Despliegue de un Modelo ML

Usando Flask

API REST

Interfaz de Programación de Aplicaciones que se ajusta a los límites de la arquitectura REST.

REST es una arquitectura de desarrollo web que puede ser utilizada en cualquier cliente HTTP; es decir, toda la información será enviada por el cliente en cada mensaje HTTP.

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones. Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).

Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.

Flask

Microframework de Python para el desarrollo de aplicaciones Web.

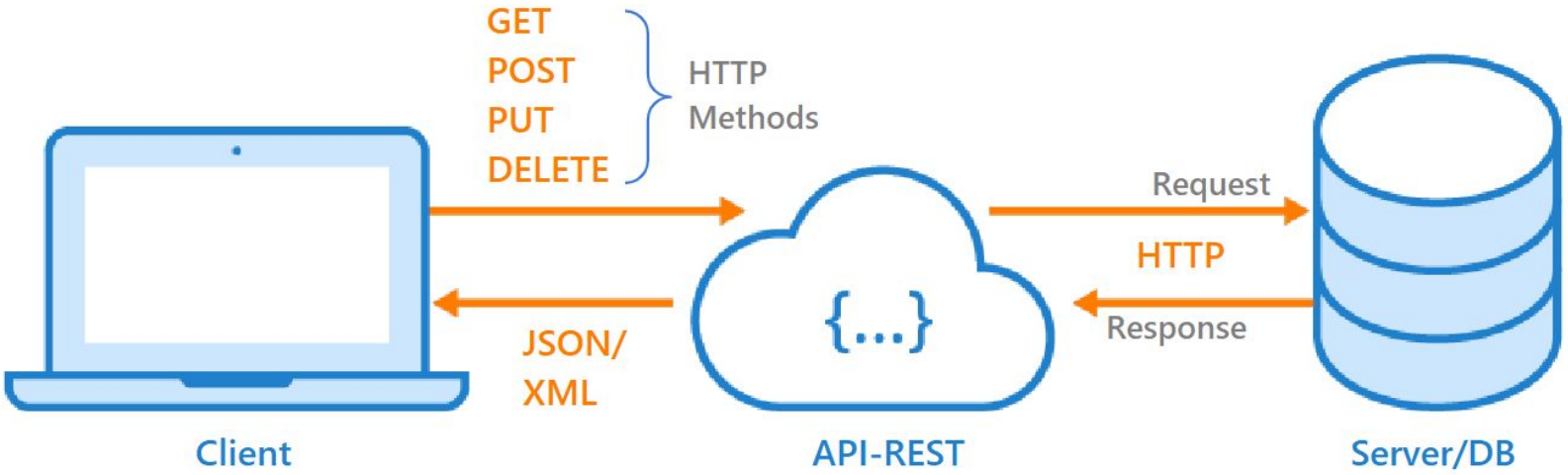
Utilizaremos solo ***Flask*** para exponer las funcionalidades de un modelo de Machine Learning:

- Extensión de Flask que permite generar ***APIs-REST*** muy fácilmente.

  ¿API-REST?

  En pocas palabras, son proveedores de datos a los cuales se les envía una petición y ellas retornan una respuesta.

API-RESTful



Una vez activa la API de RESTful, se puede hacer una solicitud (***request)*** a un determinado ***EndPoint*** con ciertos parámetros.

API-RESTful

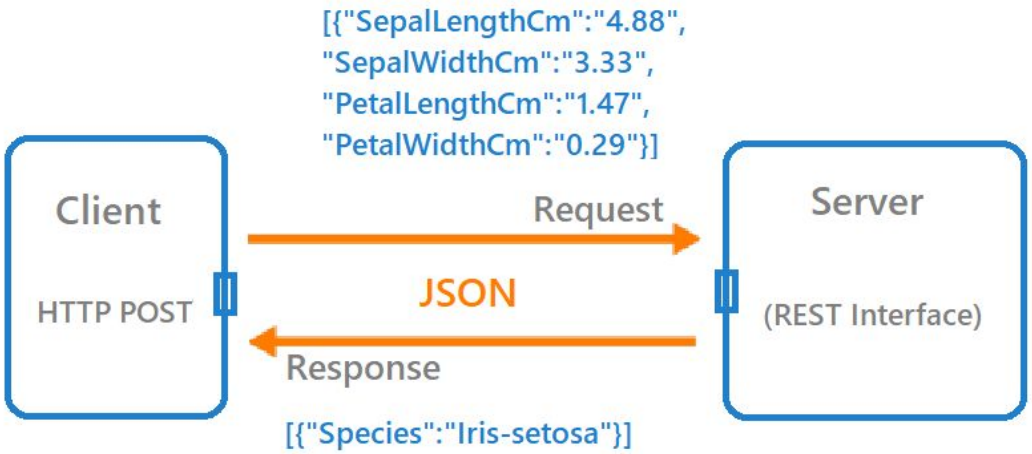
Usando la interfaz REST, el servidor recibe los parámetros, ejecuta la lógica del back end (***predicción del modelo***) y devuelve los resultados al cliente.

¿Lógica de predicción del modelo?

Deployment

Desarrollo de un Servicio Web que ejecute esa lógica

API-RESTful



En nuestro caso, los parámetros de la solicitud son ***instancias de datos*** y la respuesta del servidor son ***predicciones***.

Despliegue del Modelo ML

- ➔ Para predecir tenemos que haber instanciado y entrenado el modelo (***.fit***).
- ➔ Cuando entrenamos, el modelo queda almacenado en memoria RAM.
- ➔ Para predecir, llamamos el método ***.predict*** de ese modelo.

¿Qué pasa si llamamos el método ***.predict*** y el modelo NO está cargado en memoria?

¿Entrenamos de nuevo?

- Costoso
- Bajo rendimiento
- Inviable desde el punto de vista de las APIs.

Despliegue del Modelo ML

La solución es ***persistir*** el modelo...

*"Capacidad de **almacenar y recuperar** el estado de los objetos, de forma que sobrevivan a los procesos que lo manipulan"*

Algunas librerías de Python permiten guardar nuestros modelos ya entrenados para luego utilizarlos cuando los necesitemos (***Joblib***).

Instanciamos →	*import joblib*
Entrenamos→	*clf_rf.fit(X_train,y_train)*
Almacenamos →	*joblib.dump(clf_rf, 'modelo_entrenado.model')*
Cargamos →	*clf_rf=joblib.load('modelo_entrenado.model')*

Despliegue del Modelo ML - Ejemplo

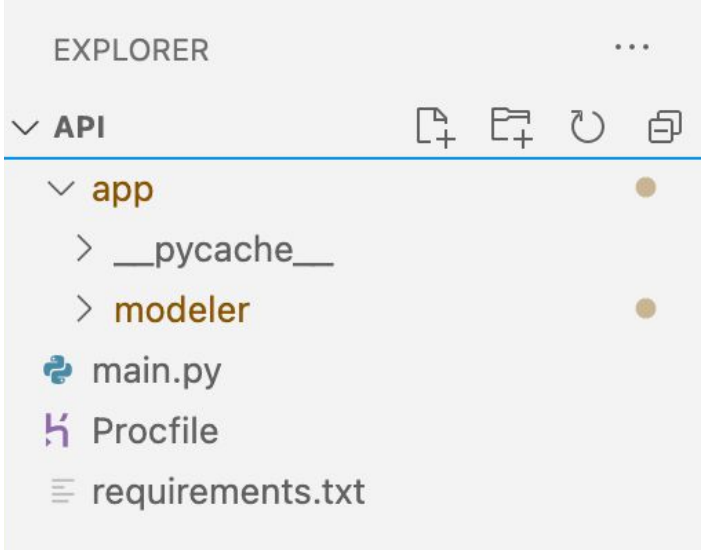
Desarrollar y desplegar un modelo de ML para predecir especies de flores a partir del dataset de Iris.

Pasos:

1. Crear la estructura de directorios de la aplicación
2. Desarrollar el modelo a implementar
3. Desarrollar el API-RESTful
4. Usar un cliente Rest

Despliegue del Modelo ML - Ejemplo

1. Crear la estructura de directorios de la aplicación



Despliegue del Modelo ML - Ejemplo

2. Desarrollar el modelo a implementar

modeler.py M

app > modeler > modeler.py > Modeler

```python
import os
import joblib
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

class Modeler:
    def __init__(self):
        self.df = pd.read_csv('https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv')
        try: self.model = joblib.load('iris.model')
        except: self.model = None

    def fit(self):
        print('Entrenando el Modelo')
        if not os.path.exists('iris.model'):
            x = self.df.drop('species', axis=1)
            y = self.df['species']
            self.model = DecisionTreeClassifier().fit(x, y)
            print('Persistiendo el Modelo')
            joblib.dump(self.model, 'iris.model')

    def predict(self, measurement):
        print(len(measurement))
        if not os.path.exists('iris.model'):
            raise Exception('Por favor, entrene el modelo...')

        prediction = self.model.predict([measurement])
        return prediction[0]

```

Despliegue del Modelo ML - Ejemplo

3. Desarrollar el API-RESTful

- Flask se estructura de una manera que cada endpoint debe tener ruta.
- Definimos funciones correspondientes al tipo de solicitud que realiza el cliente (***POST*** y ***GET*** las más populares).
- Instanciamos la clase ***Modeler*** y se llama la función ***.predict()*** con los datos de entrada.
- Devolvemos una representación JSON de la predicción.
- Conectamos la clase ***Predict*** con el endpoint.
- Hacemos que la app sea ejecutable.

main.py

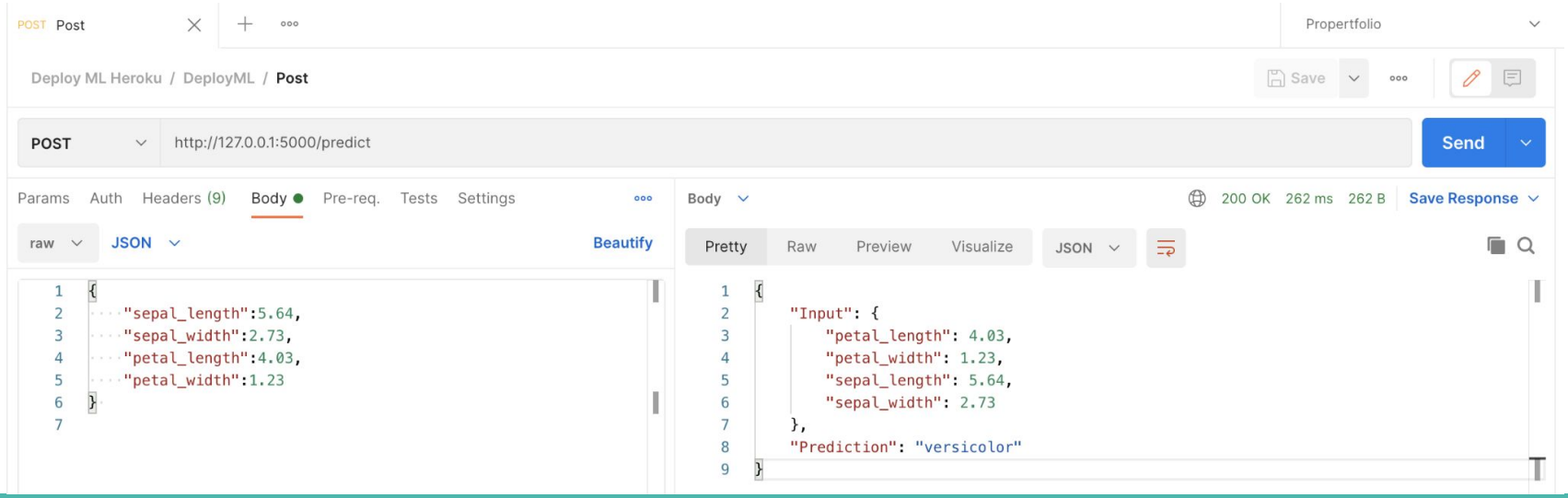
main.py > predict

```python
import os
import joblib
from flask import Flask, jsonify, request
from app.modeler.modeler import Modeler

app = Flask(__name__)

@app.route('/', methods=['GET'])
def index():
    return "Método GET de Prueb con Flask"

@app.route('/predict', methods=['POST'])
def predict():
    request_data = request.get_json()
    sepal_length = request_data['sepal_length']
    sepal_width = request_data['sepal_width']
    petal_length = request_data['petal_length']
    petal_width = request_data['petal_width']

    m = Modeler()
    m.fit()
    app.logger.debug('Modelo Entrenado...')
    prediction = m.predict([sepal_length, sepal_width, petal_length, petal_width])
    app.logger.debug('Prediction' + prediction)
    return jsonify({
            'Input' : {
                'sepal_length' : sepal_length,
                'sepal_width' : sepal_width,
                'petal_length' : petal_length,
                'petal_width' : petal_width,
            },
            'Prediction': prediction
        })


if __name__ == '__main__':
    app.run()

```

Despliegue del Modelo ML - Ejemplo

4. Usar un cliente Rest (Postman)

- La API se ejecuta ahora en localhost:5000
- Para probar si funciona correctamente, usaremos una app cliente rest.
- Este endpoint funciona de la misma forma, independientemente del tipo de aplicación cliente



Caso de uso: AWS

Veamos la ejecución...

Despliegue del Modelo ML - Complejidad

Hoy en día, la puesta en producción de un software se lleva a cabo usando prácticas y herramientas DevOps:

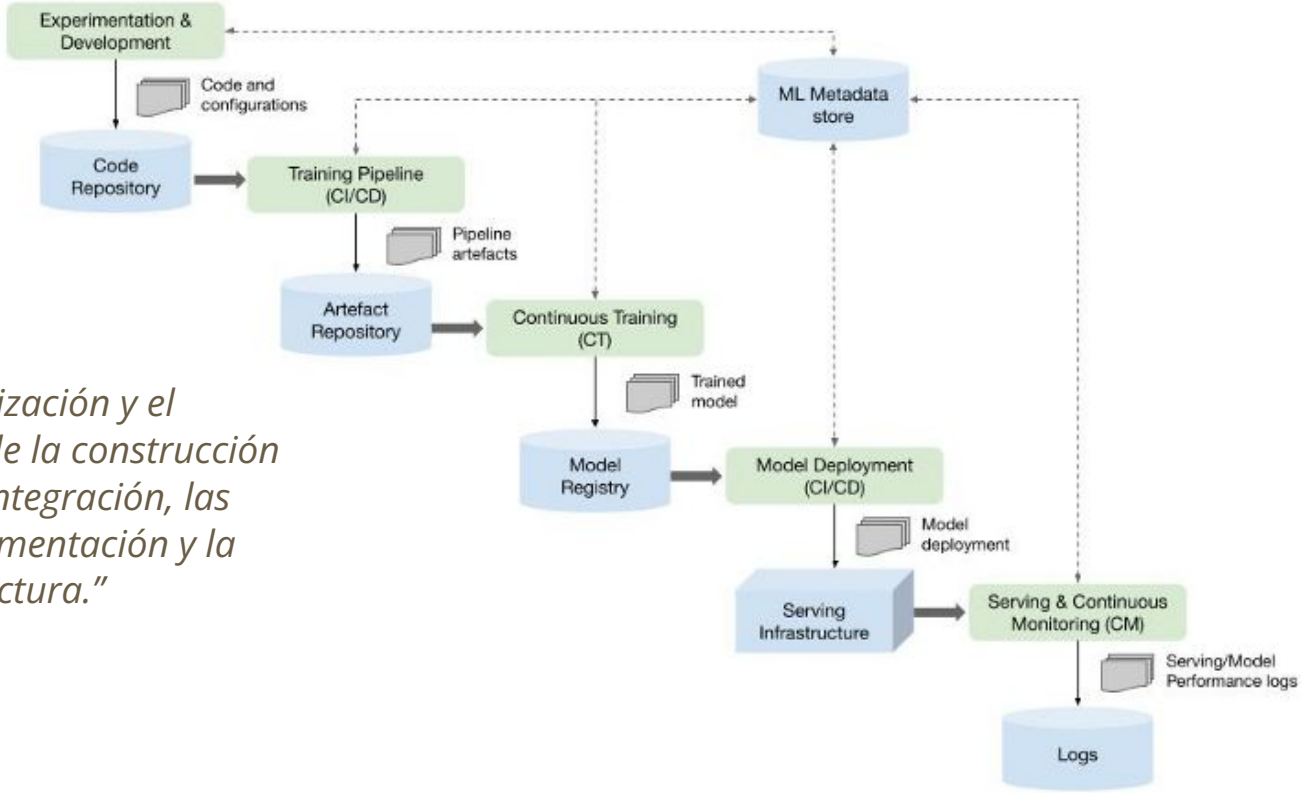
- Automatización y monitoreo de todos los pasos de la construcción de un software, desde la integración (CI), las pruebas y la liberación (CD) hasta la implementación y administración de la infraestructura.
- Requiere de varias herramientas, por ejemplo: Jenkins, GIT, Sonarqube, en fin...

Despliegue del Modelo ML - Complejidad

Ahora, un modelo de ML no es un software común:

- Tiene una complejidad asociada que no la tiene el desarrollo de software tradicional: *El comportamiento del modelo en producción es Impredecible.*
  - ML no es solo código, es código más datos (**Robustez** o sensibilidad a algún cambio en los datos de entrada)
  - El código está en un ambiente controlado (versionador, plano 1), mientras que los datos vienen del mundo real (plano 2), “están en planos separados”.
  - El reto de ML es crear un “puente” entre esos dos planos de manera controlada.
- Requiere combinar prácticas de ***DevOps*** e Ingeniería de Datos, agregando algunas que sean exclusivas de ML: ***ML Ops***

DevOps vs. ML Ops



“ML Ops aboga por la automatización y el monitoreo en todos los pasos de la construcción del sistema de ML, incluida la integración, las pruebas, la liberación, la implementación y la administración de la infraestructura.”

Fuente: **tds**