Estrategia de Despliegue & Industrialización del Modelo de Recomendación de Tratamiento

Juan Pablo Restrepo Urrea

13 de junio de 2025

1. Análisis del caso

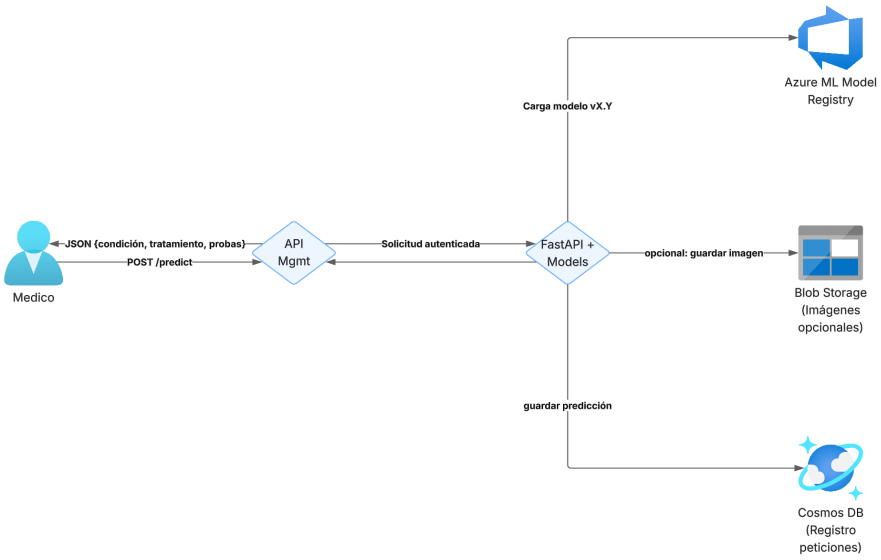
- **Retos técnicos** : orquestar inferencia con dos modelos (EfficientNet + XGBoost) manteniendo baja latencia; automatizar el ciclo de vida (versionado, pruebas, *rollback*); y garantizar trazabilidad clínica.
- **Riesgos** : deriva de datos, sobrecarga de la API ante picos de uso, fuga de datos sensibles, incumplimiento normativo (GDPR/hipaa) y pérdida de explicabilidad clínica.
- **Oportunidades** : uso de servicios gestionados de Azure que simplifican MLOps (*Azure ML*, *Container Apps*, *Monitor*); escalado elástico con pago-por-uso; y capacidad de iterar rápidamente incorporando nuevas versiones de modelos o técnicas (p. ej. vision transformers).

2. Diseño de la solución

Visión de alto nivel (solo inferencia)

1) **Ingesta.** Las peticiones llegan a un *Azure API Management* que autentica y enruta a un backend containerizado (*FastAPI*) desplegado en *Azure Container Apps*.
2) **Preprocesamiento & validación.** Dentro del contenedor se ejecuta el pipeline Python que: (i) valida tipo MIME y dimensiones de la imagen, (ii) aplica transformaciones de EfficientNet, (iii) parsea la nota clínica y normaliza las variables.
3) **Almacenamiento.** Las peticiones y resultados se registran en *Azure Cosmos DB* (esquema flexible JSON) para auditoría; las imágenes opcionalmente se archivan en *Blob Storage*.
4) **Modelos.** El contenedor monta desde *Azure ML Model Registry* la última versión estable del modelo doble (`tumor_clf` y `treatment_rec`). El despliegue usa GPU SKU si se detecta carga intensiva de imágenes.
5) **Seguridad / interoperabilidad / escalado.** *Private Link* + *VNet* → no exposición pública de datos; secretos en *Key Vault*; escalado horizontal automático en *Container Apps*; respuesta en JSON compatible FHIR.

Diagrama de arquitectura



Mejoras sobre la API actual

- **Versionado y compatibilidad**: exponer cabeceras `X-API-Version`; mantener varias rutas (`/v1/predict`, `/v2/predict`) para *blue/green deployments*.
- **Observabilidad**: incluir *OpenTelemetry* en FastAPI para trazas distribuidas; exportar métricas Prometheus (latencia, throughput, códigos de respuesta).
- **Seguridad**: OAuth2 con Azure AD; throttling y WAF en API Management; sanitizar metadatos DICOM antes de persistir.
- **Packaging**: usar un único Dockerfile para definir un entorno reproducible.
- **CI/CD**: GitHub Actions Azure ML CLI v2 → test unitarios, escaneo de vulnerabilidades, despliegue automatizado a Container Apps con validación canary (p. ej. 5

3. Evaluación de la solución

Métricas de sistema

- **Latencia P95** de inferencia (< 500 ms objetivo).
- **Disponibilidad** (SLA $\geq 99{,}5\,\%$).
- **Exactitud / F1 ponderado** del modelo con *shadow data*.
- **Deriva de datos**: PSI, KS test sobre distribuciones de entrada.
- **Fairness gap**: diferencia absoluta de F1 entre sub-grupos (< 5 p.p.).

Sesgo, explicabilidad y validación clínica

- Monitorizar *Fairlearn* semanal; alarmar si la métrica de equidad supera umbral.
- Servir SHAP local $\pm$ Grad-CAM como adjunto JSON para cada predicción.
- Pilotaje prospectivo: validar concordancia modelo-clínico $\geq 80\,\%$.

Mitigación de riesgos

- Controles de privacidad (anonimización + encriptación, *Key Vault*).
- Política de *fallback*: si el modelo está fuera de rango de confianza, devuelve "Sin recomendación – escalar a especialista".
- Comité de gobernanza de IA revisa métricas, sesgo y trazabilidad cada trimestre.

Disclaimer : *Este documento presenta una propuesta técnica elaborada como resultado de investigación personal y revisión de buenas prácticas. Mi experiencia en industrializar modelos en la nube es intermedia y principalmente holistica y teórica.*