Despliegue básico de modelos como API local

🎯 **Objetivo:** Aprender a exponer un modelo de ML como un servicio accesible vía HTTP en la máquina local para que otros programas puedan consumirlo de manera eficiente.

Gabriel Ed. Rengifo

¿Por qué necesitamos desplegar modelos?

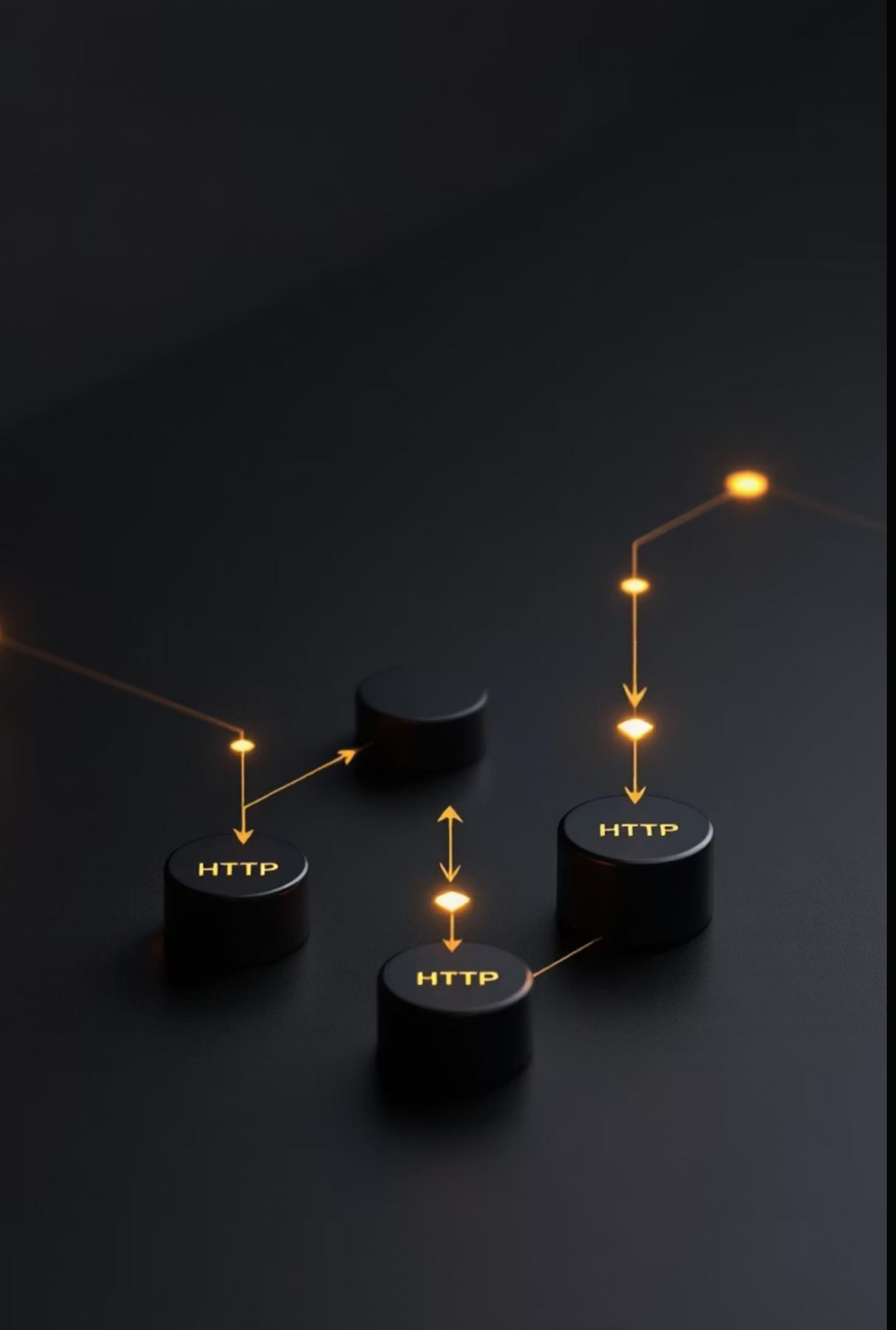
El problema

Los modelos entrenados en Jupyter Notebooks no pueden ser utilizados por otras aplicaciones de forma directa. Se quedan aislados en un entorno de desarrollo.

La solución

Necesitamos **operacionalizarlos** creando servicios que permitan:

- Integración con aplicaciones web y móviles
- Consumo por otros programas y sistemas
- Uso en tiempo real de predicciones



¿Qué es una API?

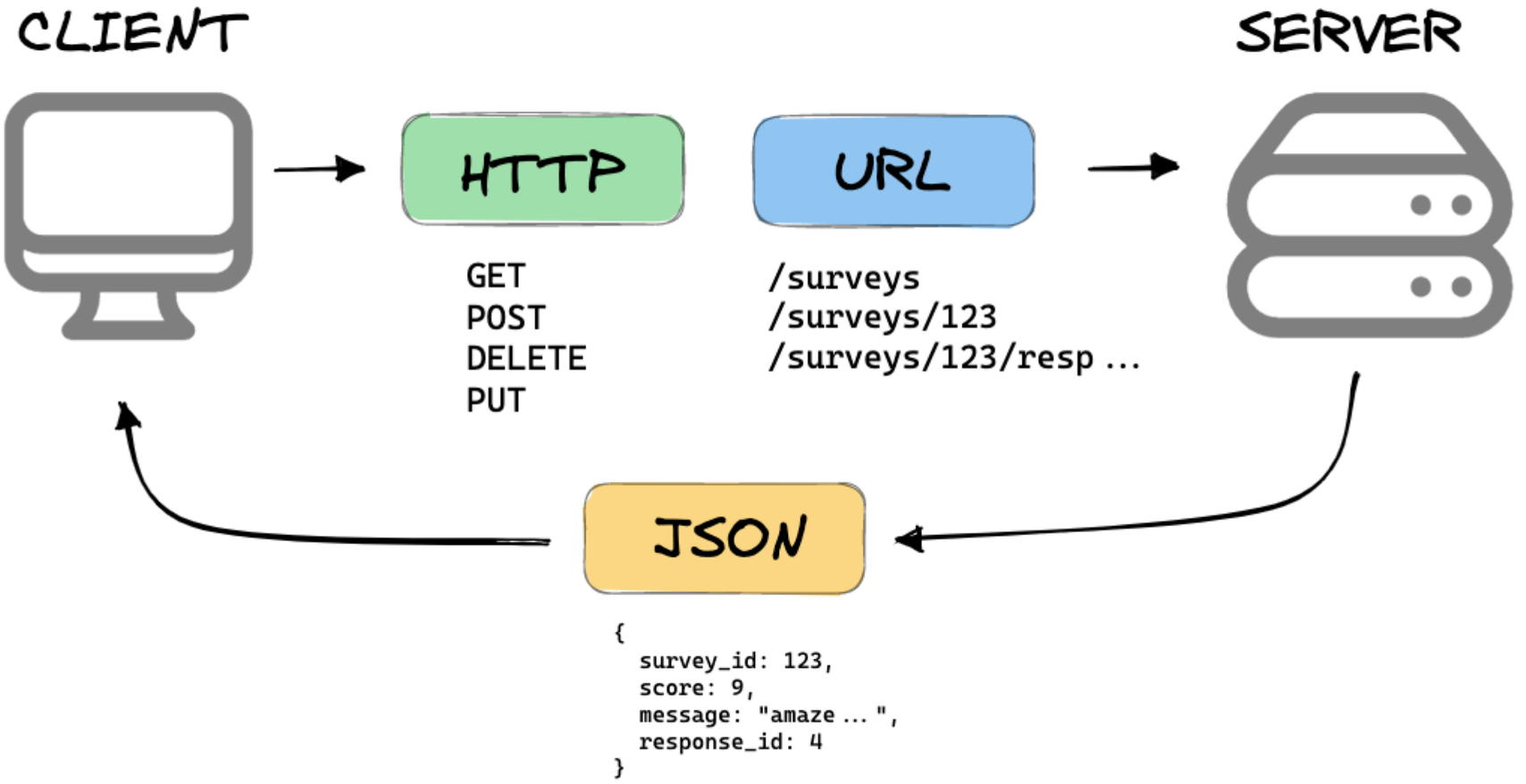
API

Application Programming Interface: una interfaz que permite que diferentes aplicaciones se comuniquen entre sí de manera estandarizada.

APIs REST

En Machine Learning utilizamos principalmente **APIs REST** basadas en el protocolo HTTP para enviar y recibir datos.

WHAT IS A REST API?



Arquitectura básica del despliegue

Modelo entrenado

Algoritmo de ML previamente entrenado y guardado en disco (pickle, joblib)

API de predicción

Servicio que recibe datos → los procesa con el modelo → devuelve predicciones

Cliente consumidor

Aplicación web, móvil u otro script que consume las predicciones de la API

Ejemplo de flujo de predicción

01

Petición del cliente

Cliente envía una petición POST con datos en formato JSON:

```
{"age": 30, "fare": 100, "sex": "male"}
```

02

Procesamiento

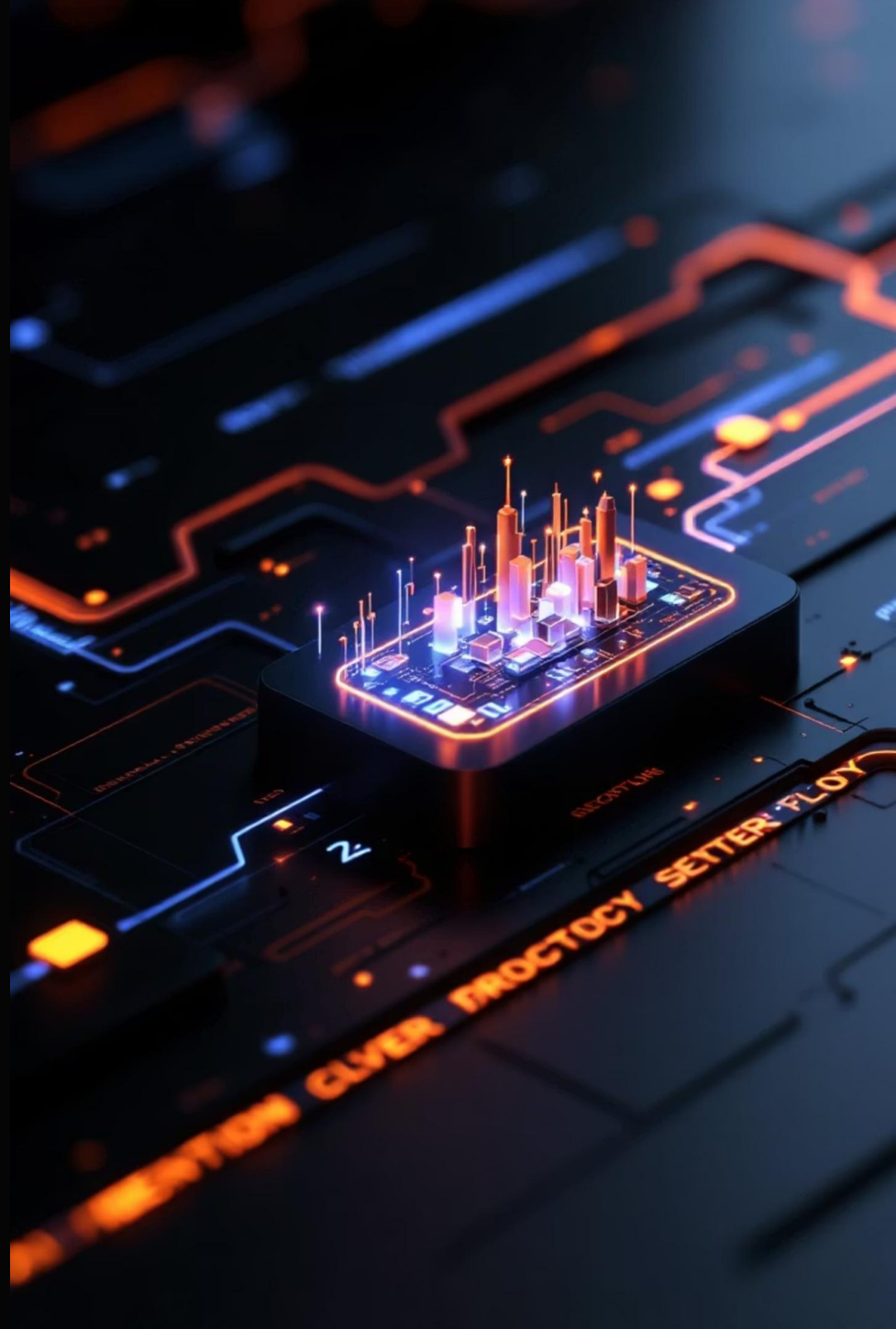
La API recibe los datos, los valida y los procesa utilizando el modelo de ML cargado en memoria.

03

Respuesta

La API devuelve la predicción en formato JSON:

```
{"prediction": "Survived"}
```



Frameworks para crear APIs

Flask

Ventajas: Simple y ligero, ideal para demos y prototipos rápidos. Fácil de aprender y configurar.

Uso: Proyectos pequeños y medianos

FastAPI

Ventajas: Muy rápido, validación automática de datos, documentación automática con Swagger.

Uso: Proyectos modernos que requieren alto rendimiento

Django REST Framework

Ventajas: Muy robusto, con muchas funcionalidades integradas. Ideal para proyectos empresariales.

Uso: Aplicaciones complejas y escalables

Pasos para desplegar localmente

Guardar el modelo

Entrenar y guardar el modelo usando librerías como joblib o pickle para persistencia.

Crear app.py

Desarrollar archivo principal con Flask/FastAPI que carga el modelo y define la lógica de la API.

Definir endpoints

Crear rutas como /predict que reciban datos y devuelvan predicciones del modelo.

Probar la API

Usar herramientas como curl, Postman o Python requests para validar el funcionamiento.

Buenas prácticas de desarrollo

Separación de código

Mantener completamente separado el código de **entrenamiento** del modelo y el código de **despliegue** para mayor claridad.

Validación de entradas

Implementar validación estricta de tipos de datos, rangos de valores y formato de las peticiones recibidas.

Manejo de errores

Crear respuestas apropiadas para inputs inválidos, errores del modelo y problemas de conectividad.

Documentación

Documentar todos los endpoints con ejemplos de uso, parámetros esperados y formatos de respuesta.

Limitaciones del despliegue local

⚠️ Restricciones

- Solo accesible desde la máquina local (127.0.0.1)
- No escalable para múltiples usuarios
- Limitado por recursos del equipo local

✅ Ventajas

- Ideal para prototipos y desarrollo
- Pruebas rápidas sin infraestructura
- Control total del entorno

Conclusiones y próximos pasos

✅ Modelos como servicios

El despliegue local transforma modelos aislados en **servicios reutilizables** que pueden integrarse en cualquier aplicación.

✅ Estándar de la industria

Las APIs REST con JSON y HTTP son el **punto de entrada estándar** para consumir modelos de ML en producción.

✅ Camino a producción

Este es el **primer paso fundamental** hacia el despliegue en la nube y entornos de producción escalables.

En las próximas sesiones exploraremos el despliegue en servidores para llevar nuestros modelos al siguiente nivel.

```
ml-api/
├─ app/
│  ├─ __init__.py
│  ├─ config.py
│  ├─ extensions.py
│  ├─ api/
│  │  ├─ __init__.py
│  │  └─ v1/
│  │     ├─ __init__.py
│  │     ├─ namespace.py
│  │     ├─ schemas.py
│  │     └─ services.py
│  └─ wsgi.py
├─ models/
│  ├─ titanic_clf.joblib
│  └─ metadata.json
├─ train/
│  └─ train_titanic.py
├─ .env.example
├─ requirements.txt
└─ README.md
```

Estructura de un proyecto IA

Stack tecnológico

```
Flask==3.0.3
flask-restx==1.3.0
flask-cors==4.0.1
python-dotenv==1.0.1
joblib==1.4.2
pandas==2.2.2
numpy==1.26.4
scikit-learn==1.5.2
```

Enviroment

```
FLASK_ENV=development
APP_NAME=ml-api
SECRET_KEY=supersecret
MODEL_PATH=models/titanic_clf.joblib
MODEL_METADATA=models/metadata.json
HOST=0.0.0.0
PORT=8000
```