Reporte de Desarrollo de Aplicación Web en Streamlit

Autor:
Oscar Hernández terán

Fecha: 3 de mayo de 2025

Índice general

Índice de figuras

Capítulo 1

Introducción

En el presente documento se expone de manera detallada el proceso de desarrollo, la metodología de ejecución y la arquitectura tecnológica empleada en la construcción de una aplicación web diseñada para la visualización interactiva de datos relacionados con emisiones de dióxido de carbono (CO_2). Se describen los componentes principales del sistema, las herramientas utilizadas, así como las mejores prácticas implementadas para garantizar su correcto funcionamiento. Asimismo, se proporcionan instrucciones precisas para la instalación, despliegue y utilización de la aplicación, con el objetivo de facilitar su reproducción y adaptación a diferentes entornos o necesidades de análisis.

Capítulo 2

Tecnologías Utilizadas

La vista general de la aplicación es la siguiente:

Figura 2.1: Vista de la aplicación

El código fuente completo de la aplicación se puede clonar desde el repositorio Visualizador de CO2.

- **Streamlit** para la creación de la interfaz web.
- **Pandas** para la manipulación de datos.
- **Docker** para la contenerización de la aplicación.

2.1. Stremalit

Streamlit es un framework de código abierto en Python diseñado específicamente para la creación rápida y sencilla de aplicaciones web enfocadas en ciencia de datos y visualización de información. Permite transformar scripts de Python en aplicaciones interactivas con apenas unas pocas líneas de código, sin necesidad de conocimientos avanzados en desarrollo web.

Entre sus principales ventajas destacan:

1. **Facilidad de uso**: Su sintaxis intuitiva permite a los desarrolladores centrarse en el análisis de datos en lugar de en el diseño de interfaces.
2. **Integración directa con bibliotecas de datos**: Se conecta de forma natural con herramientas como Pandas, NumPy, Matplotlib y otros frameworks de visualización.
3. **Despliegue rápido**: Las aplicaciones pueden ser ejecutadas localmente o desplegadas en la nube en cuestión de minutos.
4. **Interactividad**: Proporciona widgets y controles para crear interfaces dinámicas y reactivas de manera sencilla.
5. **Actualización automática**: Al modificar el código, la aplicación se actualiza automáticamente, acelerando el ciclo de desarrollo.

2.2. Docker

Por otro lado, **Docker** es una plataforma de virtualización ligera que permite empaquetar aplicaciones y sus dependencias en contenedores portables y autocontenidos. Estos contenedores garantizan que la aplicación se ejecute de manera consistente en cualquier entorno, independientemente del sistema operativo o la configuración subyacente.

Entre las principales ventajas de utilizar Docker en este proyecto destacan:

1. **Portabilidad**: El contenedor puede ser ejecutado localmente, en servidores o en plataformas en la nube como Google Cloud Platform (GCP), facilitando el despliegue en servicios como App Engine.

2. **Consistencia**: Docker asegura que el entorno de ejecución sea el mismo en todas las etapas del ciclo de vida de la aplicación, reduciendo errores causados por diferencias de configuración.

3. **Facilidad de despliegue**: Permite empaquetar la aplicación en una imagen lista para ser desplegada con comandos simples, lo que acelera el proceso de migración a producción.

4. **Eficiencia de recursos**: A diferencia de las máquinas virtuales tradicionales, los contenedores son más livianos, rápidos de iniciar y consumen menos recursos.

5. **Aislamiento**: Cada contenedor opera de forma independiente, lo que mejora la seguridad y la estabilidad del sistema.

En el contexto de este proyecto, Docker facilita no solo la ejecución local de la aplicación web desarrollada en Streamlit, sino también su preparación para ser desplegada de manera eficiente en servicios de nube como App Engine de GCP, optimizando el flujo de trabajo y garantizando escalabilidad futura.

Capítulo 3

Arquitectura de la Aplicación

El desarrollo de la aplicación se implementó siguiendo una arquitectura monolítica, en la cual todos los componentes —incluyendo el procesamiento de datos, la renderización de mapas y la gestión de capas— se ejecutan de manera conjunta desde un único archivo principal en Python (`app.py`). Para favorecer la organización, la mantenibilidad y la legibilidad del proyecto, los módulos de funciones auxiliares, los conjuntos de datos y los notebooks de pruebas se estructuran en directorios independientes, tal como se ilustra en la Figura 5.3.

Para la inspección y el análisis del código fuente, se recomienda utilizar un entorno de desarrollo integrado (IDE) que facilite la navegación entre módulos y funciones, particularmente mediante atajos que permitan acceder directamente a las definiciones. En el caso de Visual Studio Code (VS Code), esta funcionalidad puede activarse utilizando el atajo `Ctrl + Click` sobre la función o método de interés.

El archivo **app.py** actúa como punto central de la aplicación, conteniendo todas las importaciones necesarias, la definición de la función principal `main()`, así como el *entry point* que habilita la ejecución del programa.

```
1 if __name__ == "__main__":
2     main()
```

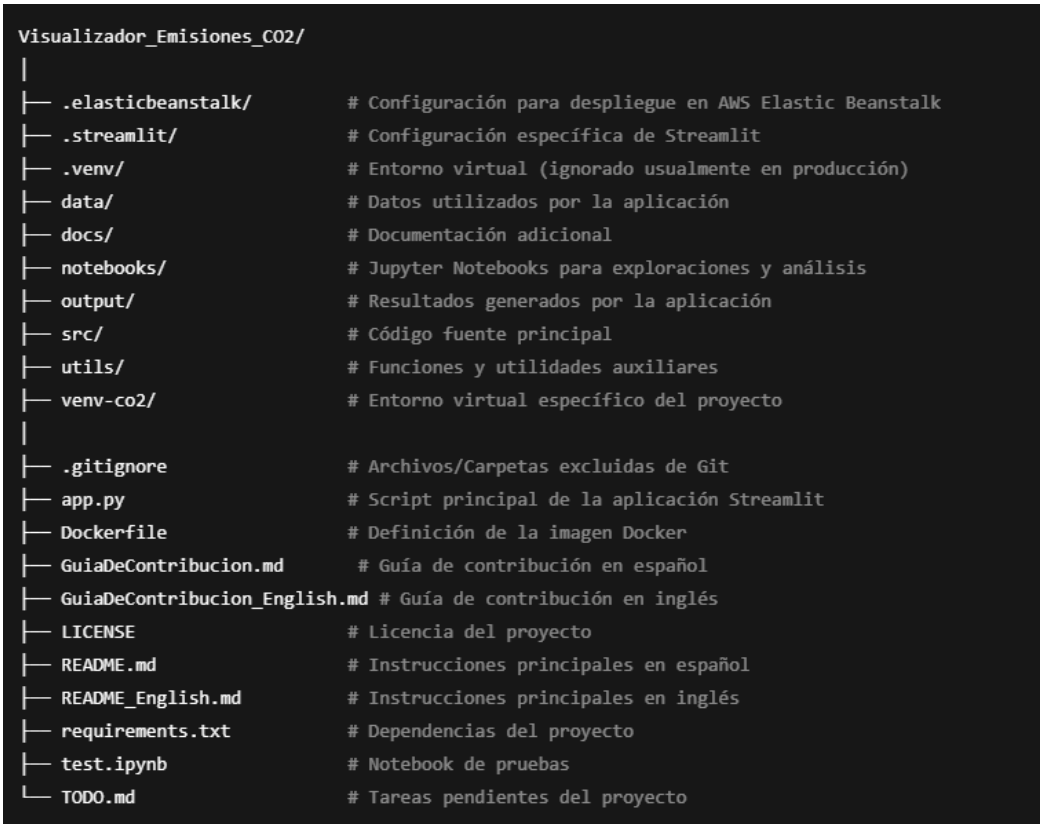

Figura 3.1: Estructura de carpetas

Como se mencionó, la aplicación cuenta con su respectivo archivo **Dockerfile** para la ejecución como contenedor, a continuación una descripción detallada de los comandos y el archivo completo.

1. **Directorio de Trabajo:** Se define `/visualizador` como el directorio de trabajo dentro del contenedor. Todas las operaciones posteriores (como la instalación de paquetes y la ejecución de la aplicación) se realizan desde esta ubicación:

```
1 WORKDIR /visualizador
```

2. **Copia de Archivos:** El contenido del directorio local se copia en su totalidad al directorio de trabajo del contenedor, asegurando que el

código fuente y los archivos de configuración estén disponibles durante el proceso de construcción:

```
COPY . /visualizador
```

3. **Actualización de `pip`:** Se actualiza la herramienta `pip` a su última versión para evitar posibles problemas de compatibilidad o advertencias durante la instalación de dependencias:

```
RUN pip install --upgrade pip
```

4. **Instalación de Dependencias:** Se instalan todas las dependencias necesarias listadas en el archivo `requirements.txt` de manera eficiente, utilizando el parámetro `-no-cache-dir` para reducir el tamaño final de la imagen:

```
RUN pip install --no-cache-dir -r requirements.txt
```

5. **Exposición de Puertos:** Se expone el puerto 8080 para permitir la comunicación externa con la aplicación web desplegada. Aunque Streamlit utiliza el puerto 8501 por defecto, en entornos como Google App Engine (GCP) se requiere mapear la aplicación a un puerto específico (comúnmente el 8080):

```
EXPOSE 8080
```

6. **Comando de Ejecución:** Finalmente, se define el comando que inicia la aplicación ejecutando `Streamlit` sobre el archivo principal `app.py`:

```
CMD ["streamlit", "run", "app.py"]
```

```
# Usa una imagen base de Python
FROM python:3.10

# Configura el directorio de trabajo dentro del contenedor
WORKDIR /visualizador

# Copia los archivos de la aplicación al contenedor
COPY . /visualizador

# Actualización de pip en contenedor
RUN pip install --upgrade pip

# Instala las dependencias de la aplicación
RUN pip install --no-cache-dir -r requirements.txt

# Expone el puerto que utiliza Streamlit (8501 por defecto)
EXPOSE 8080

# Define el comando para iniciar la aplicación
CMD ["streamlit", "run", "app.py"]

```

Capítulo 4

Implementación y ejecución

4.1. Código Fuente Principal

4.1.1. Aplicación principal

Este archivo contiene la lógica central de la aplicación y organiza el flujo de navegación entre diferentes páginas.

Funcionalidades principales:

- Configura la página de Streamlit para usar todo el ancho disponible
- Maneja un sistema de navegación mediante estados de sesión
- Carga datos de ubicaciones desde archivos JSON
- Muestra diferentes vistas según la página seleccionada por el usuario

Proceso de ejecución:

- Define la función main() que configura y ejecuta la aplicación
- Inicializa variables de estado para el control de navegación
- Carga la lista de nombres de ubicaciones desde un archivo JSON
- Renderiza el contenido correspondiente según la página actual seleccionada:
    1. Página de inicio

2. Información nacional (."Acerca de")
3. Listado de mapas disponibles
4. Mapas individuales de zonas específicas
5. Bibliografía

4.1.2. Sistema de mapas

Este archivo contiene las funciones encargadas de crear y visualizar mapas interactivos usando las bibliotecas Folium y Streamlit. Funciones principales:

grafico_de_puntos(file: str)

- Carga datos de ubicaciones desde un archivo JSON
- Convierte la información a un DataFrame de pandas
- Crea un mapa centrado en México utilizando Folium con imágenes satelitales de Esri
- Genera una cuadrícula (grid) sobre el mapa para visualizar áreas
- Coloca marcadores en el mapa para cada ubicación
- Muestra el mapa interactivo dentro de un contenedor de Streamlit

plot_map(file: str)

- Convierte coordenadas UTM a latitud/longitud
- Crea un mapa centrado en México
- Genera una cuadrícula para visualizar áreas
- Añade marcadores por cada punto de datos con información sobre los valores de CO2
- Muestra el mapa interactivo en Streamlit

mapas_individuales(file)

- Carga datos específicos de una ubicación desde un archivo CSV formateado
- Crea un mapa centrado en la ubicación promedio de los puntos de datos
- Configura una cuadrícula visual para segmentar el área
- Añade marcadores con información detallada sobre los valores de CO2 en cada punto
- Ajusta los límites del mapa para mostrar todos los puntos de datos
- Presenta el mapa interactivo en un contenedor de Streamlit

4.1.3. Gestión de páginas y vistas

Estos archivos contienen las funciones que definen la estructura visual y la navegación de la aplicación. Funciones principales en vistas.py:

definir_pagina_actual()

- Configura la barra lateral con botones de navegación
- Define las acciones para cambiar entre diferentes páginas de la aplicación

contenido_principal()

- Muestra la página principal con logos, encabezados y descripción del proyecto
- Organiza los elementos visuales en columnas

encabezado_mapa_individual(zona)

- Muestra el título y descripción para un mapa específico de una zona

bibliografía()

- Presenta recursos bibliográficos y permite la descarga de archivos de datos crudos
- Muestra una tabla con información de las ubicaciones disponibles

- Implementa botones de descarga para archivos CSV

listado_mapas()

- Permite al usuario seleccionar una ubicación específica de una lista desplegable
- Navega al mapa correspondiente cuando se selecciona una ubicación

4.1.4. Funciones en pages.py:

mostrar_datos_nacionales()

- Muestra un mapa nacional con la distribución de las principales zonas de medición
- Utiliza la función grafico_de_puntos() para visualizar los datos

4.2. Guía de Ejecución: Sistema de Visualización de Emisiones de CO2

Esta guía explica cómo ejecutar la aplicación de visualización de emisiones de CO2 en dos entornos diferentes: directamente con Python y Streamlit, y mediante contenedores Docker.

4.2.1. Sección 1: Ejecución con Python y Streamlit

Requisitos Previos

- Python 3.8 o superior instalado
  1. Verifica tu versión: python –version o python3 –version
  2. Descarga Python desde python.org si es necesario
- Gestor de paquetes pip
  1. Generalmente viene con Python, verifica con: pip –version
- Git (opcional)
  1. Para clonar el repositorio si está disponible en un control de versiones

4.2.2. Preparación del Entorno

1. **Clonar o descargar el proyecto**

```
git clone <url-del-repositorio>
```

2. **Crear un entorno virtual (recomendado)**

```
# En Linux/macOS
python3 -m venv venv
source venv/bin/activate

# En Windows
python -m venv venv
venv\Scripts\activate
```

3. **Instalar dependencias:** Instalar las dependencias necesarias para el proyecto específicadas en el archivo **requirements.txt**

```
pip install -r requirements.txt
```

4.2.3. Ejecución de la Aplicación

- **Navega al directorio del proyecto**

```
cd ruta/al/proyecto
```

- **Ejecuta la aplicación con Streamlit**

```
streamlit run app.py
```

- **Accede a la aplicación:** Streamlit abrirá automáticamente un navegador con la aplicación. Si no se abre, ve a http://localhost:8501 en tu navegador.

4.2.4. Ejecución con Docker

Requisitos previos

1. **Docker instalado:**
   - Docker Desktop para Windows y macOS
   - Docker Engine para Linux
   - Verifica la instalación con: docker –version
2. **Construcción y Ejecución del Contenedor**
   - Construir la imagen

```
docker build -t co2-visualization-app .
```

   - Ejecutar el contenerdor

```
docker run -p 8501:8501 co2-visualization-app
```

   - Acceder a la aplicación: Abre el navegador y ve a http://localhost:8501

Gestión del contenedor

1. Detener el contenedor:

```
# Si usas Docker directamente
docker stop <container-id>

# Si usas Docker Compose
docker-compose down
```

2. Ver logs:

```
# Si usas Docker directamente
docker logs <container-id>

# Si usas Docker Compose
docker-compose logs
```

3. Reconstruir la imagen (después de cambios):

```
# Si usas Docker directamente
docker build -t co2-visualization-app .

# Si usas Docker Compose
docker-compose build
```

Capítulo 5

Visualizaciones

La aplicación web desarrollada permite la visualización interactiva de distintos mapas geográficos, los cuales constituyen la base espacial para representar las emisiones de dióxido de carbono (CO_2) y otros datos asociados. En total, se utilizan tres versiones principales de mapas, cada una con características y finalidades distintas. A continuación se describe su origen, utilidad y acceso.

5.1. 1. Mapa Base Cartográfico General (OpenStreetMap)

- Fuente: Se obtiene a través de la plataforma libre OpenStreetMap.
- Función principal: Proporciona una representación general del territorio, útil como referencia visual básica para ubicar zonas geográficas, caminos, localidades y áreas administrativas.
- Utilidad en la aplicación: Sirve como mapa de fondo para las capas interactivas de emisiones y facilita la navegación para usuarios no especializados.
- Ventaja: Es liviano, gratuito y altamente compatible con bibliotecas como folium y leaflet.

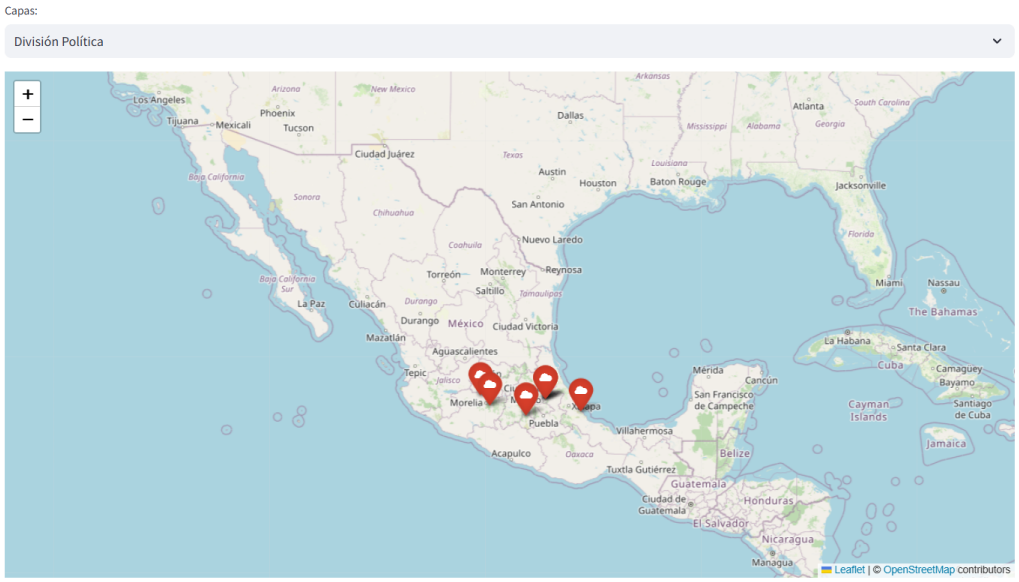

Figura 5.1: Capa de División Política

5.2. 2. Mapa Satelital (Esri World Imagery)

Fuente: Servicio de mapas satelitales proporcionado por Esri a través de su capa pública de imágenes: https://services.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer[Mapa Satelital]

- Función principal: Brindar un contexto visual más realista del terreno, incluyendo cobertura vegetal, áreas urbanas y cuerpos de agua.
- Utilidad en la aplicación: Ideal para inspecciones más detalladas, visualización ambiental o estudios en regiones específicas.
- Consideraciones: Al ser más pesado, puede aumentar los tiempos de carga en conexiones lentas.

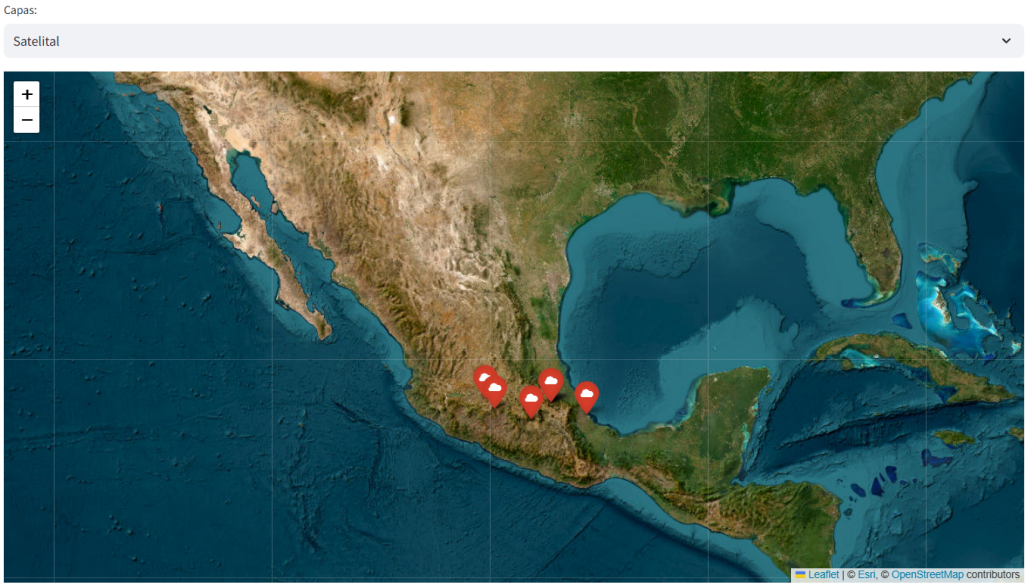

Figura 5.2: Capa Satelital

5.3. 3. Mapa Relieve Híbrido (Satélite + Etiquetas)

- Fuente: Combinación de la capa satelital de Esri con la capa de etiquetas (World Boundaries and Places) también de Esri.
- Función principal: Ofrece imágenes satelitales enriquecidas con nombres de lugares, límites políticos y otros elementos informativos superpuestos.
- Utilidad en la aplicación: Permite al usuario obtener contexto geográfico detallado sin perder información de referencia como nombres de ciudades o fronteras.
- Ventaja: Balance entre detalle visual e información contextual.

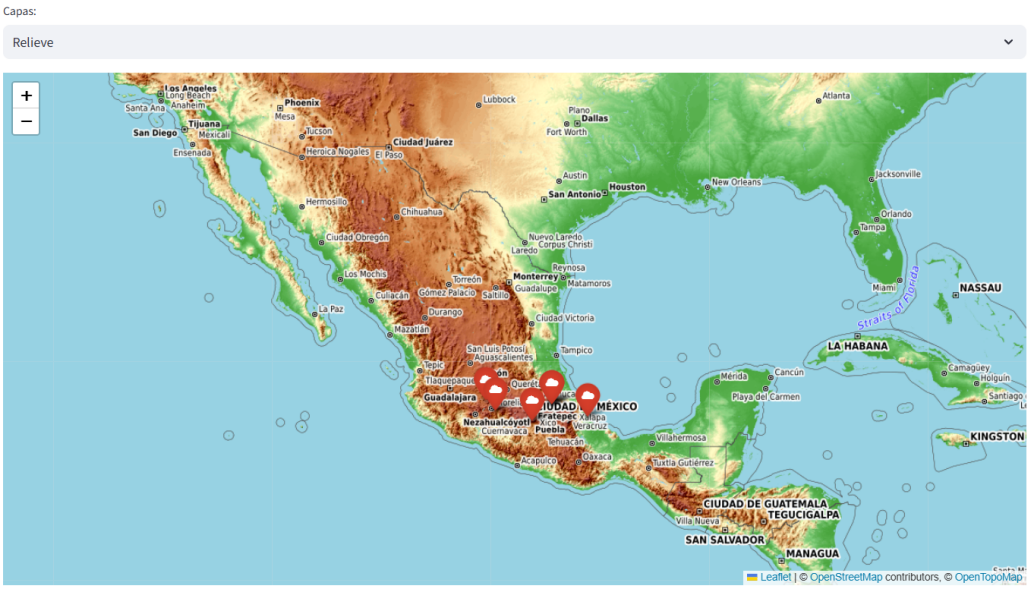

Figura 5.3: Capa de Relieve

5.4. Implementación en la Aplicación

En la implementación técnica, los mapas se integran usando la biblioteca `folium`, la cual permite alternar entre diferentes capas base mediante un control interactivo. Esto otorga flexibilidad al usuario para adaptar la visualización según el análisis deseado.

Capítulo 6

Enlaces de Interés

Puedes incluir enlaces útiles, repositorios, documentación oficial, etc.:

- Streamlit
- Pandas
- Docker
- Repositorio del Proyecto