

SISTEMA QUE PERMITE ADMINISTRAR LAS DIVERSAS ESTACIONES DE RADIO BASADAS EN SDR (EBSDRADMIN)

Contenido

I.	Introducción	2
II.	Desarrollo y Base Tecnológica de EBSDRADMIN	2
III.	Requisitos previos	3
IV.	Instalación	3
4.1.1.	Componentes del sistema.....	3
4.2.	Configuración JSON	3
4.2.1.	ID.....	4
4.2.2.	carpeta_base	5
4.2.3.	python_gnuradio	6
4.2.4.	script_gnuradio	8
4.2.5.	BROKER	10
4.2.6.	PORT	10
V.	Que compone a EBSDRADMIN y cómo usarlo.....	11
5.1.	Interfaz y Consola.....	11
5.1.1.	Sección 1	11
5.1.2.	Sección 2.....	12
5.1.3.	Sección 3.....	13
5.1.4.	Sección 4.....	14
5.2.	Primera ejecución	15
5.3.	Errores.....	18
5.3.1.	Caso 1 GNU dejo de funcionar	18
5.3.1.	Caso 2 python_gnuradio.....	19
5.3.1.	Caso 3 script_gnuradio	19
5.3.2.	Caso 4 carpeta_base.....	19
5.4.	Ejecución	20
VI.	Código.....	22
6.1.	Funcionamiento del código	23
6.2.	Creación EBSDRADMIN.exe	24
VII.	Mejoras y desarrollo futuro	26

I. Introducción

En este documento se encontrara la información correspondiente a un ejecutable “**ADMINISTRADOR DE ESTACIÓN RF** la cual por facilidad llamaremos: **EBSDRADMIN**”, el cual es una herramienta la cual fue desarrollada en Python y su función es controlar el funcionamiento de scripts de GNU Radio (.py).

El software permite:

- Asignar variables al script antes de su ejecución.
- Iniciar o detener la ejecución del script de GNU Radio.
- Interactuar mediante una interfaz gráfica, que se abre al ejecutar el programa.
- Recibir órdenes desde un servidor Node-RED a través del protocolo MQTT, lo cual permite asignar variables y dar órdenes desde la nube.
- Este sistema **EBSDRADMIN** brinda la posibilidad de controlar el script de GNU Radio, desde los equipos que se encuentran en el lugar donde se estén realizando las mediciones de espectro o desde el servidor de **Node-red**, lo cual brinda la posibilidad, de realizar el respectivo control sin estar presente en el lugar de las mediciones.
- Al cerrar la interfaz de **EBSDRADMIN**, el programa se cerrara y dejara de ejecutarse

II. Desarrollo y Base Tecnológica de EBSDRADMIN

El lenguaje de programación utilizado para desarrollar **EBSDRADMIN** fue Python, ya que es un lenguaje de código abierto con una amplia variedad de librerías y documentación disponible. Además, **GNU Radio** está basado en Python, lo que puede llegar a facilitar su integración y personalización.

La creación de **EBSDRADMIN** fue debido a la necesidad de que usando un servicio externo a los equipos que se estén usando para medir el espectro, tener la capacidad de controlar el funcionamiento y parámetros de las variables de un diagrama de bloques de **GNU radio** mediante una interfaz en **Node-red** en el dashboard, además de también tener una interfaz en los equipos que se estén usando para medir el espectro donde se pueden realizar las mismas acciones, esto con el fin de tener un solo programa el cual permita realizar el control tanto desde **Node-red**, como desde los equipos que se estén usando para medir el espectro.

III. Requisitos previos

- Se debe tener instalado **GNU radio** en el equipo.
(<https://wiki.gnuradio.org/index.php/InstallingGR>)
- Para él **EBSDRADMIN.exe** no es necesario tener instalado Python, pero si se requiere realizar modificaciones al script se debe tener instalado Python en el equipo y alguna herramienta para realizar edición de código por ejemplo: Visual studio code.

IV. Instalación

4.1.1. Componentes del sistema



Fig 1 Componentes del sistema

El sistema está compuesto por dos archivos los cuales son: "**EBSDRADMIN.exe**", el cual es el ejecutable del programa, y: "**config.json** ", archivo JSON por medio del cual se ingresan parámetros de configuración al sistema Fig 1.

Los cuales se encuentran en:

<https://drive.google.com/drive/folders/1GUJ2LGEMVtVjto02RmIwFCU37liTsOSd?usp=sharing>

Primer paso: crear alguna carpeta en el explorador de archivos del equipo, en dicha carpeta se deben guardar ambos archivos.

4.2. Configuración JSON

```
*config.json: Bloc de notas
Archivo Edición Formato Ver Ayuda
{
  "ID":ID_correspondiente a Node-red,
  "carpeta_base": "Carpeta donde se almacenan los archivos .csv",
  "python_gnuradio": "Direccion de python.exe de GNU radio",
  "script_gnuradio": "Direccion del .py que se genero apartir del diagrama de bloques de GNU",
  "BROKER":"Direccion IP donde esta instalado mosquito",
  "PORT":1883
}
```

Fig 2 Archivo JSON

Este archivo contiene parámetros los cuales son requeridos para el funcionamiento de **EBSDRADMIN**, en el archivo json como se observa en Fig 2 hay comentarios de que es lo que va en cada termino, si estos comentarios están entre comillas, la información con la cual sean remplazados también ira entre comillas.

4.2.1. ID

Es el identificador de la pestaña que se esté usando en Node red, ya que se ha planteado la posibilidad de que en el futuro Node-red funcione como un Scada, en el que se integraran diferentes sistemas que existen o existirán, la ID debe corresponder a la ID que se encuentra en la respectiva pestaña de Node red, si está mal no se puede hacer uso de las herramientas las cuales estén en esa pestaña.

Primer paso: Acceder a la Url correspondiente al dashboard la cual es la siguiente:"

http://<IP_DEL_SERVIDOR>:1880/ui", remplace **IP_DEL_SERVIDOR** por la IP de la máquina virtual o servidor en el cual este instalado Node-red.

Segundo paso: Una ves se acceda al dashboard de Node red, se verá algo como lo siguiente:



Fig 3 ID dashboard Node-red

El nombre de cada pestaña está acompañado por la ID Fig 3, lo cual se puede observar encerrado por el cuadro rojo, entonces en el archivo JSON quedaría algo como:

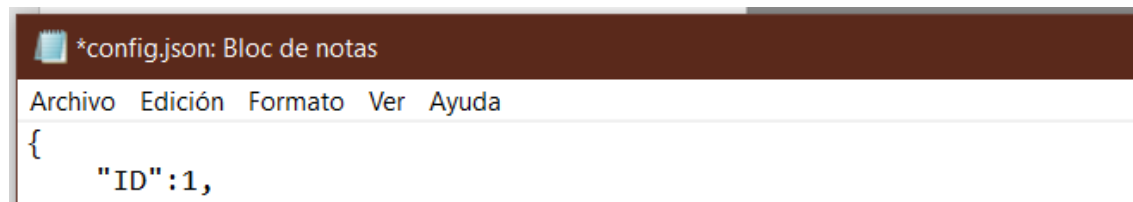


Fig 4 Ejemplo configuración ID

4.2.2. carpeta_base

Corresponde a la carpeta donde se almacenan los archivos .csv que se guardaran al hacer mediciones de espectro.

Primer paso: Se debe crear en el explorador de archivos una carpeta en alguna ubicación, se debe ingresar a la carpeta, y en la barra de direcciones en un espacio en blanco por ejemplo donde se encuentra el recuadro rojo de la siguiente imagen hay que dar un clic:

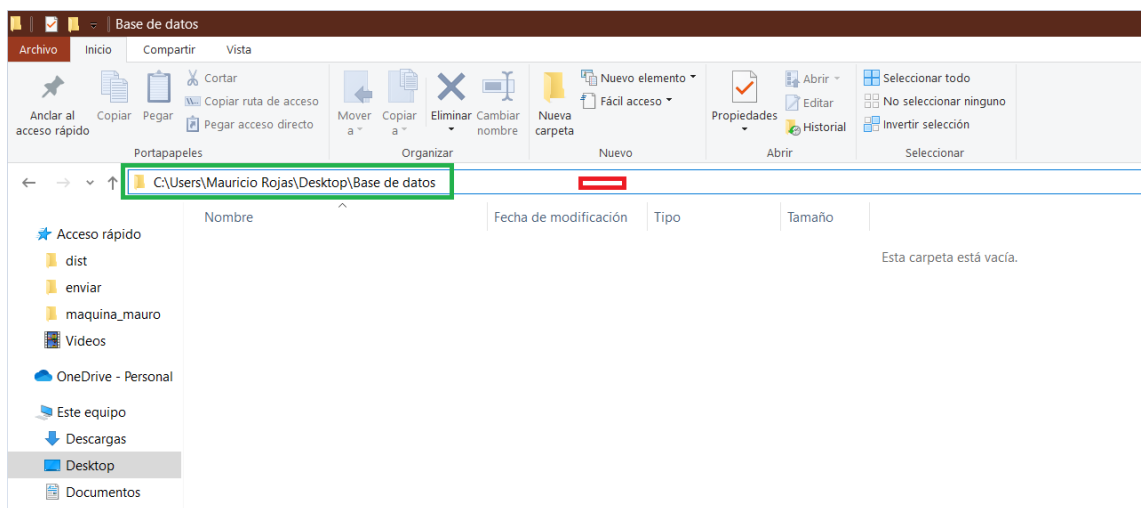
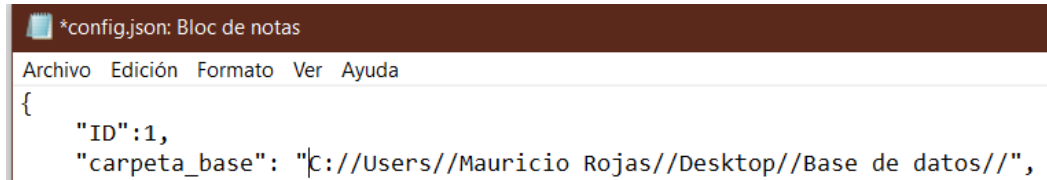


Fig 5 Obtención dirección de carpeta_base

Segundo paso: Al hacer eso aparecerá la dirección de la carpeta creada, la cual se ve en el recuadro verde Fig 5, en este caso de ejemplo es: "C:\Users\MauricioRojas\Desktop\Base de datos", (Tener en cuenta hay otro programa extra llamado loTVirtualCloud, este programa también hace uso de este directorio, pero ambos programas funcionan de forma individual, en cada uno de los 2 se debe colocar esta dirección, por lo que se debe asegurar que la carpeta usada sea la misma en ambos programas, para esto antes de realizar modificaciones a la dirección se recomienda copiarla y configurarla en el otro programa, su correspondiente documentación es encuentra [aquí](#)) y se cambia " \" por "/", además al final de esa dirección se

debe agregar:"/", después de hacer esto el ejemplo queda así: "**C://Users//Mauricio Rojas//Desktop//Base de datos//**", entonces en el archivo JSON quedaría algo como:



```
{
  "ID":1,
  "carpeta_base": "C://Users//Mauricio Rojas//Desktop//Base de datos//",
}
```

Fig 6 Ejemplo configuración carpeta_base

Nota: las direcciones de los archivos van entre comillas "".

4.2.3. *python_gnuradio*

Esto corresponde a la dirección en la cual se encuentra el Python.exe correspondiente a **GNU radio**.

Primer paso: para encontrarla, en la barra de tareas del equipo buscamos: "GNU" y damos clic en: "Abrir la ubicación del archivo"

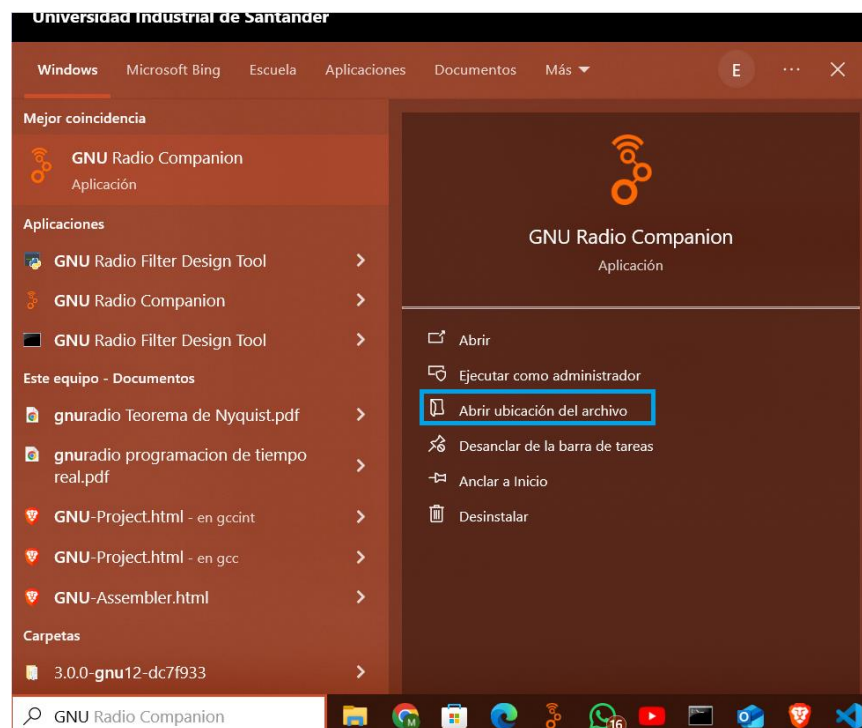


Fig 7 Búsqueda carpeta instalación GNU radio

Segundo paso: Nuevamente, damos clic derecho y damos clic sobre abrir la ubicación del archivo:

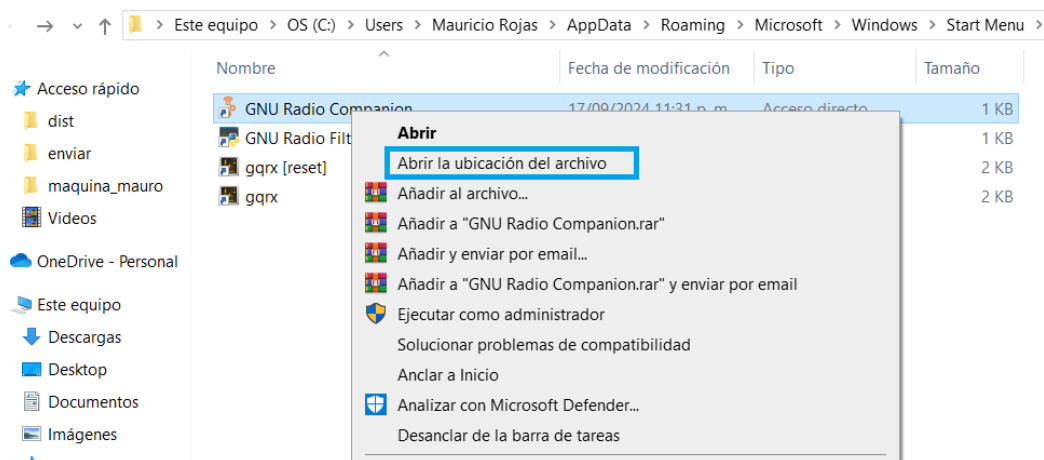


Fig 8 Búsqueda carpeta instalación GNU radio

Tercer paso: Al hacer eso se redirigirá a la carpeta donde se instaló **GNU radio** y en esa carpeta se encuentra python.exe, en la barra de direcciones en un espacio en blanco por ejemplo donde se encuentra el recuadro rojo de la siguiente imagen hay que dar un clic:

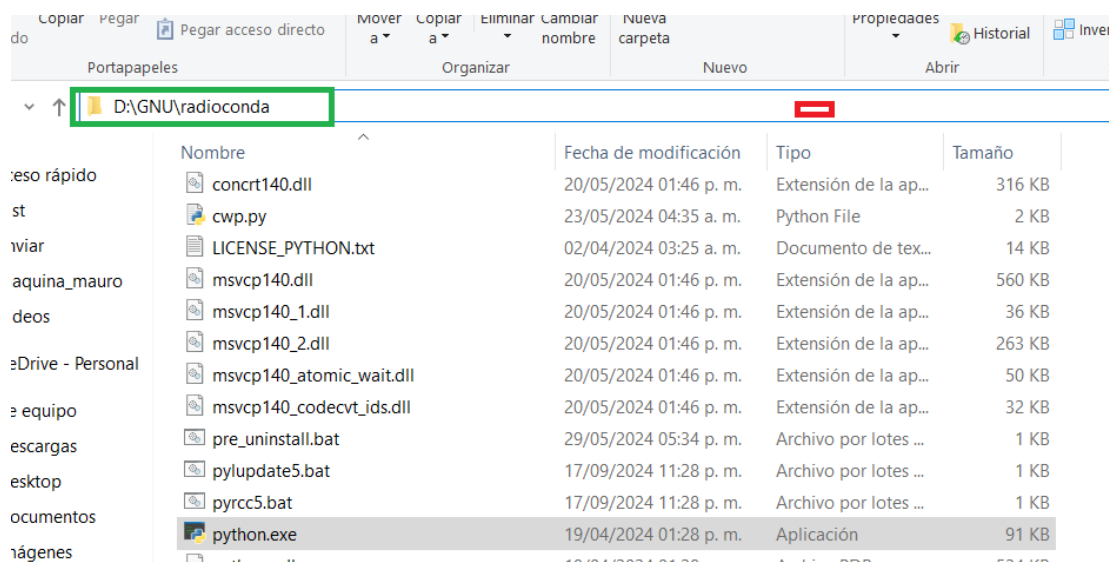
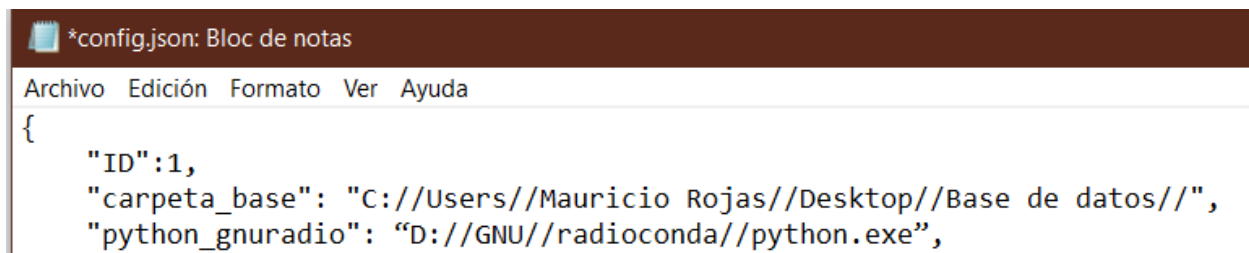


Fig 9 Obtención dirección donde se encuentra Python.exe de GNU radio

Cuarto paso: Al hacer eso aparecerá la dirección de la carpeta donde esta python.exe, la cual se ve en el recuadro verde Fig 9, la cual en este caso de ejemplo es: "D:\GNU\radioconda", y se cambia " \" por "/", además al final de esa dirección se debe agregar: "//python.exe", después

de hacer esto el ejemplo queda así: “D://GNU//radioconda//python.exe”, entonces en el archivo JSON quedaría algo como:

A screenshot of a text editor window titled '*config.json: Bloc de notas'. The window has a menu bar with 'Archivo', 'Edición', 'Formato', 'Ver', and 'Ayuda'. The main text area contains a JSON object with the following content:

```
{
  "ID": 1,
  "carpeta_base": "C://Users//Mauricio Rojas//Desktop//Base de datos//",
  "python_gnuradio": "D://GNU//radioconda//python.exe",
```

Fig 10 Ejemplo configuración python_gnuradio

4.2.4. script_gnuradio

Corresponde a la dirección del .py que se generó a partir del diagrama de bloques de **GNU radio**.

Primer paso: Por defecto se encuentra disponible un archivo .py el cual corresponde a un radiotelescopio educativo: [URL](#) , en la anterior dirección se encuentra disponible el archivo **RADIOTELESCOPE_EDUCATIVO.py** junto con los archivos

RADIOTELESCOPE_EDUCATIVO_epy_block_1_0.py y

RADIOTELESCOPE_EDUCATIVO_epy_block_2.py generados debido al uso de bloques

Python, deben ser guardados en un mismo directorio, además también esta disponible el diagrama de bloques de GNU radio usado **radiotelescop23.grc**, para conocer más sobre este diagrama de bloques se recomienda su correspondiente documentación [aquí](#), si se desea hacer uso de algún otro diagrama de bloques diferente o modificar el correspondiente al del radiotelescopio educativo debe generar y configurar el .py de **GNU radio**. Para realizarlo se debe ver este Video: <https://youtu.be/3TdTRboyMoY>.

Archivo de configuración del .py :

https://drive.google.com/file/d/14HvRymwIMdBvUjFgnisFPqjVFgy0AqYD/view?usp=drive_link

Formato variables:

- **BW_hackRF**
- **f_centro**
- **nubeVirtual**
- **azimut**
- **elevación**

- **altitud**
- **latitud**
- **longitud**
- **descripcion**

Segundo paso: Una vez se crea y configura el .py generado por **GNU radio**, se accede a la carpeta donde está el .py, generalmente es la misma carpeta donde se guardó el diagrama de bloques de **GNU radio** o en su defecto la carpeta donde se guardó los archivos .py que se proporcionó ya previamente configurados, en la barra de direcciones en un espacio en blanco por ejemplo donde se encuentra el recuadro rojo de la siguiente imagen hay que dar un clic, además identificamos el nombre del archivo .py principal, el cual toma la ID escrita en el bloque option de **GNU radio**, los otros archivos que también se generaron corresponden a Python blocks que se crearon en **GNU radio** Fig 11.

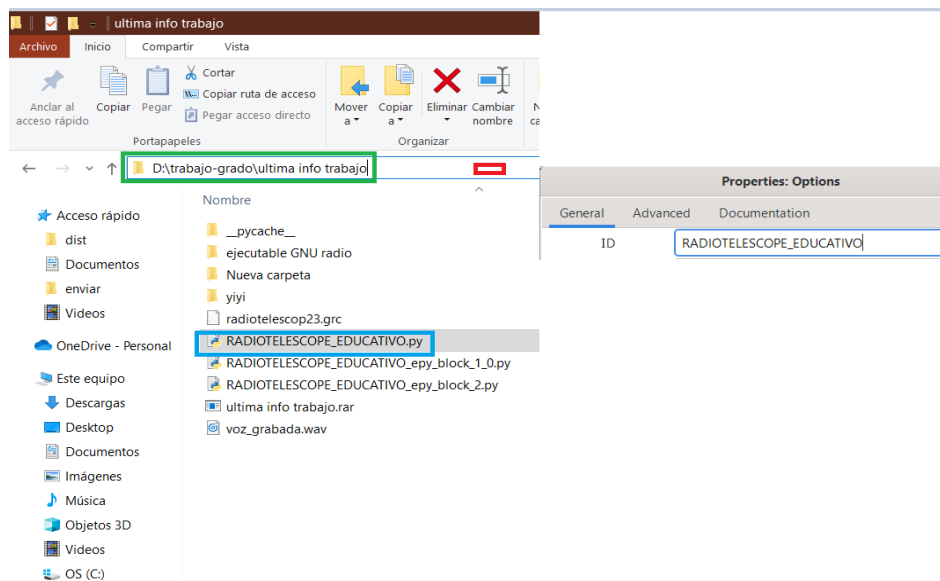
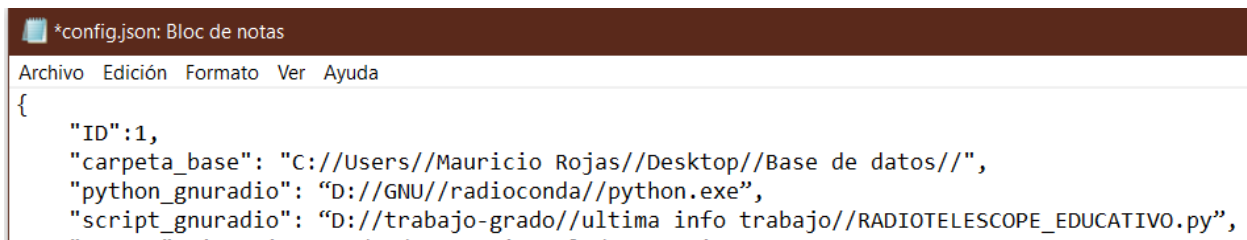


Fig 11 Obtención dirección donde se encuentra el .py de GNU radio

Tercer paso: Al hacer eso aparecerá la dirección de la carpeta donde está el .py, la cual se ve en el recuadro verde, la cual en este caso de ejemplo es: "D:\trabajo-grado\ultima info trabajo", y se cambia " \" por "//", además al final de esa dirección se debe agregar: "//nombre del archivo.py", en este ejemplo: "//RADIOTELESCOPE_EDUCATIVO.py" después de hacer esto el ejemplo queda así: "D://trabajo-grado//ultima info trabajo//RADIOTELESCOPE_EDUCATIVO.py", entonces en el archivo JSON quedaría algo como:

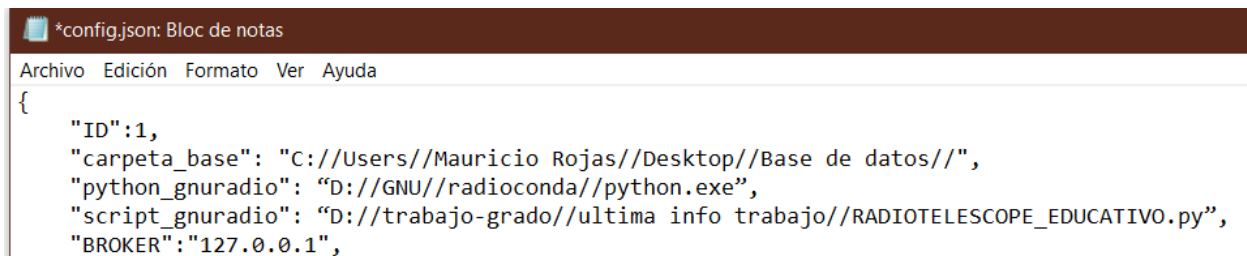


```
*config.json: Bloc de notas
Archivo Edición Formato Ver Ayuda
{
  "ID":1,
  "carpeta_base": "C://Users//Mauricio Rojas//Desktop//Base de datos//",
  "python_gnuradio": "D://GNU//radioconda//python.exe",
  "script_gnuradio": "D://trabajo-grado//ultima info trabajo//RADIOTELESCOPE_EDUCATIVO.py",
}
```

Fig 12 Ejemplo configuración script_gnuradio

4.2.5. BROKER

Corresponde a la dirección IP del servidor o máquina virtual donde está instalado **Mosquitto**. Es la misma IP en la que se encuentra instalado **Node-RED**, el JSON quedaría así:

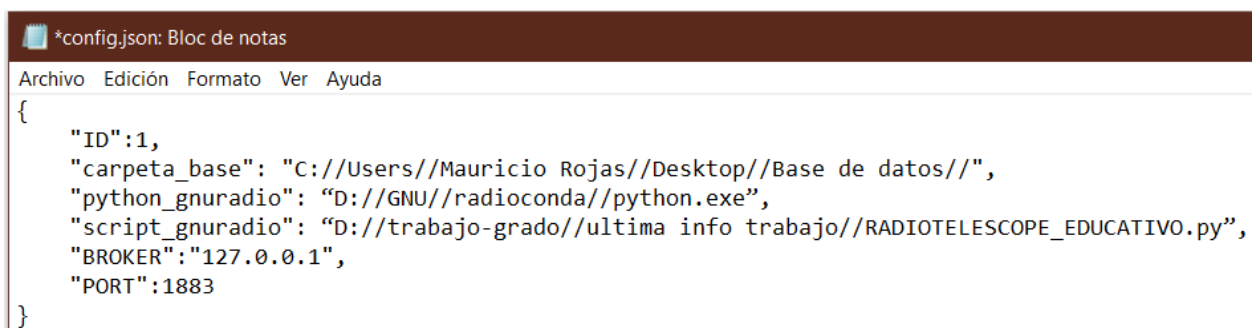


```
*config.json: Bloc de notas
Archivo Edición Formato Ver Ayuda
{
  "ID":1,
  "carpeta_base": "C://Users//Mauricio Rojas//Desktop//Base de datos//",
  "python_gnuradio": "D://GNU//radioconda//python.exe",
  "script_gnuradio": "D://trabajo-grado//ultima info trabajo//RADIOTELESCOPE_EDUCATIVO.py",
  "BROKER":"127.0.0.1",
}
```

Fig 13 Ejemplo configuración BROKER

4.2.6. PORT

El **puerto 1883** es el **puerto por defecto** que usa el protocolo **MQTT (Message Queuing Telemetry Transport)** para la comunicación entre clientes y el broker. El Json con esta última configuración estaría listo para usarse y quedaría así:



```
*config.json: Bloc de notas
Archivo Edición Formato Ver Ayuda
{
  "ID":1,
  "carpeta_base": "C://Users//Mauricio Rojas//Desktop//Base de datos//",
  "python_gnuradio": "D://GNU//radioconda//python.exe",
  "script_gnuradio": "D://trabajo-grado//ultima info trabajo//RADIOTELESCOPE_EDUCATIVO.py",
  "BROKER":"127.0.0.1",
  "PORT":1883
}
```

Fig 14 Ejemplo configuración PORT

V. Que compone a EBSDRADMIN y cómo usarlo

5.1. Interfaz y Consola

Al ejecutar EBSDRADMIN.exe, se abrirá una interfaz y una consola, si se quiere finalizar la ejecución de EBSDRADMIN esto se logra al cerrar la interfaz o en su defecto la consola.

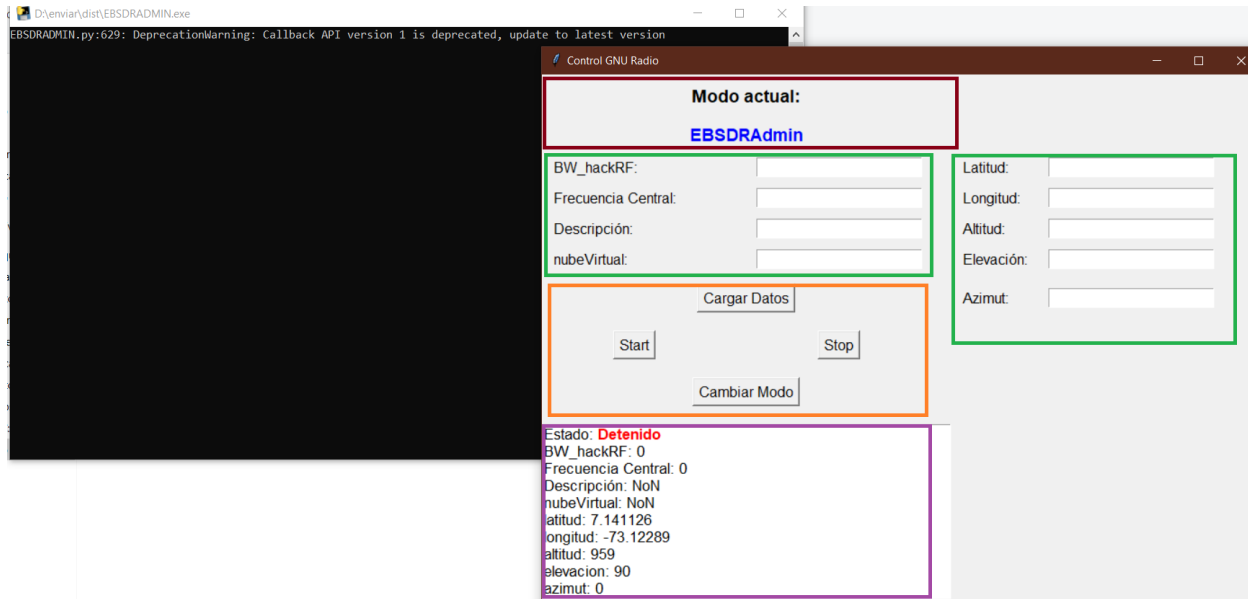


Fig 15 Interfaz de EBSDRADMIN y consola

En la consola se imprimirá información importante, la cual corresponde a cierres inesperados del .py de **GNU radio**, o fallos en el JSON referentes a las diferentes URL las cuales se configuraron previamente.

La interfaz gráfica tiene varia información, para explicarla se procedió a dividir en 4 secciones.

5.1.1. Sección 1

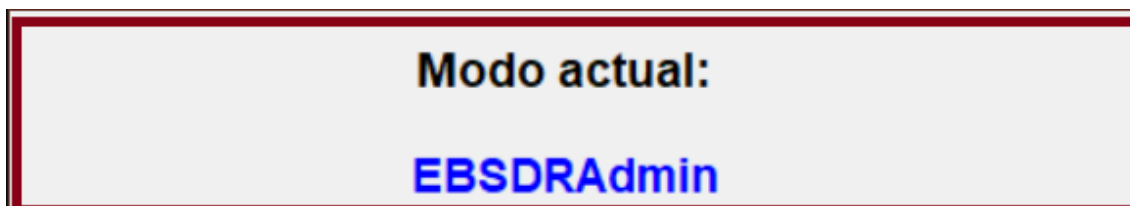


Fig 16 Sección 1 Interfaz de EBSDRADMIN

En esta parte de la interfaz se mostrara el modo en el que se encuentre en ese momento, tiene 2 posibles estados: “EBSDRADMIN” y “servidor” Fig 16.

- **EBSDRADMIN:** significa que el control sobre **GNU radio** lo tienen los equipos en el lugar donde se están haciendo mediciones de espectro.
- **servidor:** Significa que el control lo tiene el servidor de **Node-red**, por lo que un usuario puede acceder al dashboard y desde allí tener control de **GNU radio**.

5.1.2. Sección 2

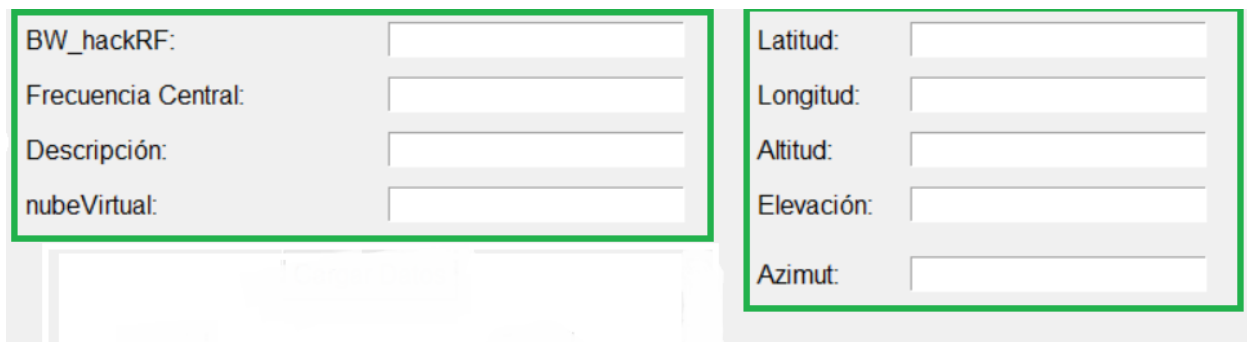


Fig 17 Sección 2 Interfaz de EBSDRADMIN

En esta parte de la interfaz es donde se pueden ingresar los valores de las variables deseados para ejecutar el diagrama de bloques de **GNU radio** Fig 17 donde:

- **BW_hackRF:** Corresponde al ancho de banda de la medición, solo se pueden ingresar valores numéricos, se admite notación científica (ejemplo:40e6) si no es un valor numérico no se permitirá cargar los datos y se generara una ventana de notificación informando que algunas de las variables no son valores numéricos validos
- **Frecuencia Central:** Corresponde al valor central de frecuencia de la medición, solo se pueden ingresar valores numéricos, se admite notación científica (ejemplo:40e6) si no es un valor numérico no se permitirá cargar los datos y se generara una ventana de notificación informando que algunas de las variables no son valores numéricos validos
- **Descripción:** Descripción de la medición, no tiene restricciones
- **nubeVirtual:** Corresponde al nombre del archivo .csv en el cual se guardaran las mediciones de espectro, hay algunos caracteres los cuales son caracteres restringidos los cuales no pueden ir en nombres de archivos:" \ / : * ? " < > |", por lo que si son ingresados no se permitirán cargar los datos y con una notificación se informara el problema

- **Latitud:** Coordenada geográfica que indica la posición de un punto en la Tierra con respecto al ecuador.
- **Longitud:** es una coordenada geográfica que indica la posición de un punto en la Tierra con respecto al meridiano de Greenwich
- **Altitud: ---,**solo se pueden ingresar valores numéricos, se admite notación científica (ejemplo:40e6) si no es un valor numérico no se permitirá cargar los datos y se generara una ventana de notificación informando que algunas de las variables no son valores numéricos validos
- **Elevación: ----,**solo se pueden ingresar valores numéricos, se admite notación científica (ejemplo:40e6) si no es un valor numérico no se permitirá cargar los datos y se generara una ventana de notificación informando que algunas de las variables no son valores numéricos validos
- **Azimut:----,** solo se pueden ingresar valores numéricos, se admite notación científica (ejemplo:40e6) si no es un valor numérico no se permitirá cargar los datos y se generara una ventana de notificación informando que algunas de las variables no son valores numéricos validos

5.1.3. Sección 3

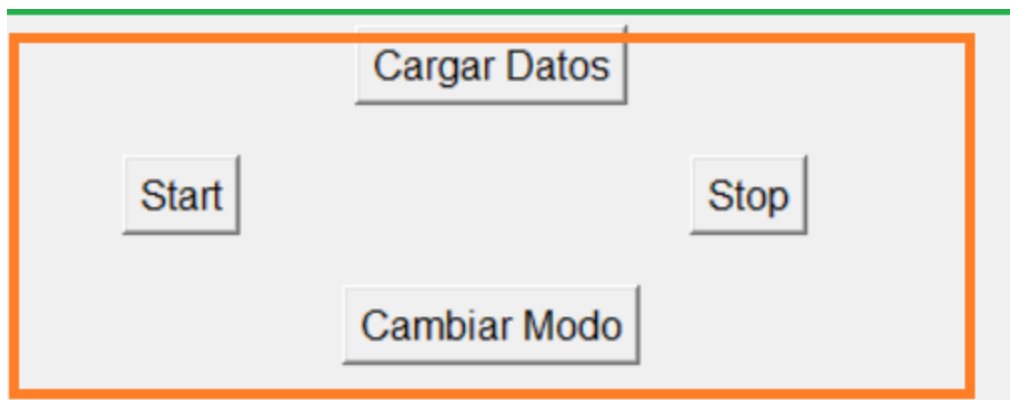
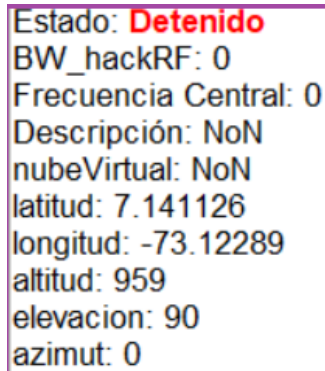


Fig 18 Sección 3 Interfaz de EBSDRADMIN

En esta parte de la interfaz se encuentran los diferentes botones por medio de los cuales se puede controlar las ordenes que da **EBSDRADMIN**, hay 4 botones [Fig 18](#).

- **Cargar datos:** botón encargado de actualizar las variables para enviar a **GNU radio**, realiza la acción de actualizar las variables, solo si las variables tienen valores válidos, si la ejecución de **GNU radio** está detenida y si el modo es **EBSDRADMIN**, de lo contrario no hará nada, solo se desplegará una ventana de notificación que informará el problema.
- **Start:** botón encargado de iniciar la ejecución de **GNU radio**, realiza la acción solo si hay datos iniciales cargados, cuando la aplicación se inicia por primera vez tiene algunos datos por defecto, pero no son datos correctos, por lo que primero se deben cargar datos adecuados para iniciar **GNU radio**, cuando la ejecución de **GNU radio** está detenida y si el modo es **EBSDRADMIN**, de lo contrario no hará nada, solo se desplegará una ventana de notificación que informará el problema.
- **Stop:** botón encargado de detener la ejecución de **GNU radio**, realiza la acción solo si hay datos iniciales cargados, cuando la aplicación se inicia por primera vez tiene algunos datos por defecto, pero no son datos correctos, por lo que primero se deben cargar datos adecuados para iniciar **GNU radio**, cuando la ejecución de **GNU radio** está iniciada y si el modo es **EBSDRADMIN**, de lo contrario no hará nada, solo se desplegará una ventana de notificación que informará el problema.
- **Cambiar Modo:** botón encargado de cambiar el modo entre **EBSDRADMIN** y servidor, realizará la acción solo si hay conexión con el servidor de **Node-red**, si no está activo no permitirá cambiar modo a servidor.

5.1.4. Sección 4

A screenshot of a software interface window with a purple border. It displays a list of configuration variables and their current values. The text is as follows:

```
Estado: Detenido
BW_hackRF: 0
Frecuencia Central: 0
Descripción: NoN
nubeVirtual: NoN
latitud: 7.141126
longitud: -73.12289
altitud: 959
elevacion: 90
azimut: 0
```

Fig 19 Sección 4 Interfaz de EBSDRADMIN

En esta parte de la interfaz es donde se imprimirá el valor que en ese momento tiene cada una de las variables que se pueden cargar para **GNU radio** Fig 19, los valores que se observan en la imagen son los valores iniciales que están al iniciar el programa, estos valores no son válidos, por lo que se deben cargar variables antes de intentar iniciar **GNU radio**, de lo contrario no permitirá ejecutarlo, cada vez que se realicen cambios en las variables estos se reflejaran aquí, además se imprime el estado, lo cual representa el estado de la ejecución de **GNU radio**, el cual es: "Detenido" e "Iniciado", también se informa si la última ejecución de **GNU radio** se cerró inesperadamente.

5.2. Primera ejecución

Cabe aclarar que con primera ejecución esto hace referencia a modificaciones que se realicen al archivo JSON, ya que al realizar algún tipo de modificación a las configuraciones, no se ha realizado una comprobación por lo que no se sabe si están de forma correcta, también hace referencia a la primera vez que se quiera usar el programa en algún equipo.

- **Primer paso:** ejecutar EBSDRADMIN.exe, se abrirá la interfaz y la consola

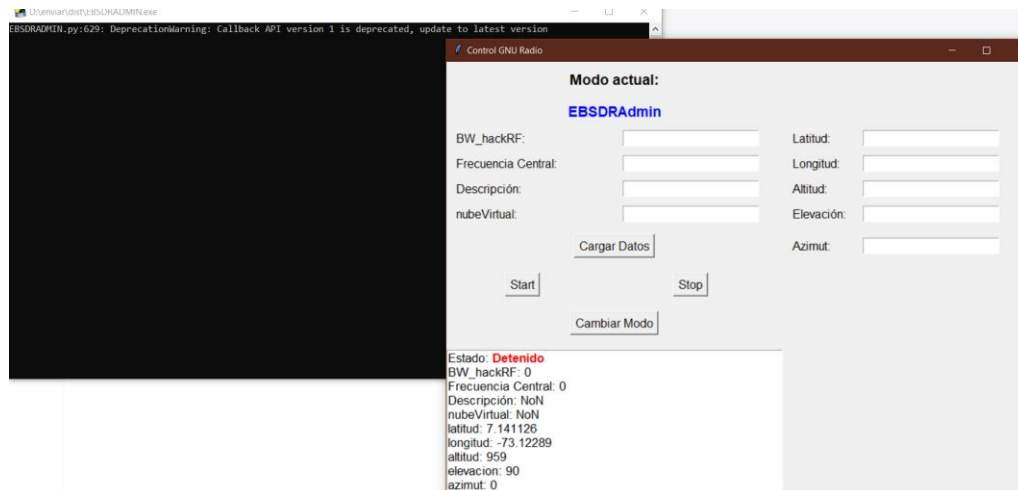


Fig 20 Interfaz de EBSDRADMIN y consola

- **Segundo paso:** En un apartado [Sección 2](#), se explicó lo correspondiente a los valores que acepta cada una de las variables, por lo que para una primera prueba se recomienda ingresar valores random, solo hay que tener en cuenta 2 factores, **BW_hackRF** debe ser diferente de 0, porque este valor es el que se usa en los valores iniciales y funciona para reconocer que no se han cargado datos iniciales al sistema. además se recomienda para **nubeVirtual**, usar **prueba** como standard esto con el fin de solo usar un archivo. csv para realizar pruebas y reconocerlo fácilmente.
- **Tercer paso:** una vez cargados los datos para realizar pruebas y que sean validos se debe pulsar el botón de **start**, si Estado=iniaciado, no se imprime algún error en la consola, esto significa que todo está funcionando correctamente, tal como se muestra en [Fig 21](#), esta significa que **EBSDRADMIN** ya esta listo para usarse.

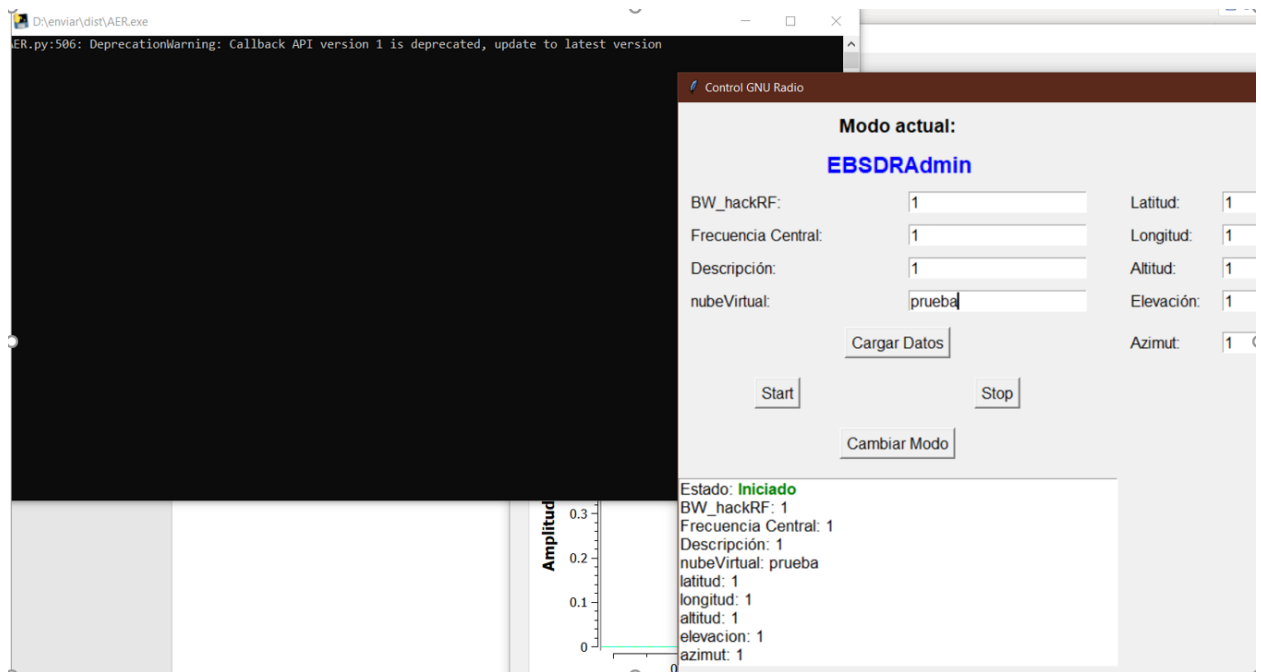


Fig 21 Pruebas de funcionamiento usando datos random

- **Cuarto paso:** Si el tercer paso no funcionó correctamente y en la interfaz debajo de Estado apareció el mensaje: "Última ejecución de **GNU falló**" y en la consola imprime un mensaje el cual inicia por: "**CRITICAL**" "ó **Error**" Fig 22 dirigirse al apartado de errores del documento.

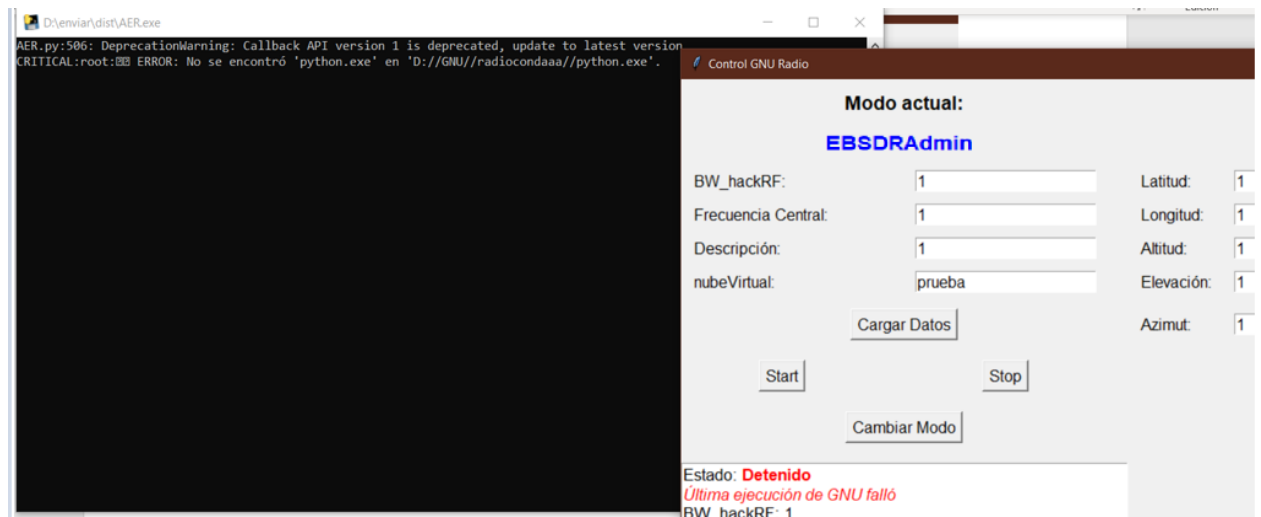
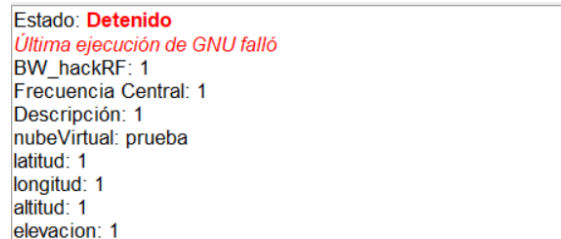


Fig 22 Demostración Fallo

5.3. Errores

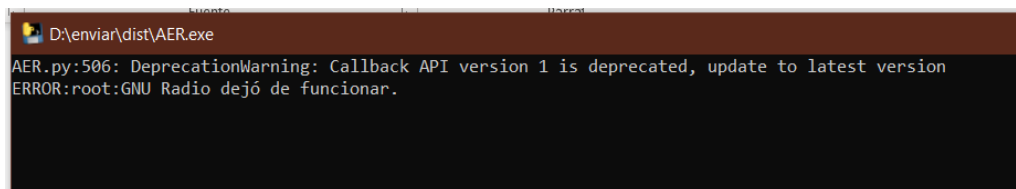
Pueden ocurrir errores a la hora de intentar realizar la ejecución de **GNU radio**, las razones son varias, pero todas en la interfaz se imprimirá el mensaje: "Última ejecución de **GNU falló**" [Fig 23](#).



```
Estado: Detenido  
Última ejecución de GNU falló  
BW_hackRF: 1  
Frecuencia Central: 1  
Descripción: 1  
nubeVirtual: prueba  
latitud: 1  
longitud: 1  
altitud: 1  
elevacion: 1
```

Fig 23 Reporte de variables en caso de fallos

5.3.1. Caso 1 GNU dejó de funcionar

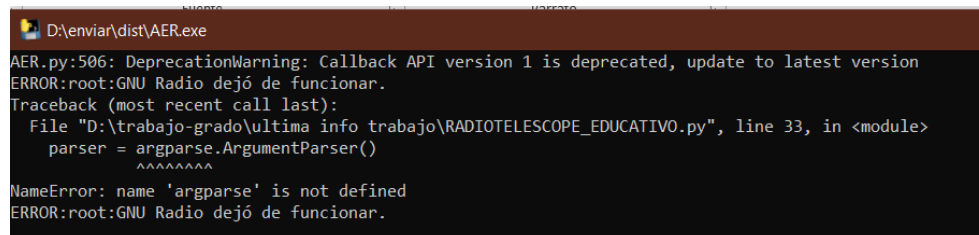


```
D:\enviar\dist\AER.exe  
AER.py:506: DeprecationWarning: Callback API version 1 is deprecated, update to latest version  
ERROR:root:GNU Radio dejó de funcionar.
```

Fig 24 Error GNU dejó de funcionar

Bueno, este caso puede ocurrir por varias situaciones:

- **Primera:** El .py no fue generado por **GNU radio**, sino es un script de Python creado de forma externa a **GNU radio**, el programa solo valida si es un .py, por ejemplo si es un script que realiza alguna acción y se finaliza, esta situación se cataloga como cierre inesperado [Fig 24](#), se debe cerrar el programa configurar un .py valido en el JSON y volverlo a iniciar.
- **Segunda:** El .py fue generado por **GNU radio**, pero puede pasar que al realizarle las respectivas modificaciones ocurrió algún problema [Fig 24](#), se recomienda revisar el apartado de [script gnuradio](#), el siguiente es un ejemplo de eso:



```
D:\enviar\dist\AER.exe  
AER.py:506: DeprecationWarning: Callback API version 1 is deprecated, update to latest version  
ERROR:root:GNU Radio dejó de funcionar.  
Traceback (most recent call last):  
  File "D:\trabajo-grado\ultima info trabajo\RADIOTELESCOPE_EDUCATIVO.py", line 33, in <module>  
    parser = argparse.ArgumentParser()  
            ^^^^^^^^^  
NameError: name 'argparse' is not defined  
ERROR:root:GNU Radio dejó de funcionar.
```

Fig 25 Error GNU dejó de funcionar, variante error al configurar el .py

- **Tercera:** El .py fue generado por **GNU radio**, y su funcionamiento es correcto, pero por ejemplo varios diagramas de **GNU radio** muchas veces se configuran para abrir interfaces para graficar, si se cierra esa interfaz se finaliza la ejecución de **GNU radio**, esto causa que se imprima este mensaje en la consola, para solucionarlo basta con solamente volver a pulsar el botón de start.

5.3.1. Caso 2 *python_gnuradio*

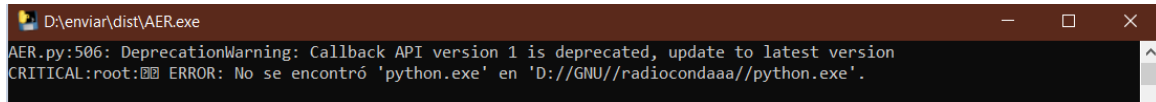


Fig 26 Error python_gnuradio

Esto ocurre cuando la dirección de **python_gnuradio** no es correcta y python.exe no se encuentra en ese directorio [Fig 26](#), se debe revisar la dirección ingresada en el JSON, se recomienda revisar [python_gnuradio](#), se debe cerrar la aplicación, colocar una dirección del archivo python.exe valida y volver a iniciar el programa.

5.3.1. Caso 3 *script_gnuradio*

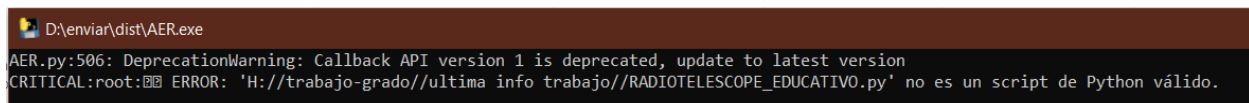


Fig 27 Error script_gnuradio

Esto [Fig 27](#) ocurre por 2 posibles situaciones:

- **Primera:** La dirección donde se encuentra el .py de **GNU radio** está incorrecta o no existe.
- **Segunda:** La dirección es correcta, pero el archivo que se está intentando usar no es un .py o su nombre está mal escrito en la dirección.

Para ambos casos se recomienda revisar el apartado [script_gnuradio](#)

5.3.2. Caso 4 *carpeta_base*

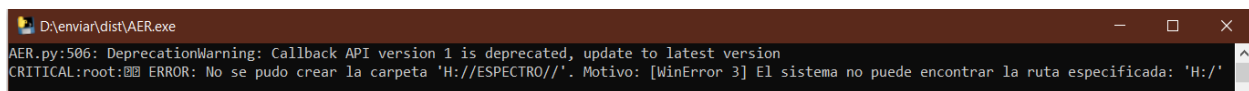


Fig 28 Error carpeta_base

Este error [Fig 28](#) ocurre cuando la ruta especificada para guardar los archivos .csv no es válida. Puede deberse a que la letra de la unidad de almacenamiento fue ingresada incorrectamente o a que la ubicación seleccionada tiene restricción de permisos.

Por ejemplo, si el equipo tiene dos discos duros:

- **Unidad C:** Disco principal
- **Unidad D:** Segundo disco

Pero en la configuración de la ruta se ingresa la letra **H:**, y el equipo no tiene una unidad con esa letra, el sistema no podrá encontrar el destino y generará un error, ya que si la ruta elegida no existe, se intentara crearla automáticamente. Se recomienda seguir las instrucciones del apartado [carpeta base](#).

5.4. Ejecución

Una vez verificada la correcta ejecución de **EBSDRADMIN.exe** mediante la [¡Error! No se encuentra el origen de la referencia.](#), se puede proceder con su uso:

- **Iniciar EBSDRADMIN.exe:** Ejecutar **EBSDRADMIN.exe**; esto abrirá la interfaz y la consola.

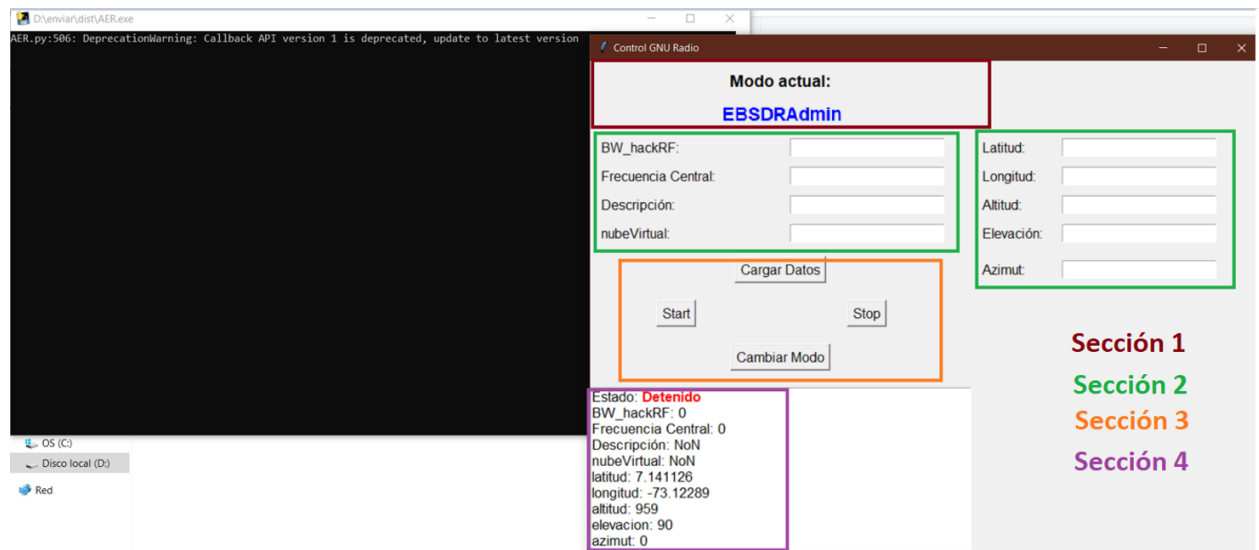


Fig 29 Interfaz de EBSDRADMIN y consola

- **Modo actual:**

Por defecto, al iniciarse, **EBSDRADMIN** estará en "**Modo actual: EBSDRADMIN**", lo que se puede visualizar en la interfaz en [*¡Error! No se encuentra el origen de la referencia.*](#)

- En este modo, la modificación de variables y el control de **GNU Radio** se realizan desde la interfaz de **EBSDRADMIN**.
- Si la interfaz muestra "**Modo actual: Servidor**", significa que la modificación de variables y el control se realizarán mediante **Node-RED**, y **EBSDRADMIN** no permitirá realizar cambios en las variables ni controlar **GNU Radio** desde su interfaz.
- Para cambiar entre "**EBSDRADMIN**" y "**Servidor**", se debe utilizar el botón "**Cambiar Modo**", cuyo funcionamiento y ubicación se explican en [*¡Error! No se encuentra el origen de la referencia.*](#)

Es importante destacar que EBSDRADMIN inicia con valores predeterminados que no son válidos, por lo que antes de intentar controlar GNU Radio, el usuario debe ingresar las variables requeridas.

También EBSDRADMIN no guarda las variables configuradas tras su cierre; cada vez que se inicie, será necesario volver a ingresarlas.

- **Configurar variables:** Para esto hay que llenar todos los campos de las variables que se muestran en [*Sección 2*](#) según las necesidades del usuario, y luego para cargar las variables se usa el botón llamado "Cargar Datos" cuyo funcionamiento y ubicación se muestran en [*¡Error! No se encuentra el origen de la referencia.*](#), el cual actualizará las variables en EBSDRADMIN y estos cambios se reflejarán en la interfaz en [*¡Error! No se encuentra el origen de la referencia.*](#)
- **Dar órdenes a GNU radio:** una vez en EBSDRADMIN se encuentren impresas en la interfaz [*¡Error! No se encuentra el origen de la referencia.*](#) las variables deseadas, se puede proceder a controlar GNU radio por medio de los botones "Start" y "Stop" cuyo funcionamiento y ubicación se muestran en [*¡Error! No se encuentra el origen de la referencia.*](#), en la interfaz cuando se presiona el botón "Start", en la interfaz [*¡Error! No se encuentra el origen de la referencia.*](#), se reflejara "Estado: **iniciado**" y la ejecución de GNU radio comenzara [*Fig 30*](#) (si hay algún error "Estado: **Detenido**" y se imprimirá en la interfaz "**Última ejecución de GNU falló**" [*Fig 30*](#) en cuyo caso se debe revisar [*¡Error!*](#)

No se encuentra el origen de la referencia.), si se pulsa el botón "Stop" en la interfaz ¡Error! No se encuentra el origen de la referencia., se reflejara "Estado: **Detenido**" y la ejecución de GNU radio finalizara.

Estado: Iniciado BW_hackRF: 6.000M Frecuencia Central: 4.500G Descripción: demostracion nubeVirtual: demostracion latitud: 8 longitud: 8 altitud: 8 elevacion: 8 azimut: 8	Estado: Detenido BW_hackRF: 6.000M Frecuencia Central: 4.500G Descripción: demostracion nubeVirtual: demostracion latitud: 8 longitud: 8 altitud: 8 elevacion: 8 azimut: 8
Orden "Start"	Orden "Stop"

Fig 30 Ejemplo de lo que se imprime en la interfaz cuando no hay fallos

- **Cerrar EBSDRADMIN:** Para cerrar EBSDRADMIN basta con cerrar la interfaz de EBSDRADMIN o la consola, esto finalizara automáticamente el programa.

VI. Código

```
EBSDRADMIN.py > ...
1  #----- inicio importar librerias necesarias
2  import paho.mqtt.client as mqtt
3  import subprocess
4  import json
5  import time
6  import threading
7  import tkinter as tk
8  from tkinter import messagebox
9  import sys
10 import re
11 from tkinter import *
12 from tkinter import ttk
13 import json
14 import os
15 import logging
16 #----- fin importe de librerias necesarias
17
18
19
20 #----- inicio Configuración de logging
21
22 # ♦ Niveles de logging disponibles:
23
24 # logging.DEBUG    -> Nivel 10  (Mensajes detallados para depuración)  menor prioridad
25 # logging.INFO     -> Nivel 20  (Mensajes informativos)
26 # logging.WARNING  -> Nivel 30  (Advertencias que no detienen el programa)
27 # logging.ERROR    -> Nivel 40  (Errores que pueden afectar el funcionamiento)
28 # logging.CRITICAL -> Nivel 50  (Errores fatales que requieren detener el programa) mayor prioridad
29
30 logging.basicConfig(level=logging.ERROR) #error,critical
31
```

Fig 31 Código de EBSDRADMIN

Para ejecutar y modificar el código de **EBSDRADMIN** Fig 31, se recomienda hacerlo en un equipo físico de uso local y utilizar entornos como **Visual Studio Code**, ya que facilita la edición eficiente del código y la ejecución de scripts en Python.

No todas las plataformas son adecuadas para este propósito. Por ejemplo, en entornos basados en la nube como **Google Colab**, no es posible abrir la interfaz gráfica de **EBSDRADMIN** ni ejecutar GNU Radio completamente.

El código fuente **EBSDRADMIN.py** y **config.json** se encuentra disponible en la siguiente URL:
https://drive.google.com/drive/folders/1GUJ2LGEMVtVjto02RmIwFCU37liTsOSd?usp=drive_link

El código se encuentra debidamente explicado por medio de comentarios los cuales describen su funcionamiento

6.1. Funcionamiento del código

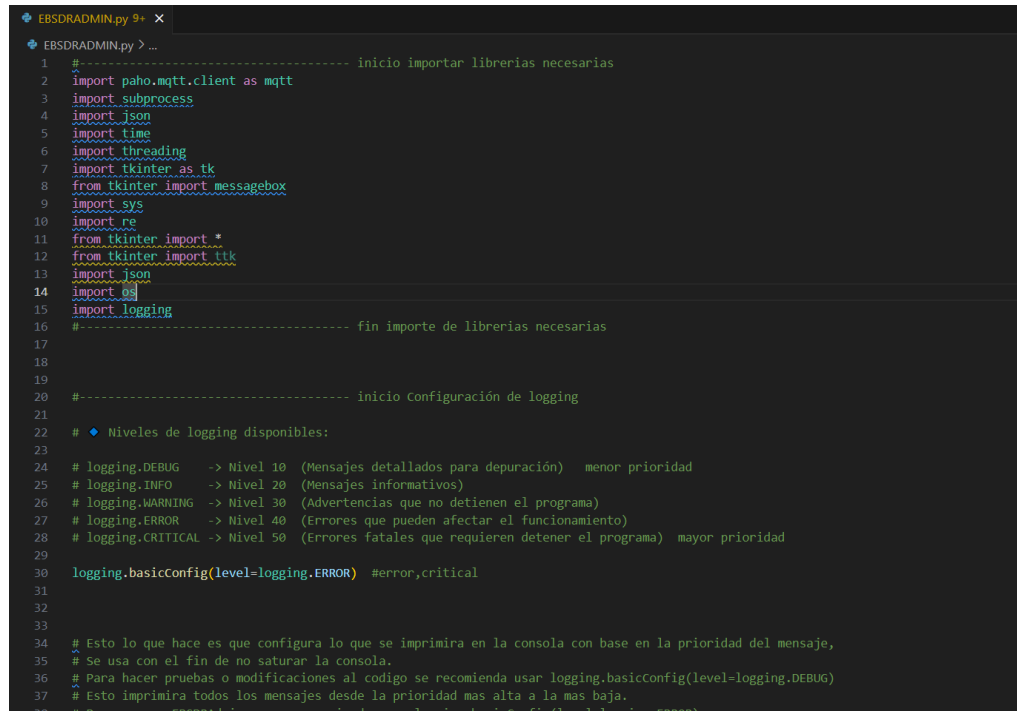
En apartados anteriores ya se ha explicado el funcionamiento de **EBSDRADMIN**, por lo que aquí se profundizara un poco mas en como es el funcionamiento del código:

- Al ejecutar **EBSDRADMIN** se abrirá una interfaz en la cual se pueden ingresar variables y dar órdenes para sobrescribir las variables que ya están cargadas, iniciar o detener **GNU radio** y configurar el Modo actual.
- **EBSDRADMIN** se comunicara con **Node-red** usando MQTT, además todos los mensajes que sean publicados o recibidos por **EBSDRADMIN**, contendrán la correspondiente ID, descrita en el apartado [ID](#), si la ID en los mensajes recibidos por **EBSDRADMIN** no corresponde a la configurada en el **JSON**, el mensaje será ignorado, lo mismo si la ID que se encuentra en los mensajes publicados por **EBSDRADMIN** no corresponde con la que se encuentra en la pestaña de **Node-red** a utilizar, serán ignorados.
- **EBSDRADMIN** al establecer comunicación con **Node-red**, cada cierto intervalo de tiempo publicara en los tópicos las variables, estado de la ejecución de **GNU radio** y Modo actual en ese instante, lo cual usa **Node-red** para mantenerse actualizado, identificar que si hay conexión con **EBSDRADMIN** y con base en lo recibido habilitar o restringir acciones a realizar.
- **Node-red**, también cada cierto intervalo de tiempo en un tópico publicara un mensaje, **EBSDRADMIN** si después de un intervalo de tiempo no llega el mensaje publicado por ese tópico determina que no hay conexión con **Node-red** y deja de publicar en los tópicos las variables, estado de la ejecución de **GNU radio**, Modo actual y cada cierto intervalo de tiempo tratara de volver a suscribirse a los tópicos y con ello restablecer conexión con **Node-red**, una vez lo logre se habilitara nuevamente la publicación en los tópicos.
- **EBSDRADMIN**, comenzara a procesar los mensajes recibidos por MQTT y a publicar en los tópicos, solo hasta que se realice la suscripción a los tópicos.

6.2. Creación EBSDRADMIN.exe

- **Primer paso:** Crear una carpeta, nombrarla y en ella guardar a **EBSDRADMIN.py** y **config.json** cuya ubicación se muestra en [Código](#)

- **Segundo paso:** Abrir la herramienta que se quiera usar para la edición de código en este caso se usara visual studio code, y abrir **EBSDRADMIN.py**



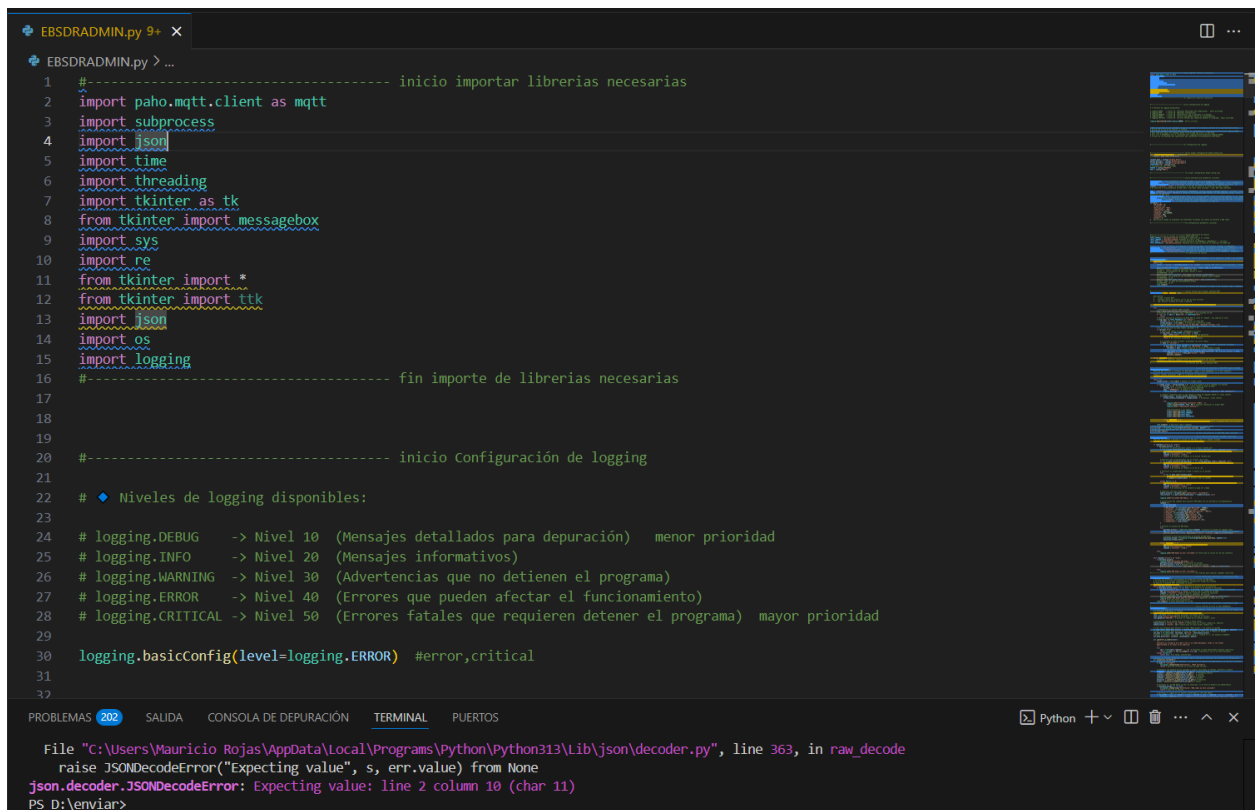
```

1  #----- inicio importar librerias necesarias
2  import paho.mqtt.client as mqtt
3  import subprocess
4  import json
5  import time
6  import threading
7  import tkinter as tk
8  from tkinter import messagebox
9  import sys
10 import re
11 from tkinter import *
12 from tkinter import ttk
13 import json
14 import os
15 import logging
16 #----- fin importe de librerias necesarias
17
18
19
20 #----- inicio Configuración de logging
21
22 # ♦ Niveles de logging disponibles:
23
24 # logging.DEBUG -> Nivel 10 (Mensajes detallados para depuración) menor prioridad
25 # logging.INFO -> Nivel 20 (Mensajes informativos)
26 # logging.WARNING -> Nivel 30 (Advertencias que no detienen el programa)
27 # logging.ERROR -> Nivel 40 (Errores que pueden afectar el funcionamiento)
28 # logging.CRITICAL -> Nivel 50 (Errores fatales que requieren detener el programa) mayor prioridad
29
30 logging.basicConfig(level=logging.ERROR) #error,critical
31
32
33
34 # Esto lo que hace es que configura lo que se imprimira en la consola con base en la prioridad del mensaje,
35 # Se usa con el fin de no saturar la consola.
36 # Para hacer pruebas o modificaciones al codigo se recomienda usar logging.basicConfig(level=logging.DEBUG)
37 # Esto imprimira todos los mensajes desde la prioridad mas alta a la mas baja.
38 # Se usa con el fin de no saturar la consola.

```

Fig 32 Codigo abierto en visual studio code

- **Tercer paso:** Abrir la consola de Python la cual una manera de hacerlo es iniciar una ejecución del código y cerrar la ventana de la interfaz de **EBSDRADMIN**.



```
1 #----- inicio importar librerias necesarias
2 import paho.mqtt.client as mqtt
3 import subprocess
4 import json
5 import time
6 import threading
7 import tkinter as tk
8 from tkinter import messagebox
9 import sys
10 import re
11 from tkinter import *
12 from tkinter import ttk
13 import json
14 import os
15 import logging
16 #----- fin importe de librerias necesarias
17
18
19
20 #----- inicio Configuración de logging
21
22 # ♦ Niveles de logging disponibles:
23
24 # logging.DEBUG -> Nivel 10 (Mensajes detallados para depuración) menor prioridad
25 # logging.INFO -> Nivel 20 (Mensajes informativos)
26 # logging.WARNING -> Nivel 30 (Advertencias que no detienen el programa)
27 # logging.ERROR -> Nivel 40 (Errores que pueden afectar el funcionamiento)
28 # logging.CRITICAL -> Nivel 50 (Errores fatales que requieren detener el programa) mayor prioridad
29
30 logging.basicConfig(level=logging.ERROR) #error,critical
31
32
```

PROBLEMAS 202 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

File "C:\Users\Mauricio Rojas\AppData\Local\Programs\Python\Python313\Lib\json\decoder.py", line 363, in raw_decode
raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 2 column 10 (char 11)
PS D:\enviar>

Fig 33 Consola de python

- **Cuarto paso:** en la consola de Python ingresaremos el comando:” **pyinstaller --onefile EBSDRADMIN.py**”, **importante** debe ser ingresado sin las comillas: “”, además de que **EBSDRADMIN.py** hace referencia al archivo, por lo que la carpeta que se debe usar en visual studio code es en la que se encuentra, un caso mas general consiste en usar la direccion en la cual se encuentre **EBSDRADMIN.py** Ejemplo:
 - Identificar la dirección de la ruta donde se guardó **EBSDRADMIN.py**

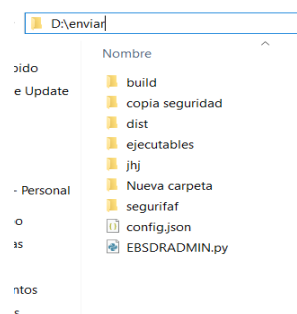
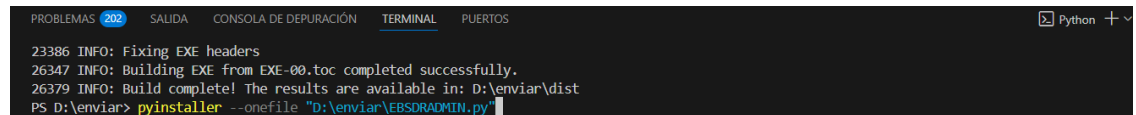


Fig 34 Directorio de EBSDRADMIN

En este caso es “D:\enviar”, se le agrega al final”: “\EBSDRADMIN.py”

Y quedaría "D:\enviar\EBSDRADMIN.py", finalmente el comando quedaría de esta forma: **pyinstaller --onefile "D:\enviar\EBSDRADMIN.py"**, se recomienda que la dirección del archivo este entre comillas esto debido a que si en alguna parte del directorio tiene un espacio, en Python no será reconocido adecuadamente.



```
PROBLEMAS 202 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Python +
23386 INFO: Fixing EXE headers
26347 INFO: Building EXE from EXE-00.toc completed successfully.
26379 INFO: Build complete! The results are available in: D:\enviar\dist
PS D:\enviar> pyinstaller --onefile "D:\enviar\EBSDRADMIN.py"
```

Fig 35 Ejemplo comando ingresado forma general

- Lo que hace este comando es crear un ejecutable independiente (.exe), de **EBSDRADMIN.py**, En la carpeta que se esté usando en visual studio code se crearan 2 carpetas, las cuales son “dist” y “build”, la que es de interés es “dist”, ya que en ella se guardara el EBSDRADMIN.exe generado, “build” son archivos temporales los cuales pueden ser eliminados sin ningún problema.

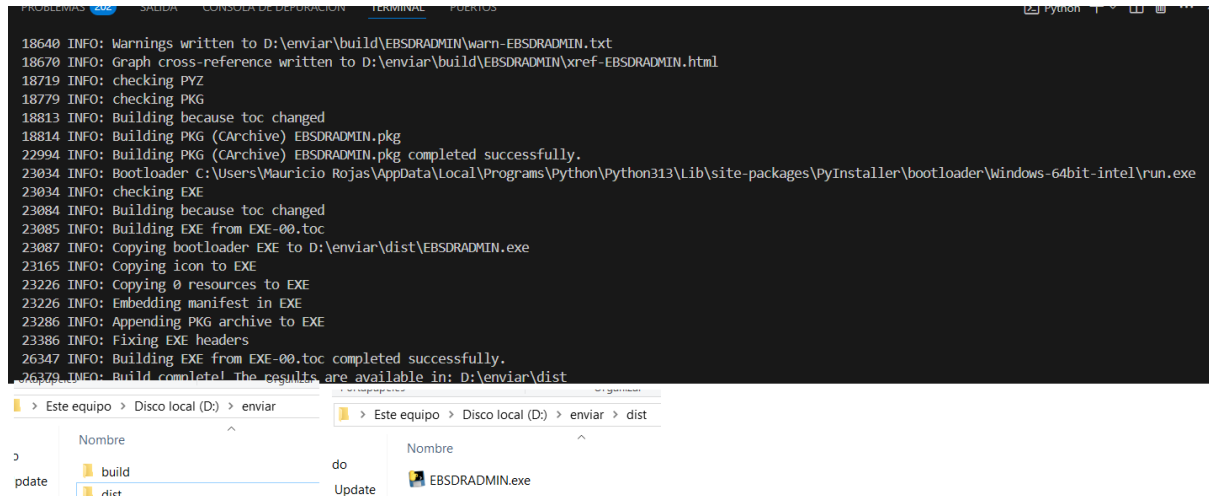


Fig 36 Resultados ejecución comando para creación de EBSDRADMIN.exe

VII. Mejoras y desarrollo futuro

En estos momentos se puede decir que esta es una primera versión funcional de **EBSDRADMIN**, por lo que esta no es una versión la cual se pueda decir que es una versión final y no que no se tenga posibilidades de mejora, algunas cosas que se pueden mejorar en el futuro pueden ser:

- Optimizar el código existente. Existe posibilidad de que existan maneras de las cuales se puedan realizar algunas de las acciones del código de forma más eficiente
- Mejora referente a los parámetros que se configuran en el JSON. Se debe diseñar una manera más eficiente y sencilla por medio de la cual los usuarios puedan configurar dichos parámetros de forma más eficiente.
- **EBSDRADMIN** cuando se reinicia no almacena las variables que se usaron en una última ejecución. En casos como por ejemplo un apagón, se apagarían todos los sistemas y por ende la ejecución de **EBSDRADMIN** se detendría y al reiniciarlo se requeriría configurarlo nuevamente y esto implica que el usuario tiene que tener un registro externo de los parámetros usados.
- **EBSDRADMIN** no se inicia automáticamente. En casos como un apagón lo ideal es que cuando se restablezca el servicio y los equipos se reinicien **EBSDRADMIN** se iniciara y reiniciara el .py con la última configuración que guardo antes del apagón.
- Optimizar el como se manejan los errores ya que actualmente si una configuración del JSON es incorrecta **EBSDRADMIN** debe cerrarse corregir el JSON y volver a ejecutar el programa