Backend Nodejs/AWS Challenge

Descripción del reto técnico

1. **Desarrollar una API RESTful** utilizando **Node.js** versión 20, con **Serverless Framework o CDK** y desplegarla en AWS Lambda.
2. **Integrar dos APIs públicas** distintas, una de ellas será la **API de Star Wars (SWAPI)**, y otra API pública de tu elección (por ejemplo, una API meteorológica).
3. Al consumir los datos de estas APIs:
    - **Fusiona los datos** obtenidos de ambas APIs en un solo modelo que tenga sentido. Por ejemplo, puedes combinar información sobre los personajes de Star Wars con datos meteorológicos basados en la ubicación de sus planetas.
    - **Normaliza y procesa** los datos, asegurando que todos los campos estén correctamente formateados (conversión de tipos, unidades de medida, etc.).
4. **Crear 3 endpoints**:
    - **GET /fusionados**: Este endpoint debe consultar ambas APIs y devolver los datos fusionados. La respuesta debe estar almacenada en una **base de datos** (DynamoDB o MySQL) para futuras consultas.
    - **POST /almacenar**: Permite almacenar información personalizada (no relacionada con las APIs externas) en la base de datos.
    - **GET /historial**: Retorna el historial de todas las respuestas almacenadas por el endpoint /fusionados, ordenado cronológicamente y paginado.
5. **Añadir un proceso de caché**: Debes implementar un sistema de **cacheo** de las respuestas obtenidas de las APIs externas. Si la información solicitada ha sido consultada en los últimos 30 minutos, devolverla desde el caché en lugar de hacer una nueva llamada a las APIs. Puedes utilizar DynamoDB como caché o Redis.
6. **Optimización de costos**: Utiliza las prácticas recomendadas de AWS para minimizar el costo del uso de **Lambda**, como el ajuste del timeout y la memoria.

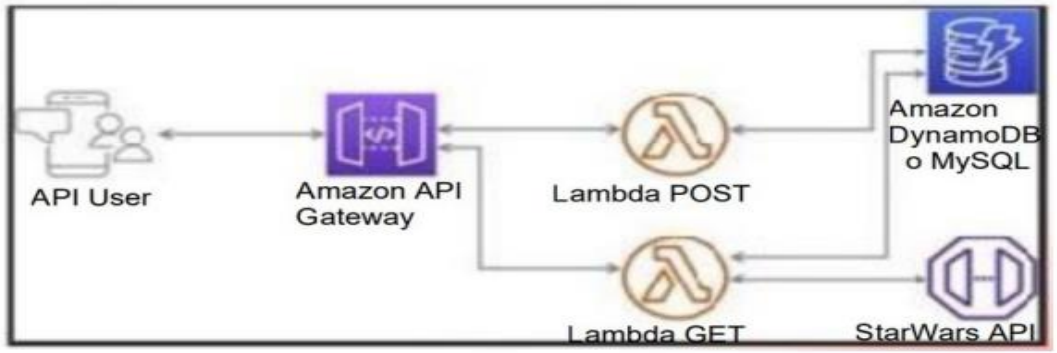
Puntos mínimos-obligatorios del MVP

- **Pruebas unitarias** y de integración usando Jest o similar.
- Uso de **TypeScript** para tipado estático y mayor seguridad en el código.
- **3 endpoints:**
  - Un GET que combine y muestre datos de las dos APIs externas.
  - Un POST para almacenar recursos propios en la base de datos.
  - Un GET para consultar el historial de datos almacenados.
- **Cacheo de resultados** para evitar múltiples llamadas a las APIs dentro de un intervalo de 30 minutos.
- **Despliegue en AWS** usando Serverless Framework o CDK.
- Almacenamiento en **DynamoDB o MySQL**.
- Uso de **AWS Lambda y API Gateway**.

Puntos Bonus:

- **Autenticación** para proteger los endpoints POST y GET /historial (puede ser con **JWT** o **AWS Cognito**).
- **Documentación** de los endpoints con **Swagger/OpenAPI**.
- Uso de **logging** avanzado con **AWS CloudWatch** para rastrear errores y rendimiento.
- Implementar un sistema de **rate-limiting** para evitar abuso de los endpoints que consumen las APIs externas.
- **Monitorización** de latencias y trazabilidad de las peticiones con **AWS X-Ray**.
- Uso de Gherkin para las pruebas unitarias y enfoque BDD.

Arquitectura Recomendada



¡Muchas Gracias!