Bertrán Vidal Campos
Javier Mendoza Guerrero

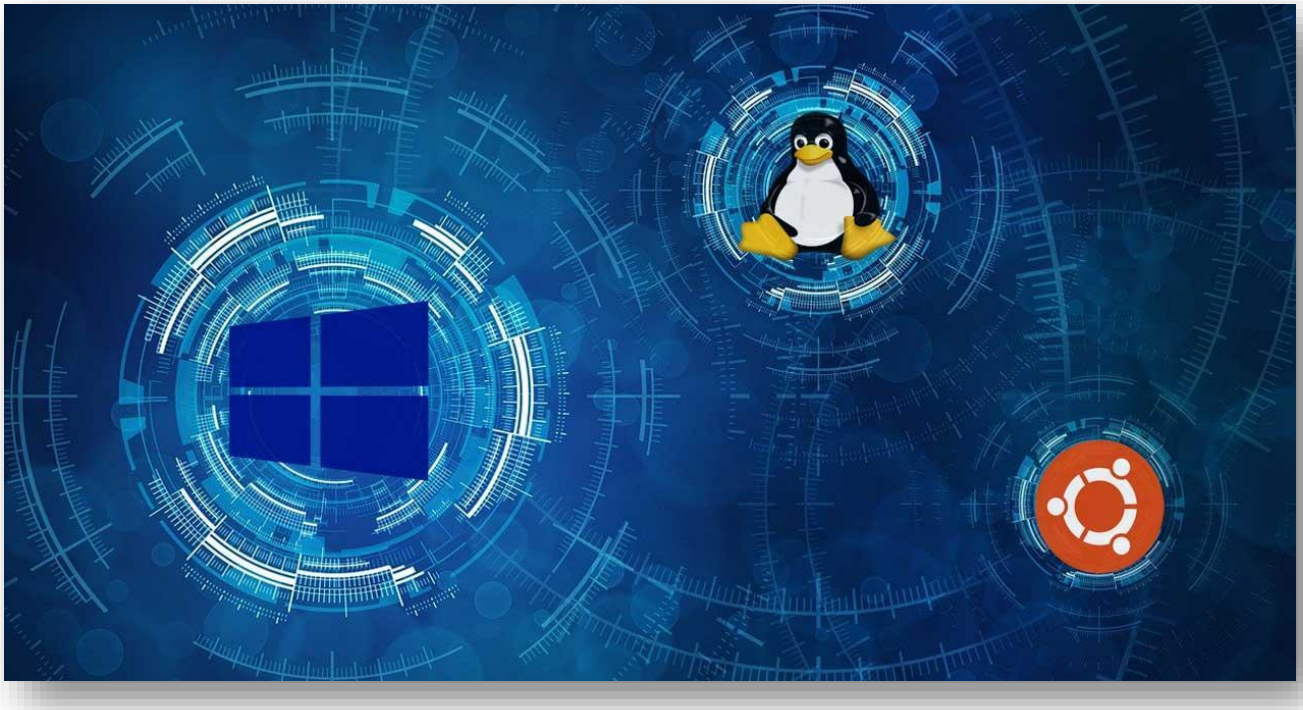
Proyecto Final de Fundamentos de los sistemas Operativos–2º curso

Grado en Ingeniería Matemática e Inteligencia Artificial

Proyecto Fundamentos de los sistemas operativos

Bertrán Vidal Campos

Javier Mendoza Guerrero



Contenido

1. INTRODUCCIÓN

Este proyecto consiste en desplegar una aplicación basada en Flask dentro de un entorno virtual de Alpine Linux. La aplicación implementa varios servicios que permiten realizar análisis y visualización de datos, incluyendo estadísticas, exploración de datos y generación de mapas interactivos. A lo largo del proyecto, se aplican conceptos, como la configuración de máquinas virtuales, la instalación de dependencias, la creación de entornos virtuales, y la automatización mediante scripts de Bash. También se incluye el despliegue de la aplicación en un servidor en la nube.

El desarrollo incluye la implementación de scripts para el procesamiento de datos, como la extracción y filtrado de columnas específicas y la creación de mapas personalizados por usuario. Además, se desarrollan funciones para comparar tiempos de ejecución secuenciales y paralelos, optimizando el rendimiento del procesamiento de datos. El proyecto incorpora el uso de herramientas de Python como requests para realizar peticiones a servicios alojados en Flask, generando resultados, que se almacenan en directorios específicos del sistema.

Finalmente, el proyecto abarca el despliegue en una instancia de máquina virtual en Google Cloud, donde se configuran claves SSH para garantizar la seguridad de las conexiones. Esta fase asegura que el proyecto sea accesible de forma remota, optimizando su usabilidad y demostrando la integración con servicios en la nube.

2. INSTALACIÓN DE PAQUETES ALPINE

```bash
#!/bin/sh

#Modificar repositories para instalar sudo

sed -i '3s/^#//' /etc/apk/repositories

#paquetes

paquetes="python3 python3-dev git nano sudo bash gcc g++ musl-dev linux-headers wget curl htop"


#Comprobar si los paquetes estan instalados

for paquete in $paquetes; do
    if ! apk list --installed | grep -q "$paquete"; then
        echo "Instalando $paquete"
        apk add "$paquete"
    else
        echo "$paquete ya instalado"
    fi
done

#Instalar libc-dev sin comprobar si ya existe

echo "Instalando libc-dev sin comprobacion"
apk add libc-dev
```

Figura 1. Script de Bash Apartado 1

El primer ejercicio del proyecto consiste en desarrollar un script en Bash para instalar y configurar varios paquetes esenciales en Alpine Linux. Este script debe comenzar verificando si cada paquete ya está instalado en el sistema utilizando el comando apk list --installed junto con grep. En caso de que algún paquete no esté instalado, el script procede a instalarlo mediante el comando apk add <nombre_paquete>. Los paquetes que se deben asegurar en el sistema incluyen python3, git, nano, sudo, bash, gcc, libc-dev, g++, musl-dev, linux-headers, wget, curl, y htop.

El diseño del script es eficiente y bien estructurado, ya que automatiza tanto la verificación como la instalación de los paquetes necesarios. Además, el script incluye la configuración de sudo al modificar el archivo /etc/apk/repositories, habilitando el uso de comandos con privilegios elevados para usuarios no root, lo cual es fundamental para la administración del sistema. Este enfoque permite ahorrar recursos al evitar instalaciones innecesarias y asegura que el sistema esté preparado para los siguientes pasos del proyecto.

```
alps:/etc# echo '%wheel ALL=(ALL) ALL' > /etc/sudoers.d/wheel
alps:/etc# adduser alumnoimat wheel
alps:/etc# sudo -lU alumnoimat
User alumnoimat may run the following commands on alps:
    (ALL) ALL
alps:/etc# sudo apk upadate
ERROR: 'upadate' is not an apk command. See 'apk --help'.
alps:/etc# sudo apk update
fetch http://dl-cdn.alpinelinux.org/alpine/v3.20/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.20/community/x86_64/APKINDEX.tar.gz
v3.20.3-264-gf6ca45f2f88 [http://dl-cdn.alpinelinux.org/alpine/v3.20/main]
v3.20.3-265-ge9b28b67a72 [http://dl-cdn.alpinelinux.org/alpine/v3.20/community]
OK: 24165 distinct packages available
```

Figura 2. Añadir permisos de super usuario

3. SCRIPT DE BASH PARA FUNCIONAMIENTO DE UN REPOSITORIO Y UN DESPLIEGUE

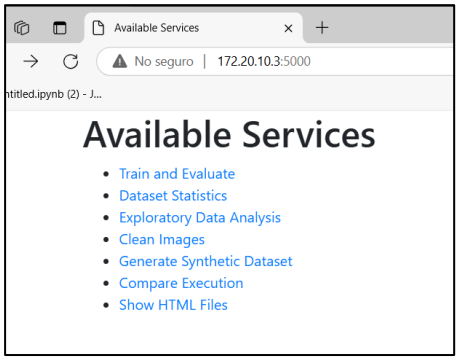


Figura 3. Servicios disponibles en el proyecto Flask

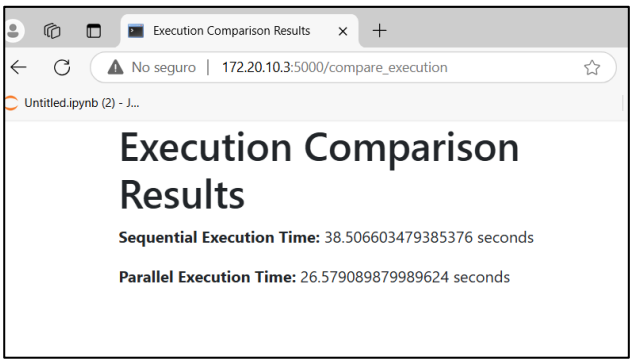


Figura 4. Multiplicación de matrices. Comparativa de resultados

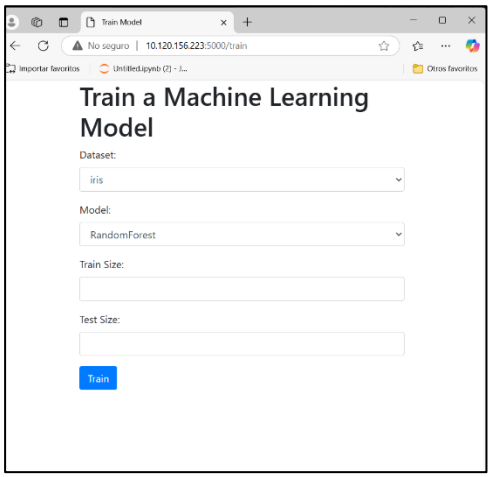


Figura 5. Train and Evaluate

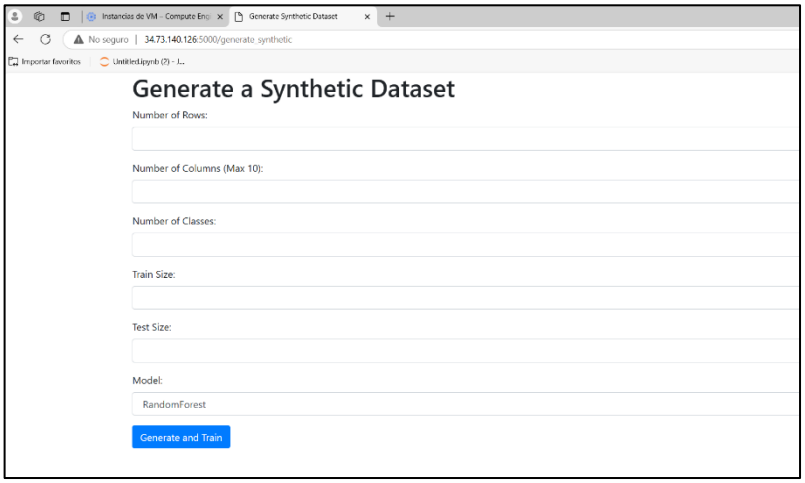


Figura 6. Datasets statistics

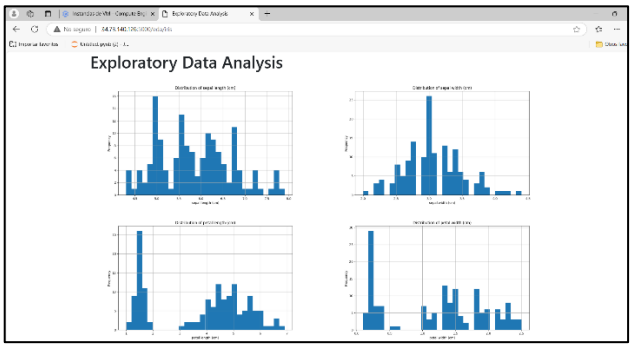


Figura 7. Exploratory Data Analysis

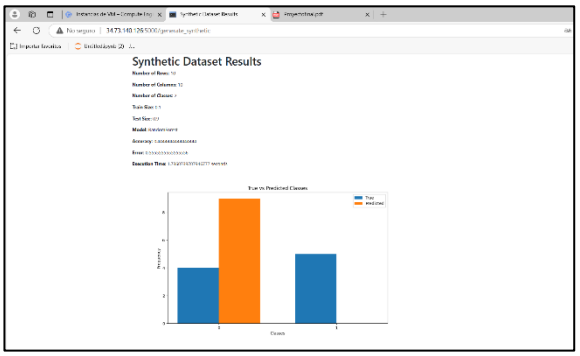


Figura 8. Generate Synthetic Dataset

Despliegue de ProyectoFUSO en un repositorio en git

```
#!/bin/sh


git clone https://github.com/pablosanchezp/ProyectoFuso.git

python3 -m venv entorno_virtual

source entorno_virtual/bin/activate

pip install -r requirements.txt

python3 ProyectoFuso/main.py
```

Figura 9. Script de Bash Apartado 2

```
#Requirements.txt

pandas==2.2.3
scikit-learn==1.5.2
matplotlib==3.9.2
numpy==2.1.3
seaborn==0.13.2
memory-profiler==0.61.0
folium==0.18.0
geopy==2.4.1
tqdm==4.66.6
imgkit==1.2.3
flask==3.0.3
```

Figura 10. Requirements.txt

Este script en Bash es una solución eficiente y completa para el despliegue de la aplicación Flask en la máquina virtual de Alpine, cumpliendo con todos los requisitos del proyecto. En primer lugar, el uso de git clone permite descargar el repositorio directamente, asegurando que se obtenga la versión más reciente del código. La creación del entorno virtual con python3 -m venv entorno_virtual garantiza un entorno aislado, evitando conflictos de dependencias con el sistema principal y permitiendo un control total sobre los paquetes utilizados en el proyecto. Además, la activación del entorno virtual con source entorno_virtual/bin/activate asegura que todas las instalaciones subsecuentes se realicen dentro de este entorno, manteniendo así la integridad del sistema.

Para instalar las dependencias correctamente, fue necesario analizar uno a uno los archivos .py del proyecto Fuso para identificar las librerías específicas que cada módulo requería. Este proceso incluyó investigar en la web y realizar pruebas y errores para determinar qué dependencias eran esenciales y cuáles podían generar conflictos en el entorno de Alpine Linux. Una vez identificadas, las dependencias se listaron en requirements.txt, lo cual permite su instalación automatizada mediante pip install -r requirements.txt. Al final, python3 ProyectoFuso/main.py ejecuta la aplicación Flask, poniendo en marcha el servidor web en la máquina virtual. Este enfoque es particularmente efectivo

porque asegura que todas las dependencias necesarias estén configuradas correctamente y evita problemas durante la ejecución de la aplicación, cumpliendo así con todos los requisitos del apartado de manera estructurada y eficaz.

4. MEJORAS PROYECTO FUSO

El proyecto ProyectoFUSO podría beneficiarse de mejoras en la documentación y la gestión de errores para hacerlo más comprensible y fácil de mantener.El archivo README.md podría ser más completo, incorporando una guía clara de instalación, la estructura del proyecto, ejemplos de ejecución, y soluciones a problemas comunes. Estos cambios harían que el proyecto sea más accesible tanto para nuevos usuarios como para desarrolladores que deseen contribuir.

En términos de gestión de errores, scripts como generate_maps.py y generate_individual_maps.py deberían incluir bloques try-except para manejar posibles fallos, como archivos faltantes, datos mal formateados o errores al guardar los resultados. Por ejemplo, al cargar un archivo, se puede capturar un FileNotFoundError y mostrar un mensaje claro para que el usuario verifique la ruta. Asimismo, al generar o guardar mapas, excepciones como ValueError o IOError pueden ser manejadas para evitar interrupciones abruptas y proporcionar mensajes que ayuden a identificar el problema rápidamente.

Estas mejoras, junto con una guía clara para futuros desarrolladores sobre cómo contribuir, ayudarían a consolidar el proyecto como un código profesional, robusto y colaborativo.

5. CREACIÓN DE UN SCRIPT PARA LA DESCARGA DE DATOS

En este apartado, el programa empieza definiendo variables que facilitan el manejo de rutas y nombres de archivos, permitiendo reutilizar el código sin necesidad de modificaciones en múltiples lugares. Luego, descargamos el archivo DatasetsGowalla.zip y lo descomprimimos para trabajar con los ficheros de datos de las ciudades. Después, use usa un bucle for que recorre cada uno de los ficheros de datos en el directorio especificado, y para cada uno de ellos se extraen las columnas necesarias con el comando cut. Las columnas 3, 4 y 5 se añaden a un archivo general llamado ALL_LOCATIONS.txt, mientras que las columnas 1, 2 y 5 se redirigen a un archivo individual para cada ciudad.

A continuación, calculamos estadísticas dentro del mismo bucle, como el número de usuarios únicos, lugares distintos y filas totales, usando los comandos cut, sort, y wc -l. También analizamos cuántas interacciones ocurrieron en los meses específicos de 2010-07 y 2010-08 mediante el comando grep. Después de calcular estas estadísticas, si la ciudad actual es una de las requeridas (El Paso, Glasgow, Manchester o Washington DC), se genera un mapa general para esa ciudad llamando al script generate_maps.py. También, seleccionamos un usuario específico para cada ciudad y se generaron mapas individuales con el script generate_individual_maps.py.

Por último, hicimos un script de Python con la librería pandas para procesar los datos filtrados de El Paso y determinar los 5 usuarios más activos ,cuyos IDs se guardan en un archivo de texto. Luego, en otro bucle, se lee este archivo línea a línea y se genera un mapa individual para cada uno de esos usuarios usando otra vez generate_individual_maps.py. Finalmente, todos los mapas generados se guardan en la carpeta html_files para que puedan ser vistos en la aplicación Flask.

Usuarios con 2 o más visitas de las ciudades especificadas que buscamos manualmente

Ciudad	Usuario
El Paso	667
Washington	22
Glasgow	268
Manchester	332

Mapas de ciudades con la función generate_maps

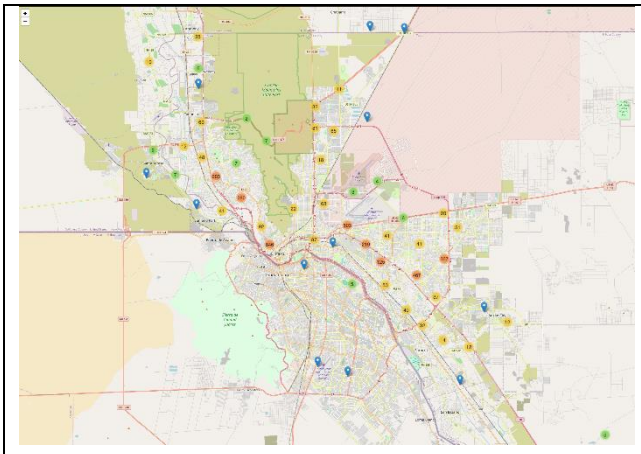


Figura 11. El Paso

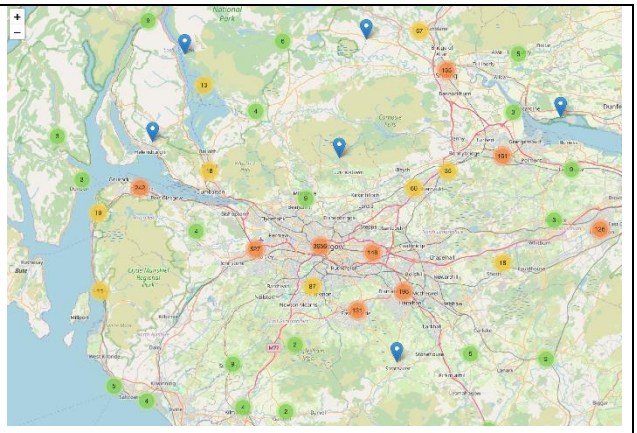


Figura 12. Glasgow

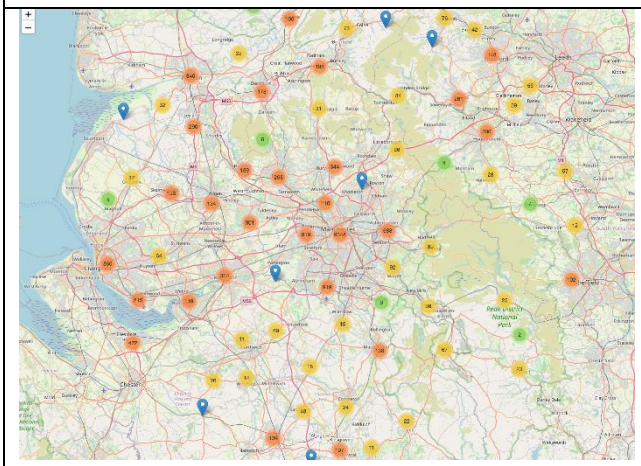


Figura 13. Manchester

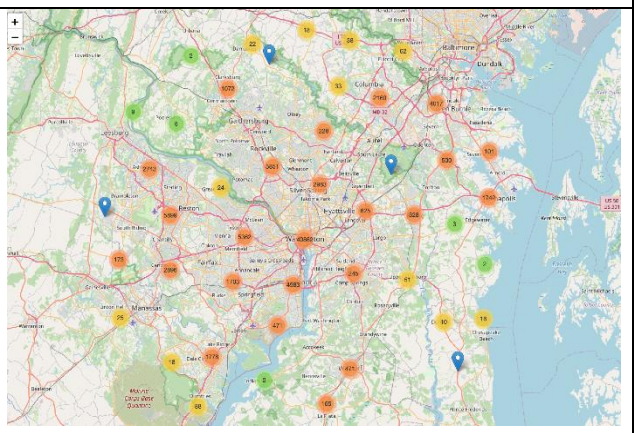


Figura 14. Washington

Mapas de visitas de usuarios individuales en las ciudades EL Paso, Washington, Glasgow y Manchester

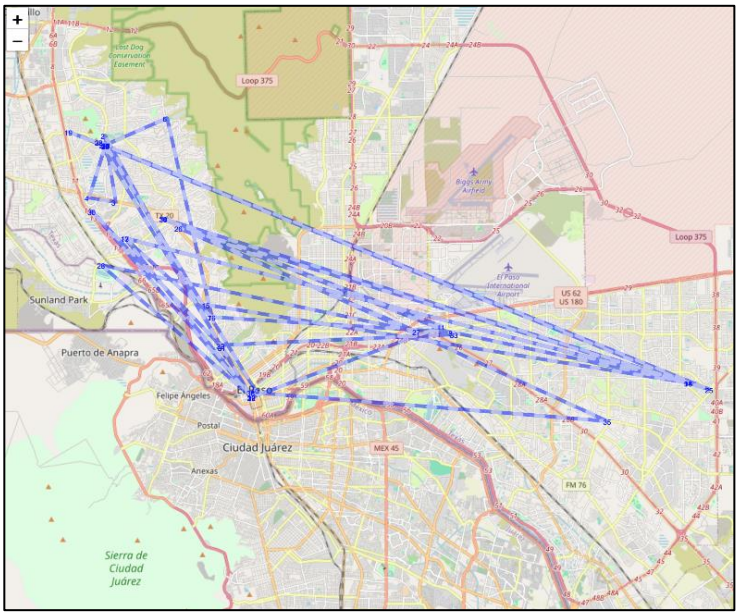


Figura 15. Mapa Usuario 667 ciudad El Paso

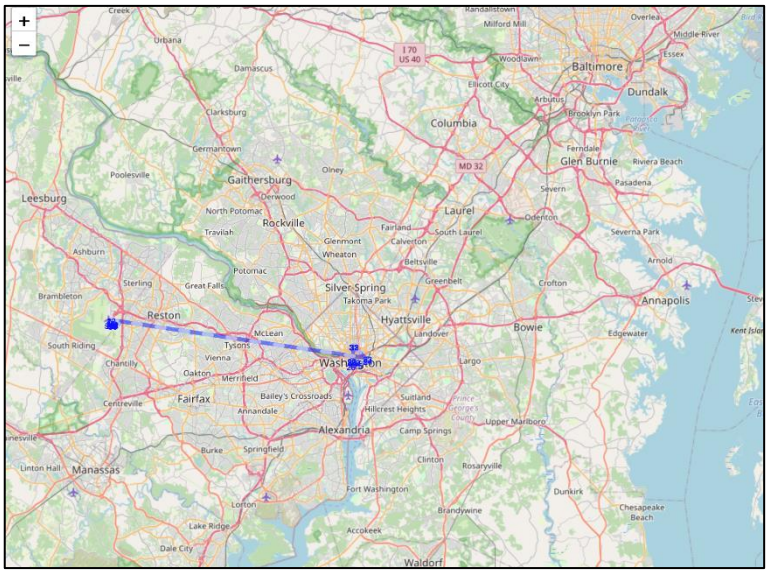


Figura 16. Mapa Usuario 22 ciudad Washington

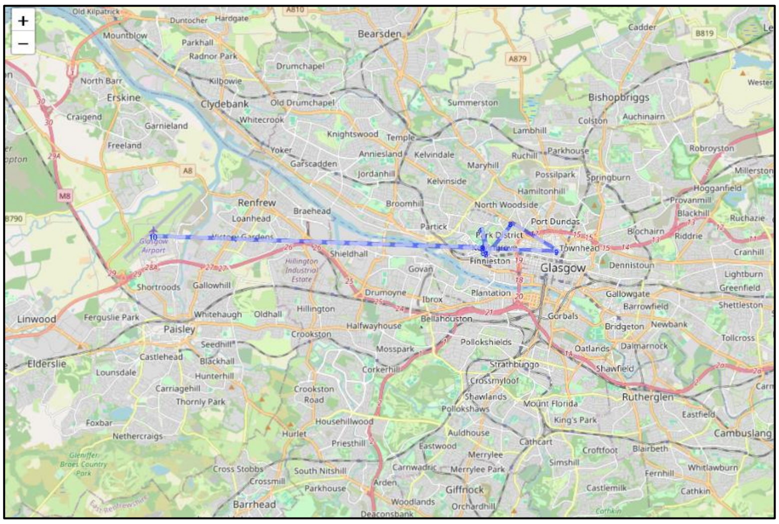


Figura 17. Mapa Usuario 268 ciudad Glasgow

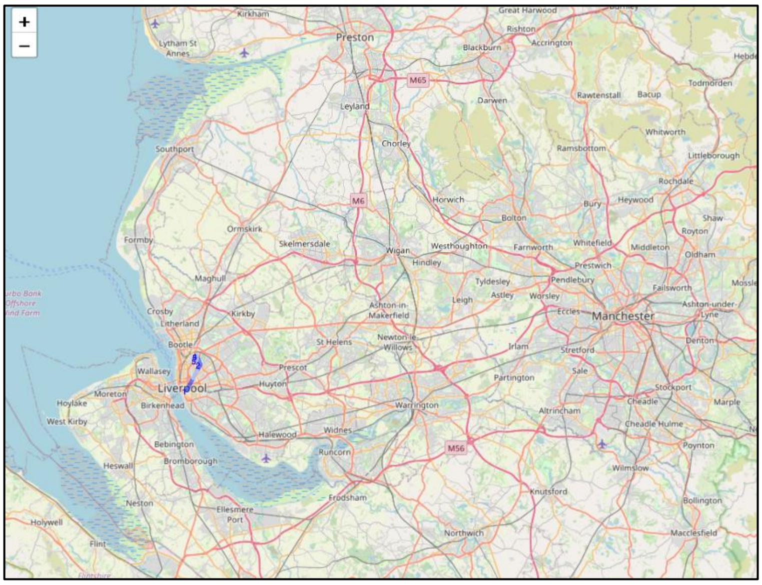


Figura 18. Mapa Usuario 332 ciudad Liverpool

Top 5 selection El Paso

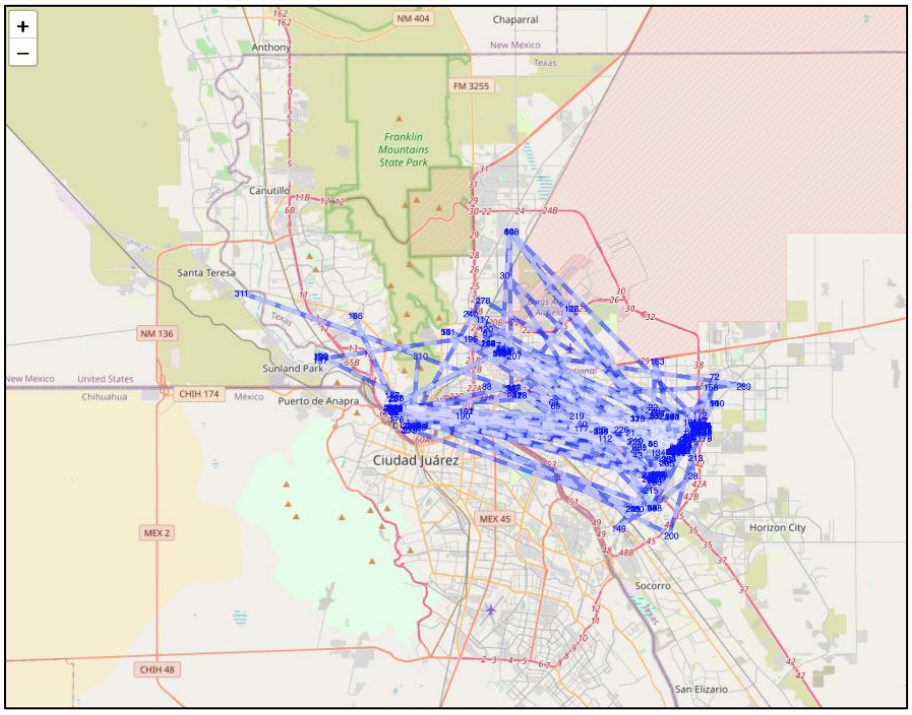


Figura 19. Visitas usuario 16440

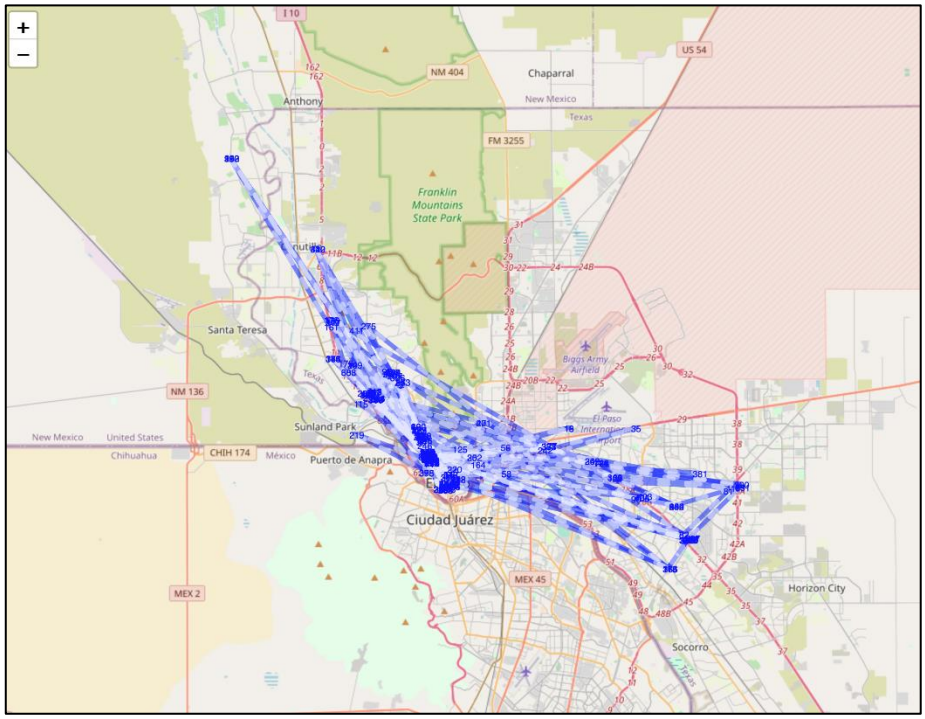


Figura 20. Visitas usuario 19601

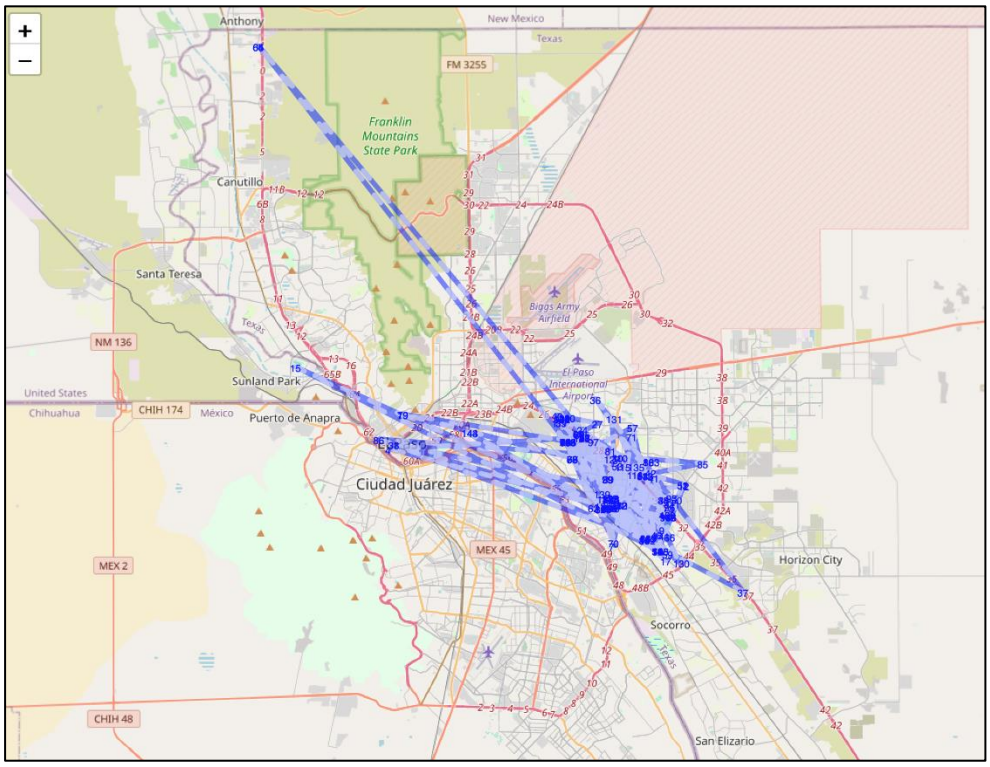


Figura 21. Visitas usuario 20086

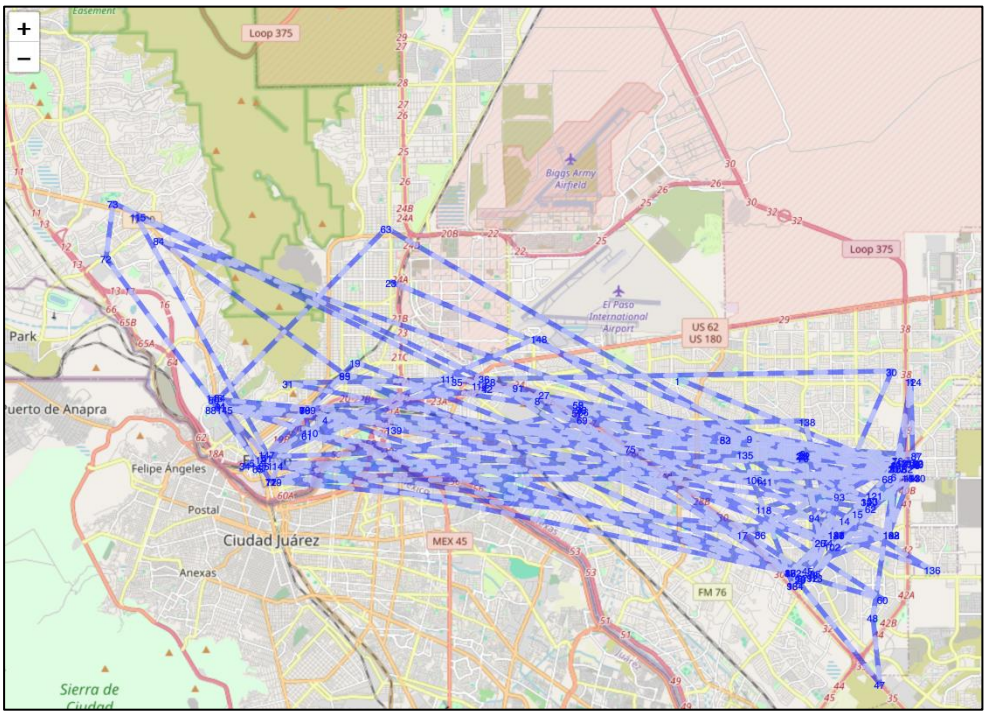


Figura 22. Visitas usuario 102157

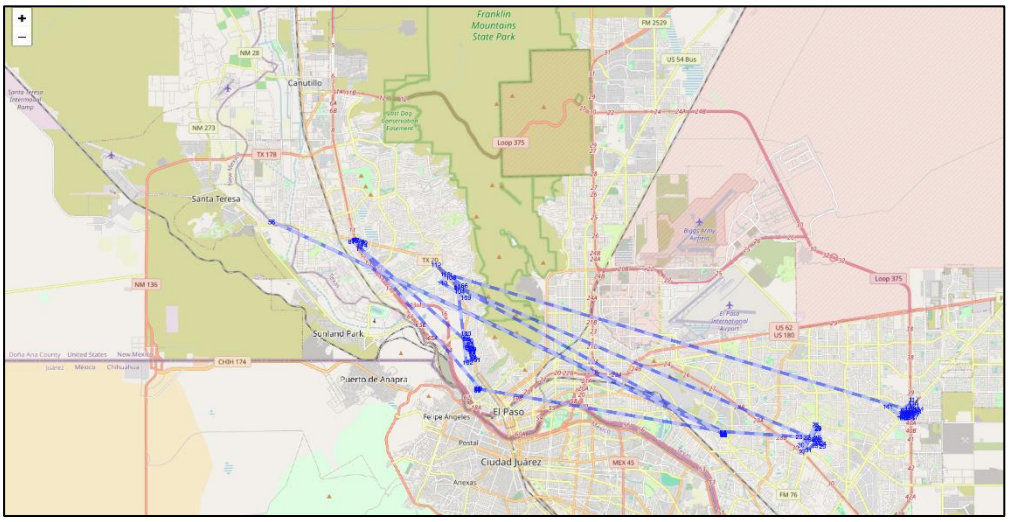


Figura 23. Visitas usuario 106485

```
URL="https://drive.google.com/uc?export=download&id=1PHWBGuwDHw4ZEIlCbgMTiEUrG8FmlJK2"

PATH_MAIN_PYTHON="ProyectoFuso/generate_maps.py"
PATH_MAIN_PYTHON_INDIVIDUAL="ProyectoFuso/generate_individual_maps.py"
PATH_GOWALLA_FILES="DatasetsGowalla/"
PATH_OUTPUT_GOWALLA_FILES="ProyectoFuso/templates/html_files/"
ARCHIVO_ZIP="DatasetsGowalla.zip"
PATH_PYTHON_TOPN="ProyectoFuso/topn_selection_javierbertran.py"


#wget --no-check-certificate "$URL" -O "$ARCHIVO_ZIP"



#unzip "$ARCHIVO_ZIP"

mkdir -p filtered_files

source entorno_virtual/bin/activate
```

Figura 24. Script Apartado 3. Variables y descarga de archivos

```
for fichero in "$PATH_GOWALLA_FILES"*; do

        # Obtener el nombre base del archivo (sin la extension)
        ciudad=$(basename "$fichero" Gowalla.txt)

        # Extraer columnas 3, 4 y 5 y anadir al archivo ALL_LOCATIONS.txt
        cut -f3-5 "$fichero" >> ALL_LOCATIONS.txt

        # Extraer columnas 1, 2 y 5 y redirigir al archivo filtrado de la ciudad
        cut -f1-2,5 "$fichero" > "filtered_files/${ciudad}_filtered.txt"




        # Calcular estadísticas
        #echo "Estadísticas para ${ciudad}.txt:"

        # Número de usuarios distintos
        num_usuarios=$(cut -f1 "$fichero" | sort | uniq | wc -l)
        #echo "Número de usuarios distintos: $num_usuarios"

        # Número de lugares distintos
        num_lugares=$(cut -f5 "$fichero" | sort | uniq | wc -l)
        #echo "Número de lugares distintos: $num_lugares"

        # Número total de filas
        num_filas=$(wc -l < "$fichero")
        #echo "Número de filas completas: $num_filas"

        # Número de interacciones en 2010-07
        interacciones_2010_07=$(grep "2010-07" "$fichero" | wc -l)
        #echo "Número de check-ins en 2010-07: $interacciones_2010_07"

        # Número de interacciones en 2010-08
        interacciones_2010_08=$(grep "2010-08" "$fichero" | wc -l)
        #echo "Número de check-ins en 2010-08: $interacciones_2010_08"
```

Figura 25. Script Apartado 3. Procesamiento de datos.

```
        # Generar el mapa general solo para las ciudades específicas
        if [[ "$ciudad" == "ElPaso" || "$ciudad" == "Glasgow" || "$ciudad" == "Manchester" || "$ciudad" == "WashingtonDC" ]]; then


                #python3 "$PATH_MAIN_PYTHON" --input_file "$fichero" --city_name "$ciudad" --output_html
"$PATH_OUTPUT_GOWALLA_FILES${ciudad}GowallaMap"

                # Asignar el user_id según la ciudad
                if [ "$ciudad" == "ElPaso" ]; then
                        user_id=667
                elif [ "$ciudad" == "WashingtonDC" ]; then
                        user_id=22
                elif [ "$ciudad" == "Glasgow" ]; then
                        user_id=268
                elif [ "$ciudad" == "Manchester" ]; then
                        user_id=332
                fi
                #python3 "$PATH_MAIN_PYTHON_INDIVIDUAL" --user_id "$user_id" --city_name "$ciudad" --input_file "$fichero" --
output_html "$PATH_OUTPUT_GOWALLA_FILES${ciudad}_${user_id}_IndividualMap.html"


        fi


done
```

Figura 26. Script Apartado 3. Creación mapas y mapas individuales.

```
input_file="filtered_files/ElPaso_filtered.txt"
output_file="filtered_files/ElPaso_top5.txt"

# Generar el archivo con los 5 usuarios más activos
python3 "$PATH_PYTHON_TOPN" --input_file "$input_file" --top 5 --output_file "$output_file"



ciudad="ElPaso"
PATH_ELPASO="DatasetsGowalla/ElPasoGowalla.txt"
output_file="filtered_files/${ciudad}_top5.txt"



# Paso 2: Leer el archivo y generar los mapas individuales para cada usuario
while read -r user_id; do

    echo "Generando mapa para el usuario ID: $user_id en la ciudad $ciudad"
    output_html="ProyectoFuso/templates/html_files/${ciudad}_${user_id}_topn_selection_Map.html"

    # Llamar al script para generar el mapa individual
    python3 "$PATH_MAIN_PYTHON_INDIVIDUAL" --user_id "$user_id" --city_name "$ciudad" --input_file "$PATH_ELPASO" --output_html
"$output_html"
done < "$output_file"


deactivate
```

Figura 27. Script Apartado 3. Creación mapas top 5 usuarios El Paso.

```python
import argparse
import pandas as pd

def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--input_file", type=str, required=True)
    parser.add_argument("--top", type=int, required=True)
    parser.add_argument("--output_file", type=str, required=True)

    return parser.parse_args()

def main():
    args = parse_args()
    data = pd.read_csv(args.input_file, delimiter='\t', header=None)
    data.columns = ['user_id', 'check-in time', 'location_id']

    user_interactions = data['user_id'].value_counts()
    top_users = user_interactions.head(args.top)

    with open(args.output_file, 'w') as f:
        for user_id, _ in top_users.items():
            f.write(f"{user_id}\n")

        print(f"Top {args.top} users saved to {args.output_file}")

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

Figura 28. Script topn_selection_javierbertran.py

6. PETICIONES DESDE PYTHON

En este apartado, nos encargamos de automatizar el proceso de peticiones al servicio Flask.. Para lograr esto, escribimos un código en Python que genera las combinaciones de tamaños de entrenamiento (train_size) y prueba (test_size) requeridas. Para ello, usamos un bucle for que itera sobre valores de train_size desde 0.1 hasta 0.9, calculando el valor complementario para test_size haciendo test_size = 1- train_size. Usamos round porque debido a la forma en la que trabaja Python, algunas restas no eran exactas. Esto nos asegura que todas las combinaciones se evalúen sin necesidad de configurarlas manualmente, ahorrando tiempo y minimizando errores.

Dentro de cada iteración, construimos un diccionario con los parámetros necesarios, como el conjunto de datos (iris), el modelo (RandomForest) y los tamaños de entrenamiento y prueba calculados. Este diccionario es enviado como una petición POST al endpoint /train de Flask, lo cual permite que se inicie el proceso de entrenamiento. Comprobamos que la respuesta del servidor tenga un estado 200 para confirmar que la petición fue exitosa. En ese caso, el servidor devuelve una referencia a la imagen generada, que descargamos con otra petición GET. Cada imagen es guardada con un nombre único con los valores de train_size y test_size.

```python
import requests

url = 'http://10.120.148.183:5000/train'

for train_size in [i/10 for i in range(1, 10)]:
    test_size = 1 - train_size
    data = {
        'dataset': 'iris',
        'model': 'RandomForest',
        'train_size': train_size,
        'test_size': test_size
          }
    response = requests.post(url, data=data)

    if response.status_code == 200:
        print(f"Entrenamiento exitoso para iris y RandomForest con train_size={train_size} y test_size={test_size}.")
        # Suponiendo que la imagen generada es referenciada en la respuesta
        img_url = response.text  # Suponiendo que la URL de la imagen esta en la respuesta
        img_response = requests.get(f'http://10.120.148.183:5000/{img_url}')
        with open(f'iris_RandomForest_{train_size}_{test_size}.png', 'wb') as f:
            f.write(img_response.content)
    else:
        print("Error en la peticion:", response.status_code)
```

Figura 29. Train model

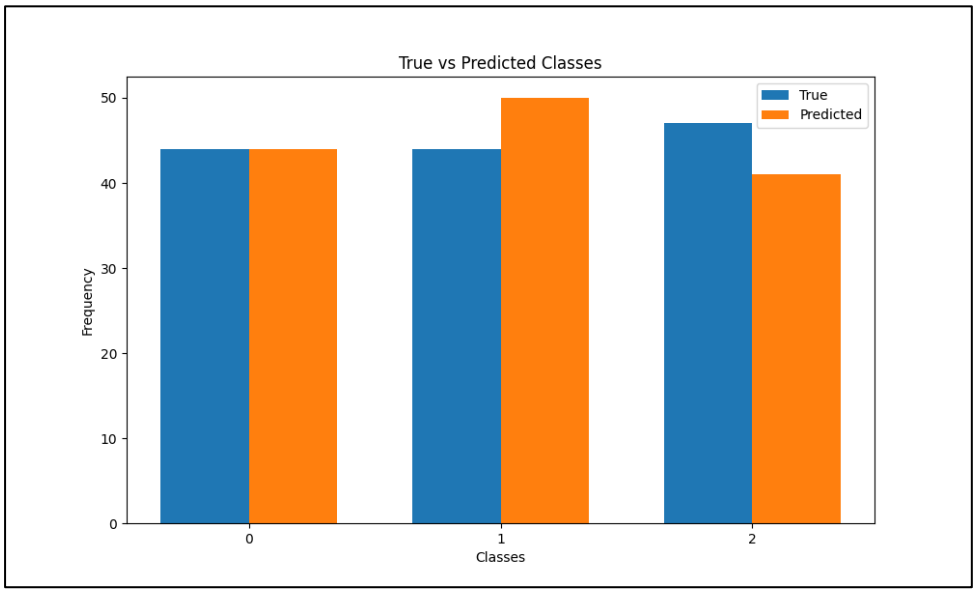


Figura 30. Imagen generada a través de Train model.

7. LA NUBE

El proceso para desplegar el proyecto implicó varias etapas, desde la creación de claves SSH en un sistema local de Ubuntu hasta la configuración y ejecución del código en una máquina virtual (VM) en Google Cloud.

Para permitir la transferencia segura de archivos y la conexión remota a otros sistemas, fue necesario generar un par de claves SSH. Este proceso se realizó con el comando ssh-keygen, que creó una clave privada y su correspondiente clave pública. Estas claves se almacenaron en el directorio ~/.ssh en Ubuntu. La clave pública fue utilizada posteriormente para permitir la autenticación en otros sistemas. Antes de continuar, se verificó que el servidor SSH en Ubuntu estaba configurado correctamente para aceptar la autenticación mediante claves, editando el archivo /etc/ssh/sshd_config.

El siguiente paso fue crear una máquina virtual en Google Cloud para alojar y ejecutar el proyecto. Se configuró la instancia de VM con un sistema operativo Ubuntu, y durante la creación se agregó la clave pública previamente generada en Ubuntu para habilitar el acceso remoto a la máquina.

Para tranferir los archivos, se usó winscp, configurando la conexión con la IP externa de la VM, el nombre de usuario de la VM y el archivo de clave privada. A la hora de ejecutar el fichero de despliegue, hubo un problema de dependencias al usar el mismo requirements que en alpine, por lo que fue necesario crear uno nuevo con las versiones necesarias para evitar problemas de dependencias. Una vez hecho esto, fue posible ejecutar los ficheros bash de los apartados 2 y 3.

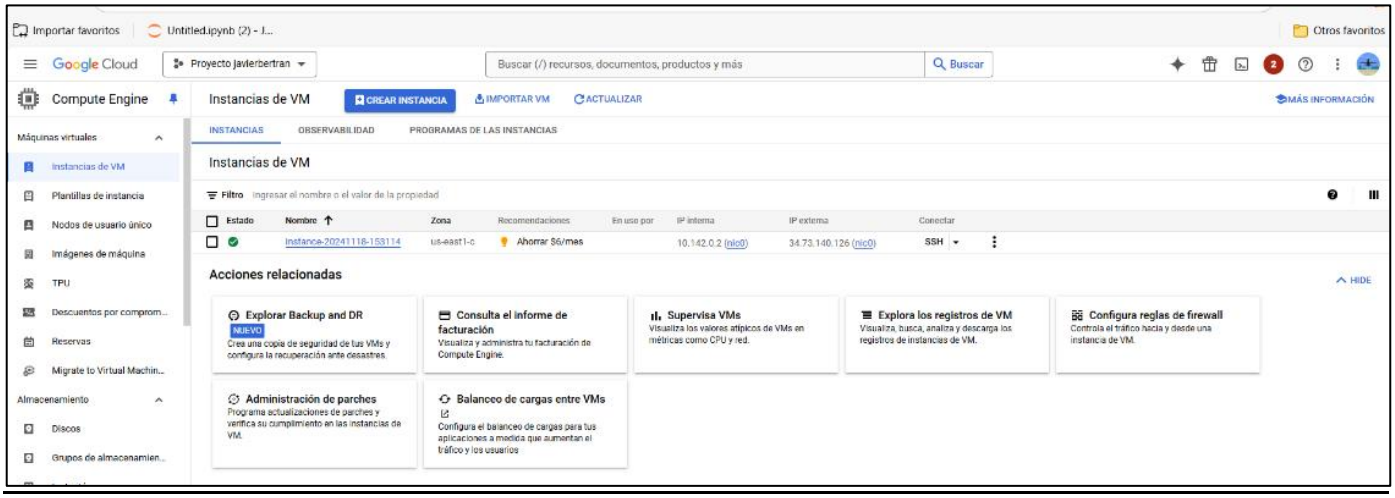


Figura 31. Instancia de VM

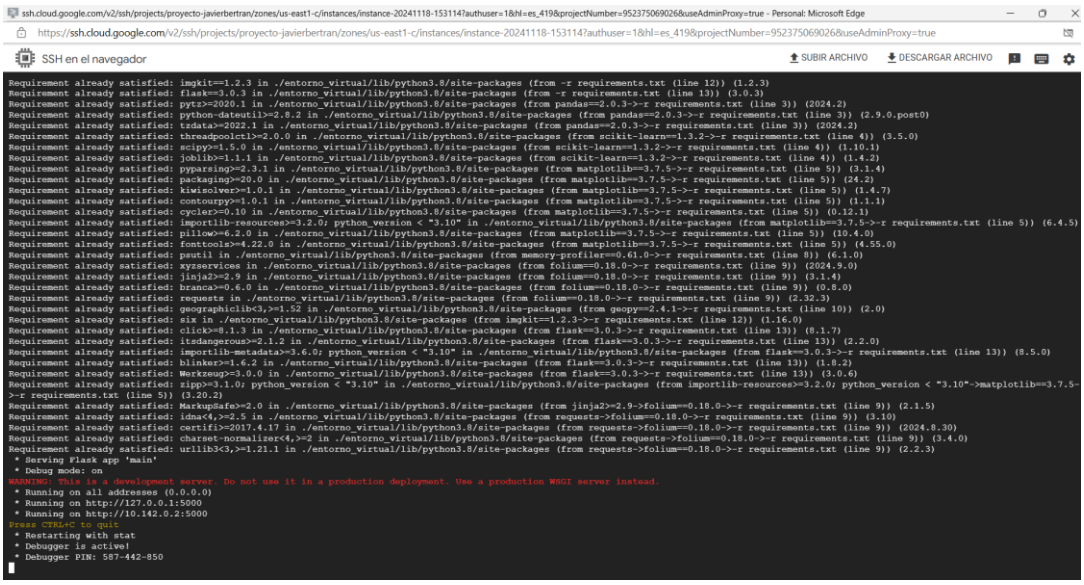


Figura 32. Terminal de Google Cloud

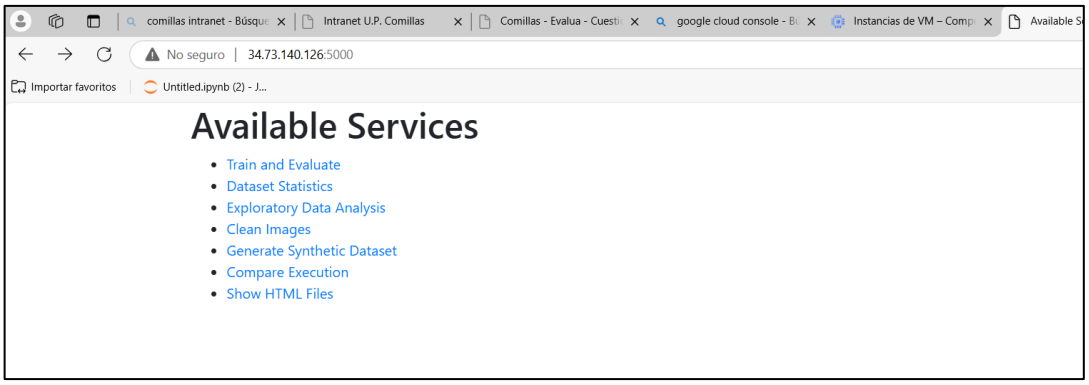


Figura 33. Proyecto Flask desplegado desde Instancia de VM

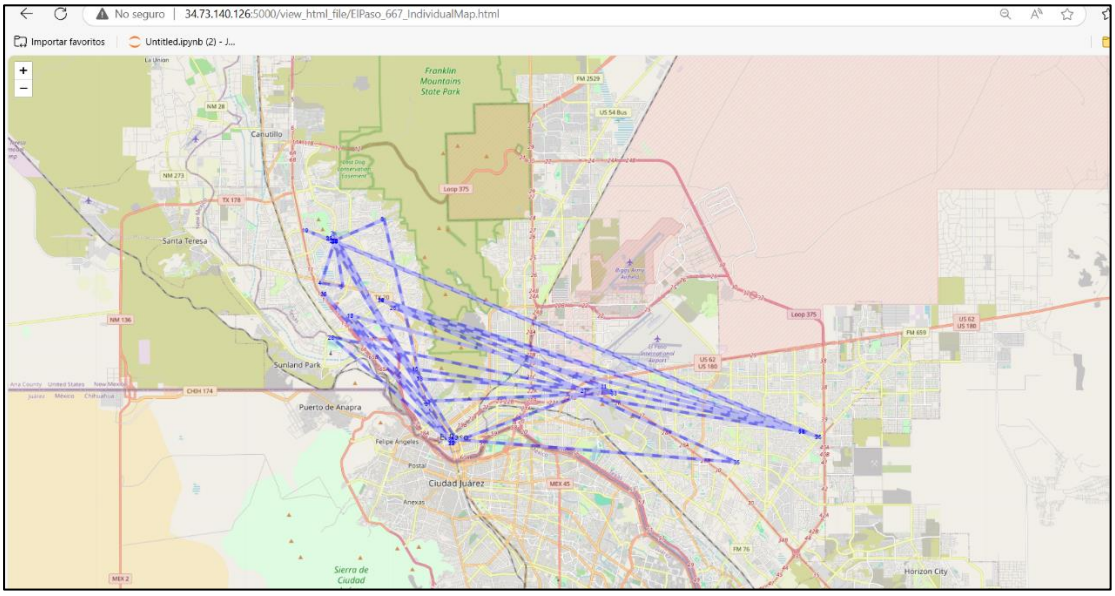


Figura 34. Mapa El Paso. Generado desde Instancia de VM.

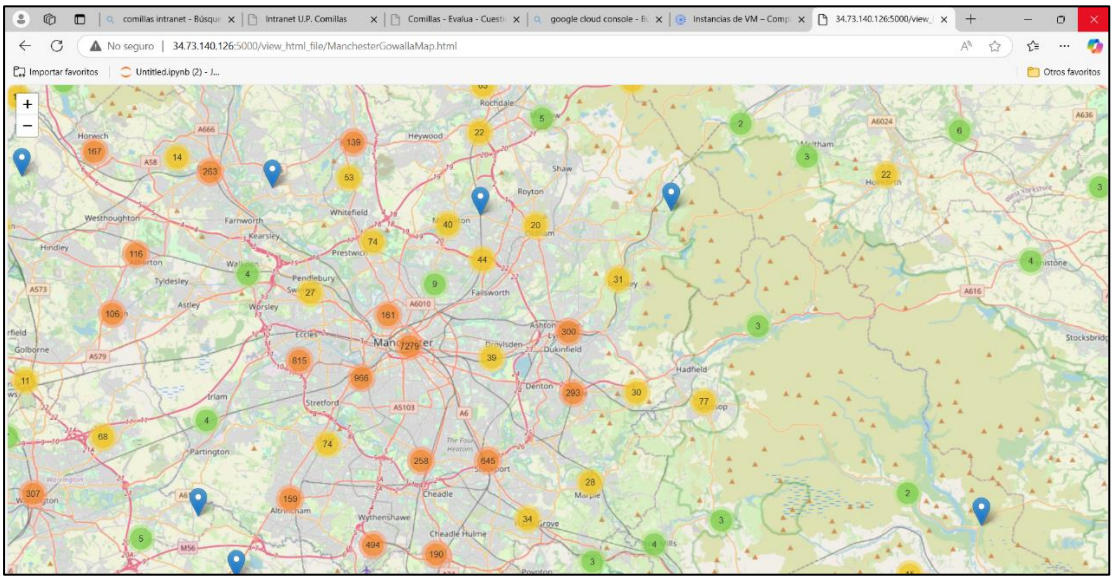


Figura 35. Mapa Manchester. Generado desde Instancia de VM.

```
#Estos son los requirements para hacer funcionar el proyecto en la instancia de Google

# Requirements
pandas==2.0.3
scikit-learn==1.3.2
matplotlib==3.7.5
numpy==1.24.4
seaborn==0.13.2
memory_profiler==0.61.0
folium==0.18.0
geopy==2.4.1
tqdm==4.67.0
imgkit==1.2.3
flask==3.0.3
```

Figura 36. Requirements para instancia VM