Enfoque de aplicaciones sin servidor

Marco de Buena Arquitectura de AWS

Diciembre de 2019

This paper has been archived.

The latest version is now available at:

https://docs.aws.amazon.com/es_es/wellarchitected/latest/serverless-applications-lens/welcome.html

Avisos

Los clientes son responsables de realizar sus propias evaluaciones de la información contenida en este documento. Este documento: (a) solo tiene fines informativos, (b) representa las prácticas y las ofertas de productos vigentes de AWS, que están sujetas a cambios sin previo aviso, y (c) no crea ningún compromiso ni garantía de AWS y sus empresas afiliadas, proveedores o licenciantes. Los productos o servicios de AWS se proporcionan “tal cual”, sin garantías, representaciones ni condiciones de ningún tipo, ya sean explícitas o implícitas. Las responsabilidades y obligaciones de AWS con respecto a sus clientes se controlan mediante los acuerdos de AWS; este documento no forma parte de ningún acuerdo entre AWS y sus clientes ni lo modifica.

Contenido

Resumen

En este documento se describe el **enfoque de aplicaciones sin servidor** para el marco de buena arquitectura de AWS. El documento abarca escenarios comunes de aplicaciones sin servidor e identifica elementos clave para garantizar que las cargas de trabajo se diseñen de acuerdo con las prácticas recomendadas.

Introducción

El marco de buena arquitectura de AWS lo ayuda a comprender las ventajas y desventajas de las decisiones que toma cuando crea sistemas en AWS[1]. Mediante el uso del marco, aprenderá las prácticas recomendadas de arquitectura para diseñar y utilizar sistemas de confianza, seguros, eficaces y rentables en la nube. Proporciona una forma de medir sus arquitecturas de forma constante en función de las prácticas recomendadas e identificar áreas de mejora. Creemos que contar con sistemas de buena arquitectura aumenta en gran medida la probabilidad de éxito empresarial.

En este "enfoque" nos centramos en cómo diseñar, implementar y diseñar sus **cargas de trabajo de aplicaciones sin servidor** en la nube de AWS. Por cuestiones de brevedad, solo hemos cubierto detalles del marco de buena arquitectura que son específicos de las cargas de trabajo sin servidor. A la hora de diseñar su arquitectura, debe tener en cuenta las prácticas recomendadas y las preguntas que no se han incluido en este documento. Recomendamos que lea el documento técnico del Marco de buena arquitectura de AWS[2].

Este documento está destinado a aquellos que ocupan puestos en tecnología, como los directores de tecnología (CTO), arquitectos, desarrolladores y miembros del equipo de operaciones. Después de leer este documento, conocerá las prácticas recomendadas y las estrategias de AWS que debe utilizar a la hora de diseñar arquitecturas para aplicaciones sin servidor.

Definiciones

El marco de buena arquitectura de AWS se basa en cinco pilares: excelencia operativa, seguridad, confiabilidad, eficiencia del rendimiento y optimización de costos. Para cargas de trabajo sin servidor, AWS proporciona varios componentes principales (sin servidor y con servidor) que permiten diseñar arquitecturas sólidas para sus aplicaciones sin servidor. En esta sección, presentaremos información general sobre los servicios que se utilizarán en todo este documento. Existen siete aspectos que debe tener en cuenta a la hora de crear una carga de trabajo sin servidor:

- Capa informática
- Capa de datos
- Capa de mensajería y streaming
- Capa de identidad y administración de usuarios
- Capa de borde
- Monitoreo e implementación de sistemas
- Enfoques de implementación

Capa informática

La capa informática de la carga de trabajo administra las solicitudes de sistemas externos. Esto controla el acceso y garantiza que las solicitudes estén debidamente autorizadas. Contiene el entorno de tiempo de ejecución que implementará y ejecutará su lógica de negocio.

AWS Lambda permite ejecutar aplicaciones sin servidor y sin estado en una plataforma administrada que admite arquitecturas de microservicios, implementación y administración de ejecución en la capa de la función.

Con **Amazon API Gateway,** puede ejecutar una API REST completamente administrada que se integra con Lambda para ejecutar su lógica de negocio e incluye administración del tráfico, control de autorizaciones y accesos, monitoreo y control de versiones de API.

AWS Step Functions organiza flujos de trabajo sin servidor, incluidos la coordinación, el estado y el encadenamiento de funciones, así como la combinación de ejecuciones prolongadas no admitidas dentro de los límites de ejecución de Lambda mediante la división en varios pasos o mediante la llamada a empleados que ejecutan en instancias de Amazon Elastic Compute Cloud (Amazon EC2) o en las instalaciones.

Capa de datos

La capa de datos de la carga de trabajo administra el almacenamiento persistente desde un sistema. Proporciona un mecanismo seguro para almacenar los estados que necesitará su lógica de negocio. Proporciona un mecanismo para desencadenar eventos en respuesta a cambios en los datos.

Amazon DynamoDB ayuda a crear aplicaciones sin servidor mediante el suministro de una base de datos NoSQL administrada para el almacenamiento persistente. En combinación con **DynamoDB Streams,** usted puede responder casi en tiempo real a los cambios en su tabla de DynamoDB al invocar las funciones de Lambda. **DynamoDB Accelerator** (DAX) agrega una caché en memoria de alta disponibilidad para DynamoDB que ofrece una mejora del rendimiento 10 veces superior de milisegundos a microsegundos.

Con **Amazon Simple Storage Service** (Amazon S3)**,** puede crear aplicaciones web y sitios web sin servidor mediante el suministro de un almacén de gran valor altamente disponible, desde el que se pueden distribuir recursos estáticos a través de una red de entrega de contenido (CDN), como **Amazon CloudFront**.

Amazon Elasticsearch Service (Amazon ES) facilita la implementación, la protección, el funcionamiento y el escalado de Elasticsearch para el análisis de registros, la búsqueda de texto completo, el monitoreo de aplicaciones y mucho más. Amazon ES es un servicio completamente administrado que proporciona un motor de búsqueda y herramientas de análisis.

AWS AppSync es un servicio de GraphQL administrado con capacidades en tiempo real y sin conexión, así como controles de seguridad de nivel empresarial que facilitan el desarrollo

de aplicaciones. AWS AppSync proporciona una API basada en datos y un lenguaje de programación consistente para que las aplicaciones y los dispositivos se conecten a servicios como DynamoDB, Amazon ES y Amazon S3.

Capa de mensajería y streaming

La capa de mensajería de la carga de trabajo administra las comunicaciones entre componentes. La capa de streaming administra el análisis y el procesamiento en tiempo real de los datos de streaming.

Amazon Simple Notification Service (Amazon SNS) proporciona un servicio de mensajería completamente administrado para patrones de publicación/suscripción mediante notificaciones de eventos asíncronos y notificaciones push móviles para microservicios, sistemas distribuidos y aplicaciones sin servidor.

Amazon Kinesis facilita la recopilación, el procesamiento y el análisis de datos de streaming en tiempo real. Con **Amazon Kinesis Data Analytics,** puede ejecutar SQL estándar o crear aplicaciones de streaming completas con SQL.

Amazon Kinesis Data Firehose captura, transforma y carga datos de streaming en Kinesis Data Analytics, Amazon S3, Amazon Redshift y Amazon ES, lo que permite realizar análisis casi en tiempo real con las herramientas de inteligencia empresarial existentes.

Capa de identidad y administración de usuarios

La capa de identidad y administración de usuarios de la carga de trabajo proporciona identidad, autenticación y autorización tanto para clientes externos como internos de las interfaces de la carga de trabajo.

Con **Amazon Cognito**, puede agregar fácilmente el registro, el inicio de sesión y la sincronización de datos de los usuarios a aplicaciones sin servidor. Los grupos de usuarios de **Amazon Cognito** proporcionan pantallas de registro integradas y federación con Facebook, Google, Amazon y Security Assertion Markup Language (SAML). Las **identidades federadas de Amazon Cognito** permiten proporcionar acceso definido a los recursos de AWS que forman parte de su arquitectura sin servidor.

Capa de borde

La capa de borde de la carga de trabajo administra la capa de presentación y la conectividad con clientes externos. Proporciona un método de entrega eficiente a clientes externos que residan en diferentes ubicaciones geográficas.

Amazon CloudFront proporciona una CDN que entrega de forma segura contenido y datos de aplicaciones web con baja latencia y altas velocidades de transferencia.

Monitoreo e implementación de sistemas

La capa de monitoreo del sistema de la carga de trabajo administra la visibilidad del sistema a través de métricas, y crea reconocimiento contextual de cómo funciona y se comporta a lo largo del tiempo. La capa de implementación define cómo se promueven los cambios en la carga de trabajo mediante un proceso de administración de versiones.

Con **Amazon CloudWatch**, puede obtener acceso a las métricas del sistema en todos los servicios de AWS que utilice, consolidar registros de nivel de sistema y aplicación, y crear indicadores clave de rendimiento (KPI) empresariales como métricas personalizadas para sus necesidades específicas. Proporciona paneles y alertas que pueden activar acciones automatizadas en la plataforma.

AWS X-Ray permite analizar y depurar aplicaciones sin servidor mediante el suministro de mapas de servicio y rastreo distribuidos para identificar fácilmente el rendimiento en cuellos de botella mediante la visualización de una solicitud integral.

AWS Serverless Application Model (AWS SAM) es una extensión de AWS CloudFormation que se utiliza para empaquetar, probar e implementar aplicaciones sin servidor. La CLI de AWS SAM también puede permitir ciclos de depuración más rápidos con el desarrollo de funciones de Lambda a nivel local.

Enfoques de implementación

Una práctica recomendada para las implementaciones en una arquitectura de microservicios es garantizar que un cambio no rompa el contrato de servicio del consumidor. Si el propietario de la API realiza un cambio que rompe el contrato de servicio y el consumidor no está preparado para ese cambio, pueden producirse errores.

Conocer qué consumidores utilizan sus API es el primer paso para garantizar que las implementaciones sean seguras. La recopilación de metadatos sobre los consumidores y su uso le permite tomar decisiones basadas en datos sobre el impacto de los cambios. Las claves de API son una forma eficaz de registrar metadatos sobre el consumidor o los clientes de la API y, a menudo, se utilizan como forma de contacto si se realiza un cambio importante en una API.

Algunos clientes que desean adoptar un enfoque reacio al riesgo para los cambios importantes pueden elegir clonar la API y direccionar a los clientes a un subdominio diferente (por ejemplo, v2.my-service.com) para garantizar que los consumidores existentes no se vean afectados. Aunque este enfoque permite nuevas implementaciones con un nuevo contrato de servicio, la desventaja es que la sobrecarga de mantener las API duales (y la infraestructura de backend posterior) requiere una sobrecarga adicional.

En la tabla se muestran los distintos enfoques de implementación:

Implementación	Impacto en el consumidor	Restauración	Factores del modelo de eventos	Velocidad de implementación
Todo a la vez	Todo a la vez	Volver a implementar la versión anterior	Cualquier modelo de eventos con una tasa de simultaneidad baja	Inmediato
Azul/Verde	Todo a la vez con cierto nivel de pruebas de entorno de producción de antemano	Revertir el tráfico al entorno anterior	Mejor para modelos de eventos asíncronos y sincronizados en cargas de trabajo de simultaneidad mediana	Minutos a horas de validación y luego inmediato para los clientes
Canary/Lineal	Cambio de tráfico inicial típico del 1 al 10 % y luego aumenta por fases o todo a la vez	Revertir el 100 % del tráfico a la implementación anterior	Mejor para cargas de trabajo de simultaneidad alta	Minutos a horas

Implementaciones todo a la vez

Las implementaciones todo a la vez implican realizar cambios además de la configuración existente. Una ventaja de este estilo de implementación es que los cambios del backend en los almacenes de datos, tales como una base de datos relacional, requieren un nivel de esfuerzo mucho menor para conciliar las transacciones durante el ciclo de cambios. Aunque este tipo de estilo de implementación es de bajo esfuerzo y puede realizarse con poco impacto en los modelos de baja concurrencia, agrega riesgo cuando se trata de una restauración y, por lo general, provoca tiempos de inactividad. Un escenario de ejemplo para utilizar este modelo de implementación es para entornos de desarrollo en los que el impacto del usuario es mínimo.

Implementaciones azul/verde

Otro patrón de desvío de tráfico es habilitar las implementaciones azul/verde. Esta versión con un tiempo de inactividad de casi cero permite que el tráfico se desvíe al nuevo entorno activo (verde), al mismo tiempo que mantiene el antiguo entorno de producción (azul) semiactivo en caso de que sea necesario realizar una restauración. Dado que API Gateway permite definir qué porcentaje de tráfico se desvía a un entorno determinado, este estilo de implementación puede ser una técnica eficaz. Dado que las implementaciones azul/verde están diseñadas para reducir el tiempo de inactividad, muchos clientes adoptan este patrón para los cambios de producción.

Las arquitecturas sin servidor que siguen las prácticas recomendadas de ausencia de estado e idempotencia son aptas para este estilo de implementación porque no hay afinidad con la infraestructura subyacente. Debe sesgar estas implementaciones hacia cambios incrementales más pequeños para que pueda revertir fácilmente a un entorno de trabajo si es necesario.

Necesita los indicadores adecuados para saber si se requiere una restauración. Como práctica recomendada, sugerimos a los clientes que utilicen métricas de alta resolución de CloudWatch, que puedan monitorear en intervalos de 1 segundo, y registren rápidamente las tendencias descendentes. Si las utiliza con alarmas de CloudWatch, puede habilitar una restauración rápida. Las métricas de CloudWatch se pueden registrar en API Gateway, Step Functions, Lambda (incluidas las métricas personalizadas) y DynamoDB.

Implementaciones de valores controlados

Las implementaciones de Canary son una forma cada vez más frecuente de aprovechar la nueva versión de un software en un entorno controlado y permitir ciclos de implementación rápidos. Las implementaciones de Canary implican la implementación de un pequeño número de solicitudes al nuevo cambio para analizar el impacto en un número reducido de sus usuarios. Como ya no tiene que preocuparse por el aprovisionamiento y el escalado de la infraestructura subyacente de la nueva implementación, la nube de AWS ha ayudado a facilitar esta adopción.

Con las implementaciones de Canary en API Gateway, puede implementar un cambio en el punto de enlace del backend (por ejemplo, Lambda) mientras mantiene el mismo punto de enlace HTTP de API Gateway para los consumidores. Además, puede controlar qué porcentaje de tráfico se dirige a una nueva implementación y para un recorte de tráfico controlado. Un escenario práctico para una implementación de Canary podría ser un nuevo sitio web. Puede monitorear las tasas de clics en un pequeño número de usuarios finales antes de desviar todo el tráfico a la nueva implementación.

Control de versiones de Lambda

Al igual que todo el software, mantener el control de versiones permite una visibilidad rápida del código que funcionaba anteriormente, así como la posibilidad de volver a una versión anterior si una nueva implementación no se realiza correctamente. Lambda permite publicar una o varias versiones inmutables para funciones de Lambda individuales, de forma que las versiones anteriores no se puedan cambiar. Cada versión de función de Lambda tiene un nombre de recurso de Amazon (ARN) único y los cambios de versión nuevos se pueden auditar a medida que se registran en CloudTrail. Como práctica recomendada en producción, los clientes deben habilitar el control de versiones para aprovechar de la mejor manera una arquitectura de confianza.

Para simplificar las operaciones de implementación y reducir el riesgo de error, los alias de Lambda permiten diferentes variaciones de la función de Lambda en el flujo de trabajo de

desarrollo, tales como el desarrollo, la versión beta y la producción. Un ejemplo de esto es cuando una integración de API Gateway con Lambda apunta al ARN de un alias de producción. El alias de producción apuntará a una versión de Lambda. El valor de esta técnica es que permite una implementación segura cuando promociona una nueva versión al entorno real, ya que el alias de Lambda dentro de la configuración del intermediario permanece estático, por lo que es necesario realizar menos cambios.

Principios generales de diseño

El marco de buena arquitectura identifica un conjunto de principios generales de diseño que facilitan un buen diseño en la nube para aplicaciones sin servidor:

- **Rápidas, sencillas y singulares**: las funciones son concisas, cortas y de una sola finalidad, y su entorno puede estar a la altura de su ciclo de vida. Las transacciones son eficientes en cuanto a costos y, por lo tanto, se prefieren ejecuciones más rápidas.
- **Piense en solicitudes simultáneas, no en solicitudes totales**: las aplicaciones sin servidor aprovechan el modelo de simultaneidad, y las alternativas a nivel de diseño se evalúan en función de la simultaneidad.
- **No comparta nada**: el entorno de tiempo de ejecución de la función y la infraestructura subyacente son de corta duración, por lo que los recursos locales como el almacenamiento temporal no están garantizados. El estado se puede manipular dentro de un ciclo de vida de ejecución de máquina de estado y se prefiere el almacenamiento persistente para los requisitos de larga duración.
- **Suponga que no hay afinidad de hardware**: la infraestructura subyacente puede cambiar. El aprovechamiento del código o las dependencias que son independientes del hardware como las banderas de CPU, por ejemplo, podrían no estar disponibles de forma coherente.
- **Organice su aplicación con máquinas de estado, no funciones**: el encadenamiento de ejecuciones de Lambda dentro del código para organizar el flujo de trabajo de su aplicación da como resultado una aplicación monolítica y estrechamente acoplada. En su lugar, utilice una máquina de estado para organizar transacciones y flujos de comunicación.
- **Utilice eventos para activar transacciones**: eventos como escribir un nuevo objeto de Amazon S3 o una actualización en una base de datos permiten la ejecución de transacciones en respuesta a funcionalidades empresariales. Este comportamiento de eventos asíncrono suele ser independiente del consumidor e impulsa el procesamiento justo a tiempo para garantizar un diseño de servicios eficiente.

- **Diseño para errores y duplicados**: las operaciones disparadas a partir de solicitudes o eventos deben ser idempotentes, ya que pueden producirse errores y una solicitud o evento determinado puede entregarse más de una vez. Incluya los reintentos adecuados para las llamadas posteriores.

Escenarios

En esta sección, se explican las cinco situaciones clave más comunes en muchas aplicaciones sin servidor y cómo influyen en el diseño y la arquitectura de las cargas de trabajo de aplicaciones sin servidor en AWS. Presentaremos los supuestos que planteamos para cada uno de estos escenarios, los impulsores comunes del diseño y una arquitectura de referencia sobre cómo deben implementarse estos escenarios.

Microservicios RESTful

Al crear un microservicio, debe pensar cómo puede entregarse un contexto empresarial como un servicio reutilizable para sus consumidores. La implementación específica se adaptará a casos de uso individuales, pero hay varios temas comunes entre los microservicios para garantizar que su implementación sea segura, resiliente y diseñada para ofrecer la mejor experiencia a sus clientes.

La creación de microservicios sin servidor en AWS le permite no solo aprovechar las capacidades sin servidor, sino también utilizar otros servicios y características de AWS, así como el ecosistema de herramientas de la red de socios de AWS y AWS (APN). Las tecnologías sin servidor se crean sobre una infraestructura tolerante a errores, lo que le permite crear servicios de confianza para sus cargas de trabajo críticas. El ecosistema de herramientas le permite simplificar la compilación, automatizar tareas, organizar dependencias, y monitorear y controlar sus microservicios. Por último, las herramientas sin servidor de AWS se cobran según el uso, lo que le permite ampliar el servicio con su empresa y mantener los costos bajos durante las fases iniciales y las horas que no son pico.

Características:

- Puede tener un marco seguro y fácil de utilizar que sea sencillo de replicar y que tenga altos niveles de resiliencia y disponibilidad.
- Puede registrar patrones de utilización y acceso para mejorar continuamente su backend y respaldar el uso de los clientes.
- Puede aprovechar los servicios administrados en la medida de lo posible para sus plataformas, lo que reduce las arduas tareas asociadas con la administración de plataformas comunes, incluida la seguridad y la escalabilidad.

Arquitectura de referencia

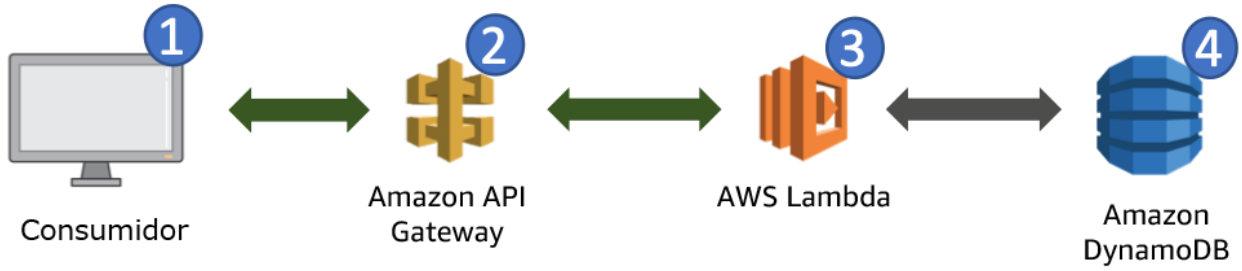

Figura 1: Arquitectura de referencia para microservicios RESTful

1. Los **clientes** aprovechan sus microservicios con la realización de llamadas a la API HTTP. Lo ideal es que los consumidores tengan un contrato de servicio estrechamente vinculado a la API para lograr expectativas consistentes en cuanto a niveles de servicio y control de cambios.
2. **Amazon API Gateway** aloja solicitudes HTTP RESTful y respuestas a los clientes. En este caso, API Gateway proporciona autorización integrada, limitación controlada, seguridad, tolerancia a errores, mapeo de solicitud/respuesta y optimizaciones de rendimiento.
3. **AWS Lambda** contiene la lógica de negocio para procesar llamadas a la API entrantes y aprovechar DynamoDB como almacenamiento persistente.
4. **Amazon DynamoDB** almacena de forma persistente datos de microservicios y escala en función de la demanda. Dado que los microservicios suelen estar diseñados para hacer una cosa bien, se incorpora periódicamente un almacén de datos NoSQL sin esquema.

Notas de configuración:

- Aproveche el registro de API Gateway para comprender la visibilidad de los comportamientos de acceso de los consumidores de microservicios. Esta información está visible en Amazon CloudWatch Logs y puede visualizarse rápidamente a través de Log Pivots, analizarse en CloudWatch Logs Insights o introducirse en otros motores que permiten realizar búsquedas, como Amazon ES o Amazon S3 (con Amazon Athena). La información entregada proporciona visibilidad clave, como:
    - Comprender las ubicaciones comunes de los clientes, que pueden cambiar geográficamente en función de la proximidad de su backend.
    - Comprender cómo las solicitudes de entrada de los clientes pueden afectar a la forma en que particiona la base de datos.
    - Comprender la semántica del comportamiento anómalo, que puede ser un indicador de seguridad.
    - Comprender errores, latencia y aciertos/fallos de caché para optimizar la configuración.

Este modelo proporciona un marco que es fácil de implementar y mantener, y un entorno seguro que se escalará a medida que aumenten sus necesidades.

Alexa Skills

Alexa Skills Kit ofrece a los desarrolladores la posibilidad de ampliar las capacidades de Alexa mediante la creación de experiencias visuales y de voz naturales y atractivas. Las habilidades exitosas son formadoras de hábito, a los que los usuarios vuelven rutinariamente porque ofrecen algo único, proporciona valor de formas nuevas, novedosas y sin fricciones.

La mayor causa de frustración por parte de los usuarios es cuando la habilidad no actúa de la manera esperada y puede tomar varias interacciones antes de lograr lo que necesitan. Es fundamental comenzar por el diseño de un modelo de interacción de voz y trabajar hacia atrás a partir de ahí, ya que algunos usuarios pueden decir demasiado poco, demasiado o posiblemente algo que usted no espera. El proceso de diseño de voz implica crear, realizar scripts y planificar enunciados esperados e inesperados.

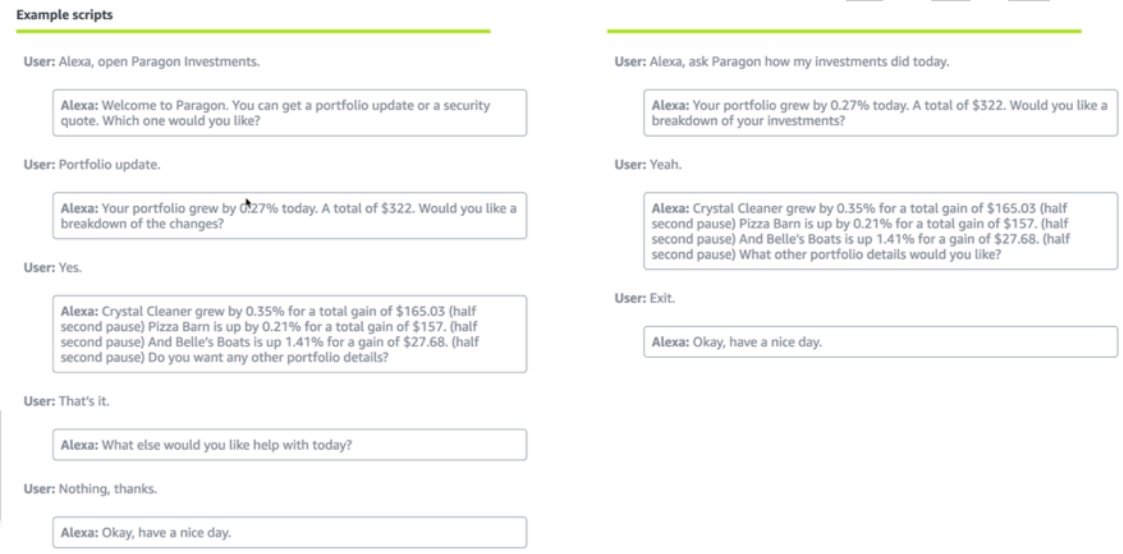

Figura 2: Script de diseño de ejemplo de Alexa Skill

Teniendo en cuenta un script básico, puede utilizar las siguientes técnicas antes de empezar a crear una habilidad:

- **Diseñe la ruta más corta hasta su finalización**
    - La ruta más corta hasta la finalización suele ser cuando el usuario proporciona toda la información y categorías a la vez, una cuenta ya está vinculada si procede y se cumplen otros requisitos previos en una única invocación de la habilidad.

- **Diseñe las rutas alternativas y árboles de decisión**
    - A menudo, lo que dice el usuario no incluye toda la información necesaria para completar la solicitud. En el flujo, identifique las rutas alternativas y las decisiones de los usuarios.
- **Diseñe las decisiones en segundo plano que la lógica del sistema tendrá que tomar**
    - Identifique las decisiones del sistema en segundo plano, por ejemplo, con usuarios nuevos o recurrentes. Una comprobación del historial del sistema podría cambiar el flujo que sigue un usuario.
- **Diseñe cómo la habilidad ayudará al usuario**
    - Incluya instrucciones claras en la ayuda sobre lo que los usuarios pueden hacer con la habilidad. En función de la complejidad de la habilidad, la ayuda podría proporcionar una respuesta simple o varias respuestas.
- **Diseñe el proceso de vinculación de cuentas, si lo hay**
    - Determine la información necesaria para el enlace de cuentas. También debe identificar cómo responderá la habilidad cuando no se haya completado el enlace de cuentas.

Características:

- Puede crear una arquitectura sin servidor completa sin administrar instancias ni servidores.
- Puede lograr que su contenido se desacople de su habilidad en la medida de lo posible.
- Busca ofrecer experiencias de voz atractivas expuestas como una API para optimizar el desarrollo en diferentes dispositivos, regiones e idiomas de Alexa.
- Puede lograr una elasticidad que se amplíe y reduzca para satisfacer las demandas de los usuarios y gestionar patrones de uso inesperados.

Arquitectura de referencia

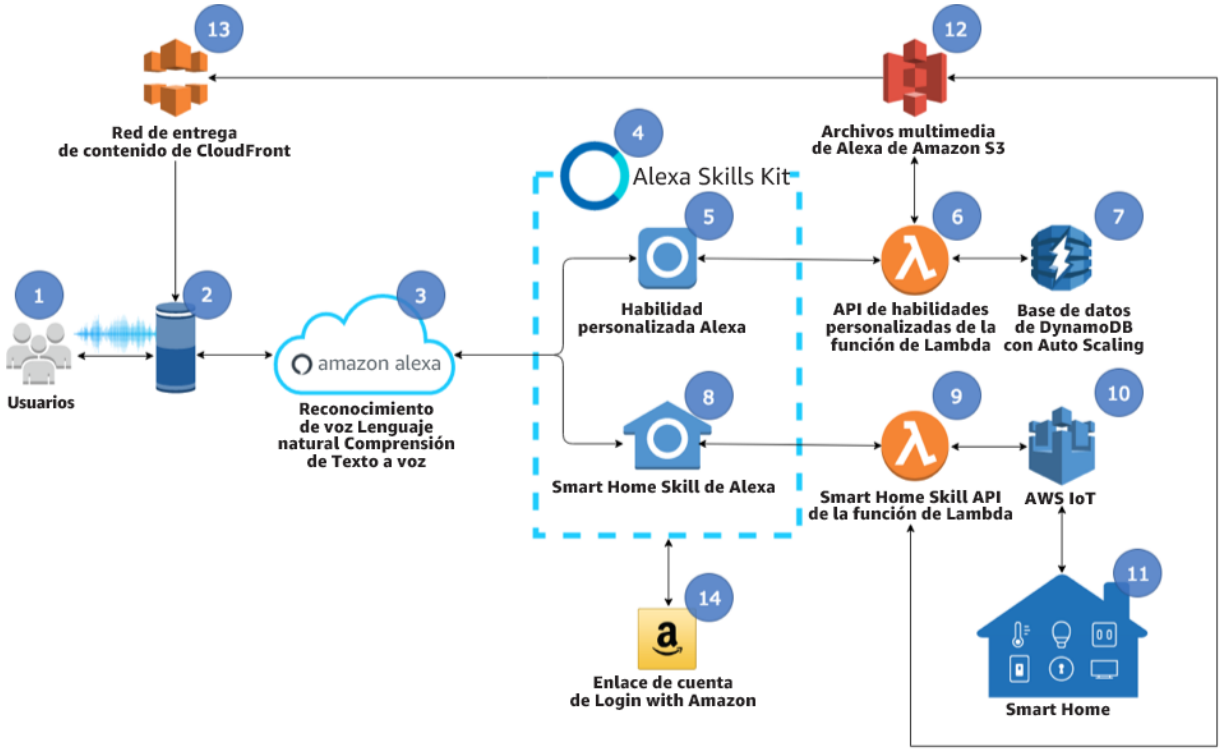

Figura 3: Arquitectura de referencia para una habilidad de Alexa

1. **Los usuarios de Alexa** interactúan con Alexa Skills al hablar con dispositivos habilitados para Alexa y utilizan la voz como método principal de interacción.
2. **Los dispositivos habilitados para Alexa** escuchan una palabra de activación y se activan tan pronto como se reconoce una. Las palabras de activación admitidas son Alexa, Computer y Echo.
3. El **servicio de Alexa** realiza un procesamiento común de la comprensión del lenguaje del habla (SLU) en nombre de su Alexa Skill, incluido el reconocimiento automático del habla (ASR), la comprensión del lenguaje natural (NLU) y la conversión de texto a voz (TTS).
4. **Alexa Skills Kit (ASK)** es una colección de API de autoservicio, herramientas, documentación y ejemplos de código que le permiten agregar habilidades a Alexa de manera rápida y sencilla. ASK es un activador de AWS Lambda de confianza que permite una integración perfecta.
5. **Alexa Custom Skill** ofrece control sobre la experiencia del usuario, lo que le permite crear un modelo de interacción personalizado. Es el tipo de habilidad más flexible, pero también la más compleja.

6. Una **función Lambda** que utiliza Alexa Skills Kit, lo que le permite crear habilidades sin problemas y evitar la complejidad innecesaria. Al utilizarlo, puede procesar diferentes tipos de solicitudes enviadas desde el servicio de Alexa y crear respuestas de voz.
7. Una **base de datos de DynamoDB** puede proporcionar un almacén de datos NoSQL que se puede escalar de manera elástica con el uso del umbral. Las habilidades suelen utilizarla para conservar el estado y las sesiones del usuario.
8. **Alexa Smart Home Skill** permite controlar dispositivos como luces, termostatos, televisores inteligentes, etc. mediante la API Smart Home. Las habilidades de Smart Home son más sencillas para crear habilidades personalizadas, ya que no le proporcionan control sobre el modelo de interacción.
9. Se utiliza una **función Lambda** para responder a las solicitudes de detección y control de dispositivos del servicio de Alexa. Los desarrolladores la utilizan para controlar un amplio número de dispositivos, incluidos los dispositivos de entretenimiento, las cámaras, la iluminación, los termostatos, las cerraduras y muchos más.
10. El **Internet de las cosas (IoT) de AWS** permite a los desarrolladores conectar de forma segura sus dispositivos a AWS y controlar la interacción entre sus habilidades de Alexa y sus dispositivos.
11. Un **Smart Home** habilitado para Alexa puede tener un número ilimitado de dispositivos conectados IoT que reciben y responden a las directivas de una Alexa Skill.
12. **Amazon S3** almacena los recursos estáticos de sus habilidades, incluidos las imágenes, el contenido y los medios. Su contenido se distribuye de forma segura con CloudFront.
13. **La red de entrega de contenido (CDN) de Amazon CloudFront** proporciona una CDN que distribuye contenido con mayor rapidez a usuarios de dispositivos móviles distribuidos geográficamente e incluye mecanismos de seguridad para recursos estáticos en Amazon S3.
14. **La vinculación de cuentas** es necesaria cuando su habilidad debe autenticarse con otro sistema. Esta acción asocia al usuario de Alexa a un usuario específico del otro sistema.

Notas de configuración:

- Valide las cargas de solicitud y respuesta de Smart Home al validar con el esquema JSON todos los mensajes posibles de Alexa Smart Home que envía una habilidad a Alexa.
- Asegúrese de que el tiempo de espera de la función Lambda sea inferior a ocho segundos y pueda gestionar las solicitudes dentro de ese periodo de tiempo. (El tiempo de espera del servicio de Alexa es de 8 segundos).

- Siga las prácticas recomendadas[7] para crear sus tablas de DynamoDB. Utilice tablas bajo demanda cuando no esté seguro de cuánta capacidad de lectura/escritura necesita. De lo contrario, elija una capacidad aprovisionada con escalado automático habilitado. En el caso de las habilidades que están muy preparadas, DynamoDB Accelerator (DAX) puede mejorar enormemente los tiempos de respuesta.
- La vinculación de cuentas proporciona información de usuario que puede almacenarse en un sistema externo. Utilice esa información para ofrecer una experiencia contextual y personalizada al usuario. Alexa tiene directrices sobre la vinculación de cuentas para ofrecer experiencias sin fricciones.
- Utilice la versión beta de la herramienta de pruebas de habilidades para recopilar comentarios anticipados sobre el desarrollo de habilidades y para el control de versiones de habilidades con el fin de reducir el impacto en las habilidades que ya están en funcionamiento.
- Utilice la CLI de ASK para automatizar el desarrollo y la implementación de habilidades.

Backend móvil

Los usuarios esperan cada vez más que sus aplicaciones móviles tengan una experiencia de usuario rápida, coherente y con una gran cantidad de características. Al mismo tiempo, los patrones de los usuarios de dispositivos móviles son dinámicos con picos de uso impredecibles y, a menudo, tienen una presencia global.

La creciente demanda de los usuarios de dispositivos móviles significa que las aplicaciones necesitan un conjunto avanzado de servicios móviles que funcionen perfectamente sin sacrificar el control y la flexibilidad de la infraestructura del backend. De forma predeterminada, se esperan determinadas capacidades entre las aplicaciones móviles:

- Capacidad para consultar, mutar y suscribirse a los cambios en la base de datos
- Persistencia de datos sin conexión y optimizaciones de ancho de banda cuando se conecta
- Búsqueda, filtrado y detección de datos en aplicaciones
- Análisis del comportamiento de los usuarios
- Mensajería dirigida a través de varios canales (notificaciones push, SMS, correo electrónico)
- Contenido avanzado, tales como imágenes y vídeos
- Sincronización de datos entre varios dispositivos y varios usuarios
- Controles de autorización exhaustivos para ver y manipular datos

La creación de un backend móvil sin servidor en AWS le permite proporcionar estas capacidades a la vez que administra automáticamente la escalabilidad, la elasticidad y la disponibilidad de una manera eficiente y rentable.

Características:

- Puede controlar el comportamiento de los datos de la aplicación desde el cliente y seleccionar explícitamente los datos que desea de la API.
- Puede lograr que su lógica de negocio se desacople de su aplicación móvil en la medida de lo posible.
- Busca proporcionar funcionalidades empresariales en forma de API para optimizar el desarrollo en varias plataformas.
- Busca aprovechar los servicios administrados para reducir la tarea pesada no diferenciada que supone mantener la infraestructura de backend móvil y, al mismo tiempo, ofrecer altos niveles de escalabilidad y disponibilidad.
- Puede optimizar los costos del backend móvil en función de la demanda real de los usuarios en lugar de pagar por recursos inactivos.

Arquitectura de referencia

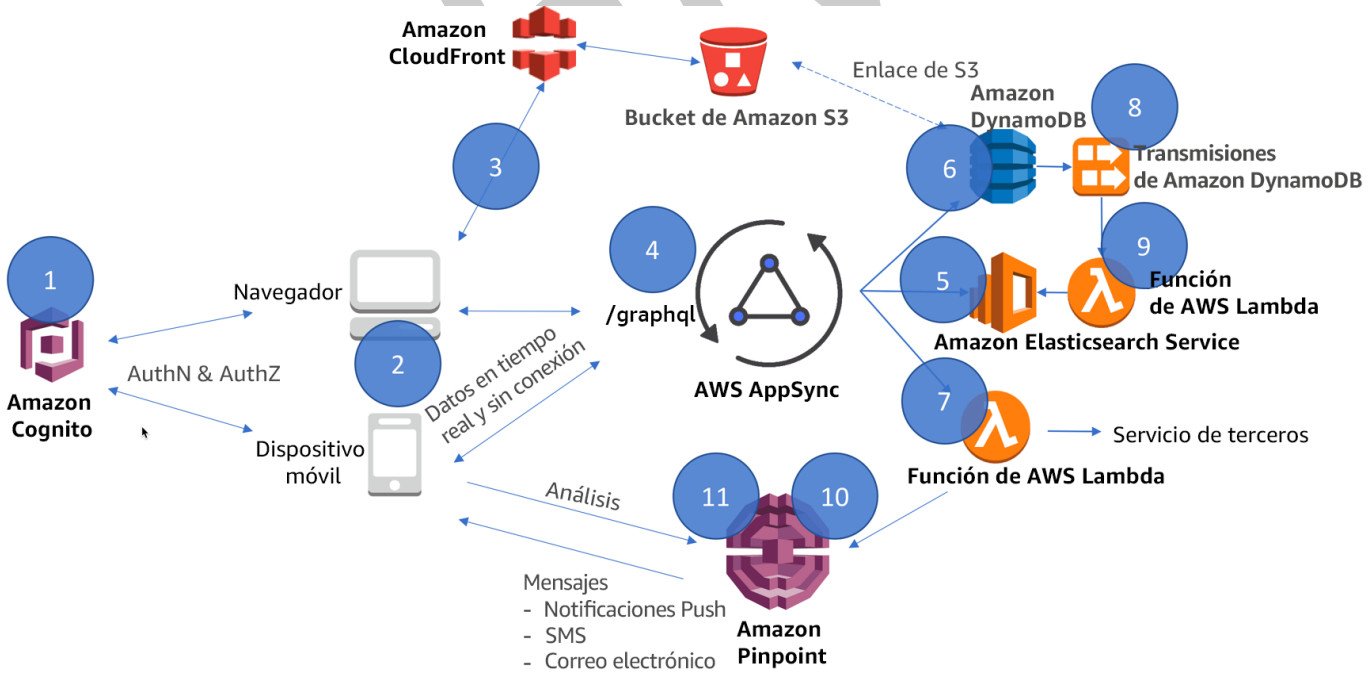

Figura 4: Arquitectura de referencia para un backend móvil

1. **Amazon Cognito** se utiliza para la administración de usuarios y como proveedor de identidad para su aplicación móvil. Además, permite a los usuarios de dispositivos móviles aprovechar las identidades sociales existentes, como Facebook, Twitter, Google+ y Amazon para iniciar sesión.
2. Los **usuarios de dispositivos móviles** interactúan con el backend de la aplicación móvil mediante la realización de operaciones de GraphQL en AWS AppSync y las API de servicio de AWS (por ejemplo, Amazon S3 y Amazon Cognito).
3. **Amazon S3** almacena recursos estáticos de aplicaciones móviles, incluidos determinados datos de usuarios de dispositivos móviles, como las imágenes de perfil. Su contenido se distribuye de forma segura a través de CloudFront.
4. **AWS AppSync** aloja solicitudes HTTP de GraphQL y respuestas a usuarios de dispositivos móviles. En este caso, los datos de AWS AppSync se encuentran en tiempo real cuando los dispositivos están conectados y los datos también están disponibles sin conexión. Los orígenes de datos para este escenario son **Amazon DynamoDB**, **Amazon Elasticsearch Service**, o las funciones de **AWS Lambda**.
5. **Amazon Elasticsearch Service** actúa como un motor de búsqueda principal para su aplicación móvil y también como análisis.
6. **DynamoDB** proporciona almacenamiento persistente para su aplicación móvil, incluidos mecanismos para hacer que los datos no deseados de usuarios de dispositivos móviles inactivos venzan mediante una característica de tiempo de vida (TTL).
7. Una función de **Lambda** gestiona la interacción con servicios de otros proveedores o llama a otros servicios de AWS para flujos personalizados que pueden formar parte de la respuesta de GraphQL a los clientes.
8. **DynamoDB Streams** captura cambios en el nivel de elemento y permite que una función de Lambda actualice los orígenes de datos adicionales.
9. Una función de **Lambda** administra los datos de streaming entre DynamoDB y Amazon ES, lo que permite a los clientes combinar tipos y operaciones de orígenes de datos lógicos de GraphQL.
10. **Amazon Pinpoint** registra análisis de clientes, incluidas sesiones de usuario y las métricas personalizadas para obtener información sobre las aplicaciones.
11. **Amazon Pinpoint** entrega mensajes a todos los usuarios/dispositivos o a un subconjunto específico en función de los análisis recopilados. Los mensajes se pueden personalizar y enviar mediante notificaciones push, correo electrónico o canales de SMS.

Notas de configuración:

- Pruebe el rendimiento[3] de las funciones de Lambda con diferentes ajustes de memoria y tiempos de espera para asegurarse de que utiliza los recursos más adecuados para el trabajo.
- Siga las prácticas recomendadas[4] cuando crea sus tablas de DynamoDB y considere la posibilidad de que AWS AppSync las aprovisione automáticamente a partir de un esquema de GraphQL, el cual utilizará una clave hash bien distribuida y creará índices para sus operaciones. Asegúrese de calcular la capacidad de lectura/escritura y la partición de la tabla para garantizar tiempos de respuesta razonables.
- Utilice el almacenamiento en caché de datos del lado del servidor de AWS AppSync para optimizar la experiencia de la aplicación, ya que todas las solicitudes de consulta posteriores a la API se devolverán desde la caché. Esto significa que no se contactarán directamente los orígenes de datos a menos que el TTL caduque.
- Siga las prácticas recomendadas[5] cuando administre dominios de Amazon ES. Además, Amazon ES proporciona una guía[6] exhaustiva sobre el diseño de partición y los patrones de acceso que también se aplican aquí.
- Utilice los controles de acceso minuciosos de AWS AppSync, configurados en solucionadores, para filtrar las solicitudes de GraphQL hasta el nivel por usuario o grupo si es necesario. Esto se puede aplicar a la autorización de AWS Identity and Access Management (IAM) o a los grupos de usuarios de Amazon Cognito con AWS AppSync.
- Utilice AWS Amplify y la CLI de Amplify para componer e integrar su aplicación con varios servicios de AWS. La consola de Amplify también se encarga de implementar y administrar pilas.

Para requisitos de baja latencia en los que se requiere una lógica empresarial casi nula, Amazon Cognito Federated Identity puede proporcionar credenciales específicas para que su aplicación móvil se comunique directamente con un servicio de AWS, por ejemplo, al cargar la imagen de perfil de un usuario, recuperar archivos de metadatos de Amazon S3 relacionados con un usuario, etc.

Procesamiento de streaming

La incorporación y el procesamiento de datos de streaming en tiempo real requieren escalabilidad y baja latencia para respaldar una variedad de aplicaciones, como el seguimiento de actividades, el procesamiento de órdenes de transacciones, el análisis de secuencias de clics, la limpieza de datos, la generación de métricas, el filtrado de registros, la indexación, el análisis de redes sociales y la telemetría y medición de datos de dispositivos IoT. Estas aplicaciones suelen presentar picos y procesar miles de eventos por segundo.

Con AWS Lambda y Amazon Kinesis, puede crear un proceso de transmisión sin servidor que se escale automáticamente sin aprovisionar ni administrar servidores. Los datos que procesa AWS Lambda pueden almacenarse en DynamoDB y analizarse más adelante.

Características:

- Puede crear una arquitectura sin servidor completa sin administrar ninguna instancia ni servidor para procesar datos de streaming.
- Puede utilizar Amazon Kinesis Producer Library (KPL) para encargarse de la incorporación de datos desde la perspectiva del productor de datos.

Arquitectura de referencia

Aquí presentamos un escenario para el procesamiento de transmisiones comunes, el cual es una arquitectura de referencia para el análisis de datos de redes sociales.

Figura 5: Arquitectura de referencia para el procesamiento de transmisiones

1. Los **productores de datos** utilizan Amazon Kinesis Producer Library (KPL) para enviar datos de streaming de redes sociales a un flujo de Kinesis. También se pueden usar el agente de Amazon Kinesis y los productores de datos personalizados que utilizan la API de Kinesis.
2. Una **transmisión de Amazon Kinesis** recopila, procesa y analiza datos de streaming en tiempo real que producen los productores de datos. Los datos introducidos en el flujo pueden ser procesados por un consumidor, que, en este caso, es Lambda.
3. **AWS Lambda** actúa como consumidor de la transmisión que recibe una matriz de los datos introducidos como un único evento o invocación. La función Lambda realiza el procesamiento adicional. Los datos transformados se almacenan en un almacenamiento persistente que, en este caso, es DynamoDB.
4. **Amazon DynamoDB** proporciona un servicio de base de datos NoSQL rápido y flexible que incluye disparadores que se pueden integrar con AWS Lambda para que estos datos estén disponibles en otro lugar.

5. **Los usuarios empresariales** aprovechan una interfaz de informes sobre DynamoDB para recopilar información de los datos de tendencias de las redes sociales.

Notas de configuración:

- Siga las prácticas recomendadas[7] cuando reparticione las transmisiones de Kinesis para acomodar una tasa de incorporación mayor. El número de particiones y el factor de paralelización determinan la simultaneidad del procesamiento de flujos de datos. Por lo tanto, debe ajustarlo de acuerdo con sus requisitos de desempeño.
- Considere la posibilidad de revisar el documento técnico Soluciones de datos de streaming[8] para el procesamiento por lotes, el análisis de transmisiones y otros patrones útiles.
- Cuando no utilice KPL, asegúrese de tener en cuenta los errores parciales de las operaciones no atómicas, como PutRecords, ya que la API de Kinesis devuelve los registros[9] procesados correctamente y sin éxito en el momento de la incorporación.
- Pueden producirse registros duplicados[10] y debe aprovechar los reintentos y la idempotencia dentro de la aplicación tanto para los consumidores como para los productores.
- Considere la posibilidad de utilizar Kinesis Data Firehose sobre Lambda cuando los datos introducidos deban cargarse continuamente en Amazon S3, Amazon Redshift o Amazon ES.
- Considere la posibilidad de utilizar Kinesis Data Analytics sobre Lambda cuando SQL estándar pueda utilizarse para consultar datos de streaming y cargar solo sus resultados en Amazon S3, Amazon Redshift, Amazon ES o Kinesis Streams.
- Siga las prácticas recomendadas para la invocación basada en transmisiones de AWS Lambda[11] ya que cubre los efectos en el tamaño de los lotes, la simultaneidad por partición y el monitoreo del procesamiento de transmisiones con más detalle.
- Utilice la cantidad máxima de intentos de reintentos Lambda, la antigüedad máxima de los registros, la bisección en errores de función, y los controles de destino para errores para crear aplicaciones de procesamiento de transmisiones más resistentes.

Aplicación web

Las aplicaciones web suelen tener requisitos exigentes para garantizar una experiencia de usuario coherente, segura y de confianza. Para garantizar una alta disponibilidad, disponibilidad global y la capacidad de escalar a miles o potencialmente millones de usuarios, a menudo se debía que reservar un exceso de capacidad considerable para gestionar solicitudes web con la mayor demanda prevista. Esto a menudo requería administrar flotas de servidores y componentes de infraestructura adicionales que, a su vez, generaban importantes gastos de capital y tiempos de espera prolongados para el aprovisionamiento de capacidad.

Con la informática sin servidor en AWS, puede implementar toda la pila de aplicaciones web sin realizar la ardua tarea no diferenciada de administrar servidores, estimar la capacidad de aprovisionamiento o pagar por recursos inactivos. Además, no tiene que poner en riesgo la seguridad, la confiabilidad ni el rendimiento.

Características:

- Puede obtener una aplicación web escalable que pueda globalizarse en cuestión de minutos con un alto nivel de resiliencia y disponibilidad.
- Puede lograr una experiencia de usuario coherente con tiempos de respuesta adecuados.
- Busca aprovechar los servicios administrados en la medida de lo posible para sus plataformas a fin de limitar las tareas pesadas asociadas con la administración de plataformas comunes.
- Puede optimizar los costos en función de la demanda real de los usuarios en lugar de pagar por recursos inactivos.
- Puede crear un marco que sea fácil de configurar y utilizar, y que pueda ampliar con un impacto limitado más adelante.

Arquitectura de referencia

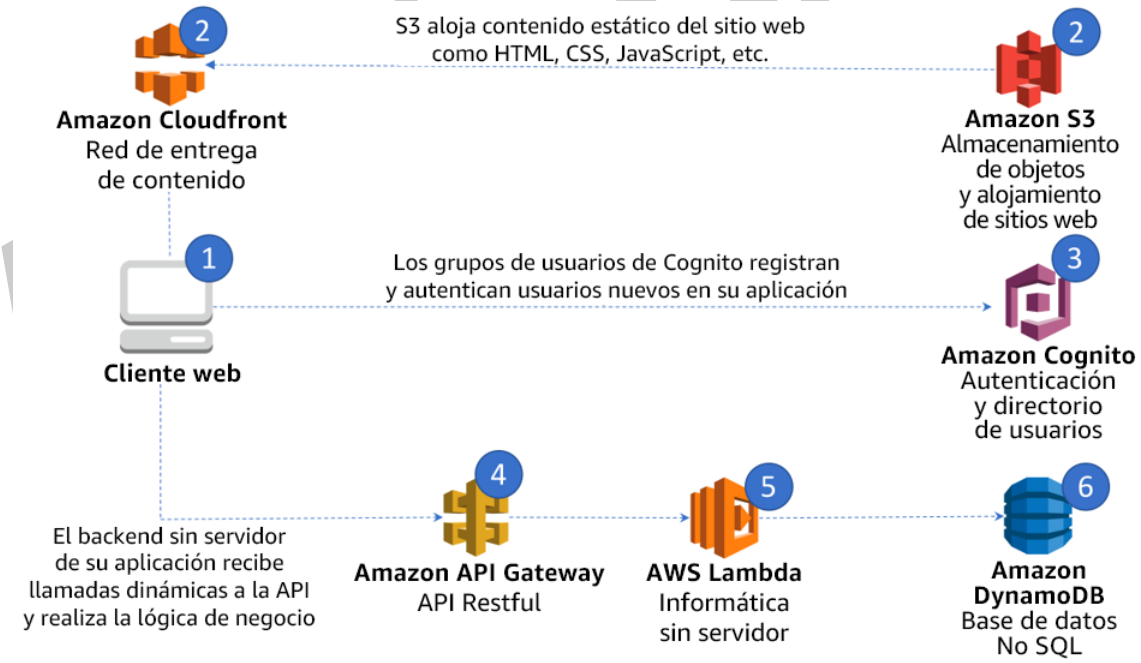

Figura 6: Arquitectura de referencia para una aplicación web

1. **Los consumidores** de esta aplicación web podrían estar concentrados geográficamente o distribuidos en todo el mundo. El uso de Amazon CloudFront no solo proporciona una mejor experiencia de rendimiento para estos consumidores mediante el almacenamiento en caché y el direccionamiento de origen óptimo, sino que también limita las llamadas redundantes a su backend.
2. **Amazon S3** aloja recursos estáticos de aplicaciones web y se distribuye de forma segura a través de CloudFront.
3. Un **grupo de usuarios de Amazon Cognito** proporciona características de administración de usuarios y proveedor de identidad para su aplicación web.
4. En muchos casos, ya que el consumidor descarga contenido estático de Amazon S3, es necesario enviar a su aplicación o que esta reciba contenido dinámico. Por ejemplo, cuando un usuario envía datos a través de un formulario, **Amazon API Gateway** actúa como punto de enlace seguro para realizar estas llamadas y devolver las respuestas que se muestran a través de su aplicación web.
5. Una función de **AWS Lambda** proporciona operaciones de creación, lectura, actualización y eliminación (CRUD) sobre DynamoDB para su aplicación web.
6. **Amazon DynamoDB** puede proporcionar el almacén de datos NoSQL backend para escalar de manera elástica con el tráfico de su aplicación web.

Notas de configuración:

- Siga las prácticas recomendadas para implementar su frontend de aplicaciones web sin servidor en AWS. Puede encontrar más información en el pilar de excelencia operativa.
- Para aplicaciones web de una sola página, utilice la consola de AWS Amplify para administrar las implementaciones atómicas, el vencimiento de la caché, el dominio personalizado y las pruebas de interfaz de usuario (IU).
- Consulte el pilar de seguridad para obtener recomendaciones sobre autenticación y autorización.
- Consulte el escenario de microservicios RESTful para obtener recomendaciones sobre el backend de aplicaciones web.
- En el caso de las aplicaciones web que ofrecen servicios personalizados, puede utilizar los planes de uso[12] de API Gateway y los grupos de usuarios de Amazon Cognito para determinar qué diferentes conjuntos de usuarios tienen acceso. Por ejemplo, un usuario premium puede tener un mayor rendimiento para llamadas a la API, acceso a API adicionales, almacenamiento adicional, etc.
- Consulte el escenario de Backend móvil si su aplicación utiliza capacidades de búsqueda que este escenario no abarca.

Los pilares del marco de buena arquitectura

Esta sección describe cada uno de los pilares e incluye definiciones, prácticas recomendadas, preguntas, consideraciones y servicios clave de AWS pertinentes a la hora de diseñar soluciones para aplicaciones sin servidor.

Por cuestiones de brevedad, solo hemos seleccionado las preguntas del marco de buena arquitectura específicas de las cargas de trabajo sin servidor. Las preguntas que no se han incluido en este documento deben tenerse en cuenta al diseñar su arquitectura. Recomendamos que lea el documento técnico del Marco de buena arquitectura de AWS.

Pilar de excelencia operativa

El pilar de **excelencia operativa** incluye la capacidad de ejecutar y monitorear sistemas para ofrecer valor empresarial y mejorar continuamente los procesos y procedimientos de soporte.

Definición

Existen tres áreas de prácticas recomendadas para la excelencia operativa en la nube:

- Preparación
- Operación
- Evolución

Además de lo que cubre el marco de buena arquitectura en relación con procesos, runbooks y días de juego, existen áreas específicas que debe observar para impulsar la excelencia operativa en aplicaciones sin servidor.

Prácticas recomendadas

Preparación
No hay prácticas operativas exclusivas para las aplicaciones sin servidor que pertenezcan a esta subsección.

Operación

> OPS 1: ¿Cuál es su entendimiento sobre el estado de la aplicación sin servidor?

Métricas y alertas

Es importante comprender las dimensiones y métricas de Amazon CloudWatch para cada servicio de AWS que vaya a utilizar a fin de que pueda establecer un plan en un lugar para evaluar su comportamiento y agregar métricas personalizadas donde considere conveniente.

Amazon CloudWatch proporciona paneles automatizados entre servicios y por servicio para ayudarlo a comprender las métricas clave de los servicios de AWS que utiliza. En el caso de las métricas personalizadas, utilice el formato de métricas incrustadas de Amazon CloudWatch para registrar un lote de métricas que CloudWatch procesará de forma asíncrona sin afectar el rendimiento de la aplicación sin servidor.

Las siguientes directrices pueden utilizarse tanto si crea un panel como si desea formular un plan para aplicaciones nuevas y existentes en relación con las métricas de:

- **Métricas empresariales**
    - Los KPI empresariales que medirán el rendimiento de su aplicación con respecto a los objetivos empresariales y que son importantes para saber cuándo algo afecta de manera crítica a su empresa en general, tanto en lo que respecta a los ingresos como a lo que no.
    - **Ejemplos**: pedidos realizados, operaciones con tarjeta de débito o crédito, vuelos adquiridos, etc.
- **Métricas de experiencia del cliente**
    - Los datos de experiencia del cliente no solo determinan la eficacia general de su UI/UX, sino también si los cambios o las anomalías afectan la experiencia del cliente en una sección determinada de su aplicación. A menudo, se miden en percentiles para evitar valores atípicos cuando se intenta comprender el impacto a lo largo del tiempo y cómo se extiende por toda la base de clientes.
    - **Ejemplos**: latencia percibida, tiempo que se tarda en agregar un elemento a una cesta o comprobarlo, tiempos de carga de la página, etc.
- **Métricas del sistema**
    - Las métricas de proveedores y de aplicaciones son importantes para respaldar las causas raíz de las secciones anteriores. También le informan si sus sistemas están en buen estado, en riesgo o si ya son sus clientes.
    - **Ejemplos**: porcentaje de errores/éxitos HTTP, utilización de memoria, duración/error/limitación controlada de la función, longitud de la cola, longitud de los registros de transmisión, latencia de integración, etc.

- **Métricas operativas**
    - Las métricas operativas son igualmente importantes para comprender la sostenibilidad y el mantenimiento de un sistema determinado y son cruciales para respaldar cómo progresó o se degradó la estabilidad a lo largo del tiempo.
    - **Ejemplos**: número de tickets (resoluciones correctas y fallidas, etc.), número de veces que se llamó a las personas disponibles, disponibilidad, estadísticas de canalización de CI/CD (implementaciones exitosas/fallidas, tiempo de retroalimentación, ciclo y tiempo de espera, etc.)

Las alarmas de CloudWatch deben configurarse tanto en niveles individuales como en niveles agregados. Un ejemplo de nivel individual es la alarma en la métrica *Duration* de Lambda o *IntegrationLatency* de API Gateway cuando se invoca a través de la API, ya que es probable que diferentes partes de la aplicación tengan perfiles diferentes. En esta instancia, puede identificar rápidamente una implementación incorrecta que hace que una función se ejecute durante mucho más tiempo del habitual.

Entre los ejemplos de nivel de agregación se incluyen alarmas, pero no se limitan a las siguientes métricas:

- **AWS Lambda**: *Duration*, *Errors*, *Throttling* y *ConcurrentExecutions*. Para las invocaciones basadas en flujos, alerta en *IteratorAge*. Para las invocaciones asíncronas, alerta sobre *DeadLetterErrors*.
- **Amazon API Gateway**: *IntegrationLatency, Latency, 5XXError*
- **Balanceador de carga de aplicaciones**: *HTTPCode_ELB_5XX_Count, RejectedConnectionCount, HTTPCode_Target_5XX_Count, UnHealthyHostCount, LambdaInternalError, LambdaUserError*
- **AWS AppSync**: 5XX y *Latency*
- **Amazon SQS:** *ApproximateAgeOfOldestMessage*
- **Amazon Kinesis Data Streams:** *ReadProvisionedThroughputExceeded, WriteProvisionedThroughputExceeded, GetRecords.IteratorAgeMilliseconds, PutRecord.Success, PutRecords.Success* (si se utiliza Kinesis Producer Library) y *GetRecords.Success*
- **Amazon SNS**: *NumberOfNotificationsFailed, NumberOfNotificationsFilteredOut-InvalidAttributes*
- **Amazon SES:** *Rejects, Bounces, Complaints, Rendering Failures*
- **AWS Step Functions:** *ExecutionThrottled, ExecutionsFailed, ExecutionsTimedOut*
- **Amazon EventBridge:** *FailedInvocations, ThrottledRules*

- **Amazon S3:** *5xxErrors, TotalRequestLatency*
- **Amazon DynamoDB:** *ReadThrottleEvents, WriteThrottleEvents, SystemErrors, ThrottledRequests, UserErrors*

Registro centralizado y estructurado

Estandarizar el registro de aplicaciones para emitir información operativa sobre transacciones, identificadores de correlación, identificadores de solicitudes entre componentes y resultados comerciales. Utilice esta información para responder a preguntas arbitrarias sobre el estado de la carga de trabajo.

A continuación, se muestra un ejemplo de un registro estructurado que utiliza JSON como salida:

```
{
   "timestamp":"2019-11-26 18:17:33,774",
   "level":"INFO",
   "location":"cancel.cancel_booking:45",
   "service":"booking",
   "lambda_function_name":"test",
   "lambda_function_memory_size":"128",
   "lambda_function_arn":"arn:aws:lambda:eu-west-1:
12345678910:function:test",
   "lambda_request_id":"52fdfc07-2182-154f-163f-5f0f9a621d72",
   "cold_start": "true",
   "message":{
      "operation":"update_item",
      "details":{
         "Attributes":{
            "status":"CANCELLED"
         },
         "ResponseMetadata":{

"RequestId":"G7S3SCFDEMEINPG6AOC6CL5IDNVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "HTTPStatusCode":200,
            "HTTPHeaders":{
               "server":"Server",
               "date":"Thu, 26 Nov 2019 18:17:33 GMT",
               "content-type":"application/x-amz-json-1.0",
               "content-length":"43",
               "connection":"keep-alive",
               "x-amzn-
requestid":"G7S3SCFDEMEINPG6AOC6CL5IDNVV4KQNSO5AEMVJF66Q9ASUAAJG",
               "x-amz-crc32":"1848747586"
            },
```

```
            "RetryAttempts":0
         }
      }
   }
}
```

El registro centralizado lo ayuda a buscar y analizar los registros las de aplicaciones sin servidor. El registro estructurado facilita la obtención de consultas para responder a preguntas arbitrarias sobre el estado de la aplicación. A medida que el sistema crezca y se incorporen más registros, debe considerar la posibilidad de utilizar los niveles de registro adecuados y un mecanismo de muestreo para registrar un pequeño porcentaje de registros en modo DEBUG.

Rastreo distribuido

Al igual que las aplicaciones sin servidor, pueden producirse anomalías a mayor escala en sistemas distribuidos. Debido a la naturaleza de las arquitecturas sin servidor, es fundamental disponer de rastreo distribuido.

Realizar cambios en su aplicación sin servidor implica muchos de los mismos principios de implementación, cambio y administración de lanzamientos que se utilizan en las cargas de trabajo tradicionales. Sin embargo, hay cambios sutiles en el uso de las herramientas existentes para lograr estos principios.

El rastreo activo con AWS X-Ray debe habilitarse para proporcionar capacidades de rastreo distribuidas, así como para habilitar mapas visuales de los servicios para una resolución de problemas más rápida. X-Ray lo ayuda a identificar la degradación del rendimiento y comprender rápidamente las anomalías, incluidas las distribuciones de latencia.

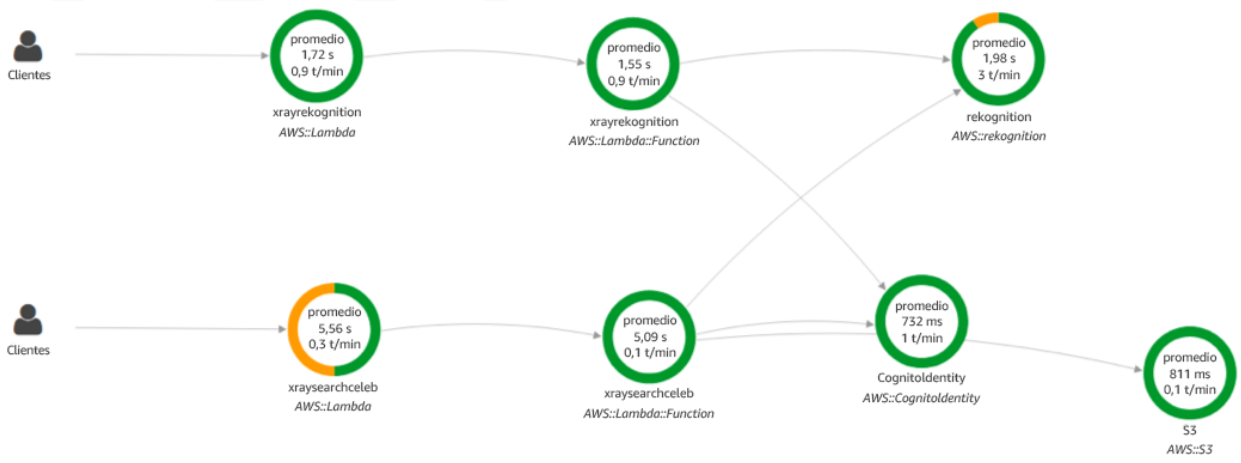

Figura 7: Mapa de servicios de AWS X-Ray que visualiza dos servicios

Los mapas de servicios son útiles para comprender los puntos de integración que necesitan atención y prácticas de resiliencia. Para las llamadas de integración, se necesitan reintentos, retardos y posiblemente interruptores para evitar que los errores se propaguen a servicios posteriores.

Otro ejemplo son las anomalías en la red. No confíe en los tiempos de espera predeterminados ni en la configuración de reintentos. En su lugar, puede configurarlos para fallar rápidamente si sucede un tiempo de espera de lectura/escritura de socket donde el tiempo predeterminado puede ser de segundos o minutos en algunos clientes.

X-Ray también ofrece dos potentes características que pueden mejorar la eficacia en la identificación de anomalías en aplicaciones: anotaciones y subsegmentos.

Los subsegmentos son útiles para comprender cómo se construye la lógica de la aplicación y con qué dependencias externas tiene que comunicarse. Las anotaciones son pares clave-valor con valores de cadena, número o booleans que AWS X-Ray indexa de manera automática.

Si se combinan, pueden ayudarlo a identificar rápidamente estadísticas de rendimiento de operaciones y transacciones empresariales específicas; por ejemplo, cuánto tiempo se tarda en consultar una base de datos o cuánto tiempo se tarda en procesar imágenes con grandes multitudes.

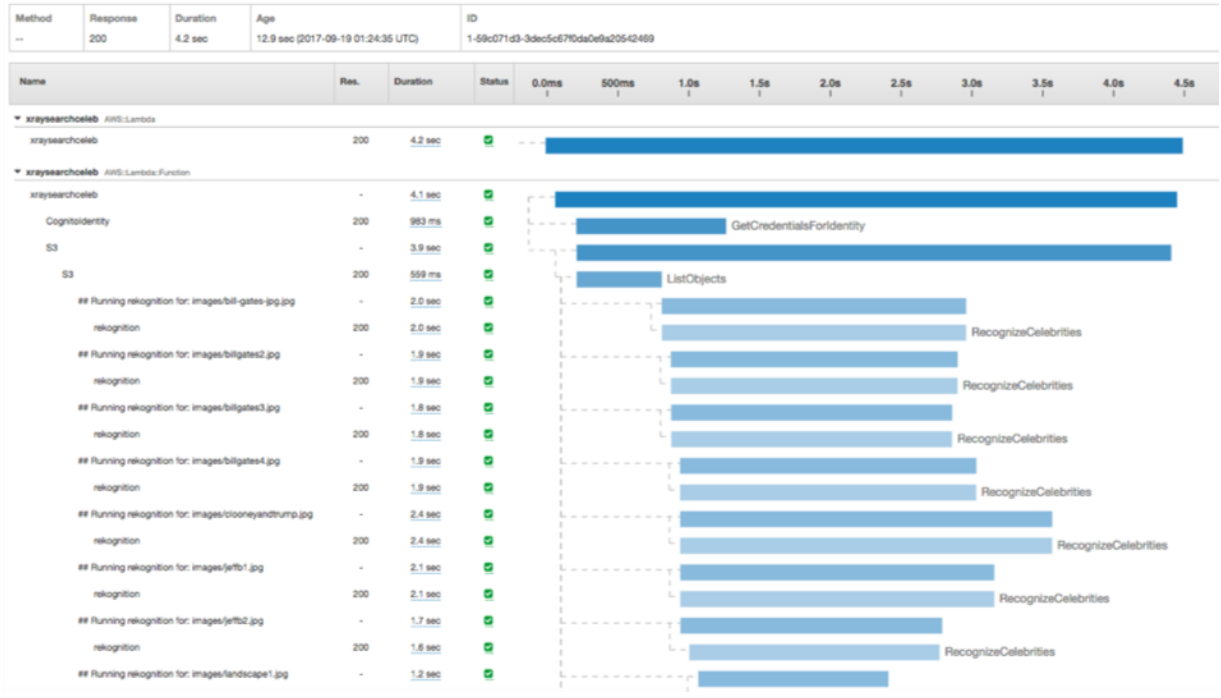

Figura 8: Rastreo de AWS X-Ray con subsegmentos que comienzan con ##

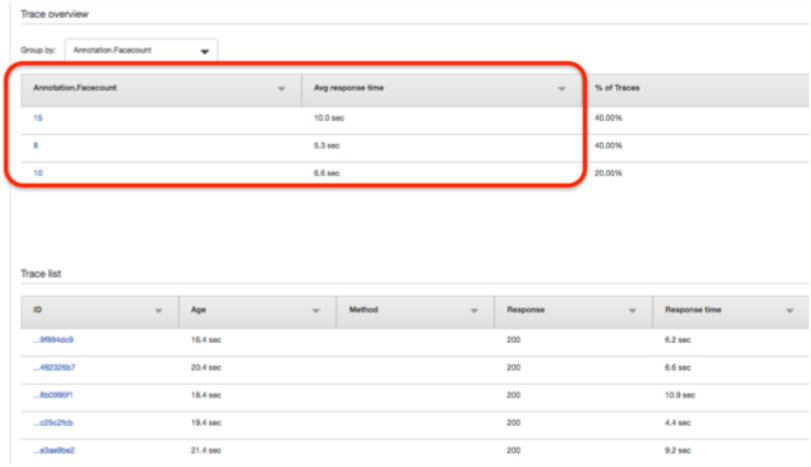

Figura 9: Rastreos de AWS X-Ray agrupados por anotaciones personalizadas

OPS 2: ¿Cómo se aborda la administración del ciclo de vida de las aplicaciones?

Creación de prototipos

Utilice la infraestructura como código para crear entornos temporales para nuevas características para las que desee crear prototipos y desglosar a medida que las completa. Puede utilizar cuentas dedicadas por equipo o por desarrollador de acuerdo con el tamaño del equipo y el nivel de automatización dentro de la organización.

Los entornos temporales permiten mayor fidelidad cuando se trabaja con servicios administrados e incrementan los niveles de control para ayudarlo a obtener la seguridad que la carga de trabajo integra y opera según lo previsto.

Para la administración de la configuración, utilice variables de entorno para los cambios poco frecuentes, como el nivel de registro y las cadenas de conexión a la base de datos. Utilice el almacén de parámetros de AWS System Manager para la configuración dinámica, como las alternativas de características y almacene información confidencial con AWS Secrets Manager.

Pruebas

Las pruebas suelen realizarse a través de pruebas unitarias, de integración y de aceptación. Desarrollar estrategias de pruebas sólidas le permite emular su aplicación sin servidor en diferentes cargas y condiciones.

Las pruebas unitarias no deben ser diferentes de las aplicaciones sin servidor y, por lo tanto, pueden ejecutarse localmente sin ningún cambio.

Las pruebas de integración no deben simular los servicios que no puede controlar, ya que pueden cambiar y proporcionar resultados inesperados. Estas pruebas se realizan mejor cuando se utilizan servicios reales, ya que pueden proporcionar el mismo entorno que utilizaría una aplicación sin servidor cuando procesa solicitudes en producción.

Las pruebas de aceptación o integrales deben realizarse sin ningún cambio, ya que el objetivo principal es simular las acciones de los usuarios finales a través de la interfaz externa disponible. Por lo tanto, no hay ninguna recomendación única que deba tener en cuenta aquí.

En general, las herramientas de Lambda y de terceros que están disponibles en AWS Marketplace pueden utilizarse como un conjunto de prueba en el contexto de las pruebas de rendimiento. A continuación, se indican algunas consideraciones que deben tenerse en cuenta durante el desarrollo de las pruebas de rendimiento:

- Las métricas como la memoria máxima invocada utilizada y la duración inicial están disponibles en CloudWatch Logs. Para obtener más información, lea la sección del pilar de rendimiento.
- Si la función de Lambda se ejecuta dentro de Amazon Virtual Private Cloud (VPC), preste atención al espacio de direcciones IP disponible dentro de su subred.
- La creación de código modularizado como funciones independientes fuera del controlador permite realizar funciones más comprobables en unidades.
- El establecimiento del código de conexión externalizado (como un grupo de conexiones a una base de datos relacional) al que se hace referencia en el código de inicialización/constructor estático de la función Lambda (es decir, el alcance global, fuera del controlador) garantizará que no se alcancen los umbrales de conexión externos si se reutiliza el entorno de ejecución de Lambda.
- Utilice la tabla bajo demanda de DynamoDB a menos que las pruebas de rendimiento superen los límites actuales de su cuenta.
- Tenga en cuenta cualquier otro límite de servicio que se pueda utilizar en la aplicación sin servidor en pruebas de rendimiento.

Implementación

Utilice la infraestructura como código y el control de versiones para permitir el seguimiento de cambios y versiones. Aísle las etapas de desarrollo y producción en entornos independientes. Esto reduce los errores que causan los procesos manuales y ayuda a aumentar los niveles de control para ayudarlo garantizar que su carga de trabajo funcione según lo previsto.

Utilice un marco sin servidor para modelar, crear prototipos, crear, empaquetar e implementar aplicaciones sin servidor, como AWS SAM o Serverless Framework. Con la infraestructura como código y un marco de trabajo, puede parametrizar su aplicación sin servidor y sus dependencias para facilitar la implementación en etapas aisladas y en cuentas de AWS.

Por ejemplo, una etapa Beta de canalización de CI/CD puede crear los siguientes recursos en una cuenta beta de AWS y para las respectivas etapas que también puede querer tener en cuentas diferentes (Gamma, Dev, Prod): *OrderAPIBeta, OrderServiceBeta, OrderStateMachineBeta, OrderBucketBeta, OrderTableBeta*.

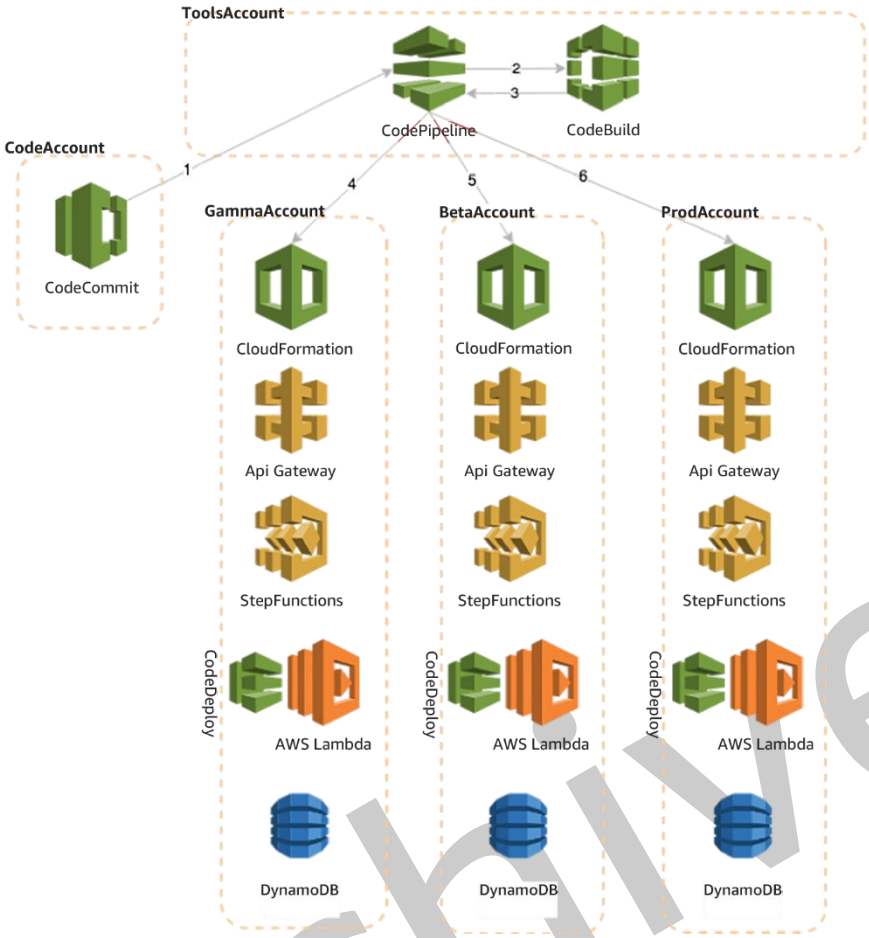

Figura 10: Canalización de CI/CD para varias cuentas

La implementación en producción favorece las implementaciones seguras en lugar de los sistemas "todo a la vez", ya que las nuevas modificaciones cambiarán gradualmente con el tiempo hacia el usuario final en una implementación "canary" o "linear". Utilice enlaces CodeDeploy (*BeforeAllowTraffic*, *AfterAllowTraffic*) y alarmas para obtener más control sobre la validación de la implementación, la restauración y cualquier personalización que necesite para su aplicación.

También puede combinar el uso de tráfico sintético, métricas personalizadas y alertas como parte de una implementación de despliegue. Esto lo ayuda a detectar errores de forma proactiva con nuevos cambios que, de lo contrario, hubieran afectado su experiencia de cliente.

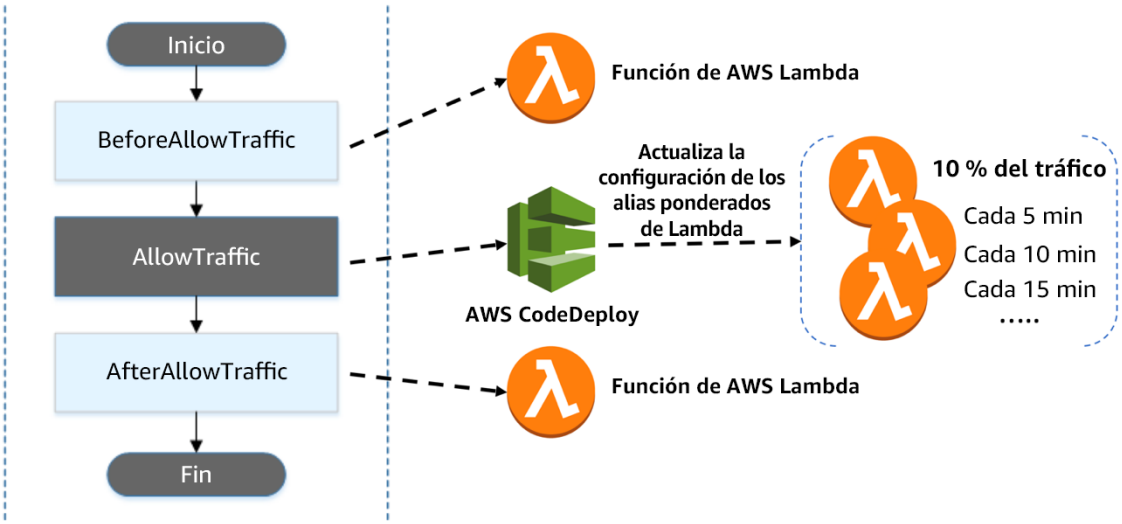

Figura 11: Implementación y enlaces de Lambda de AWS CodeDeploy

Evolución
No hay prácticas operativas exclusivas para las aplicaciones sin servidor que pertenezcan a esta subsección.

Servicios de AWS clave

Entre los servicios de AWS clave para lograr una excelencia operativa se incluyen el almacén de parámetros de AWS Systems Manager, AWS SAM, CloudWatch, AWS CodePipeline, AWS X-Ray, Lambda y API Gateway.

Recursos

Consulte los siguientes recursos para obtener más información sobre nuestras prácticas recomendadas de excelencia operativa.

Documentación y blogs

- Variables de etapa de API Gateway[13]
- Variables de entorno de Lambda[14]
- CLI DE AWS SAM[15]
- Distribución de latencia de X-Ray[16]
- Solución de problemas de aplicaciones basadas en Lambda con X-Ray[17]
- Almacén de parámetros de System Manager (SSM)[18]

- Publicación de blog sobre la implementación continua para aplicaciones sin servidor[19]
- SamFarm: ejemplo de CI/CD[20]
- Ejemplo de aplicación sin servidor con CI/CD
- Ejemplo de aplicación sin servidor que automatiza alertas y paneles
- Biblioteca de formatos de métricas incorporadas de CloudWatch para Python
- Biblioteca de formatos de métricas incorporadas de CloudWatch para Node.js
- Biblioteca de ejemplo para implementar el rastreo, el registro estructurado y las métricas personalizadas
- Límites generales de AWS
- Stackery: prácticas recomendadas para varias cuentas

Documento técnico

- Práctica de la integración continua/entrega continua en AWS[21]

Herramientas de terceros

- Página de herramientas para desarrolladores sin servidor, incluidos marcos y herramientas de terceros[22]
- Stelligent: Panel de CodePipeline para métricas operativas

Pilar de seguridad

El pilar de **seguridad** incluye la capacidad de proteger la información, los sistemas y los recursos al mismo tiempo que ofrece valor de negocio mediante evaluaciones de riesgos y estrategias de mitigación.

Definición

Existen cinco áreas de prácticas recomendadas para la seguridad en la nube:

- Identity and Access Management
- Controles de detección
- Protección de la infraestructura
- Protección de los datos
- Respuesta ante incidentes

El servicio sin servidor aborda algunos de los mayores problemas de seguridad de hoy en día, ya que elimina las tareas de administración de la infraestructura, como la aplicación de parches en el sistema operativo, la actualización de archivos binarios, etc. Aunque la superficie de ataque se reduce en comparación con las arquitecturas sin servidor, el Proyecto de seguridad de aplicaciones web abiertas (OWASP) y las prácticas recomendadas de seguridad de aplicaciones siguen aplicándose.

Las preguntas de esta sección están diseñadas para ayudarlo a abordar formas específicas en las que un intruso podría intentar obtener acceso a permisos configurados incorrectamente o explotarlos, lo que podría dar lugar a abusos. Las prácticas descritas en esta sección influyen enormemente en la seguridad de toda su plataforma en la nube, por lo que deben validarse detenidamente y revisarse con frecuencia.

La categoría de **respuesta ante incidentes** no se describirá en este documento, ya que las prácticas del marco de buena arquitectura de AWS siguen aplicándose.

Prácticas recomendadas

Identity and Access Management

SEC 1: ¿Cómo se controla el acceso a la API sin servidor?

Las API suelen ser el objetivo de los intrusos por las operaciones que pueden realizar y los valiosos datos que pueden obtener. Existen varias prácticas recomendadas de seguridad para defenderse de estos ataques.

Desde el punto de vista de la autenticación y autorización, actualmente existen cuatro mecanismos para autorizar una llamada a la API en API Gateway:

- Autorización de AWS_IAM
- Grupos de usuarios de Amazon Cognito
- Autorizador de Lambda de API Gateway
- Políticas de recursos

Principalmente, desea saber si se implementa alguno de estos mecanismos y cómo. Para los consumidores que se encuentren actualmente en su entorno de AWS o que tengan los medios necesarios para recuperar credenciales temporales de AWS Identity and Access Management (IAM) para obtener acceso a su entorno, puede utilizar la autorización de AWS_IAM y agregar permisos con privilegios mínimos al rol de IAM correspondiente para invocar a su API de forma segura.

En el siguiente diagrama se ilustra el uso de la autorización de AWS_IAM en este contexto:

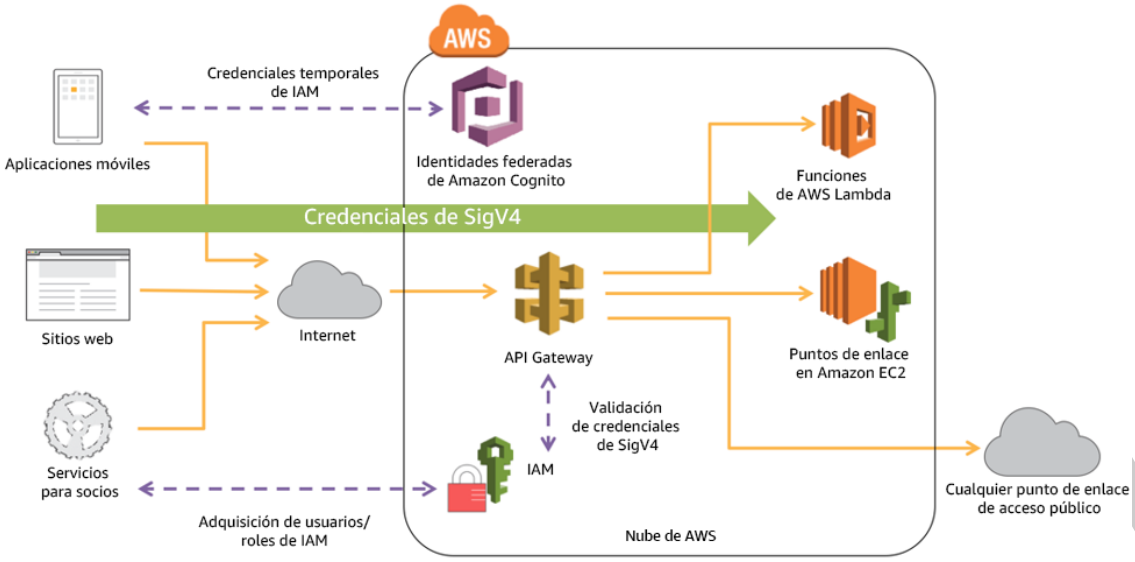

Figura 12: Autorización de AWS_IAM

Si ya dispone de un proveedor de identidad (IdP), puede utilizar un autorizador de Lambda de API Gateway para invocar una función de Lambda y autenticar o validar a un usuario determinado en su proveedor de identidad. Puede utilizar un autorizador de Lambda para la lógica de validación personalizada basada en metadatos de identidad.

Un autorizador de Lambda puede enviar información adicional derivada de un token de portador o valores de contexto de solicitud a su servicio de backend. Por ejemplo, el autorizador puede devolver un mapa que contenga ID de usuario, nombres de usuario y ámbito. Cuando utiliza autorizadores de Lambda, el backend no necesita asignar tokens de autorización a datos centrados en el usuario. Esto permite limitar la exposición de dicha información solo a la función de autorización.

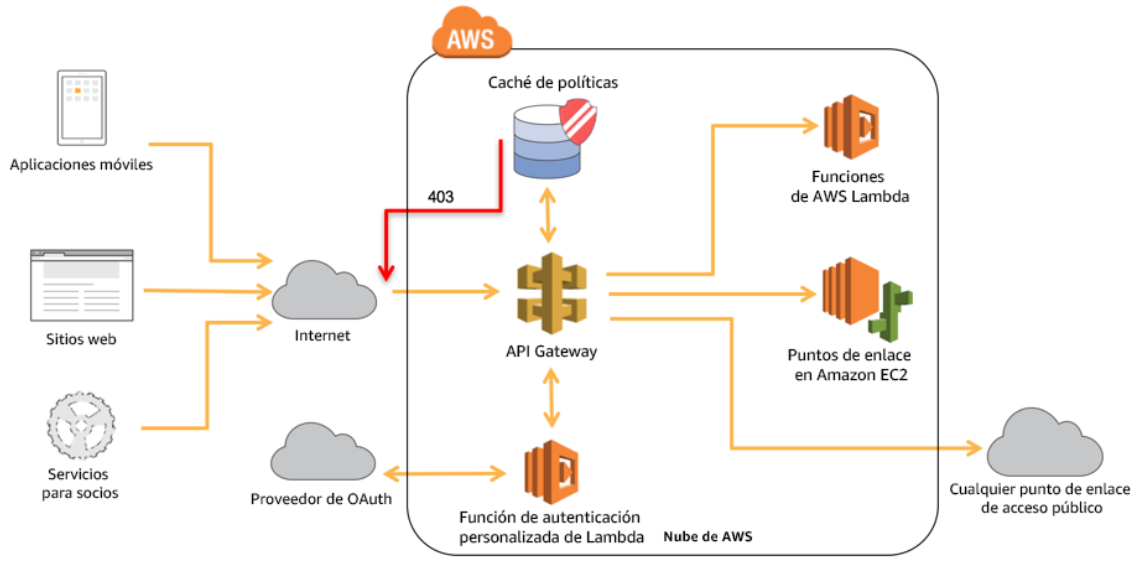

Figura 13: Autorizador de Lambda de API Gateway

Si no dispone de un proveedor de identidad, puede utilizar los grupos de usuarios de Amazon Cognito para proporcionar una administración de usuarios integrada o integrarse con proveedores de identidad externos, como Facebook, Twitter, Google+ y Amazon.

Esto se observa normalmente en el caso del backend móvil, en el que los usuarios se autentican mediante cuentas existentes en las plataformas de redes sociales, al mismo tiempo que pueden registrarse e iniciar sesión con su dirección de correo electrónico o nombre de usuario. Este enfoque también proporciona autorización granular a través de Ámbitos de OAuth.

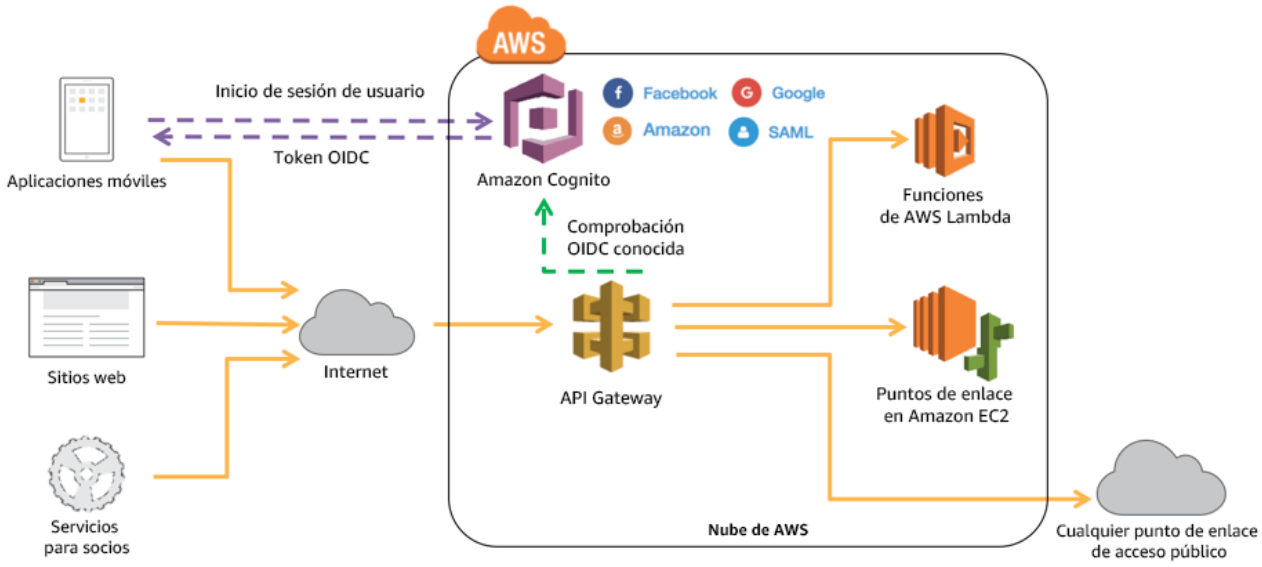

Figura 14: Grupos de usuarios de Amazon Cognito

Las claves de API de API Gateway no son un mecanismo de seguridad y no deben utilizarse para la autorización a menos que se trate de una API pública. Debe utilizarse principalmente para realizar un seguimiento del uso de un consumidor en toda la API y podría utilizarse además de los autorizadores mencionados anteriormente en esta sección.

Cuando se utilizan autorizadores de Lambda, recomendamos estrictamente que no se transfieran credenciales o cualquier tipo de información confidencial a través de encabezados o parámetros de cadenas de consulta; de lo contrario, es posible que exponga al sistema a un abuso.

Las políticas de recursos de Amazon API Gateway son documentos de políticas JSON que se pueden asociar a una API para controlar si una entidad principal de AWS especificada puede invocar la API.

Este mecanismo permite restringir las invocaciones a la API mediante:

- Usuarios de una cuenta de AWS especificada o cualquier identidad de AWS IAM
- Bloques de CIDR o rangos de direcciones IP de origen especificados
- Nubes virtuales privadas (VPC) o puntos de enlace de la VPC (en cualquier cuenta) especificados

Con las políticas de recursos, puede restringir situaciones comunes, como permitir solo solicitudes procedentes de clientes conocidos con un rango de IP específico o de otra cuenta de AWS. Si tiene previsto restringir las solicitudes procedentes de direcciones IP privadas, se recomienda utilizar puntos de enlace privados de API Gateway en su lugar.

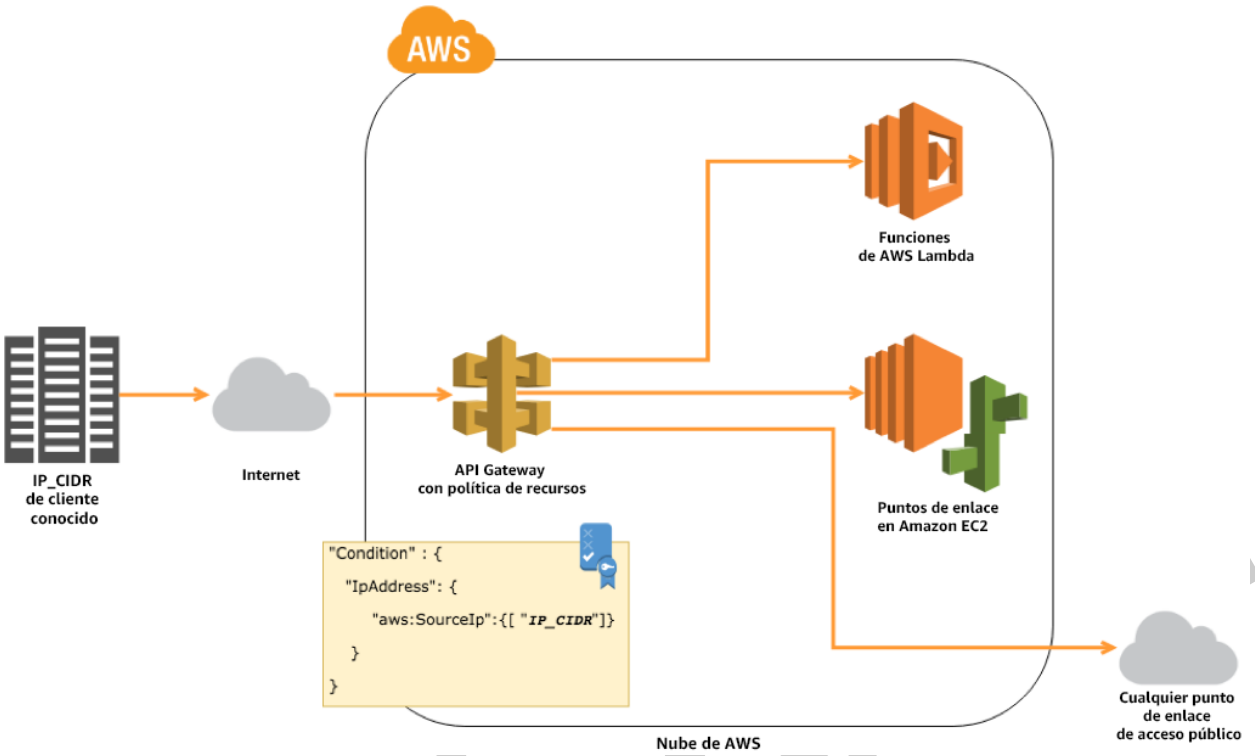

Figura 15: Política de recursos de Amazon API Gateway basada en CIDR de IP

Con los puntos de enlace privados, API Gateway restringirá el acceso a los servicios y recursos de su VPC, o a los conectados mediante Direct Connect a sus propios centros de datos.

Cuando combina puntos de enlace privados y políticas de recursos, una API puede limitarse a invocaciones de recursos específicas dentro de un rango de IP privadas específico. Esta combinación se utiliza principalmente en microservicios internos donde pueden estar en la misma cuenta o en otra cuenta.

Cuando se trata de implementaciones de gran tamaño y varias cuentas de AWS, las organizaciones pueden utilizar autorizadores de Lambda entre cuentas en API Gateway para reducir el mantenimiento y centralizar las prácticas de seguridad. Por ejemplo, API Gateway tiene la capacidad de utilizar grupos de usuarios de Amazon Cognito en una cuenta independiente. También se pueden crear y administrar autorizadores de Lambda en una cuenta independiente y luego reutilizarlos en varias API que administra API Gateway. Ambos escenarios son comunes para las implementaciones con varios microservicios que necesitan estandarizar las prácticas de autorización en las distintas API.

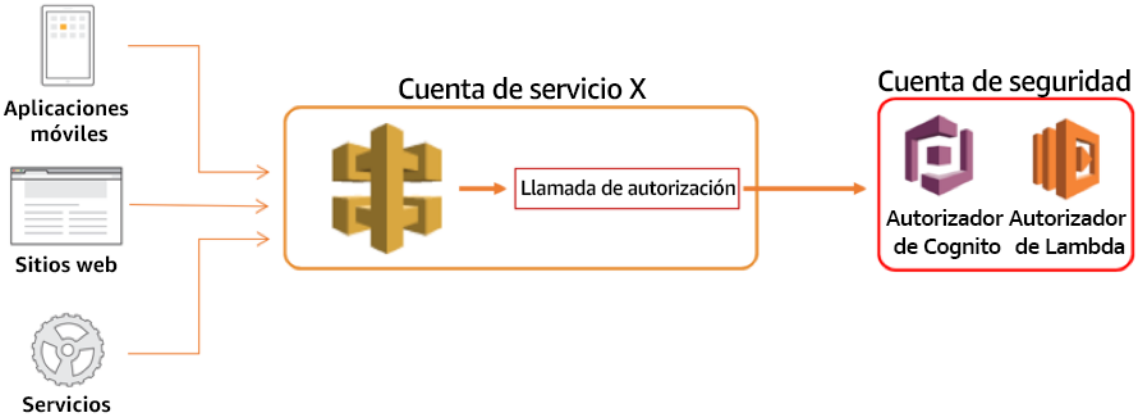

Figura 16: Autorizadores entre cuentas de API Gateway

SEC 2: ¿Cómo administra los límites de seguridad de su aplicación sin servidor?

Con las funciones de Lambda, se recomienda seguir el acceso con privilegios mínimos y permitir únicamente el acceso necesario para realizar una operación determinada. Asociar un rol con más permisos de los necesarios puede exponer a los sistemas a un abuso.

Con el contexto de seguridad, disponer de funciones más pequeñas que realizan actividades específicas contribuye a lograr una aplicación sin servidor con buena arquitectura. En cuanto a los roles de IAM, compartir un rol de IAM dentro de más de una función de Lambda probablemente infringirá el acceso con privilegios mínimos.

Controles de detección

La administración de registros es una parte importante de un diseño con buena arquitectura por razones que van desde la seguridad y el análisis forense hasta los requisitos normativos o legales.

Es igualmente importante que realice un seguimiento de las vulnerabilidades en las dependencias de la aplicación, ya que los intrusos pueden aprovechar las vulnerabilidades conocidas que se encuentran en las dependencias independientemente del lenguaje de programación que se utilice.

Para los análisis de vulnerabilidades de dependencias de aplicaciones, existen varias soluciones comerciales y de código abierto, como OWASP Dependency Check, que se pueden integrar en la canalización de CI/CD. Es importante incluir todas sus dependencias, entre las que se encuentran los SDK de AWS, como parte del repositorio de software de control de versiones.

Protección de la infraestructura

Para situaciones en las que la aplicación sin servidor necesita interactuar con otros componentes implementados en una nube virtual privada (VPC) o aplicaciones que residen en las instalaciones, es importante asegurarse de que se tienen en cuenta los límites de red.

Las funciones de Lambda se pueden configurar para obtener acceso a los recursos de una VPC. Controle el tráfico en todas las capas tal como se describe en el marco de buena arquitectura de AWS. Para cargas de trabajo que requieren filtrado de tráfico saliente por motivos de conformidad, los proxies se pueden utilizar de la misma manera que se aplican en arquitecturas que no son sin servidor.

No se recomienda aplicar límites de red únicamente en el nivel del código de la aplicación y proporcionar instrucciones sobre los recursos a los que se puede obtener acceso debido a la separación de intereses.

Para la comunicación de servicio a servicio, favorezca la autenticación dinámica, como credenciales temporales con AWS IAM en lugar de claves estáticas. API Gateway y AWS AppSync admiten la autorización de IAM, lo cual es ideal para proteger la comunicación hacia y desde los servicios de AWS.

Protección de los datos

Considere la posibilidad de habilitar los registros de acceso de API Gateway y elegir selectivamente solo lo que necesita, ya que los registros pueden contener información confidencial, en función del diseño de la aplicación sin servidor. Por este motivo, recomendamos que cifre cualquier información confidencial que atraviese su aplicación sin servidor.

API Gateway y AWS AppSync emplean TLS en todas las comunicaciones, clientes e integraciones. Aunque las cargas HTTP se cifran en tránsito, la ruta de solicitud y las cadenas de consulta que forman parte de una URL podrían no estarlo. Por lo tanto, la información confidencial se puede exponer accidentalmente a través de CloudWatch Logs si se envía a una salida estándar.

Además, la entrada incorrecta o interceptada se puede utilizar como vector de ataque, ya sea para obtener acceso a un sistema o para provocar un mal funcionamiento. La información confidencial debe protegerse en todo momento y en todas las capas posibles, tal y como se explica en detalle en el marco de buena arquitectura de AWS. Las recomendaciones de ese documento técnico siguen aplicándose aquí.

Con respecto a API Gateway, la información confidencial debe cifrarse en el lado del cliente antes de abrirse camino como parte de una solicitud HTTP o enviarse como carga útil como parte de una solicitud HTTP POST. Esto también incluye el cifrado de cualquier encabezado que pueda contener información confidencial antes de realizar una solicitud determinada.

En cuanto a las funciones de Lambda o cualquier integración con la que API Gateway pueda configurarse, la información confidencial debe cifrarse antes de cualquier procesamiento

o manipulación de datos. Esto evitará la filtración de datos si dichos datos se exponen en un almacenamiento persistente o en una salida estándar que CloudWatch Logs transmite y almacena de forma persistente.

En los escenarios descritos anteriormente en este documento, las funciones de Lambda conservarían los datos cifrados en DynamoDB, Amazon ES o Amazon S3 junto con el cifrado en reposo. Recomendamos estrictamente no enviar, registrar ni almacenar información confidencial no cifrada, ya sea como parte de las cadenas de consulta/ruta de solicitud HTTP o en la salida estándar de una función de Lambda.

Tampoco se aconseja habilitar el registro en API Gateway cuando la información confidencial no está cifrada. Como se mencionó en la subsección Controles de detección, debe consultar a su equipo de conformidad antes de habilitar el registro de API Gateway en estos casos.

SEC 3: ¿Cómo se implementa la seguridad de las aplicaciones en su carga de trabajo?

Revise los documentos de concientización de seguridad creados por los boletines de seguridad de AWS y la inteligencia de amenazas del sector, tal y como se aborda en el marco de buena arquitectura de AWS. Se siguen aplicando las directrices de OWASP para la seguridad de las aplicaciones.

Valide y sanee los eventos entrantes, y realice una revisión del código de seguridad como lo haría normalmente para aplicaciones sin servidor. Para API Gateway, configure la validación de solicitud básica como primer paso para garantizar que la solicitud se adhiera al modelo de solicitud de esquema JSON configurado, así como a cualquier parámetro requerido en URI, cadena de consulta o encabezados. Debe implementarse una validación profunda específica de la aplicación, ya sea como una función de Lambda, una biblioteca, un marco o un servicio independientes.

Almacene su información secreta, como contraseñas de base de datos o claves de API, en un administrador de información secreta que permita la rotación, la seguridad y el acceso auditado. Secrets Manager permite políticas detalladas para la información secreta, incluida la auditoría.

Servicios de AWS clave

Los servicios de AWS clave para la seguridad son Amazon Cognito, IAM, Lambda, CloudWatch Logs, AWS CloudTrail, AWS CodePipeline, Amazon S3, Amazon ES, DynamoDB y Amazon Virtual Private Cloud (Amazon VPC).

Recursos

Consulte los siguientes recursos para obtener más información sobre nuestras prácticas recomendadas en materia de seguridad.

Documentación y blogs

- Ejemplo de rol de IAM para función de Lambda con Amazon S3[23]
- Validación de solicitudes de API Gateway[24]
- Autorizadores de Lambda de API Gateway[25]
- Protección del acceso a la API con identidades federadas de Amazon Cognito, grupos de usuarios de Amazon Cognito y Amazon API Gateway[26]
- Configuración del acceso a la VPC para AWS Lambda[27]
- Filtrado del tráfico saliente de la VPC con proxies de Squid[28]
- Uso de AWS Secrets Manager con Lambda
- Auditoría de información secreta con AWS Secrets Manager
- Hoja de referencia de validación de entrada de OWASP
- Taller de seguridad sin servidor de AWS

Documentos técnicos

- Prácticas recomendadas de codificación segura de OWASP[29]
- Prácticas recomendadas de seguridad de AWS[30]

Soluciones para socios

- Seguridad sin servidor de PureSec
- Seguridad sin servidor de Twistlock[31]
- Seguridad sin servidor de Protego
- Snyk – Comprobación de dependencias y bases de datos de vulnerabilidad comercial[32]
- Uso de Hashicorp Vault con Lambda y API Gateway

Herramientas de terceros

- Comprobación de dependencias de vulnerabilidad de OWASP[33]

Pilar de confiabilidad

El pilar de **confiabilidad** incluye la capacidad de un sistema para recuperarse de las interrupciones de la infraestructura o del servicio, adquirir dinámicamente recursos informáticos para satisfacer la demanda y mitigar interrupciones como las configuraciones incorrectas o los problemas transitorios de red.

Definición

Existen tres áreas de prácticas recomendadas para la confiabilidad en la nube:

- Fundamentos
- Administración de cambios
- Administración de errores

Para ser confiable, un sistema debe disponer de una base y monitoreo bien planificados, con mecanismos para gestionar los cambios en la demanda, los requisitos o la posible defensa ante un ataque de denegación de servicio no autorizado. El sistema debe estar diseñado para detectar errores y, preferiblemente, recuperarse de forma automática.

Prácticas recomendadas

Fundamentos

REL 1: ¿Cómo se regulan las tasas de solicitudes entrantes?

Limitación controlada

En una arquitectura de microservicios, los consumidores de API pueden estar en equipos independientes o incluso fuera de la organización. Esto crea una vulnerabilidad debido a patrones de acceso desconocidos, así como el riesgo de que las credenciales del consumidor se vean comprometidas. La API de servicio puede verse afectada potencialmente si el número de solicitudes supera lo que puede gestionar la lógica de procesamiento/backend.

Además, los eventos que activan nuevas transacciones, como una actualización en una fila de base de datos o nuevos objetos que se agregan a un bucket de S3 como parte de la API, activarán ejecuciones adicionales en toda una aplicación sin servidor.

La limitación controlada debe habilitarse en el nivel de API para aplicar los patrones de acceso establecidos en un contrato de servicio. Definir una estrategia de patrón de acceso de solicitudes es fundamental para establecer cómo un consumidor debe utilizar un servicio, ya sea en el nivel de recurso o global.

Devolver los códigos de estado HTTP adecuados dentro de la API (como un 429 para la limitación controlada) ayuda a los consumidores a planificar el acceso limitado mediante la implementación de retardos y reintentos en consecuencia.

Para una limitación más granular y un uso de medición más detallado, la emisión de claves de API a los consumidores con planes de uso, además de la limitación controlada global, permite a API Gateway aplicar patrones de cuota y acceso en comportamientos inesperados.

Las claves de API también simplifican el proceso para que los administradores corten el acceso si un consumidor individual realiza solicitudes sospechosas.

Una forma común de registrar las claves de API es a través de un portal para desarrolladores. Esto le proporciona, como proveedor del servicio, metadatos adicionales asociados con los consumidores y las solicitudes. Puede registrar la aplicación, la información de contacto y el área/propósito empresarial, y almacenar estos datos en un almacén de datos duradero, como DynamoDB. Esto le ofrece una validación adicional de sus consumidores y proporciona trazabilidad de registros con identidades, para que pueda ponerse en contacto con los consumidores y resolver problemas o actualizaciones de cambios.

Tal y como se ha explicado en el pilar de seguridad, las claves de API no son un mecanismo de seguridad para autorizar solicitudes y, por lo tanto, solo deben utilizarse con una de las opciones de autorización disponibles en API Gateway.

A veces, los controles de simultaneidad son necesarios para proteger cargas de trabajo específicas frente a errores de servicio, ya que es posible que no se escalen tan rápido como Lambda. Los controles de simultaneidad permiten controlar la asignación de cuántas invocaciones simultáneas de una función de Lambda concreta se establecen en el nivel individual de función de Lambda.

Las invocaciones de Lambda que superen el conjunto de simultaneidades de una función individual se verán limitadas por el servicio de AWS Lambda y el resultado variará en función del origen de eventos. Las invocaciones síncronas devuelven un error HTTP 429, las invocaciones asíncronas se pondrán en cola y se reintentarán, mientras que los orígenes de eventos basados en transmisiones volverán a intentar hasta la fecha de vencimiento de su registro.

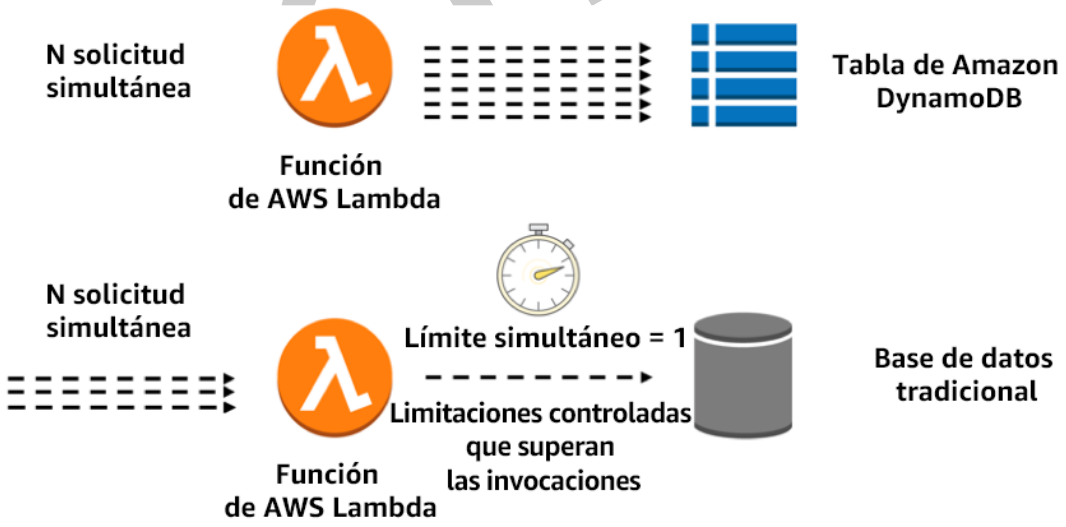

Figura 17: Controles de simultaneidad de AWS Lambda

Controlar la simultaneidad es especialmente útil para los siguientes escenarios:

- Backend confidencial o sistemas integrados que pueden tener limitaciones de escalado.
- Restricciones del grupo de conexiones de base de datos, como una base de datos relacional, que pueden imponer límites simultáneos.
- Servicios de ruta crítica: funciones de Lambda de mayor prioridad, como la autorización frente a las funciones de menor prioridad (por ejemplo, back-office) contra los límites de la misma cuenta.
- Capacidad para deshabilitar la función de Lambda (simultaneidad = 0) en caso de anomalías.
- Limitación de la simultaneidad de ejecución deseada para protegerse frente a ataques de denegación de servicio distribuido (DDoS).

Los controles de simultaneidad para las funciones de Lambda también limitan su capacidad de escalar más allá del conjunto de simultaneidades y se basa en el grupo de simultaneidad reservado de su cuenta. Para el procesamiento asíncrono, utilice Kinesis Data Streams a fin de controlar de forma eficaz la simultaneidad con una única partición en lugar del control de simultaneidad de las funciones de Lambda. Esto ofrece la flexibilidad de aumentar el número de particiones o el factor de paralelización para incrementar la simultaneidad de la función de Lambda.

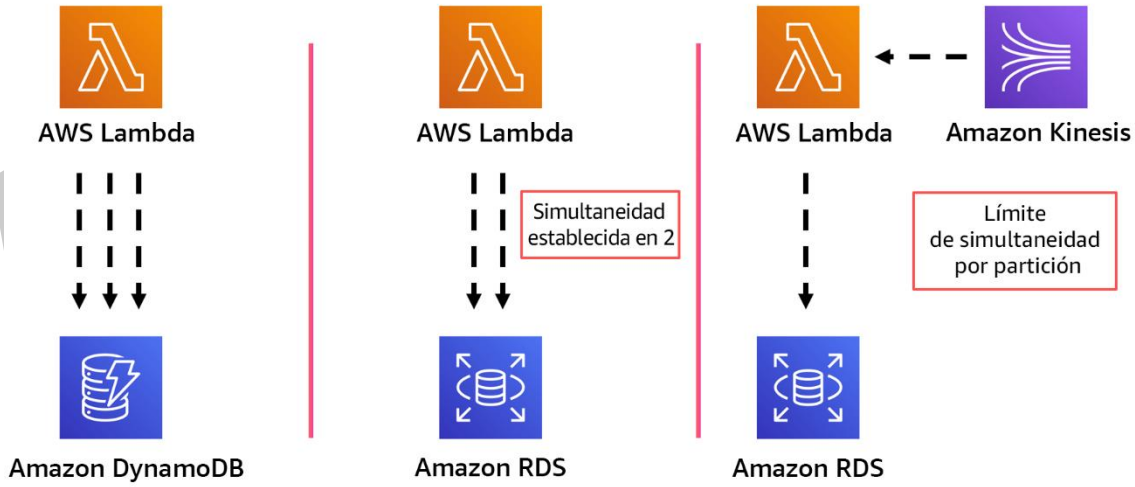

Figura 18: Controles de simultaneidad para solicitudes síncronas y asíncronas

REL 2: ¿Cómo crea resiliencia en su aplicación sin servidor?

Llamadas y eventos asíncronos
Las llamadas asíncronas reducen la latencia en las respuestas HTTP. Varias llamadas síncronas, así como ciclos de espera prolongados, pueden dar lugar a tiempos de espera y código "bloqueado" que impide la lógica de reintentos.

Las arquitecturas basadas en eventos permiten simplificar las ejecuciones asíncronas de código, lo que limita los ciclos de espera del consumidor. Estas arquitecturas suelen implementarse de forma asíncrona mediante colas, transmisiones, publicación/suscripción, webhooks, máquinas de estado y administradores de reglas de eventos en varios componentes que realizan una funcionalidad empresarial.

La experiencia del usuario se desacopla con llamadas asíncronas. En lugar de bloquear toda la experiencia hasta que se complete la ejecución general, los sistemas frontend reciben un ID de referencia/trabajo como parte de su solicitud inicial y se suscriben a cambios en tiempo real, o utilizan una API adicional para sondear su estado en sistemas heredados. Este desacoplamiento permite que el frontend sea más eficiente mediante el uso de bucles de eventos, técnicas paralelas o de simultaneidad al mismo tiempo que realiza dichas solicitudes y carga progresivamente partes de la aplicación cuando una respuesta está disponible parcial o completamente.

El frontend se convierte en un elemento clave en las llamadas asíncronas, ya que se vuelve más robusto con los reintentos personalizados y el almacenamiento en caché. Puede detener una solicitud en tránsito si no se ha recibido ninguna respuesta dentro de un SLA aceptable, ya sea debido a una anomalía, una condición transitoria, una red o entornos degradados.

Por otra parte, cuando sea necesario realizar llamadas síncronas, se recomienda, como mínimo, asegurarse de que el tiempo total de ejecución no supere el tiempo máximo de espera de la API Gateway o de la AWS AppSync. Utilice un servicio externo (por ejemplo, AWS Step Functions) para coordinar transacciones empresariales entre varios servicios, para controlar el estado y controlar la gestión de errores que se produce a lo largo del ciclo de vida de la solicitud.

Administración de cambios
Esto se aborda en el marco de buena arquitectura de AWS, y en el pilar de excelencia operativa encontrará información específica sobre las aplicaciones sin servidor.

Administración de errores
Algunas partes de una aplicación sin servidor se dictan mediante llamadas asíncronas a varios componentes en función de eventos, como publicación/suscripción y otros patrones. Cuando las llamadas asíncronas fallan, deben registrarse y reintentarse siempre que sea posible. De lo contrario, puede producirse una pérdida de datos, lo que daría como resultado una experiencia degradada para el cliente.

Para las funciones de Lambda, cree una lógica de reintento en las consultas de Lambda para garantizar que las cargas de trabajo con picos no sobrecarguen el backend. Utilice el registro estructurado tal y como se indica en el pilar de excelencia operativa para registrar reintentos, incluida la información contextual sobre errores, ya que se pueden registrar como una métrica personalizada. Utilice destinos de Lambda para enviar información contextual sobre errores, rastros de pila y reintentos en colas de mensajes fallidos (DLQ) dedicadas, como los temas de SNS y las colas de SQS. También puede desarrollar un plan para sondear mediante un mecanismo independiente para volver a dirigir estos eventos fallidos de vuelta al servicio previsto.

Los SDK de AWS proporcionan mecanismos de retardos y reintentos de forma predeterminada cuando se comunican con otros servicios de AWS que son suficientes en la mayoría de los casos. Sin embargo, debe revisarlos y ajustarlos para que se adapten a sus necesidades, especialmente los tiempos de espera de keepalive HTTP, conexión y socket.

Siempre que sea posible, utilice Step Functions para minimizar la cantidad de intentos/capturas personalizados, retardos y lógica de reintentos dentro de sus aplicaciones sin servidor. Para obtener más información, consulte la sección del pilar de optimización de costos. Utilice la integración de Step Functions para guardar las ejecuciones de estado fallidas y su estado en una DLQ.

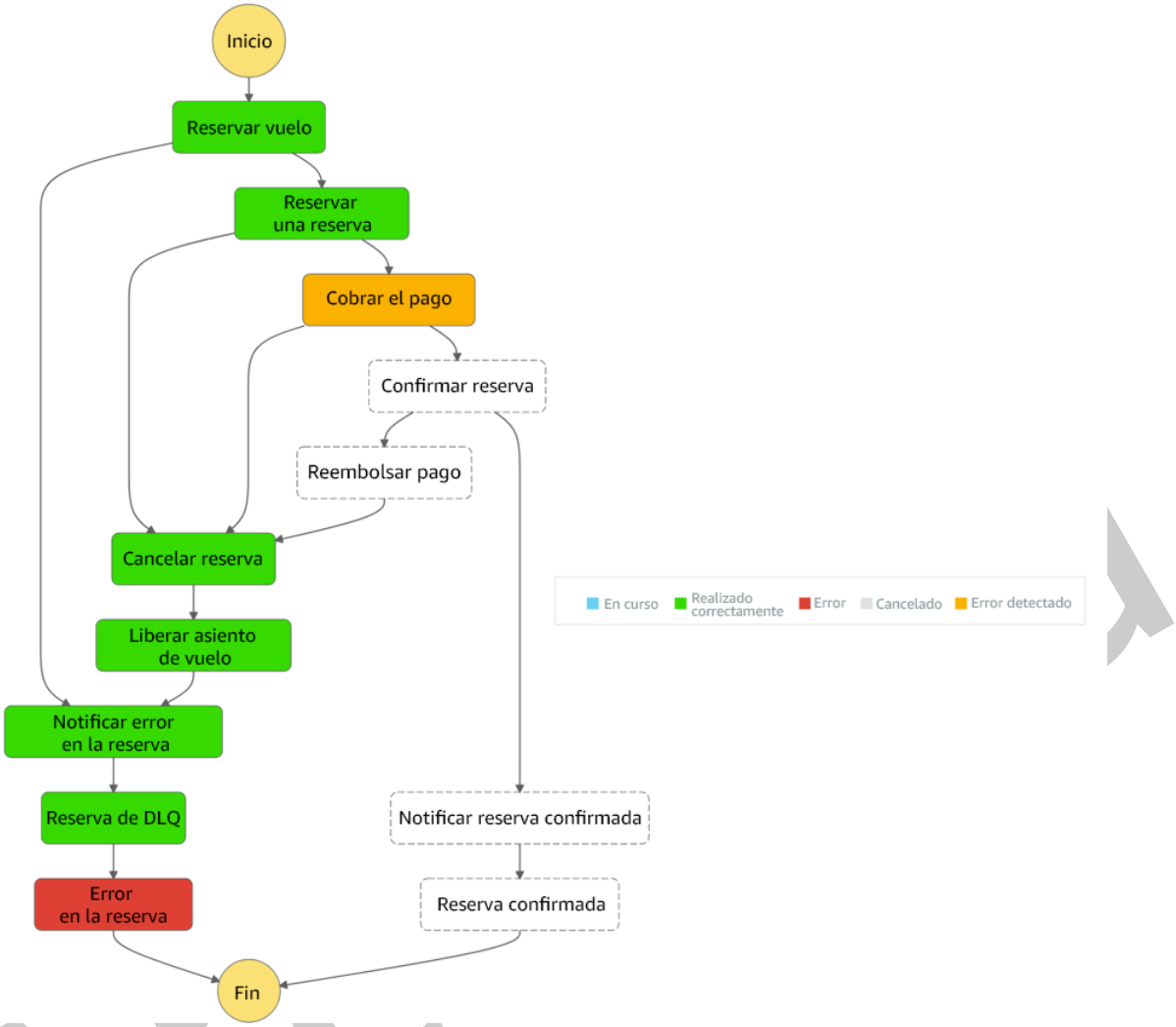

Figura 19: Máquina de estado de Step Functions con el paso de DLQ

Pueden producirse errores parciales en operaciones no atómicas, como PutRecords (Kinesis) y BatchWriteItem (DynamoDB), ya que se devuelven correctamente si al menos un registro se ha recibido correctamente. Siempre inspeccione la respuesta cuando utilice este tipo de operaciones y aborde de forma programática las fallas parciales.

Cuando consuma flujos de Kinesis o DynamoDB, utilice controles de gestión de errores de Lambda, como la antigüedad máxima del registro, los intentos máximos de reintentos, las DLQ en errores y la bisección en errores de función, para crear resiliencia adicional en su aplicación.

Para las partes síncronas basadas en transacciones y que dependen de determinadas garantías y requisitos, la anulación de las transacciones fallidas tal y como se describe en el patrón Saga[34] también se puede lograr mediante el uso de máquinas de estado de Step Functions, que desacoplarán y simplificarán la lógica de su aplicación.

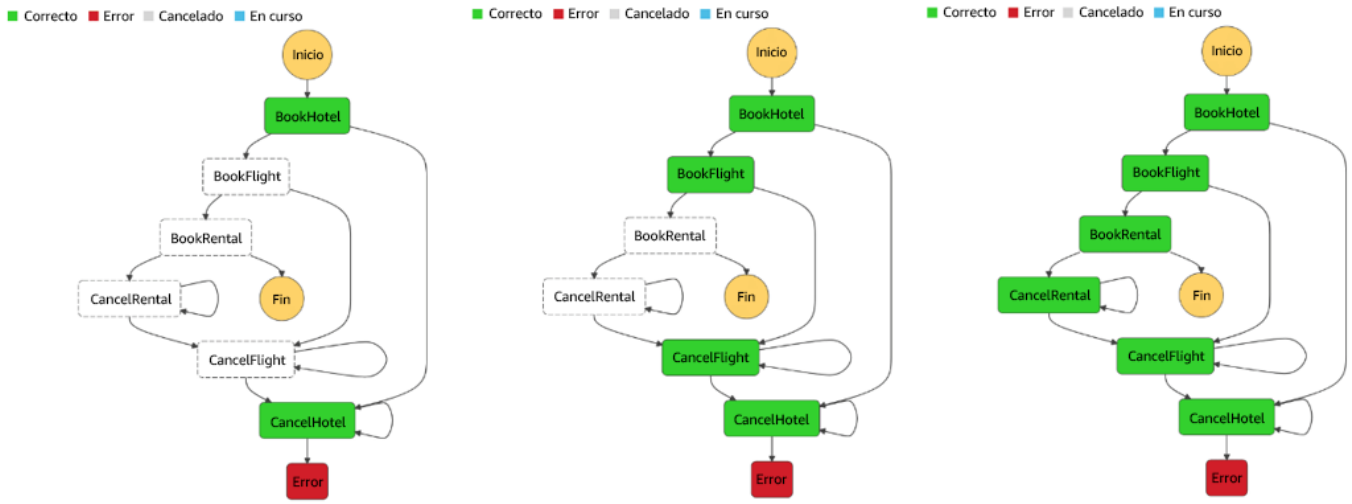

Figura 20: Patrón Saga en Step Functions de Yan Cui

Límites

Además de lo que se cubre en el marco de buena arquitectura, considere la posibilidad de revisar los límites para casos de uso de ráfagas y picos. Por ejemplo, API Gateway y Lambda tienen diferentes límites para las tasas de solicitudes estables y de ráfaga. Utilice capas de escalado y patrones asíncronos siempre que sea posible, y realice una prueba de carga para garantizar que los límites de su cuenta actual puedan admitir la demanda real de los clientes.

Servicios de AWS clave

Los servicios de AWS clave para mayor confiabilidad son AWS Marketplace, Trusted Advisor, CloudWatch Logs, CloudWatch, API Gateway, Lambda, X-Ray, Step Functions, Amazon SQS y Amazon SNS.

Recursos

Consulte los siguientes recursos para obtener más información sobre nuestras prácticas recomendadas de confiabilidad.

Documentación y blogs

- Límites en Lambda[35]
- Límites en API Gateway[36]
- Límites en Kinesis Streams[37]
- Límites en DynamoDB[38]

- Límites en Step Functions[39]
- Patrones de gestión de errores[40]
- Pruebas sin servidor con Lambda[41]
- Monitoreo de registros de funciones de Lambda[42]
- Control de versiones de Lambda[43]
- Etapas en API Gateway[44]
- Reintentos de API en AWS[45]
- Gestión de errores de Step Functions[46]
- X-Ray[47]
- DLQ de Lambda[48]
- Patrones de gestión de errores con API Gateway y Lambda[49]
- Estado de espera de Step Functions[50]
- Patrón Saga[51]
- Aplicación del patrón Saga a través de Step Functions[52]
- Aplicación de repositorio de aplicaciones sin servidor – DLQ Redriver
- Solución de problemas de reintento y de tiempo de espera con el SDK de AWS
- Controles de resiliencia de Lambda para el procesamiento de transmisiones
- Destinos de Lambda
- Aplicación de repositorio de aplicaciones sin servidor – Repetición de eventos
- Aplicación de repositorio de aplicaciones sin servidor – Almacenamiento y copia de seguridad de eventos

Documentos técnicos

- Microservicios en AWS[53]

Pilar de eficiencia del rendimiento

El pilar de **eficiencia del rendimiento** se centra en el uso eficiente de los recursos informáticos para satisfacer los requisitos y el mantenimiento de dicha eficiencia a medida que la demanda cambia y las tecnologías evolucionan.

Definición

La eficiencia del rendimiento en la nube se compone de cuatro áreas:

- Selección
- Revisión
- Monitoreo
- Compensaciones

Adopte un enfoque basado en datos para seleccionar una arquitectura de alto rendimiento. Recopile datos sobre todos los aspectos de la arquitectura, desde el diseño de alto nivel hasta la selección y configuración de los tipos de recursos. Mediante la revisión cíclica de sus opciones, se asegurará de aprovechar la continua evolución de la nube de AWS.

El monitoreo garantizará que sea consciente de cualquier desviación del rendimiento esperado y podrá tomar medidas al respecto. Por último, puede realizar compensaciones en su arquitectura para mejorar el rendimiento, tales como el uso de la compresión o el almacenamiento en caché, o mitigar los requisitos de consistencia.

PER 1: ¿Cómo ha optimizado el rendimiento de su aplicación sin servidor?

Selección

Ejecute pruebas de rendimiento en su aplicación sin servidor con tasas estables y de ráfagas. Con el resultado, pruebe ajustar las unidades de capacidad y realizar una prueba de carga después de los cambios a fin de poder seleccionar la mejor configuración:

- **Lambda:** pruebe diferentes configuraciones de memoria, ya que las IOPS de CPU, red y almacenamiento se asignan proporcionalmente.
- **API Gateway:** utilice puntos de enlace de borde para clientes dispersos geográficamente. Utilice Regional para clientes regionales y cuando utilice otros servicios de AWS dentro de la misma región.
- **DynamoDB:** utilice bajo demanda para el tráfico de aplicaciones impredecible, de lo contrario, el modo aprovisionado para el tráfico constante.
- **Kinesis:** utilice la distribución ramificada mejorada para el canal de entrada/salida especializado por consumidor en varios escenarios de consumidores. Utilice una ventana de lotes ampliada para transacciones de bajo volumen con Lambda.

Configure el acceso de VPC a las funciones de Lambda solo cuando sea necesario. Configure una gateway NAT si la función de Lambda habilitada para la VPC necesita acceso a Internet.

Tal y como se explica en el marco de buena arquitectura, configure su gateway NAT en varias zonas de disponibilidad para lograr una alta disponibilidad y rendimiento.

Las API optimizadas para límites de API Gateway proporcionan una distribución de CloudFront completamente administrada para optimizar el acceso a consumidores dispersos geográficamente. Las solicitudes de API se dirigen al punto de presencia (POP) de CloudFront más cercano, lo que normalmente mejora el tiempo de conexión.

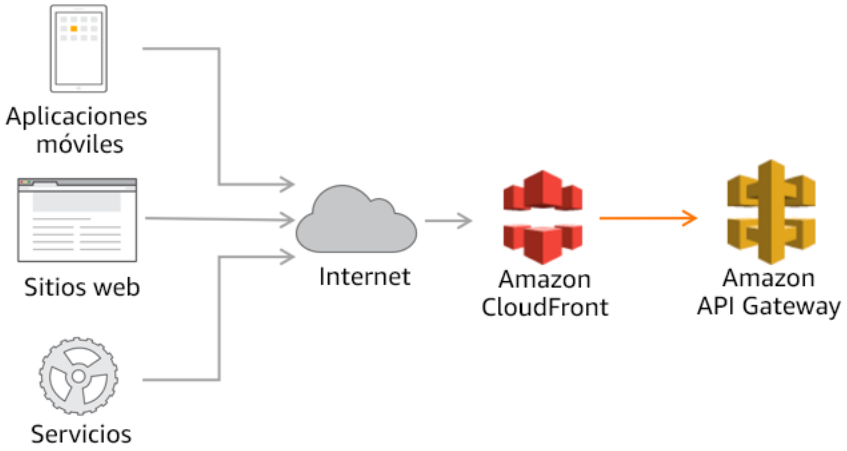

Figura 21: Implementación de API Gateway optimizada para límites

El punto de enlace regional de API Gateway no proporciona una distribución de CloudFront y habilita HTTP2 de forma predeterminada, lo que ayuda a reducir la latencia general cuando las solicitudes se originan en la misma región. Los puntos de enlace regionales también permiten asociar su propia distribución de Amazon CloudFront o una CDN existente.

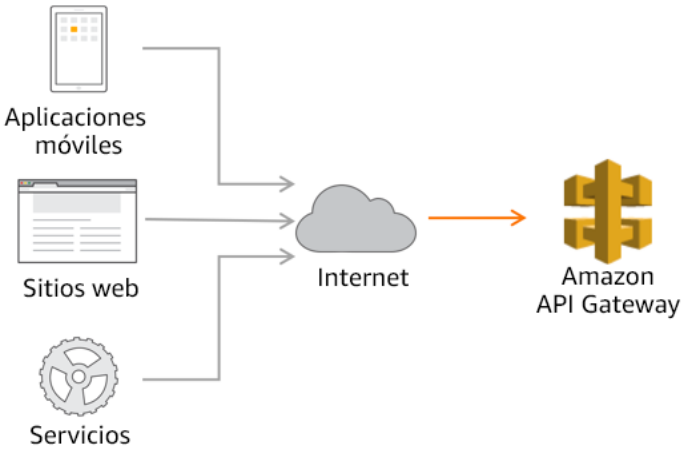

Figura 22: Implementación de API Gateway de punto de enlace regional

Esta tabla puede ayudarlo a decidir si desea implementar una API optimizada para límites o un punto de enlace de API regional:

	API optimizada para límites	Punto de enlace de API regional
El acceso a la API se realiza a través de las regiones. Incluye la distribución de CloudFront administrada por API Gateway.	X	
El acceso a la API se realiza dentro de la misma región. Menor latencia de solicitud cuando se accede a la API desde la misma región en la que esta se implementa.		X
Capacidad para asociar su propia distribución de CloudFront.		X

Este árbol de decisiones puede ayudarlo a decidir cuándo implementar la función de Lambda en una VPC.

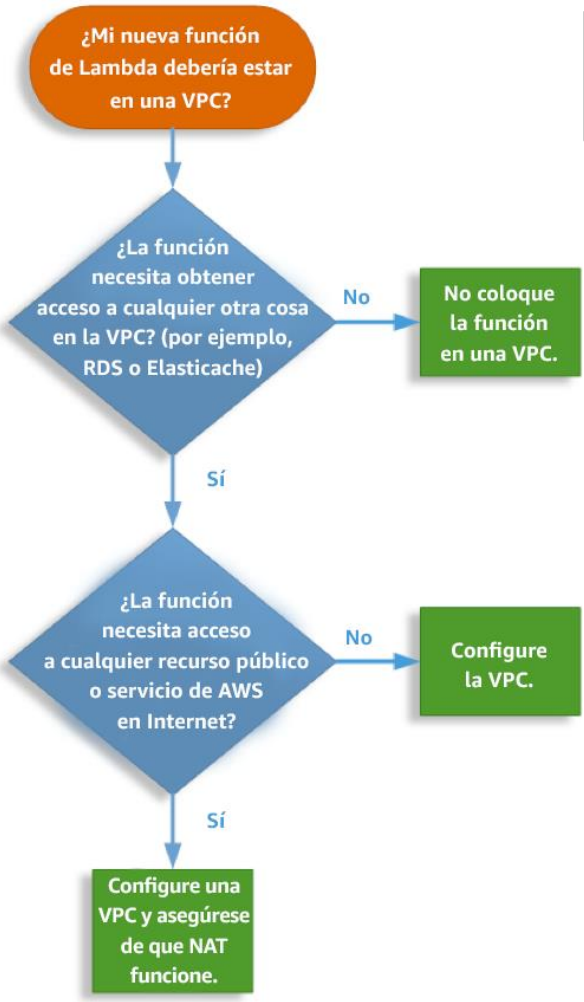

Figura 23: Árbol de decisiones para implementar una función de Lambda en una VPC

Optimice

A medida que una arquitectura sin servidor crece de forma orgánica, existen ciertos mecanismos que se utilizan habitualmente en una variedad de perfiles de carga de trabajo. A pesar de las pruebas de rendimiento, las compensaciones de diseño deben considerarse para aumentar el rendimiento de su aplicación, sin olvidar su SLA y sus requisitos.

Se puede habilitar el almacenamiento en caché de API Gateway y AWS AppSync para mejorar el rendimiento de las operaciones aplicables. DAX puede mejorar significativamente las respuestas de lectura, así como los índices secundarios locales y globales para evitar las operaciones de análisis de tablas completas de DynamoDB. Estos detalles y recursos se describieron en el escenario del backend móvil.

La codificación de contenido de API Gateway permite a los clientes de API solicitar la compresión de la carga antes de enviarla de nuevo en respuesta a una solicitud de API. Esto reduce el número de bytes que se envían desde API Gateway a los clientes de API y reduce el tiempo que se tarda en transferir los datos. Puede habilitar la codificación de contenido en la definición de la API y también puede establecer el tamaño de respuesta mínimo que activa la compresión. De forma predeterminada, las API no tienen habilitada la compatibilidad con la codificación de contenido.

Establezca el tiempo de espera de la función unos segundos más alto que el promedio de ejecución para tener en cuenta cualquier problema transitorio en los servicios posteriores que se utilizaron en la ruta de comunicación. Esto también se aplica cuando se trabaja con actividades, tareas y visibilidad de mensajes de SQS de Step Functions.

Elegir una configuración de memoria predeterminada y un tiempo de espera en AWS Lambda puede tener un efecto no deseado en el rendimiento, el costo y los procedimientos operativos.

Establecer un tiempo de espera mucho más alto que el promedio de ejecución puede provocar que las funciones se ejecuten durante más tiempo si el código no funciona correctamente. Esto da como resultado costos más elevados y el posible alcance de límites de simultaneidad en función de cómo se invoquen dichas funciones.

Si se establece un tiempo de espera que equivale a una ejecución satisfactoria de la función, una aplicación sin servidor podría detener bruscamente una ejecución en caso de que se produzca un problema transitorio de red o una anomalía en los servicios posteriores.

Establecer un tiempo de espera sin realizar pruebas de carga y, lo que es más importante, sin tener en cuenta los servicios ascendentes puede dar lugar a errores siempre que cualquier parte alcance su tiempo de espera en primer lugar.

Siga las prácticas recomendadas para trabajar con funciones de Lambda[54], tales como la reutilización de contenedores, la minimización del tamaño del paquete de implementación en función de sus necesidades de tiempo de ejecución y la minimización de la complejidad de sus dependencias, incluidos los marcos que podrían no estar optimizados para un inicio rápido. El percentil 99 de latencia (P99) debe tenerse en cuenta siempre, ya que uno no puede afectar el SLA de la aplicación acordado con otros equipos.

Para las funciones de Lambda en la VPC, evite la resolución de DNS de nombres de host públicos de recursos subyacentes en su VPC. Por ejemplo, si la función de Lambda obtiene acceso a una instancia de base de datos de Amazon RDS en su VPC, lance la instancia con la opción no accesible públicamente.

Después de que se haya ejecutado una función de Lambda, AWS Lambda mantiene el contexto de ejecución durante un tiempo arbitrario y se anticipa a otra invocación de la función de Lambda. Esto permite utilizar el ámbito global para operaciones costosas y excluidas, como, por ejemplo, el establecimiento de una conexión a la base de datos o cualquier lógica de inicialización. En las invocaciones posteriores, puede verificar si sigue siendo válida y reutilizar la conexión existente.

Transacciones asíncronas

Dado que sus clientes esperan interfaces de usuario más modernas e interactivas, ya no podrá mantener flujos de trabajo complejos mediante transacciones síncronas. Cuanta más interacción de servicio necesite, más terminará encadenando llamadas, lo que podría aumentar el riesgo de estabilidad del servicio, así como el tiempo de respuesta.

Los marcos de trabajo modernos de IU, como Angular.js, VueJS y React, las transacciones asíncronas y los flujos de trabajo nativos en la nube proporcionan un enfoque sostenible para satisfacer la demanda de los clientes, además de ayudarlo a desacoplar componentes y centrarse en los dominios de procesos y negocios.

Estas transacciones asíncronas (o, a menudo, descritas como una arquitectura controlada por eventos) inician eventos coreográficos posteriores en la nube en lugar de obligar a los clientes a bloquear y esperar (bloqueo de E/S) para obtener una respuesta. Los flujos de trabajo asíncronos gestionan una variedad de casos de uso, incluidos, entre otros: la entrada de datos, las operaciones de ETL y el cumplimiento de pedidos/solicitudes.

En estos casos de uso, los datos se procesan a medida que llegan y se recuperan a medida que cambian. Describimos las prácticas recomendadas para dos flujos de trabajo asíncronos comunes donde puede aprender algunos patrones de optimización para la integración y el procesamiento asíncrono.

Procesamiento de datos sin servidor

En un flujo de trabajo de procesamiento de datos sin servidor, los datos se introducen desde los clientes en Kinesis (mediante el agente de Kinesis, el SDK o la API) y llegan a Amazon S3.

Los nuevos objetos inician una función de Lambda que se ejecuta automáticamente. Esta función se suele utilizar para transformar o particionar datos para su posterior procesamiento y posiblemente se almacenan en otros destinos, como DynamoDB u otro bucket de S3 en el que los datos se encuentran en su formato final.

Dado que usted puede tener diferentes transformaciones para diferentes tipos de datos, recomendamos dividir minuciosamente las transformaciones en diferentes funciones de

Lambda para lograr un rendimiento óptimo. Con este enfoque, dispone de la flexibilidad de ejecutar la transformación de datos en paralelo y aumentar la velocidad y mejorar los costos.

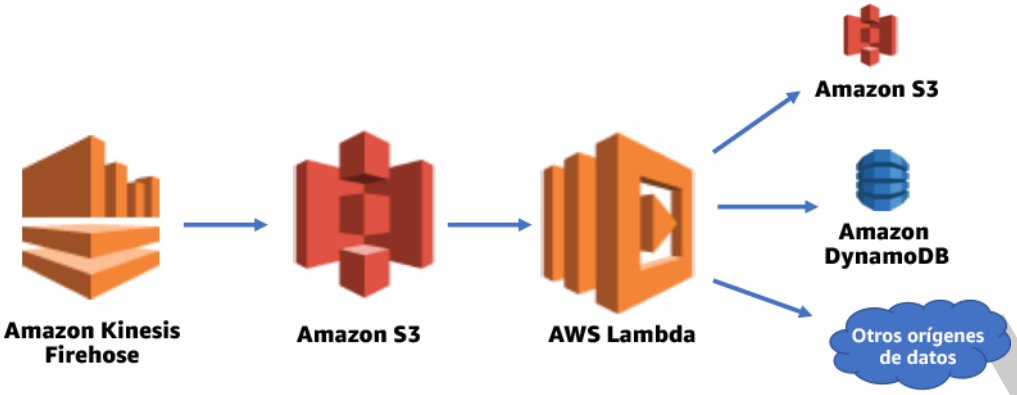

Figura 24: Entrada asincrónica de datos

Kinesis Data Firehose ofrece transformaciones de datos nativas que se pueden utilizar como una alternativa a Lambda, donde no se necesita lógica adicional para transformar registros en registros de Apache Log/System a CSV, JSON, JSON a Parquet u ORC.

Envío de eventos sin servidor con actualizaciones de estado

Supongamos que tiene un sitio de comercio electrónico y un cliente envía un pedido que inicia un proceso de deducción y envío de inventario; o una aplicación empresarial que envía una consulta grande que puede tardar minutos en responder.

Es posible que los procesos necesarios para completar esta transacción común requieran varias llamadas al servicio, lo que puede tardar un par de minutos en completarse. Dentro de esas llamadas, desea protegerse frente a posibles errores al agregar reintentos y retardos exponenciales. Sin embargo, esto puede provocar una experiencia de usuario poco óptima para quien espera que se complete la transacción.

Para flujos de trabajo largos y complejos similares a este, puede integrar API Gateway o AWS AppSync con Step Functions que, tras nuevas solicitudes autorizadas, iniciará este flujo de trabajo empresarial. Step Functions responde inmediatamente con un ID de ejecución al intermediario (aplicación móvil, SDK, servicio web, etc.).

En el caso de sistemas heredados, puede utilizar el ID de ejecución para sondear Step Functions y ver el estado del flujo de trabajo empresarial a través de otra API REST. Con WebSockets, independientemente de si utiliza REST o GraphQL, puede recibir el estado del flujo de trabajo empresarial en tiempo si proporciona actualizaciones en cada paso del flujo de trabajo.

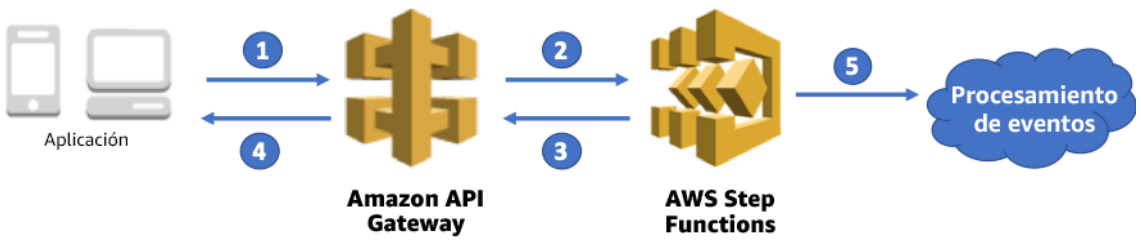

Figura 25: Flujo de trabajo asíncrono con máquinas de estado de Step Functions

Otra situación habitual es la integración de API Gateway directamente con SQS o Kinesis como capa de escalado. Una función de Lambda solo sería necesaria si se espera información empresarial adicional o un formato de ID de solicitud personalizado del intermediario.

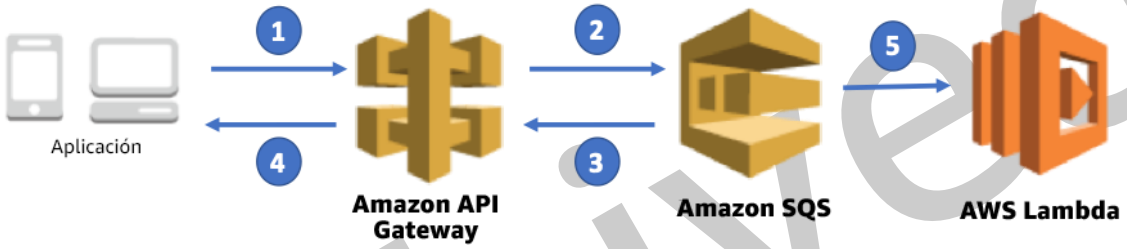

Figura 26: Uso de una cola como capa de escalado para el flujo de trabajo asíncrono

En este segundo ejemplo, SQS tiene varios fines:

1. Almacenar el registro de solicitud de forma duradera es importante porque el cliente puede continuar con confianza durante todo el flujo de trabajo con la seguridad de que la solicitud se procesará eventualmente.
2. Tras una ráfaga de eventos que pueden sobrecargar temporalmente el backend, se puede sondear la solicitud para su procesamiento cuando los recursos estén disponibles.

En comparación con el primer ejemplo sin una cola, Step Functions almacena los datos de forma duradera sin la necesidad de una cola u orígenes de datos de seguimiento de estado. En ambos ejemplos, la práctica recomendada consiste en buscar un flujo de trabajo asíncrono después de que el cliente envíe la solicitud y evitar la respuesta resultante como código de bloqueo si la finalización tardara varios minutos.

Con WebSockets, AWS AppSync proporciona esta capacidad de manera inmediata a través de suscripciones de GraphQL. Con las suscripciones, un cliente autorizado podría escuchar las mutaciones de datos que le interesen. Esto es ideal para los datos que se transmiten o que pueden producir más de una única respuesta.

Con AWS AppSync, a medida que las actualizaciones de estado cambian en DynamoDB, los clientes pueden suscribirse y recibir actualizaciones automáticamente a medida que se producen, y es el patrón perfecto para cuando los datos se manejan en la interfaz de usuario.

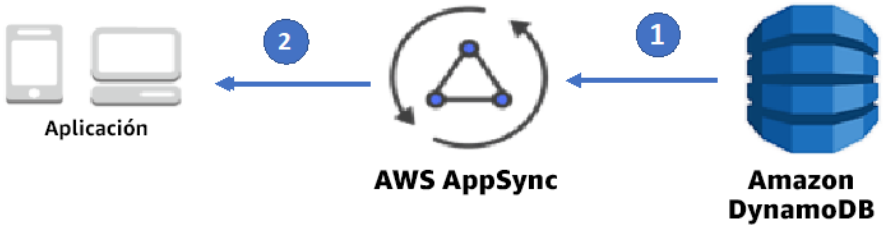

Figura 27: Actualizaciones asíncronas a través de WebSockets con AWS AppSync y GraphQL

Los webhooks se pueden implementar con suscripciones HTTP de tema SNS. Los consumidores pueden alojar un punto de enlace HTTP al que SNS volverá a llamar a través de un método POST en caso de que se produzca un evento (por ejemplo, un archivo de datos que llegue a Amazon S3). Este patrón es ideal cuando los clientes se pueden configurar, como otro microservicio, que podría alojar un punto de enlace. De forma alternativa, Step Functions admite devoluciones de llamada en las que una máquina de estado se bloqueará hasta que reciba una respuesta para una tarea determinada.

Figura 28: Notificación asíncrona a través de Webhook con SNS

Por último, el sondeo podría ser costoso desde el punto de vista tanto del costo como del recurso, ya que varios clientes sondean constantemente una API para comprobar el estado. Si el sondeo es la única opción debido a restricciones de entorno, se recomienda establecer SLA con los clientes para limitar el número de “sondeos vacíos”.

Figura 29: Sondeo del cliente para actualizaciones de transacciones realizadas recientemente

Por ejemplo, si una consulta de almacenamiento de datos grande tarda un promedio de dos minutos para obtener una respuesta, el cliente debe sondear la API después de dos minutos con retardo exponencial si los datos no están disponibles. Existen dos patrones comunes para garantizar que los clientes no sondean con más frecuencia de la esperada: la limitación controlada y la marca temporal para cuando es seguro volver a sondear.

Para las marcas temporales, el sistema que se sondea puede devolver un campo adicional con una marca temporal o un periodo de tiempo en el que es seguro para el consumidor volver a sondear. Este enfoque sigue un escenario optimista en el que el consumidor respetará y utilizará esto sabiamente y, en caso de abuso, también podrá aplicar una limitación controlada para una implementación más completa.

Revisión

Consulte el documento técnico del marco de buena arquitectura de AWS para conocer las prácticas recomendadas en el área de **revisión** sobre la eficiencia del rendimiento que se aplica a las aplicaciones sin servidor.

Monitoreo

Consulte el documento técnico del marco de buena arquitectura de AWS para conocer las prácticas recomendadas en el área de **monitoreo** sobre la eficiencia del rendimiento que se aplica a las aplicaciones sin servidor.

Compensaciones

Consulte el documento técnico del marco de buena arquitectura de AWS para conocer las prácticas recomendadas en el área de **compensaciones** sobre la eficiencia del rendimiento que se aplica a las aplicaciones sin servidor.

Servicios de AWS clave

Los servicios de AWS clave para mejorar el rendimiento son DynamoDB Accelerator, API Gateway, Step Functions, Gateway NAT, Amazon VPC y Lambda.

Recursos

Consulte los siguientes recursos para obtener más información sobre nuestras prácticas recomendadas en la eficiencia del rendimiento.

Documentación y blogs

- Preguntas frecuentes sobre AWS Lambda[55]
- Prácticas recomendadas para trabajar con funciones de AWS Lambda[56]
- AWS Lambda: Funcionamiento[57]
- Comprensión de la reutilización de contenedores en AWS Lambda[58]
- Configuración de una función de Lambda para obtener acceso a los recursos en una VPC de Amazon[59]
- Habilitación del almacenamiento en caché de la API para mejorar la capacidad de respuesta[60]
- DynamoDB: Índices secundarios globales[61]
- Amazon DynamoDB Accelerator (DAX)[62]
- Guía para desarrolladores: Kinesis Streams[63]
- SDK de Java: Configuración de la mejora del rendimiento
- SDK de Node.js: Habilitación de HTTP Keep Alive
- SDK de Node.js: Mejora de las importaciones
- Uso de colas de Amazon SQS y AWS Lambda para lograr un rendimiento alto
- Aumento del rendimiento de procesamiento de transmisiones con distribución ramificada mejorada
- Ajuste de potencia de Lambda
- Cuándo utilizar el modo aprovisionado y bajo demanda de Amazon DynamoDB
- Análisis de datos de registro con Amazon CloudWatch Logs Insights
- Integración de varios orígenes de datos con AWS AppSync
- Integraciones de servicios de Step Functions
- Patrones de almacenamiento en caché
- Almacenamiento en caché de aplicaciones sin servidor
- Prácticas recomendadas para Amazon Athena y AWS Glue

Pilar de optimización de costos

El pilar de **optimización de costos** incluye el proceso continuo de refinamiento y mejora de un sistema durante todo su ciclo de vida. Desde el diseño inicial de su primera prueba de concepto hasta el funcionamiento continuo de las cargas de trabajo de producción, la adopción de las prácticas de este documento le permitirá crear y operar sistemas con control de costos que logren resultados empresariales y minimicen costos, lo que permitirá a su negocio maximizar el retorno de la inversión.

Definición

Existen cuatro áreas de prácticas recomendadas para la optimización de costos en la nube:

- Recursos rentables
- Coincidencia de la oferta y la demanda
- Conciencia de gastos
- Optimización a lo largo del tiempo

Al igual que ocurre con los demás pilares, hay alternativas que deberá considerar. Por ejemplo, ¿desea optimizar la velocidad de comercialización o el costo? En algunos casos, es mejor optimizar la velocidad: salda rápida al mercado, envío de nuevas características o simplemente el cumplimiento de una fecha límite en lugar dela inversión en la optimización de costos iniciales.

Las decisiones de diseño a veces se guían por la prisa en lugar de los datos empíricos, ya que siempre existe la tentación de compensar en exceso “por si acaso” en lugar de dedicar tiempo en realizar un análisis comparativo para la implementación más rentable.

Esto suele dar lugar a implementaciones sobreaprovisionadas drásticamente y poco optimizadas. En las secciones siguientes se proporcionan técnicas y orientación estratégica para la optimización de costos inicial y continua de su implementación.

Por lo general, las arquitecturas sin servidor tienden a reducir costos porque algunos de los servicios, como AWS Lambda, no cuestan nada mientras están inactivos. Sin embargo, seguir determinadas prácticas recomendadas y evaluar las alternativas lo ayudará a reducir aún más el costo de estas soluciones.

Prácticas recomendadas

COSTO 1: ¿Cómo se optimizan los costos?

Recursos rentables
Las arquitecturas sin servidor son más fáciles de administrar en términos de asignación correcta de recursos. Debido a su modelo de precios "pago de acuerdo al valor" y su escala en función de la demanda, el servicio sin servidor reduce de forma efectiva el esfuerzo de planificación de la capacidad.

Tal y como se explica en los pilares de excelencia operativa y rendimiento, la optimización de su aplicación sin servidor tiene un impacto directo en el valor que produce y en su costo.

A medida que Lambda asigna proporcionalmente IOPS de CPU, red y almacenamiento en función de la memoria, cuanto más rápida sea la ejecución, más económico y más valor producirá la función. Esto se debe principalmente a la dimensión incremental de facturación de 100 ms.

Coincidencia de la oferta y la demanda
La arquitectura sin servidor de AWS está diseñada para escalar en función de la demanda y, por tanto, no hay prácticas aplicables a seguir.

Conciencia de gastos
Tal y como se explica en el marco de buena arquitectura de AWS, la mayor flexibilidad y agilidad que permite la nube fomenta la innovación, además del desarrollo y la implementación rápidos. Elimina los procesos manuales y el tiempo asociados con el aprovisionamiento de la infraestructura en las instalaciones, incluida la identificación de especificaciones de hardware, la negociación de presupuestos de precios, la administración de órdenes de compra, la programación de envíos y la posterior implementación de los recursos.

A medida que su arquitectura sin servidor crece, se multiplicará el número de funciones de Lambda, API, etapas y otros recursos. La mayoría de estas arquitecturas deben presupuestarse y preverse en términos de costos y administración de recursos. El etiquetado puede ayudarlo aquí. Puede asignar costos de su factura de AWS a funciones y API individuales y obtener una vista detallada de los costos por proyecto en AWS Cost Explorer.

Una buena implementación consiste en compartir la misma etiqueta de valor clave para los recursos que pertenecen al proyecto mediante programación y crear informes personalizados de acuerdo con las etiquetas que usted creó. Esta característica lo ayudará no solo a asignar los costos, sino también a identificar qué recursos pertenecen a cada proyecto.

Optimización a lo largo del tiempo
Consulte el documento técnico del marco de buena arquitectura de AWS para conocer las prácticas recomendadas en el área de **Optimización a lo largo del tiempo** y obtener información sobre la optimización de costos que se aplica a las aplicaciones sin servidor.

Registro de incorporación y almacenamiento
AWS Lambda utiliza CloudWatch Logs para almacenar la salida de las ejecuciones con el fin de identificar y solucionar problemas en las ejecuciones y de monitorear la aplicación sin servidor. Esto afectará al costo del servicio de CloudWatch Logs en dos dimensiones: incorporación y almacenamiento.

Establezca los niveles de registro adecuados y elimine la información de registro innecesaria para optimizar la incorporación de registros. Utilice variables de entorno para controlar el nivel de registro de aplicaciones y el registro de muestra en el modo DEBUG a fin de garantizar que dispone de información adicional cuando sea necesario.

Establezca períodos de retención de registros para grupos de CloudWatch Logs nuevos y existentes. Para archivar registros, exporte y defina las clases de almacenamiento rentables que mejor se adapten a sus necesidades.

Integraciones directas
Si la función de Lambda no realiza una lógica personalizada mientras se integra con otros servicios de AWS, lo más probable es que sea innecesaria.

Los destinos de API Gateway, AWS AppSync, Step Functions, EventBridge y Lambda se pueden integrar directamente con una serie de servicios, y proporcionarle más valor y menos gastos generales operativos.

La mayoría de las aplicaciones sin servidor públicas ofrecen una API con una implementación independiente del contrato correspondiente, tal y como se describe en el Escenario de microservicios.

Un escenario de ejemplo en el que una integración directa es una mejor opción es la incorporación de datos de secuencias de clics a través de una API REST.

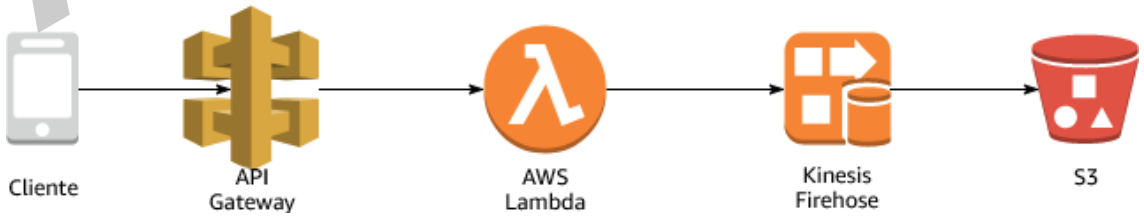

Figura 30: Envío de datos a Amazon S3 mediante Kinesis Data Firehose

En este caso, API Gateway ejecutará una función de Lambda que simplemente incorporará el registro entrante en Kinesis Data Firehose. Este, posteriormente, agrupará los registros antes de almacenarlos en un bucket de S3. Dado que no es necesaria ninguna lógica adicional para este ejemplo, podemos utilizar un proxy de servicio de API Gateway para integrarlo directamente con Kinesis Data Firehose.

Figura 31: Reducción del costo de envío de datos a Amazon S3 mediante la implementación de un proxy de servicio de AWS

Con este enfoque, eliminamos el costo de uso de Lambda y las invocaciones innecesarias mediante la implementación del proxy de servicio de AWS en API Gateway. Como desventaja, esto podría suponer cierta complejidad adicional si se necesitan varias particiones para satisfacer la tasa de incorporación.

Si es sensible a la latencia, puede transmitir datos directamente a Kinesis Data Firehose con las credenciales correctas a costa de las características de abstracción, contrato y API.

Figura 32: Reducción del costo de envío de datos a Amazon S3 mediante la transmisión directa con el SDK de Kinesis Data Firehose

Para situaciones en las que necesite conectarse con recursos internos dentro de su VPC o en las instalaciones y no se requiera una lógica personalizada, utilice la integración privada de API Gateway.

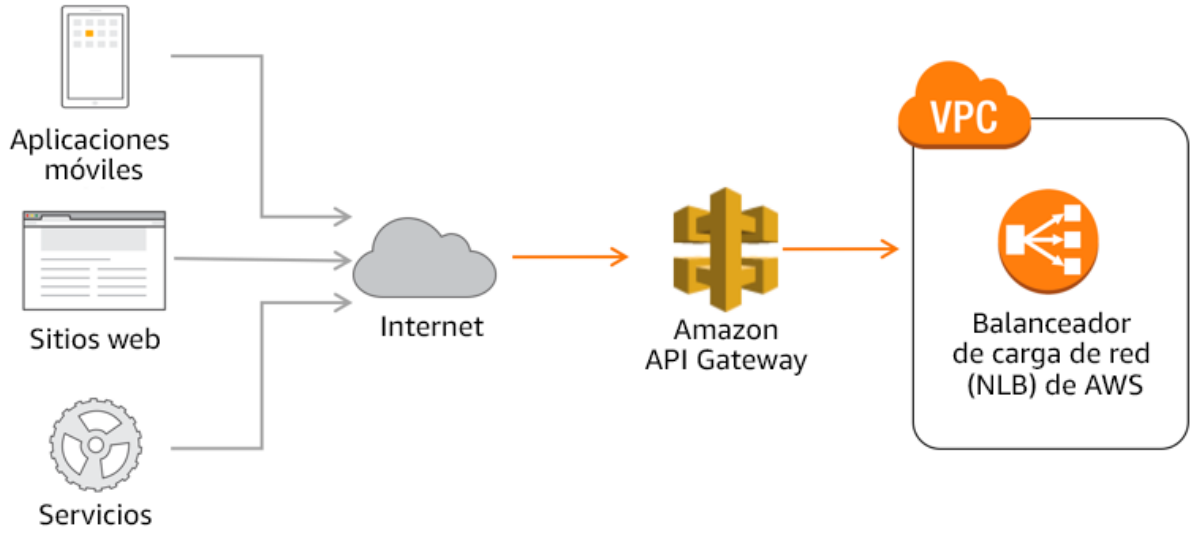

Figura 33: Integración privada de Amazon API Gateway a través de Lambda en VPC para obtener acceso a recursos privados

Con este enfoque, API Gateway envía cada solicitud entrante a un balanceador de carga de red interno de su propiedad en su VPC. Este reenvía el tráfico a cualquier backend, ya sea en la misma VPC o en las instalaciones a través de la dirección IP.

Este enfoque tiene beneficios tanto de costos como de rendimiento, ya que no necesita un salto adicional para enviar solicitudes a un backend privado con los beneficios adicionales de autorización, limitación controlada y mecanismos de almacenamiento en caché.

Otra escenario consiste en un patrón de distribución ramificada en el que Amazon SNS transmite mensajes a todos sus suscriptores. Este enfoque requiere una lógica de aplicación adicional para filtrar y evitar una invocación de Lambda innecesaria.

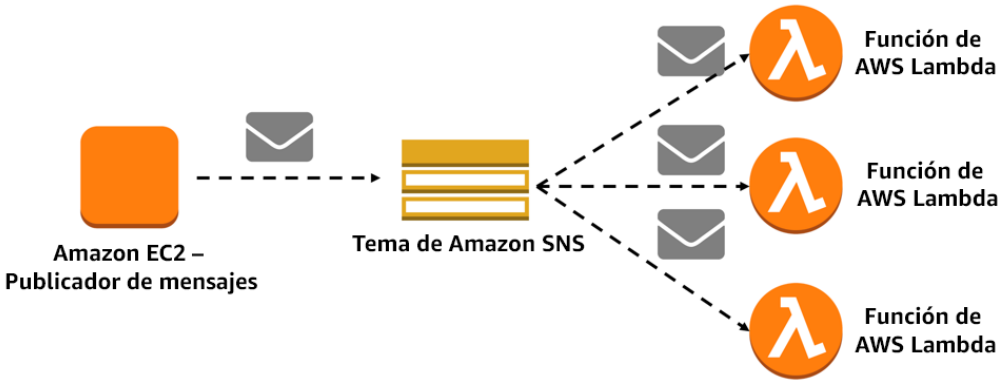

Figura 34: Amazon SNS sin filtrado de atributos de mensajes

SNS puede filtrar eventos en función de los atributos de los mensajes y entregar el mensaje de forma más eficiente al suscriptor correcto.

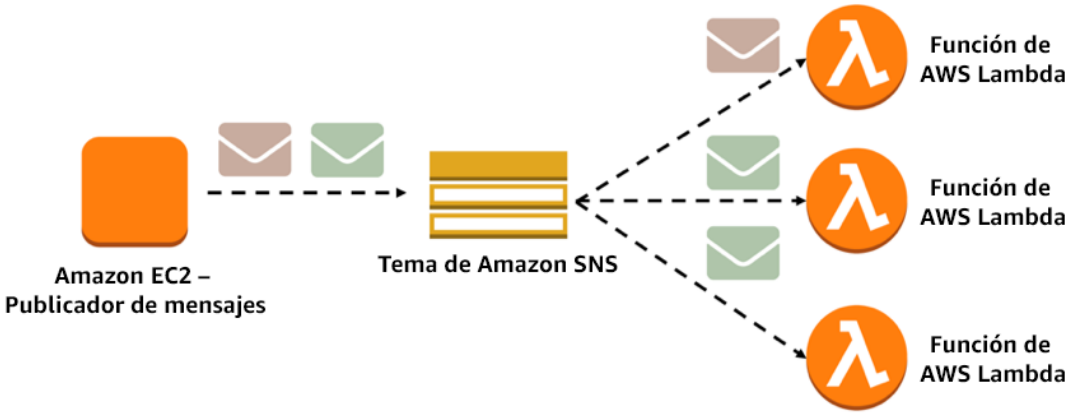

Figura 35: Amazon SNS con filtrado de atributos de mensajes

Otro ejemplo son las tareas de procesamiento de ejecución prolongada en las que es posible que tenga que esperar a que se complete la tarea antes de continuar con el siguiente paso. Este estado de espera puede implementarse dentro del código de Lambda. Sin embargo, es mucho más eficiente transformar el procesamiento asíncrono mediante eventos o implementar el estado de espera mediante Step Functions.

Por ejemplo, en la siguiente imagen, sondeamos un trabajo de AWS Batch y revisamos su estado cada 30 segundos para ver si se ha completado. En lugar de codificar esta espera dentro de la función de Lambda, implementamos un sondeo (*GetJobStatus*) + espera (*Wait30Seconds*) + factor decisivo (*CheckJobStatus*).

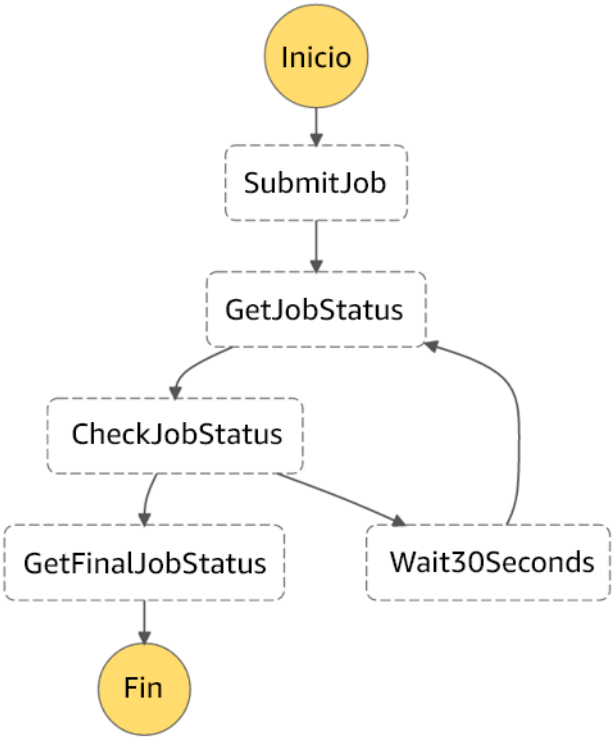

Figura 36: Implementación de un estado de espera con AWS Step Functions

La implementación de un estado de espera con Step Functions no incurrirá en ningún costo adicional, ya que el modelo de precios de Step Functions se basa en las transiciones entre estados y no en el tiempo empleado en un estado.

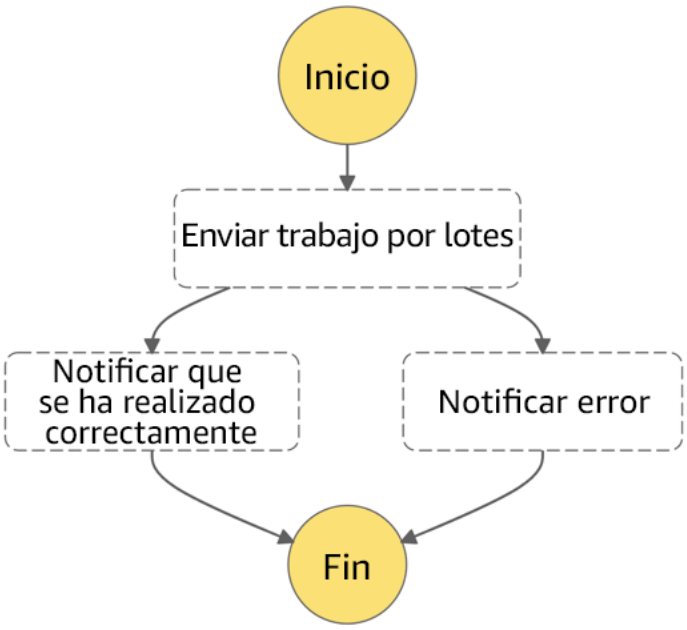

Figura 37: Espera síncrona de integración del servicio de Step Functions

En función de la integración que tenga que esperar, Step Functions puede esperar de forma síncrona antes de pasar a la siguiente tarea, lo que le ahorrará una transición adicional.

Optimización de código

Tal y como se explica en el pilar de rendimiento, la optimización de la aplicación sin servidor puede mejorar de forma eficaz el valor que produce por ejecución.

El uso de variables globales para mantener las conexiones con sus almacenes de datos u otros servicios y recursos aumentará el rendimiento y reducirá el tiempo de ejecución, lo que también reducirá el costo. Para obtener más información, consulte la sección del pilar de rendimiento.

Un ejemplo en el que el uso de las características de servicio administrado puede mejorar el valor por ejecución es mediante la recuperación y el filtrado de objetos de Amazon S3, ya que para recuperar objetos grandes de Amazon S3 se requiere más memoria para las funciones de Lambda.

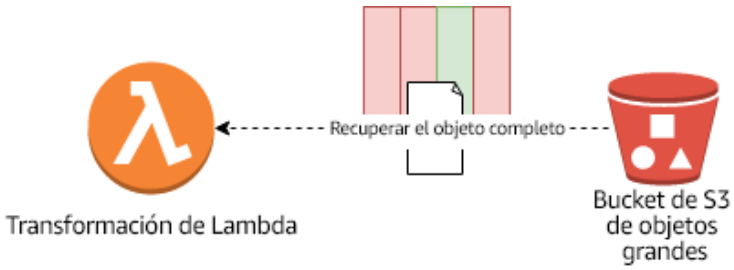

Figura 38: Función de Lambda que recupera el objeto de S3 completo

En el diagrama anterior, se observa que, cuando se recuperan objetos grandes de Amazon S3, es posible aumentar el consumo de memoria de Lambda, aumentar el tiempo de ejecución (para que la función pueda transformar, iterar o recopilar los datos necesarios) y, en algunos casos, solo se necesita parte de esta información.

Esto se representa con tres columnas en rojo (no se requieren datos) y una columna en verde (se requieren datos). El uso de consultas SQL de Athena para recopilar información detallada necesaria para su ejecución reduce el tiempo de recuperación y el tamaño del objeto en el que se realizan las transformaciones.

Figura 39: Lambda con recuperación de objetos de Athena

En el siguiente diagrama, se observa que, cuando se consulta a Athena para obtener los datos específicos, se reduce el tamaño del objeto recuperado y, como beneficio adicional, se ´puede reutilizar dicho contenido. Esto se debe a que Athena guarda los resultados de las consultas en un bucket de S3 e invoca la invocación de Lambda cuando los resultados llegan a Amazon S3 de forma asíncrona.

Se podría usar un enfoque similar con S3 Select. S3 Select permite a las aplicaciones recuperar solo un subconjunto de datos de un objeto mediante expresiones SQL sencillas. Al igual que en el ejemplo anterior con Athena, la recuperación de un objeto más pequeño de Amazon S3 reduce el tiempo de ejecución y la memoria utilizada por la función de Lambda.

200 segundos

```
# Download and process all keys
for key in src_keys:
  response =
s3_client.get_object(Bucket=src_bucket,
Key=key)
  contents = response['Body'].read()
  for line in contents.split('\n')[:-1]:
      line_count +=1
      try:
          data = line.split(',')
           srcIp = data[0][:8]


....
```

95 segundos

```
# Select IP Address and Keys
for key in src_keys:
  response =
s3_client.select_object_content
      (Bucket=src_bucket, Key=key,
expression =
       SELECT SUBSTR(obj._1, 1, 8),
obj._2 FROM s3object as obj)
  contents = response['Body'].read()
  for line in contents:
      line_count +=1
      try:


....
```

Figura 40: Estadísticas de rendimiento de Lambda con Amazon S3 y S3 Select

Recursos

Consulte los siguientes recursos para obtener más información sobre nuestras prácticas recomendadas sobre la optimización de costos.

Documentación y blogs

- Retención de CloudWatch Logs[64]
- Exportación de CloudWatch Logs a Amazon S3[65]
- Transmisiones de CloudWatch Logs a Amazon ES[66]
- Definición de estados de espera en máquinas de estado de Step Functions[67]
- Máquina de estado de pase de venta de Coca-Cola con tecnología de Step Functions[68]
- Creación de flujos de trabajo por lotes para genómica de alto rendimiento en AWS[69]
- Simplifique su mensajería de publicación/suscripción con el filtrado de mensajes de Amazon SNS
- S3 Select y Glacier Select
- Arquitectura de referencia de Lambda para MapReduce
- Aplicación de repositorio de aplicaciones sin servidor – Ajuste automático de la retención de grupos de CloudWatch Logs
- Diez recursos que cada arquitecto sin servidor debe conocer

Documento técnico

- Optimización de la economía empresarial con arquitecturas sin servidor[70]

Conclusión

Si bien las aplicaciones sin servidor eliminan las arduas tareas no diferenciadas a los desarrolladores, todavía hay principios importantes que aplicar.

Por motivos de confiabilidad, con la prueba periódica de las rutas de error, es más probable que detecte errores antes de que lleguen a producción. Para obtener rendimiento, retroceder hacia las expectativas de los clientes le permitirá diseñar para obtener una experiencia óptima. También hay una serie de herramientas de AWS que ayudan a optimizar el rendimiento.

Para optimizar los costos, puede reducir el residuo innecesario dentro de su aplicación sin servidor mediante el ajuste del tamaño de los recursos de acuerdo con la demanda de tráfico y mejorar el valor mediante la optimización de su aplicación. Para las operaciones, la arquitectura debe esforzarse por automatizar la respuesta a los eventos.

Por último, una aplicación segura protegerá los recursos de información confidencial de su organización y cumplirá cualquier requisito de conformidad en cada capa.

El panorama de las aplicaciones sin servidor evoluciona con el ecosistema de herramientas y procesos que crecen y maduran. A medida que esto ocurra, actualizaremos este documento para ayudarlo a garantizar que sus aplicaciones sin servidor tengan una buena arquitectura.

Colaboradores

Las siguientes personas y organizaciones contribuyeron a redactar este documento:

- Adam Westrich: arquitecto senior de soluciones, Amazon Web Services
- Mark Bunch: arquitecto de soluciones empresariales, Amazon Web Services
- Ignacio Garcia Alonso: arquitecto de soluciones, Amazon Web Services
- Heitor Lessa: director principal de Well-Architected sin servidor, Amazon Web Services
- Philip Fitzsimons: director senior de Well-Architected, Amazon Web Services
- Dave Walker: arquitecto principal y especialista en soluciones, Amazon Web Services
- Richard Threlkeld: director senior de productos móviles, Amazon Web Services
- Julian Hambleton-Jones: arquitecto senior de soluciones, Amazon Web Services

Documentación adicional

Para obtener información adicional, consulte lo siguiente:

- Marco de Buena Arquitectura de AWS[71]

Revisiones del documento

Fecha	Descripción
Diciembre de 2019	Actualizaciones para características nuevas y evolución de las prácticas recomendadas.
Noviembre de 2018	Nuevas situaciones para Alexa y Mobile, y actualizaciones para reflejar las nuevas características y la evolución de las prácticas recomendadas.
Noviembre de 2017	Publicación inicial.

Notas

[1] https://aws.amazon.com/well-architected

[2] http://d0.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

[3] https://github.com/alexcasalboni/aws-lambda-power-tuning

[4] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/BestPractices.html

[5] http://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/es-managedomains.html

[6] https://www.elastic.co/guide/en/elasticsearch/guide/current/scale.html

[7] http://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-scaling.html

[8] https://d0.awsstatic.com/whitepapers/whitepaper-streaming-data-solutions-on-aws-with-amazon-kinesis.pdf

[9] http://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecords.html

[10] http://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-duplicates.html

[11] http://docs.aws.amazon.com/lambda/latest/dg/best-practices.html#stream-events

[12] http://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-usage-plans.html

[13] http://docs.aws.amazon.com/apigateway/latest/developerguide/stage-variables.html

[14] http://docs.aws.amazon.com/lambda/latest/dg/env_variables.html

[15] https://github.com/awslabs/serverless-application-model

[16] https://aws.amazon.com/blogs/aws/latency-distribution-graph-in-aws-x-ray/

[17] http://docs.aws.amazon.com/lambda/latest/dg/lambda-x-ray.html

[18] http://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-paramstore.html

[19] https://aws.amazon.com/blogs/compute/continuous-deployment-for-serverless-applications/

[20] https://github.com/awslabs/aws-serverless-samfarm

[21] https://d0.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf

[22] https://aws.amazon.com/serverless/developer-tools/

[23] http://docs.aws.amazon.com/lambda/latest/dg/with-s3-example-create-iam-role.html

[24] http://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-method-request-validation.html

[25] http://docs.aws.amazon.com/apigateway/latest/developerguide/use-custom-authorizer.html

[26] https://aws.amazon.com/blogs/compute/secure-api-access-with-amazon-cognito-federated-identities-amazon-cognito-user-pools-and-amazon-api-gateway/

[27] http://docs.aws.amazon.com/lambda/latest/dg/vpc.html

[28] https://aws.amazon.com/pt/articles/using-squid-proxy-instances-for-web-service-access-in-amazon-vpc-another-example-with-aws-codedeploy-and-amazon-cloudwatch/

[29] https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

[30] https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf

[31] https://www.twistlock.com/products/serverless-security/

[32] https://snyk.io/

[33] https://www.owasp.org/index.php/OWASP_Dependency_Check

[34] http://theburningmonk.com/2017/07/applying-the-saga-pattern-with-aws-lambda-and-step-functions/

[35] http://docs.aws.amazon.com/lambda/latest/dg/limits.html

[36] http://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html#api-gateway-limits

[37] http://docs.aws.amazon.com/streams/latest/dev/service-sizes-and-limits.html

[38] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Limits.html

[39] http://docs.aws.amazon.com/step-functions/latest/dg/limits.html

[40] https://aws.amazon.com/blogs/compute/error-handling-patterns-in-amazon-api-gateway-and-aws-lambda/

[41] https://aws.amazon.com/blogs/compute/serverless-testing-with-aws-lambda/

[42] http://docs.aws.amazon.com/lambda/latest/dg/monitoring-functions-logs.html

[43] http://docs.aws.amazon.com/lambda/latest/dg/versioning-aliases.html

[44] http://docs.aws.amazon.com/apigateway/latest/developerguide/stages.html

[45] http://docs.aws.amazon.com/general/latest/gr/api-retries.html

[46] http://docs.aws.amazon.com/step-functions/latest/dg/tutorial-handling-error-conditions.html#using-state-machine-error-conditions-step-4

[47] http://docs.aws.amazon.com/xray/latest/devguide/xray-services-lambda.html

[48] http://docs.aws.amazon.com/lambda/latest/dg/dlq.html

[49] https://aws.amazon.com/blogs/compute/error-handling-patterns-in-amazon-api-gateway-and-aws-lambda/

[50] http://docs.aws.amazon.com/step-functions/latest/dg/amazon-states-language-wait-state.html

[51] http://microservices.io/patterns/data/saga.html

[52] http://theburningmonk.com/2017/07/applying-the-saga-pattern-with-aws-lambda-and-step-functions/

[53] https://d0.awsstatic.com/whitepapers/microservices-on-aws.pdf

[54] http://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

[55] https://aws.amazon.com/lambda/faqs/

[56] http://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

[57] http://docs.aws.amazon.com/lambda/latest/dg/lambda-introduction.html

[58] https://aws.amazon.com/blogs/compute/container-reuse-in-lambda/

[59] http://docs.aws.amazon.com/lambda/latest/dg/vpc.html

[60] http://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-caching.html

[61] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html

[62] https://aws.amazon.com/dynamodb/dax/

[63] http://docs.aws.amazon.com/streams/latest/dev/amazon-kinesis-streams.html

[64] http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/SettingLogRetention.html

[65] http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/S3ExportTasksConsole.html

[66] http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_ES_Stream.html

[67] http://docs.aws.amazon.com/step-functions/latest/dg/amazon-states-language-wait-state.html

[68] https://aws.amazon.com/blogs/aws/things-go-better-with-step-functions/

[69] https://aws.amazon.com/blogs/compute/building-high-throughput-genomics-batch-workflows-on-aws-workflow-layer-part-4-of-4/

[70] https://d0.awsstatic.com/whitepapers/optimizing-enterprise-economics-serverless-architectures.pdf

[71] https://aws.amazon.com/well-architected