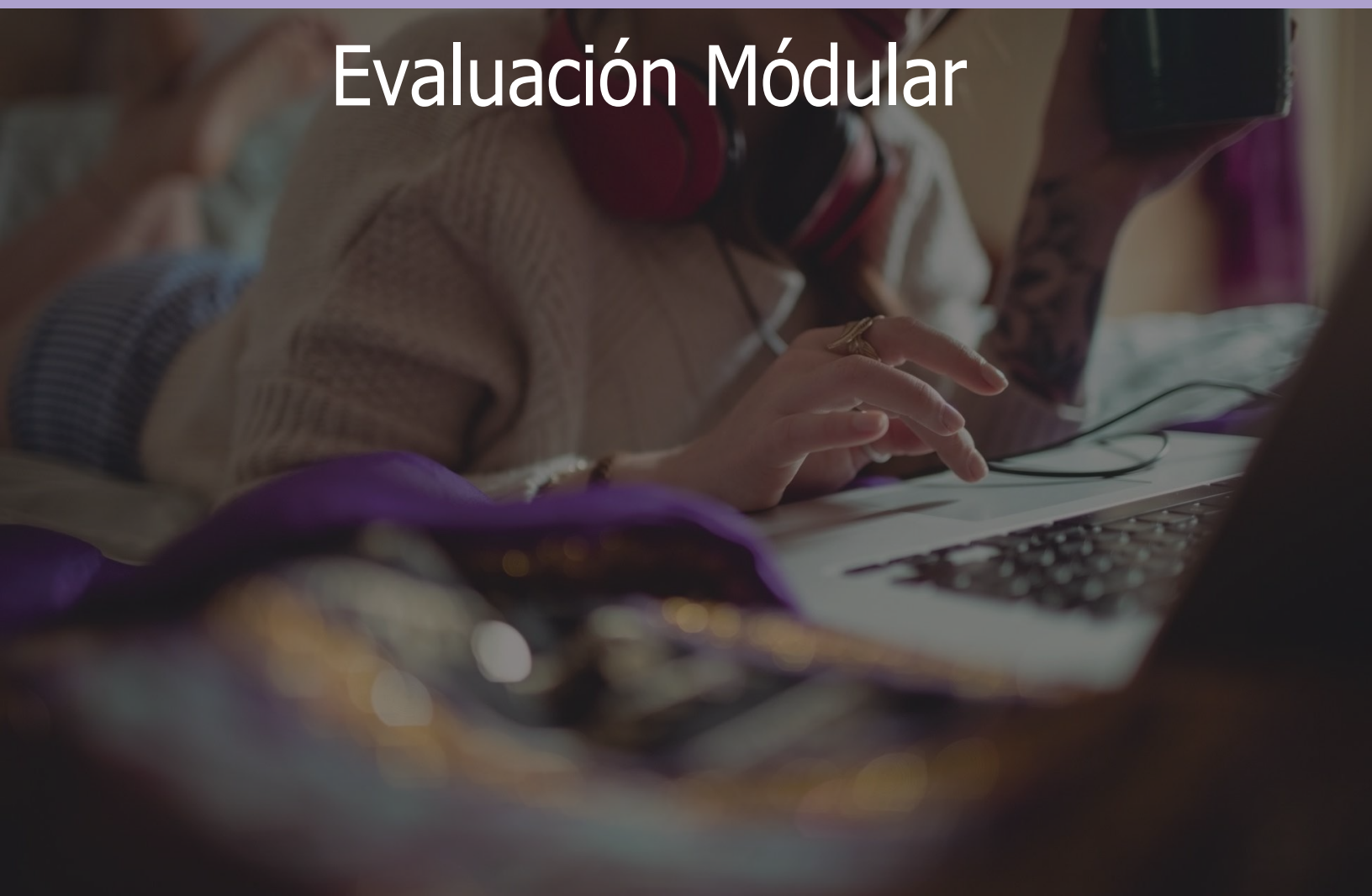


# Módulo 10

Evaluación Modular



## ACTIVIDAD:

# MLOps en la Nube: Despliegue automatizado de un modelo predictivo real

### Objetivo:

Desarrollar un sistema completo que integre un modelo de Machine Learning (clasificación o regresión), exponerlo como API REST mediante Flask, contenedorizado con Docker, y con un flujo básico de CI/CD para automatizar pruebas y despliegues locales. Se espera demostrar buenas prácticas de versionado, empaquetado y documentación.

### Contexto:

Has sido contratado por una fintech/startup de salud para integrar un modelo de predicción (puede ser de scoring de crédito o diagnóstico preventivo) dentro de su infraestructura tecnológica. Necesitan que este servicio sea:

- Escalable localmente mediante contenedores
- Fácil de actualizar
- Accesible vía REST
- Documentado y probado con CI/CD

Tiempo estimado de desarrollo: 120 minutos.

Formato de entrega: archivo comprimido .zip o .rar con código fuente y PDF.

Modalidad: grupal.



# Requerimiento:

## 1. Modelo Predictivo

- Dataset a utilizar:
  - Salud: Breast Cancer (<https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data> )
- Entrenar y guardar el modelo usando joblib, pickle o .h5

## 2. API con Flask

- Crear una API con al menos dos rutas:
  - GET /: para probar el estado del servicio
  - POST /predict: para recibir un JSON y retornar predicción
- Validación de entradas, manejo de errores, logging


## 3. Dockerización

- Dockerfile funcional, con dependencias y entorno reproducible.
- Pruebas locales con Docker run.

## 4. CI/CD Automatizado (opcional, recomendado)

- Workflow en GitHub Actions o GitLab CI.
- Automatizar: build, test de endpoint básicos, push a registro.





# Rúbrica:

Indicador / Criterio	Insuficiente	Por lograrlo	Medianamente logrado	Logrado	Sobresaliente
Entrenamiento y serialización del modelo	No entrena modelo o no lo guarda.	Modelo entrenado, pero mal guardado o con errores.	Modelo entrenado y guardado correctamente, sin documentación clara.	Entrenamiento correcto, guardado con joblib/pickle/h5, y con prueba básica.	Entrenamiento optimizado, probado, con explicación clara del modelo y sus decisiones.
Desarrollo de API con Flask	No implementa API funcional.	API básica sin manejo de errores ni validación.	API funcional con rutas requeridas, validación limitada.	API bien estructurada, con validación, logging y manejo de errores.	API robusta, documentada, con pruebas de endpoints (manuales o automatizadas).
Dockerización del sistema	No entrega Dockerfile o está incompleto.	Dockerfile con errores o sin pruebas.	Dockerfile funcional, reproducible, probado localmente.	Dockerfile correcto, imagen funcional, dependencias claras.	Dockerfile optimizado, eficiente en capas, pruebas locales exitosas.
Automatización CI/CD (opcional)	No implementa.	Workflow básico sin pruebas.	CI/CD con build funcional y push a registro.	CI/CD con build, test de endpoints y push automatizado.	Flujo CI/CD completo y profesional con integración continua y pruebas automáticas.
Documentación y entrega	Entrega incompleta, desordenada o ilegible.	Código sin explicación ni estructura.	Proyecto funcional con documentación básica (README, PDF).	Entrega ordenada con README claro, PDF explicativo y capturas.	Documentación clara, completa, con buenas prácticas y reflexiones finales.