

Instalación en Nelfify

Para desplegar una aplicación Python que use las librerías Flask, pandas, flask_cors y StringIO en Netlify, debes asegurarte de que todas estas dependencias estén correctamente listadas en tu archivo requirements.txt y configurar adecuadamente tu proyecto.

1. Preparar tu proyecto

Estructura de archivos:

Asegúrate de que tu proyecto tiene la siguiente estructura:

```
mi_app/  
├── app.py # archivo principal de la aplicación Flask  
├── requirements.txt # dependencias de Python  
└── netlify.toml # archivo de configuración de Netlify
```

Contenido de app.py

Aquí tienes un ejemplo básico de una aplicación Flask que usa las librerías que mencionaste (Flask, request, jsonify, pandas, CORS, StringIO):

```
from flask import Flask, request, jsonify  
import pandas as pd  
from flask_cors import CORS  
from io import StringIO  
  
app = Flask(__name__)  
CORS(app) # Habilitar CORS si es necesario  
  
@app.route('/upload', methods=['POST'])  
def upload_file():  
    # Recuperar el archivo CSV desde la solicitud  
    file = request.files['file']  
    # Leer el archivo CSV usando pandas  
    df = pd.read_csv(StringIO(file.stream.read().decode('utf-8')))
```

```
    # Ejemplo: devolver las primeras filas del CSV  
    return jsonify(df.head().to_dict())
```

```
if __name__ == "__main__":  
    app.run(debug=True)
```

Este es un ejemplo simple que:

Recibe un archivo CSV a través de un POST request.
Utiliza pandas para leer el archivo CSV.
Devuelve las primeras filas del archivo CSV como un objeto JSON.

2. Archivo requirements.txt

Para que Netlify pueda instalar las dependencias necesarias para ejecutar tu aplicación Flask, debes crear un archivo requirements.txt que incluya las librerías necesarias:

```
Flask==2.0.1
flask-cors==3.1.1
pandas==1.3.3
gunicorn==20.1.0
```

Asegúrate de usar las versiones de las librerías que sean compatibles con tu proyecto. Puedes generar este archivo automáticamente ejecutando:

```
pip freeze > requirements.txt
```

3. Archivo netlify.toml

Este archivo de configuración le dice a Netlify cómo manejar el despliegue de tu aplicación. Crea un archivo llamado netlify.toml en la raíz de tu proyecto y agrega el siguiente contenido:

```
[build]
publish = "/"
command = "pip install -r requirements.txt && gunicorn app:app"

[context.production.environment]
GUNICORN_CMD_ARGS = "--bind 0.0.0.0 --workers 4"
```

Este archivo configura Netlify para que ejecute el comando `pip install -r requirements.txt` para instalar las dependencias y luego ejecute la aplicación Flask usando Gunicorn. El parámetro `--bind 0.0.0.0` es necesario para que Gunicorn pueda escuchar en todas las interfaces de red.

4. Sube tu proyecto a un repositorio Git

Si no has subido tu proyecto a un repositorio Git (GitHub, GitLab o Bitbucket), hazlo ahora:

```
git init
git add .
git commit -m "Primer commit"
git remote add origin
git push -u origin master
```

5. Conecta tu repositorio a Netlify

Inicia sesión en Netlify.

Haz clic en "New site from Git" para crear un nuevo sitio.

Conecta tu cuenta de GitHub, GitLab o Bitbucket y selecciona el repositorio que contiene tu

aplicación.

En la sección "Build settings", asegúrate de configurar el directorio de construcción como / (si tu archivo app.py está en la raíz del repositorio).

Configura el comando de construcción como:

```
pip install -r requirements.txt && gunicorn app:app
```

6. Desplegar tu aplicación

Una vez que Netlify haya configurado el proyecto, hará una construcción automática e intentará desplegar tu aplicación. Si todo está correctamente configurado, deberías ver tu aplicación Python funcionando en la URL proporcionada por Netlify.

7. Accede a tu aplicación

Cuando Netlify complete el despliegue, te proporcionará una URL en su dominio (algo como <https://tu-sitio.netlify.app>) donde podrás acceder a tu aplicación Python.

Notas adicionales:

Netlify es más adecuado para aplicaciones estáticas, pero es posible usarlo con servicios de backend simples usando Flask y Gunicorn.

Si tu aplicación requiere más configuraciones avanzadas de backend (por ejemplo, bases de datos o servicios más complejos), puedes considerar plataformas como Heroku, AWS, o Google Cloud.

Este flujo te permitirá desplegar una aplicación Python con Flask, pandas, y flask-cors en Netlify.

Commit con problemas

El error que estás viendo indica que intentaste subir un archivo (resultados_finales.csv) que supera el límite de tamaño de archivo de GitHub de 100 MB. GitHub no permite cargar archivos que superen este tamaño directamente en el repositorio.

Para solucionar este problema, tienes dos opciones:

Opción 1: Usar Git LFS (Git Large File Storage)

Git LFS es una extensión de Git que permite gestionar archivos grandes, como los CSV que mencionas. Con Git LFS, los archivos grandes no se almacenan directamente en el repositorio, sino que se almacenan en un servidor separado, mientras que en el repositorio solo se mantiene una referencia a esos archivos.

Pasos para usar Git LFS:

Instalar Git LFS: Si aún no tienes instalado Git LFS, primero necesitas instalarlo. Puedes hacerlo con el siguiente comando (dependiendo de tu sistema operativo):

Linux (con apt):

`sudo apt-get install git-lfs`

macOS (con brew):

```
brew install git-lfs
```

Windows: Descarga e instala Git LFS para Windows.

Configurar Git LFS en tu repositorio: Una vez instalado, inicializa Git LFS en tu repositorio con el siguiente comando:

```
git lfs install
```

Rastrear el archivo grande: Ahora, debes decirle a Git LFS que debe gestionar el archivo `resultados_finales.csv` (o cualquier otro archivo grande) con el siguiente comando:

```
git lfs track "resultados_finales.csv"
```

Esto crea un archivo `.gitattributes` en tu repositorio que le indica a Git LFS qué archivos deben ser gestionados.

Agregar y confirmar el archivo: A continuación, agrega y confirma los cambios. Asegúrate de que el archivo `resultados_finales.csv` esté siendo rastreado por Git LFS:

```
git add .gitattributes  
git add resultados_finales.csv  
git commit -m "Configurar Git LFS para archivos grandes"
```

Subir el archivo a GitHub: Ahora puedes intentar hacer el push nuevamente:

```
git push origin main
```

Git LFS se encargará de subir el archivo grande a su almacenamiento en la nube y GitHub solo almacenará una referencia.

Verificación:

Puedes verificar que Git LFS esté funcionando correctamente con:

```
git lfs ls-files
```

Esto debería mostrar los archivos que Git LFS está gestionando.

Opción 2: Reducir el tamaño del archivo antes de subirlo

Si no deseas usar Git LFS, otra opción sería reducir el tamaño de tu archivo CSV. Algunas sugerencias incluyen:

Dividir el archivo en partes más pequeñas: Puedes dividir el archivo CSV en varios archivos más pequeños (por ejemplo, dividirlo por fecha o categorías) para que cada archivo sea más pequeño y se ajuste al límite de tamaño de GitHub.

Comprimir el archivo: Puedes comprimir el archivo CSV en un archivo .zip o .gzip antes de subirlo. Sin embargo, ten en cuenta que GitHub aún tiene límites para archivos comprimidos.

Resumen:

Usar Git LFS es la mejor solución si necesitas almacenar archivos grandes en tu repositorio. Te permite manejar archivos mayores de 100 MB sin problemas. Reducir el tamaño del archivo o comprimirlo también es una opción si no deseas usar Git LFS.

Te recomiendo optar por Git LFS si planeas trabajar con archivos grandes regularmente.