**Despliegue de la solución en AWS: Servicios, Infraestructura y Escenarios de Optimización**

Para desplegar esta solución de generación de tickets basada en modelos de lenguaje, propongo utilizar la nube de Amazon Web Services (AWS) por su escalabilidad, seguridad y variedad de servicios administrados. A continuación, se detallan los servicios necesarios, la arquitectura recomendada y un escenario de optimización ante una futura demanda elevada.

**Servicios AWS necesarios**

* **Amazon ECS / AWS Fargate**: Para ejecutar contenedores Docker. Se utilizaría para desplegar la API desarrollada con FastAPI.
* **Amazon ECR**: Repositorio donde se almacenarán las imágenes Docker de la aplicación desarrollada.
* **Amazon API Gateway**: Se podría utilizar para exponer la API al público con control de tráfico, autenticación y límites de uso.
* **Amazon S3**: Se utilizaría para almacenar archivos como los provistos en la prueba cases.json, departments.json o archivos de configuración como el prompt.
* **AWS Secrets Manager**: Se utilizaría para almacenar y acceder a credenciales de manera segura, como la clave de OpenAI.
* **Amazon CloudWatch**: Utilizada para el monitoreo, métricas y logs de la aplicación desplegada en producción.
* **VPC, Subnets y Security Groups**: Para definir una red privada segura y controlar el acceso a los recursos.
* **IAM (Identity and Access Management)**: Para gestionar permisos de acceso entre los servicios utilizados.

**Arquitectura propuesta**

El cliente o sistema frontend realizará solicitudes a una API REST desplegada en un contenedor dentro de ECS (o Fargate). Esta API accede a los archivos necesarios en S3 y utiliza la clave de OpenAI almacenada en Secrets Manager para realizar la generación del ticket. Todos los logs y métricas son enviados a CloudWatch para su monitoreo.

A diagram of a software application

AI-generated content may be incorrect.

**Escenario posible de optimización**

Suponiendo que la API empieza a recibir un gran volumen de tráfico (miles de solicitudes por hora), este escenario puede generar cuellos de botella, sobre todo teniendo en cuenta la utilización de OpenAI y la anonimización de la información de los clientes con SpaCy. Para solucionarlo se podría utilizar:

1. **Separar el procesamiento** mediante una arquitectura basada en colas (Amazon SQS) y funciones Lambda que procesen los casos en segundo plano.
2. **Implementar caché** con Redis (Amazon ElastiCache) para evitar llamadas duplicadas a OpenAI si el mismo caseID ya fue procesado.
3. **Guardar los resultados** en una base de datos como DynamoDB o Aurora Serverless, permitiendo auditoría y análisis posteriores.
4. **Escalar horizontalmente** con más contenedores Fargate adicionales o tareas Lambda paralelas según la carga.