Arquitectura Resiliente para UltraSeguros S.A.

Introducción

UltraSeguros S.A. ofrece servicios críticos que requieren **alta disponibilidad**, **tolerancia a fallos** y **capacidad de adaptación ante condiciones adversas**. Durante picos de tráfico o escenarios de error, un fallo total del sistema puede generar impactos económicos significativos y afectar la confianza de los clientes.

Ante este contexto, se diseñó e implementó una **arquitectura resiliente**, capaz de **degradarse progresivamente**, **mantener activos los servicios críticos** y **recuperarse automáticamente** cuando las condiciones del sistema se estabilizan, sin intervención manual.

Decisiones de Arquitectura

Uso de AWS API Gateway

Se seleccionó AWS API Gateway como punto único de entrada al sistema debido a que permite:

- Centralizar el acceso a los servicios.
- Manejar picos de tráfico de forma automática.
- Desacoplar a los clientes de la lógica interna del sistema.

API Gateway cumple el rol de **fachada del sistema**, protegiendo la capa de cómputo, controlando el acceso y facilitando la observabilidad de las solicitudes entrantes.

Uso de AWS Lambda

Se utilizó AWS Lambda para implementar la lógica principal del sistema debido a que:

- Permite **escalado automático** sin necesidad de gestionar infraestructura.
- Aísla fallos a nivel de ejecución.
- Facilita la implementación de lógica condicional basada en el estado de salud del sistema.
- Reduce costos operativos al ejecutarse únicamente bajo demanda.

La función Lambda contiene la lógica de evaluación de salud, conteo de errores, recuperación automática y determinación dinámica del nivel de servicio activo.

Uso de AWS CloudWatch Logs

AWS CloudWatch Logs se utilizó como componente central de **observabilidad**, permitiendo:

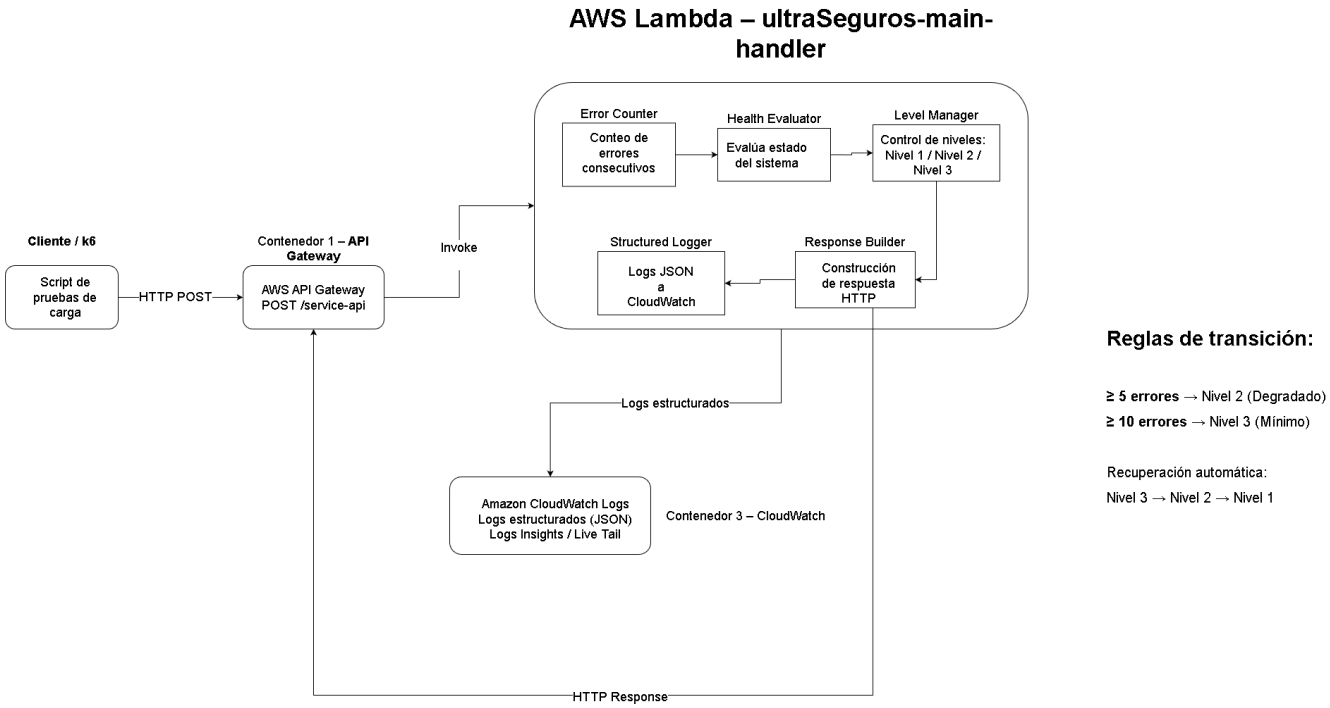
- Registrar cada solicitud procesada por el sistema.
- Evidenciar explícitamente los cambios de nivel de servicio.
- Validar el comportamiento del sistema bajo pruebas de carga.
- Auditar los procesos de degradación y recuperación automática.

El uso de logs estructurados en formato JSON facilita el análisis posterior mediante CloudWatch Logs Insights.

Diagrama de Arquitectura (Modelo C4)

El siguiente diagrama representa la arquitectura del sistema utilizando el **modelo C4**, mostrando:

- El contexto del sistema.
- Los contenedores principales.
- Los componentes internos de la función Lambda.
- El flujo de solicitudes, degradación y recuperación automática.



Atributo de Calidad Priorizado

Resiliencia

El atributo de calidad priorizado fue la **resiliencia**, entendida como la capacidad del sistema de **continuar operando ante fallos parciales** y condiciones adversas.

Se priorizó la resiliencia por encima de la disponibilidad total, permitiendo que el sistema **reduzca progresivamente capacidades no críticas** antes de llegar a un fallo completo. Este enfoque garantiza continuidad del servicio y una mejor experiencia para el usuario final.

Niveles de Servicio Implementados

Nivel 1 – Servicio Completo (Full Service)

- Todas las funcionalidades del sistema se encuentran disponibles.
- Mensaje de respuesta:

```
Nivel 1: OK
```

Nivel 2 – Servicio Degradado

- Solo se mantienen las funcionalidades esenciales.
- Se activa cuando se detectan **5 errores o más**.
- Permite continuidad operativa con capacidades reducidas.

Nivel 3 – Operación Mínima

- Se activa cuando se detectan **10 errores o más**.
- El sistema **nunca deja de responder**.
- Mensajes claros al usuario:

Error:

```
Nivel 3: Sistema bajo mantenimiento, intente más tarde
```

Respuesta mínima exitosa:

```
Nivel 3: Operación al mínimo
```

Tácticas de Arquitectura Aplicadas

Degradación progresiva (Graceful Degradation)

El sistema reduce sus capacidades de forma controlada conforme aumentan los errores, evitando fallos totales y manteniendo los servicios críticos disponibles.

Monitoreo de salud

Cada petición evalúa métricas internas como:

- `errorCount`
- `healthyCount`
- `level`

Estas métricas determinan dinámicamente el nivel de servicio activo.

Recuperación automática

Cuando los indicadores de salud regresan a valores aceptables, el sistema recupera gradualmente sus capacidades sin necesidad de intervención manual.

Observabilidad

Todos los eventos relevantes quedan registrados en CloudWatch Logs, permitiendo análisis posterior y validación del comportamiento esperado del sistema.

Logs y Métricas

El sistema genera logs estructurados en formato JSON para cada ejecución. Ejemplo de log registrado:

```
{
  "time": "2025-12-12T19:23:08.509Z",
  "level": 1,
  "errorCount": 0,
  "healthyCount": 5,
  "statusCode": 200,
  "message": "Nivel 1: OK"
}
```

Estos logs permiten identificar claramente:

- El nivel activo del sistema.
- La cantidad de errores acumulados.
- El proceso de degradación y recuperación.

Pruebas Bajo Carga

Se ejecutó un script k6 durante aproximadamente **6 minutos**, simulando carga constante y fallos intencionales.

Durante la ejecución, el sistema:

- Transitó por los **tres niveles de servicio**.
- Registró las transiciones en CloudWatch Logs.
- Se recuperó automáticamente al Nivel 1 al finalizar la prueba.

Tipo de Arquitectura: Monolítica y Distribuida

La solución implementa una **arquitectura monolítica lógica y distribuida físicamente**.

Monolítica (lógica)

- Toda la lógica de negocio reside en una única función Lambda.
- Simplifica la consistencia del estado.
- Facilita el control de degradación y recuperación.

Distribuida (física)

Los componentes están desacoplados y administrados de forma independiente:

- API Gateway
- AWS Lambda
- CloudWatch Logs

Cada componente escala y falla de manera independiente.

Justificación

Para este reto, una arquitectura completamente distribuida basada en múltiples microservicios habría introducido complejidad innecesaria. La elección de una arquitectura monolítica lógica permitió enfocarse en la **resiliencia**, evitando falacias comunes de arquitecturas de microservicios como:

- Complejidad excesiva.
- Sobrecosto operativo.
- Dificultad de observabilidad.
- Problemas de consistencia de estado.

Conclusión

La arquitectura diseñada demuestra que es posible construir un sistema altamente resiliente utilizando servicios administrados de AWS. El sistema es capaz de adaptarse dinámicamente a condiciones adversas, degradarse de forma controlada y recuperarse automáticamente, cumpliendo con los objetivos del negocio y los requerimientos técnicos del reto.

Comandos Utilizados

```
k6 run reto3.js
```

Ejecuta la prueba de carga que simula picos de tráfico y errores.

```
fields @timestamp, @message
| filter @message like /Nivel/
| sort @timestamp asc
```

Evidencia transiciones de nivel en CloudWatch Logs.

```
parse @message '"level":*,' as level
```

Extrae métricas estructuradas desde logs JSON.
