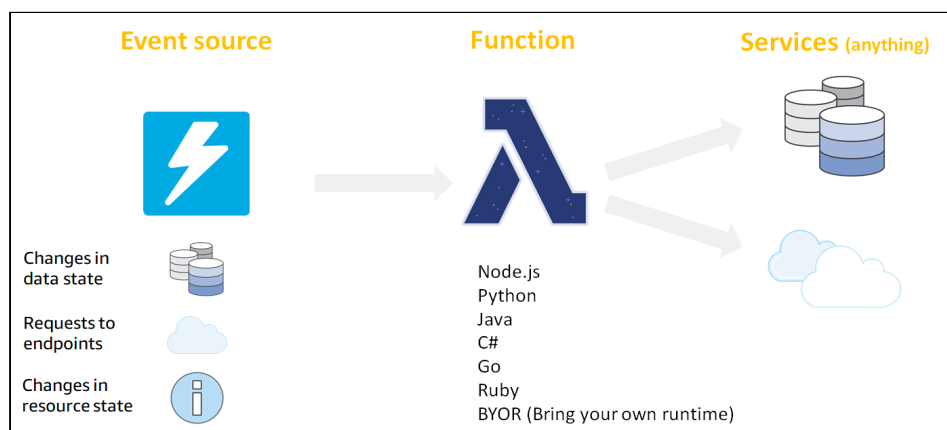
Hay muchas opciones para disparar una función de Lambda y para crear fuentes de eventos personalizadas y adaptarlas a las necesidades de cada servicio. Los principales tipos de fuentes de evento son:

- **Depósitos de datos:** Algunos ejemplos son **Amazon S3**, **Amazon DynamoDB** o **Amazon Kinesis**, que son servicios de bases de datos. En resumen, la primera permite almacenar datos en la nube, la segunda ofrece una base de datos NoSQL y la tercera destaca por permitir recopilar, procesar y analizar datos de streaming en tiempo real.

En cualquier caso, el usuario que utilice Lambda podrá **rastrear** los posibles **cambios** en la base de datos y utilizar esto como fuente de evento para lanzar funciones Lambda.

- **Puntos de enlace que emiten eventos:** Cualquier punto de enlace puede **invocar Lambda** y servir como una fuente de evento. Por ejemplo, si un usuario le da una orden a Alexa, se emitirá un evento que disparará una función de AWS Lambda.
- **Servicios de mensajería:** Algunos ejemplos de servicios de mensajería que pueden ser fuentes de eventos para invocar Lambda son **Amazon SQS** o **Amazon SNS**. La primera es un servicio de colas de mensajes, y la segunda es un servicio de mensajería administrado especialmente para la comunicación aplicación a aplicación (A2A) y aplicación a persona (A2P).
- **Acciones dentro de un repositorio:** La ejecución de una **acción** dentro de un repositorio también se considera una **fuentes de evento** y, como tal, puede disparar funciones de Lambda. Un ejemplo puede ser el repositorio **AWS CodeCommit**, que es un servicio de control de código fuente que aloja repositorios de Git privados.

Por último, este esquema representa la comunicación entre las fuentes de eventos y AWS Lambda, que a su vez se comunicará con los distintos servicios para poder proporcionar o ejecutar las acciones necesarias para las fuentes.





### 3. Arquitecturas de referencia

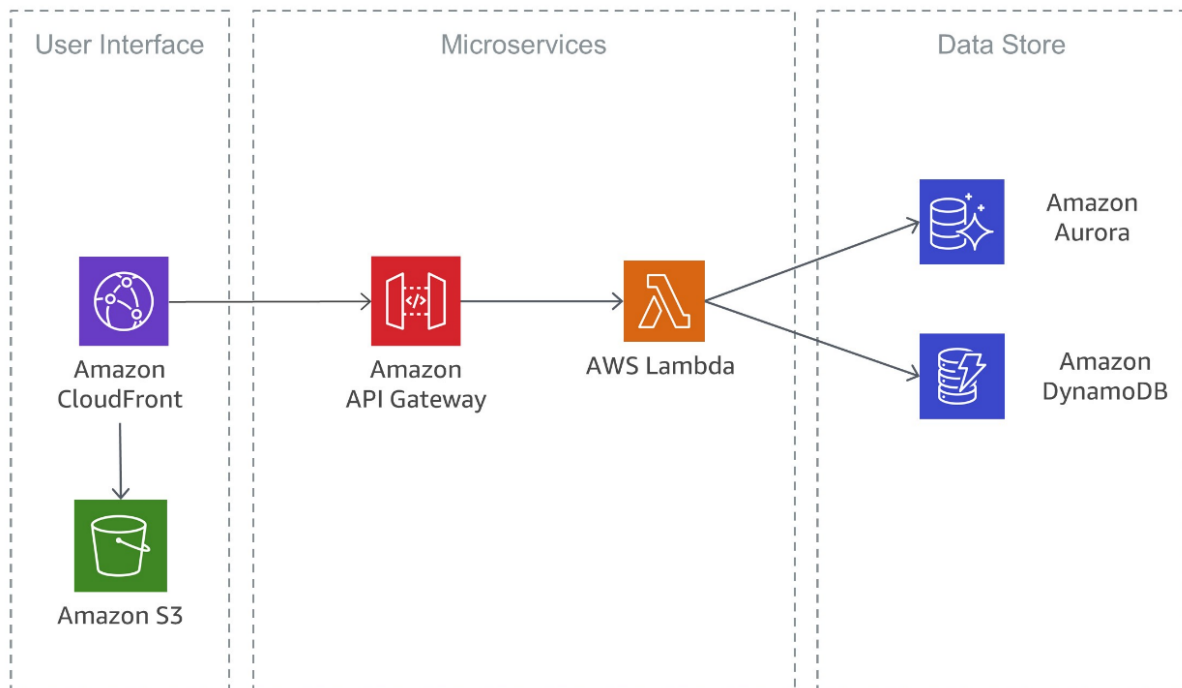
En este apartado se explicarán varias arquitecturas de referencia que cubrirán distintos y variados casos de usos y ayudarán a ejemplificar cómo puede ser útil el uso de aplicaciones serverless.

#### Microservicios RESTful

Las tecnologías serverless necesitan de una infraestructura que se mantenga siempre disponible para peticiones y que tengan una alta tolerancia a fallos, puesto que si no no serían capaces de ofrecer servicios que permitan recibir altas cargas de trabajo y gestionarlas de manera adecuada.

Para ello, Amazon ofrece muchos servicios AWS, siendo el más importante la tecnología AWS Serverless. Todos estos servicios forman un ecosistema que simplifica al usuario todo el proceso de creación, automatización de tareas, organización de servicios y supervisión de microservicios. Además, su punto más fuerte es el poder aumentar el uso de AWS conforme el negocio vaya creciendo, y mantener el coste al mínimo cuando el uso disminuya.

Un esquema que muestra mejor un posible ejemplo de diseño de una arquitectura con microservicio RESTful utilizando AWS:



Explicaremos un poco qué representa cada apartado y cómo funciona este ejemplo de arquitectura:

- **User interface:** En este apartado es donde los clientes harán **llamadas API HTTP** a los microservicios. En este caso, se utiliza conjuntamente el depósito de datos en la nube Amazon S3 y el servicio web de distribución de contenido Amazon CloudFront.

- **Microservices:** Este apartado es similar a lo explicado anteriormente. **Amazon API Gateway** aloja las solicitudes y **responde** a los clientes. En este caso, también podría encargarse de proporcionar autorización, la seguridad, la tolerancia a fallos, optimizar el rendimiento...

Por otro lado, **AWS Lambda** hace exactamente lo mismo que en el ejemplo anterior. Contiene la **lógica de negocio** y utiliza la base de datos Dynamo BD cuando es necesario para almacenar datos de manera persistente.

- **Data Store:** Aquí **DynamoDB** se encarga de **almacenar los datos** de los microservicios. En este caso, DynamoDB nos es muy útil para almacenar datos sin tener muchos problemas con esquemas estrictos al ser NoSQL. También es importante resaltar que esta puede **escalar** en función de la demanda por parte de los clientes.

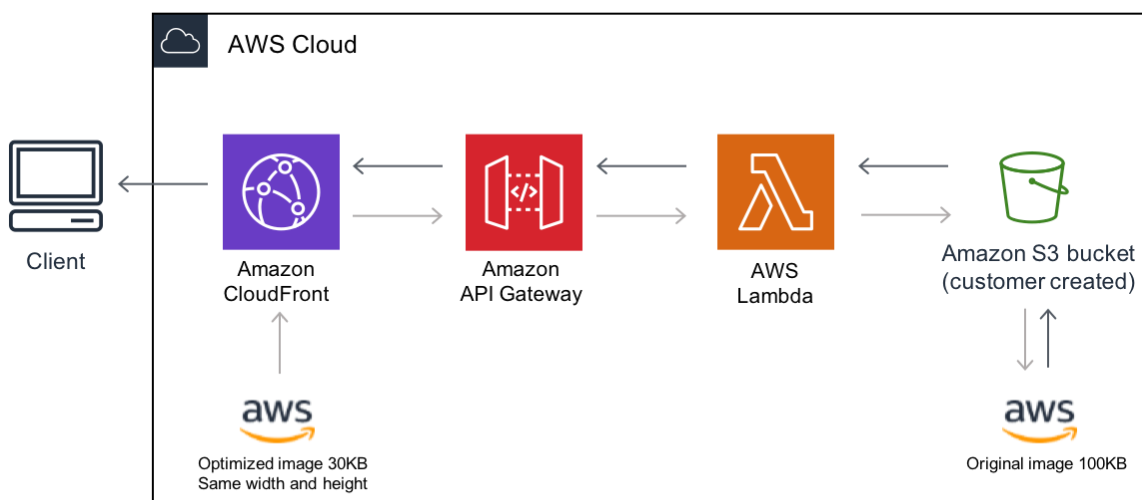
En caso de desear una base de datos relacional clásica (o querer disponer de las dos) existe la posibilidad de utilizar **Amazon Aurora**.

## Procesamiento de imágenes

Otro caso de uso donde las aplicaciones serverless pueden ser muy útiles es el caso del **procesamiento de imágenes**. Los eventos nos vendrán muy bien y la facilidad que ofrecen los servicios serverless para **ampliar y reducir** la carga de trabajo de forma dinámica nos ayudarán mucho.

Un ejemplo sencillo utilizando servicios de Amazon podría ser un conjunto de imágenes almacenadas en Amazon S3 (base de datos capaz de disparar funciones en AWS Lambda). Mediante eventos, se podrían procesar las imágenes almacenadas en la base de datos utilizando eventos y haciendo llamadas a Lambda.

Lo ilustraremos mejor con el **ejemplo** representado en el siguiente esquema:



- A la derecha del todo podemos ver cómo hay una **imagen** almacenada en **Amazon S3**.
- **AWS Lambda** se encargará de recuperar esta imagen, **procesarla** y devolverla modificada a la Amazon API Gateway cuando reciba una petición.
- A continuación se generará una URL en Amazon CloudFront desde donde se podrá **acceder** a la imagen.
- El cliente podrá acceder a ella desde una interfaz de usuario o un punto de enlace a la API del gestor de imágenes, según como prefiera el usuario que esté desarrollando la arquitectura.

En resumen, la utilización de AWS Lambda y eventos para acceder a imágenes y procesarlas de manera rápida y eficiente es una decisión muy acertada.

## Procesamiento de transmisiones

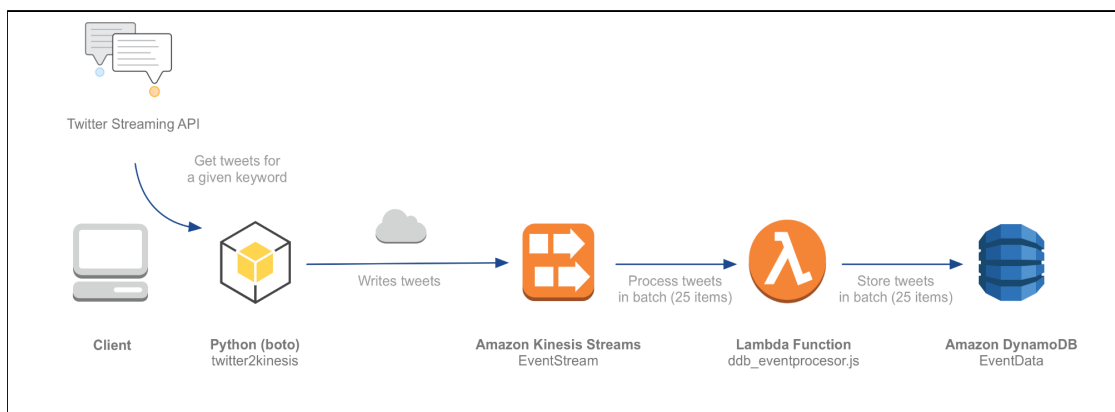
Para el procesamiento de datos de streaming contamos con **AWS Lambda** y **Amazon Kinesis**. Estas herramientas nos permiten realizar seguimiento de actividades de las aplicaciones, procesamiento de órdenes de transacciones, indexación, análisis en redes sociales y mediciones y telemetría de datos de dispositivos compatibles con IoT, etc.

Ahora hablaremos un poco más en detalle de Amazon Kinesis y como este facilita la **recopilación**, el **procesamiento** y el **análisis** de datos de streaming. Amazon Kinesis nos ofrece la posibilidad de **incorporar** datos en **tiempo real** (videos, audios, secuencias de clics de sitios web, registro de aplicaciones...) permitiéndonos procesar estos datos a medida que se reciben y **responder** de forma **instantánea**.

Las capacidades de Amazon Kinesis son las siguientes:

- **Kinesis Video Streams**: servicio de transmisión segura de videos de dispositivos conectados a AWS.
- **Kinesis Data Streams**: servicio de streaming de datos en tiempo real.
- **Kinesis Data Firehose**: servicio de registro, transformación y carga de transmisiones de datos almacenados en almacenes de datos de AWS.
- **Kinesis Data Analytics**: servicio de procesamiento de datos SQL o Apache Flink en tiempo real.

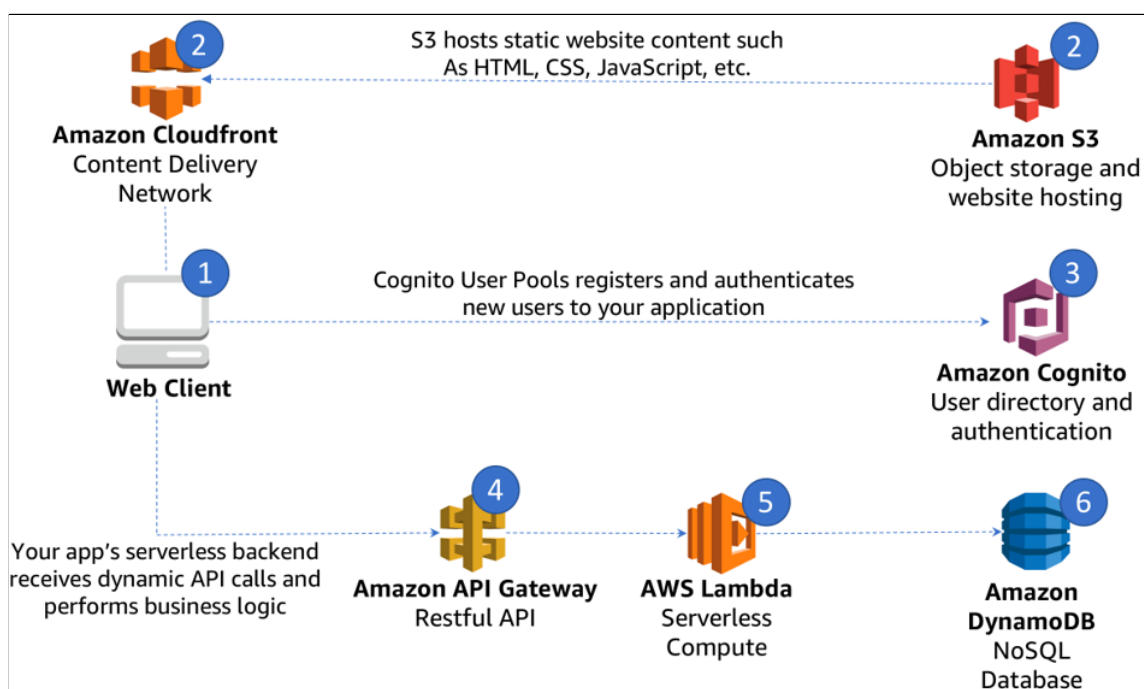
Por último, mostrar un ejemplo de arquitectura de procesamiento de transmisiones:



En esta arquitectura podemos observar cómo una vez que el usuario se conecta a la aplicación web, en este caso twitter, y escribe un tweet, no solo este usuario lo está haciendo si no que multitud de usuarios también crean sus tweets a la vez. Este tweet llega al servicio de **Kinesis Streams** y este a través de una **función lambda** escribe los tweets en la base de datos almacenada en **Amazon DynamoDB**.

## Aplicación web

El uso de AWS nos permite implementar una aplicación web **sin** la necesidad del uso de **servidores** y ofreciendo una seguridad, confiabilidad y rendimiento mejores. También nos proporciona una **alta disponibilidad** y la posibilidad de un **escalado** dependiendo de nuestra demanda.



1. Los consumidores obtienen un **mejor rendimiento** gracias al almacenamiento en caché y la dirección óptima original y limitación de llamadas repetitivas al backend, sino que también cuentan con la posibilidad de consumir la aplicación desde **cualquier punto del planeta**.
2. **Amazon S3** guarda la **parte estática** y la entrega de forma segura al usuario a través de CloudFront.
3. **Usuarios de Amazon Cognito** son los que proporcionan las funciones de **proveedor** de gestión e **identificación de usuarios**.
4. **API Gateway** es la encargada de la **entrega y envío de contenido dinámico**, le aportará un punto de enlace seguro a la aplicación Web para la realización de llamadas y le regresará las respuestas.
5. **AWS Lambda** proporciona de forma directa las operaciones **CRUD** por sobre **DynamoDB**.
6. DynamoDB ofrece una base de datos **NoSQL** la cual proporciona un manera de **escalado flexible** con el tráfico de la aplicación Web.

## 4. Implementación

Es recomendable a la hora de implementar una arquitectura de microservicios que un cambio en esta **no interrumpa** el contrato de servicio del cliente, ya que si el consumidor no está preparado para un cambio que interrumpe el contrato de servicio esto hará que se produzca un **fallo**.

El impacto de la implementación de cambios dependerá en gran medida de los **consumidores** que usen tu API por lo que es muy importante recopilar metadatos sobre el uso utilizando **claves API**. Una clave API es un string encriptado simple que se utiliza como un identificador único para autenticar solicitudes asociadas a tu proyecto.

Para suavizar el impacto de los cambios de interrupción se puede clonar la API y dirigir a los clientes a diferentes **subdominios**. Una de las **desventajas** que tiene esta implementación es el **mantenimiento** de dos versiones de la API (gastos generales por el manejo y aprovisionamiento de la infraestructura). Las diferentes formas de dirigir a los clientes al API clonada son las siguientes:

### Todos de una vez

Se crea una **copia de la versión anterior** mientras que en la actual se introducen todos los cambios, una ventaja que tiene esto es que los **cambios** en las **bases de datos** son más **sencillos** de coordinar con las transacciones durante el periodo de cambio.

Este estilo de implementación se aconseja sobre todo en modelos de **baja concurrencia** donde el impacto del usuario es mínimo, ya que en casa de reversión puede causar **tiempo de inactividad**.

### Azul/verde

El tráfico es redirigido al **nuevo entorno** (verde) mientras que mantenemos el **antiguo entorno** (azul) por si tuviéramos que revertir el despliegue.

Este patrón es uno de los **más utilizados** ya que **reduce** los tiempo de **inactividad**, aunque solo es recomendable en despliegues con **cambios** incrementales **pequeños** para que el retorno al anterior entorno sea más sencillo

### Canary/lineal

Los despliegues Canary consiste en dirigir una **pequeña cantidad** de las solicitudes al nuevo entorno comprobando así como funciona este en un número reducido de usuarios. Con este despliegue se puede implementar un cambio en su endpoint backend mientras se mantiene el mismo endpoint HTTP para los consumidores.

Este despliegue es recomendable en modelos con **alta simultaneidad**, ya que podemos decidir en todo momento el porcentaje de tráfico que se redirige a la nueva implementación.



# Tutorial conseguir “Hola Mundo”

Explicaremos cómo conseguir configurar AWS y poder dar nuestros primeros pasos en esta tecnología.

**Registrarse en AWS**


Dirección de correo electrónico  
Utilizará esta dirección de correo electrónico para iniciar sesión en su nueva cuenta de AWS.

Contraseña

Confirmar la contraseña

Nombre de la cuenta de AWS  
Elija un nombre para la cuenta. Podrá cambiarlo en la configuración de la cuenta después de registrarse.

Comprobación de seguridad



Escriba los caracteres como se indica arriba

**Continuar (paso 1 de 5)**

Para comenzar, deberemos registrarnos en Amazon AWS. Nos pedirán nuestros datos (es un formulario de registro con 4 pantallas), pero una vez terminado podremos comenzar a configurar nuestro servicio. Es importante avisar de que se solicitarán datos bancarios, pero para este tutorial solo se utilizarán los servicios de las capas gratuitas de AWS.

Una vez finalizado el registro, iniciaremos sesión en la cuenta que acabamos de registrar.

**aws**

**Inicio de sesión de usuarios de cuentas raíces**

Correo electrónico: janiraelx@hotmail.es

Contraseña [¿Ha olvidado la contraseña?](#)

**Iniciar sesión**

Tras esto, habremos accedido por primera vez a la consola:

**Página de inicio de la consola**

**Visitados recientemente**

Servicios no visitados recientemente

Explore uno de estos servicios de AWS visitados habitualmente.

[IAM](#) [EC2](#) [S3](#) [RDS](#) [Lambda](#)

**Le damos la bienvenida a A...**

- Introducción a AWS**  
Conozca los aspectos fundamentales y encuentre información valiosa para sacar el máximo provecho de AWS.
- Formación y certificación**  
Aprenda de expertos de AWS, mejore sus habilidades y aumente sus conocimientos.
- ¿Cuáles son las novedades de AWS?**  
Descubra los nuevos servicios, características y regiones de AWS.

**AWS Health**

Problemas abiertos: 0 (Últimos 7 días)

Cambios programados: 0 (Próximos 7 días y últimos 7 días)

Otras notificaciones: 0 (Últimos 7 días)

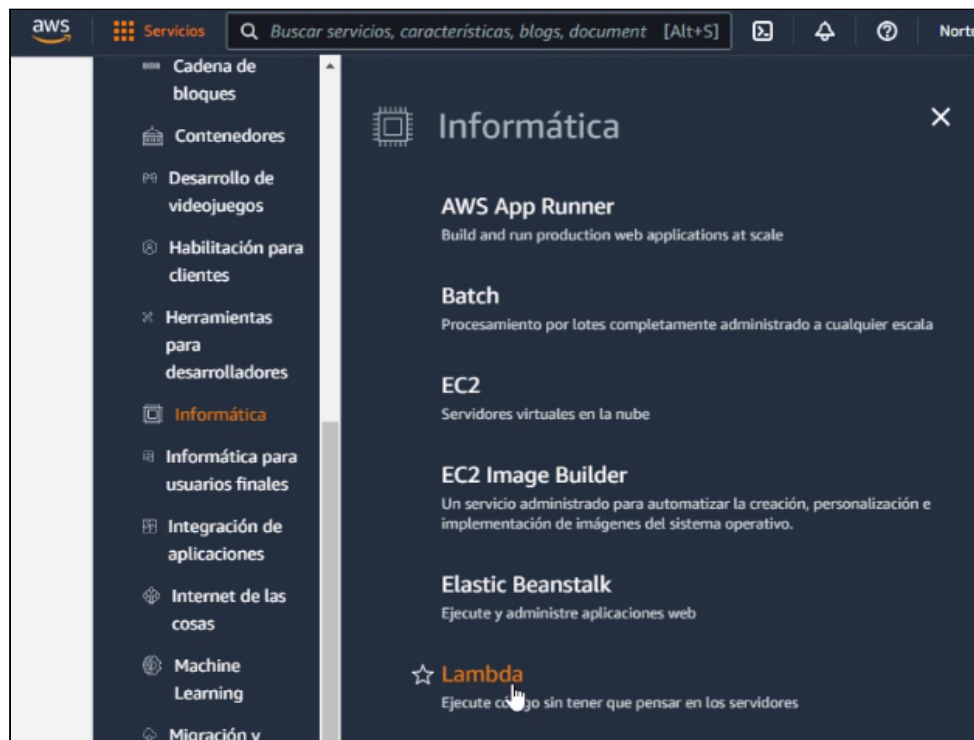
**Costo y uso**

Sin costo ni uso

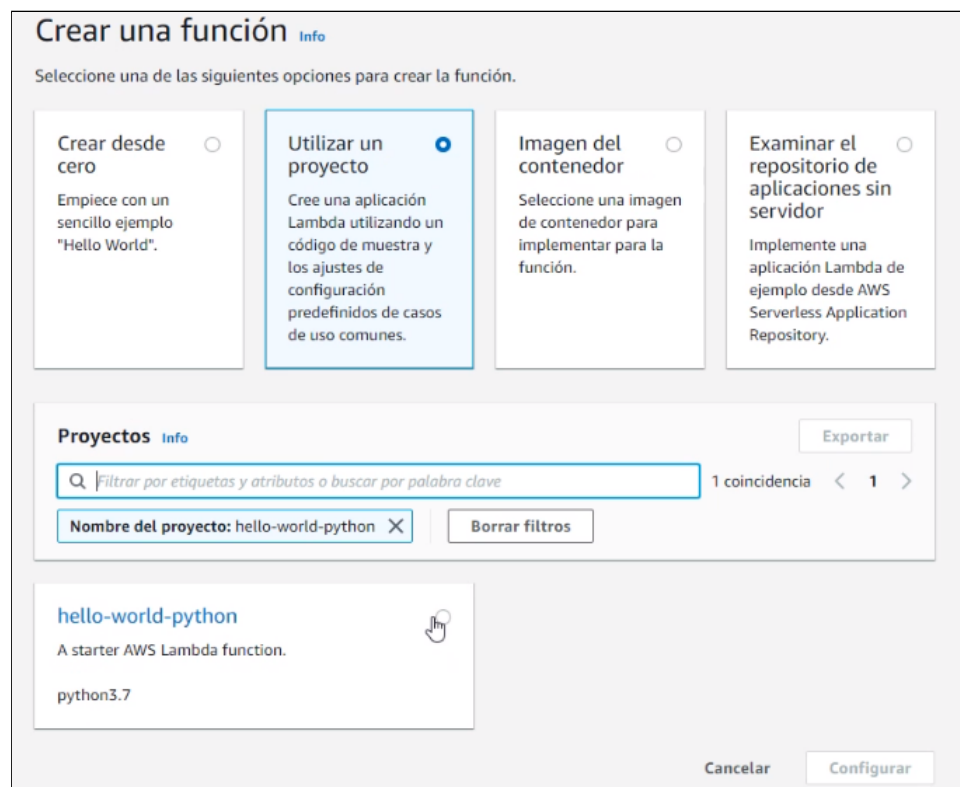
Esto podría deberse a que no ha configurado el Administrador de costos de AWS o a que no tiene permisos.

[Ir a Administración de costos de AWS](#)

A continuación, arriba en el apartado “Servicios” seleccionamos “Informática” y luego “Lambda”.



A continuación crearemos una función seleccionando “Utilizar un proyecto” e introduciendo en el buscador uno con nombre “hello-world-python”. Este proyecto base nos proporcionará herramientas ya implementadas que nos ayudarán en este ejemplo. Lo seleccionamos y le damos a “Configurar”:



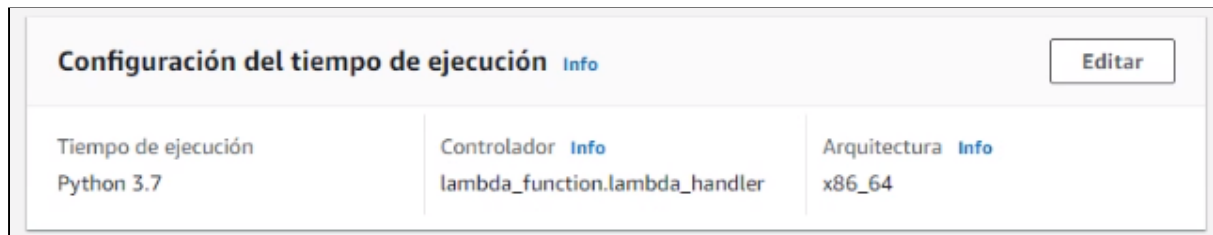
Ahora tendremos que configurar las características de nuestra función. Pondremos el nombre, el rol y el nombre del rol, tal y como se muestra en la imagen. Le damos a “Crear una función” para continuar:

The screenshot shows the 'Crear una función' (Create function) page in the AWS Lambda console. The breadcrumb trail at the top is 'Lambda > Funciones > Crear una función > Configurar el proyecto hello-world-python'. The main section is titled 'Información básica' (Basic information). It contains three main configuration areas: 1. 'Nombre de la función' (Function name) with a text input field containing 'hello-world-python'. 2. 'Rol de ejecución' (Execution role) with three radio button options: 'Creación de un nuevo rol con permisos básicos de Lambda' (selected), 'Uso de un rol existente', and 'Creación de un nuevo rol desde la política de AWS templates'. Below these is a blue information box stating: 'La creación de roles puede tardar unos minutos. No elimine el rol ni edite las políticas de confianza o de permisos de este rol.' 3. 'Nombre del rol' (Role name) with a text input field containing 'lambda\_basic\_execution' and a note: 'Utilice exclusivamente letras, números, guiones o guiones bajos. No incluya espacios.' Below this is a 'Plantillas de política' (Policy templates) section with a dropdown menu and a refresh button.

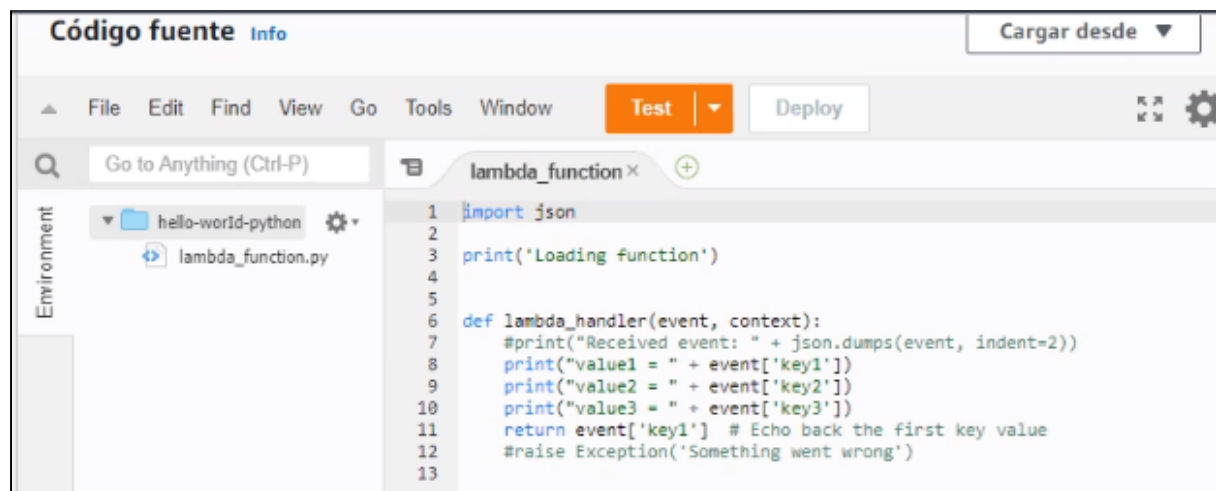
Tras esto, aparecerá esta página:

The screenshot shows the 'hello-world-python' function page in the AWS Lambda console. The breadcrumb trail is 'Lambda > Funciones > hello-world-python'. The page title is 'hello-world-python'. There are three buttons at the top right: 'Limitación', 'Copiar ARN', and 'Acciones'. The main section is titled 'Información general de la función' (General function information). It displays the function icon (a stylized 'A' logo), the name 'hello-world-python', and 'Layers (0)'. Below this are two buttons: '+ Agregar desencadenador' (Add trigger) and '+ Agregar destino' (Add destination). On the right side, there is a 'Descripción' (Description) section with the text 'A starter AWS Lambda function.', an 'Última modificación' (Last modified) section with the text 'hace 7 minutos' (7 minutes ago), and an 'ARN de la función' (Function ARN) section with the text 'arn:aws:lambda:us-east-1:127541961800:function:hello-world-python'.

Nos aseguraremos de que en el apartado “Tiempo de ejecución” esté seleccionado “Python 3.7” (la versión de Python del proyecto). En caso de que no sea así, seleccionamos “Editar” y lo cambiamos:



También podemos ver que Lambda automáticamente detectará que la función donde tiene que empezar a ejecutar su código es “lambda\_handler()”:

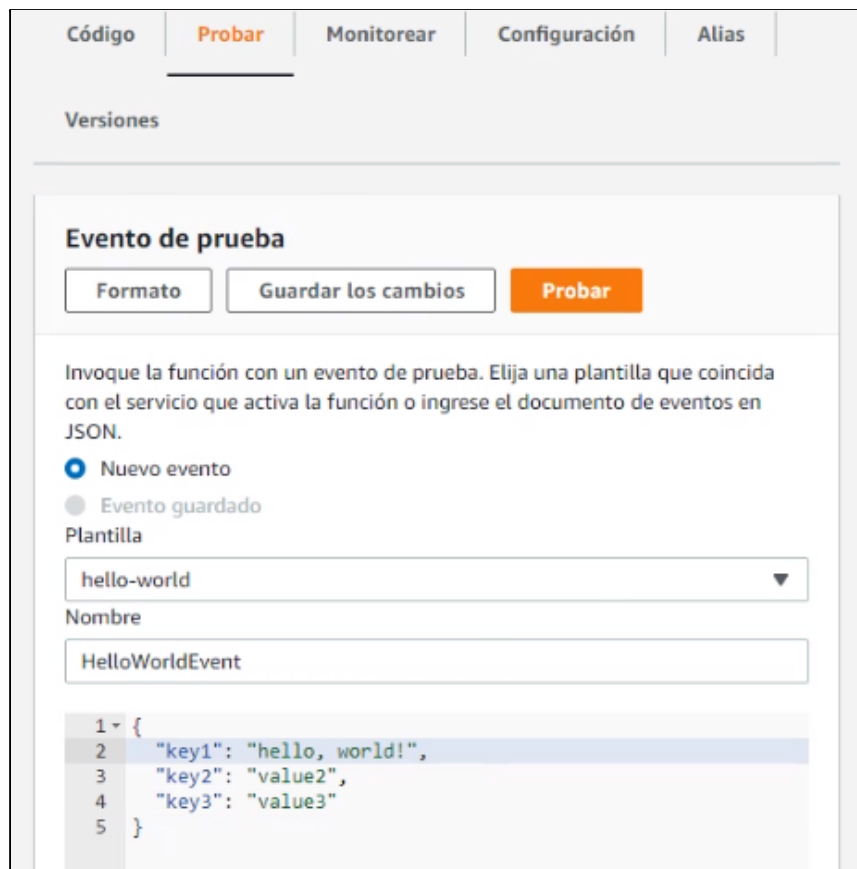


A continuación, podemos ver que Amazon AWS nos permite modificar el tamaño de la memoria, el tiempo de espera y el VPC (si deseamos ejecutar las funciones en la nube). En caso de desearlo, iremos al apartado “Configuración” y a “Editar”:



En nuestro caso lo dejaremos con los valores predeterminados. Cabe resaltar que Amazon ofrece el servicio “Amazon AWS Compute Optimizer”, encargado de ayudar a optimizar al máximo el uso de memoria y los tiempos de espera.

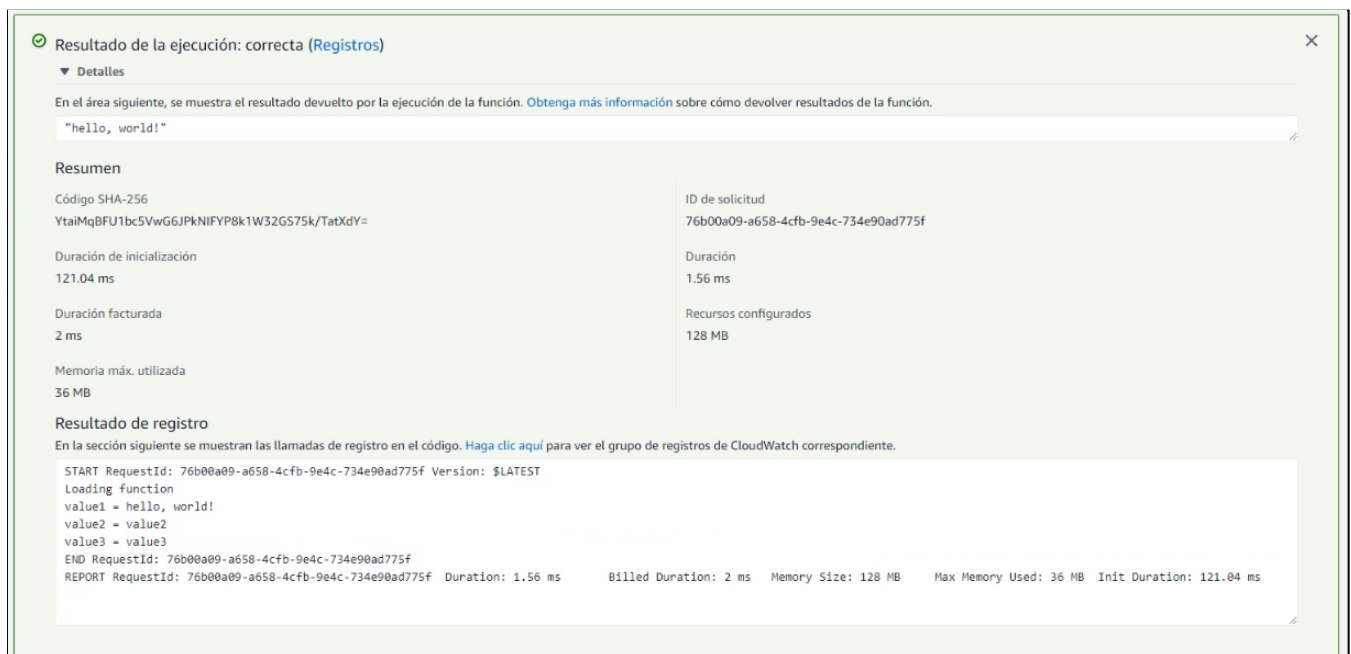
Ahora que ya hemos terminado de configurar nuestra función, podemos proceder a probar que funciona utilizando un “Evento de prueba”. Iremos al apartado “Probar” y lo configuraremos de la siguiente manera:



The screenshot shows the 'Probar' (Test) tab in the AWS Lambda console. At the top, there are tabs for 'Código', 'Probar', 'Monitorear', 'Configuración', and 'Alias'. Below the tabs is a 'Versiones' section. The main area is titled 'Evento de prueba' (Test Event). It contains three buttons: 'Formato', 'Guardar los cambios', and 'Probar'. Below these buttons, there is a text instruction: 'Invoca la función con un evento de prueba. Elige una plantilla que coincida con el servicio que activa la función o ingrese el documento de eventos en JSON.' There are two radio buttons: 'Nuevo evento' (selected) and 'Evento guardado'. Below the radio buttons is a 'Plantilla' (Template) dropdown menu with 'hello-world' selected. There is also a 'Nombre' (Name) text field with 'HelloWorldEvent' entered. At the bottom, there is a JSON editor showing a test event structure:

```
1 {  
2   "key1": "hello, world!",  
3   "key2": "value2",  
4   "key3": "value3"  
5 }
```

¡E voilà! Hemos conseguido crear un “hello, world!” con éxito en Amazon AWS:



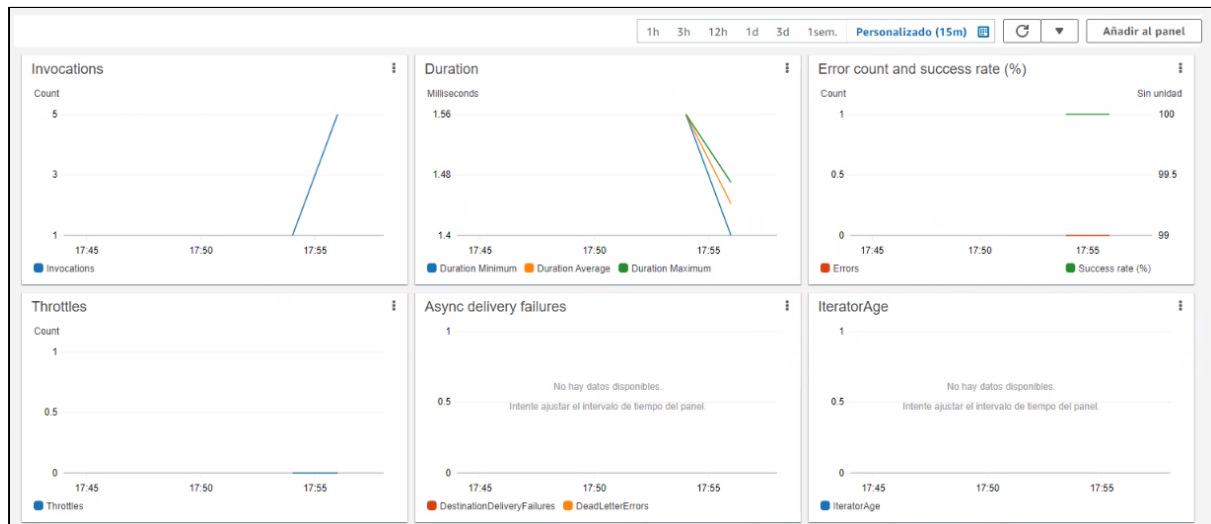
The screenshot shows the 'Resultado de la ejecución: correcta (Registros)' (Execution result: correct (Logs)) window in the AWS Lambda console. It displays the following information:

- Detalles:** En el área siguiente, se muestra el resultado devuelto por la ejecución de la función. [Obtenga más información](#) sobre cómo devolver resultados de la función. The result is "hello, world!".
- Resumen:**
  - Código SHA-256: YtaiMqBFU1bc5VwG6JpKniFYP8k1W32GS75k/TatXdY=
  - ID de solicitud: 76b00a09-a658-4cfb-9e4c-734e90ad775f
  - Duración de inicialización: 121.04 ms
  - Duración: 1.56 ms
  - Duración facturada: 2 ms
  - Recursos configurados: 128 MB
  - Memoria máx. utilizada: 36 MB
- Resultado de registro:** En la sección siguiente se muestran las llamadas de registro en el código. [Haga clic aquí](#) para ver el grupo de registros de CloudWatch correspondiente. The logs show:

```
START RequestId: 76b00a09-a658-4cfb-9e4c-734e90ad775f Version: $LATEST  
Loading function  
value1 = hello, world!  
value2 = value2  
value3 = value3  
END RequestId: 76b00a09-a658-4cfb-9e4c-734e90ad775f  
REPORT RequestId: 76b00a09-a658-4cfb-9e4c-734e90ad775f Duration: 1.56 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 36 MB Init Duration: 121.04 ms
```



Pero aquí no terminamos. Podemos monitorear el funcionamiento de nuestra función. Si probamos varias veces el evento de prueba y vamos al apartado “Monitorear” podremos visualizar gráficas de las distintas métricas obtenidas al procesar los eventos y ejecutar la función. Esto es posible gracias a que Lambda envía las métricas de tiempo de ejecución a Amazon CloudWatch. Con CloudWatch podemos, además, configurar alarmas por si, por ejemplo, las peticiones comienzan a durar demasiado tiempo.



Estas gráficas muestran el recuento de invocaciones, la duración de estas, los errores de invocación, las invocaciones limitadas, la edad del iterador y los errores de DLQ. Esta información nos puede ser muy útil para ver cómo de óptima es nuestra función y si ocurren demasiados errores cuando es invocada.

En conclusión, con estas nociones básicas podremos comenzar a escribir nuestras propias funciones Lambda y probarlas con varios eventos de prueba.

# Bibliografía

## Amazon:

- [Kinesis](#)
- [SQS](#)
- [S3](#)
- [DynamoDB](#)
- [Aurora](#)
- [AWS Serverless](#)
- [AWS CodeCommit](#)
- [Microservicios](#)
- [CloudFront](#)

## Otras:

- [Arquitectura de software dirigida por eventos](#)
- [Arquitectura dirigida por eventos](#)
- [EventBridge vs SNS vs SQS](#)
- [Consumiendo eventos](#)
- [¿Qué es la arquitectura dirigida por eventos?](#)
- [Estrategias de Despliegue para Aplicaciones Serverless](#)

## Podcast:

- “Charlas técnicas de AWS” - [Episodio 5: Empezando con Serverless en AWS](#)