

**AGRICULTURA DE PRECISIÓN PARA RIEGO INTELIGENTE DE CULTIVOS DE CAFÉ
UTILIZANDO TECNOLOGÍAS DE INTERNET DE LAS COSAS**

ANEXO B



Edinsson Alberto López López

Juan Diego Yasnó Collo

Director: PhD. Miguel Ángel Niño Zambrano

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Sistemas

Grupo I+D en Tecnologías de la Información - GTI

Popayán, julio de 2022

Contenido

Índice de figuras	4
Anexo B	6
Instalación IDE Arduino	6
Configuración entorno de desarrollo IDE Arduino	9
Ajustes IDE para creación de la malla con las placas ESP32	11
Configuración del bróker MQTT en la Raspberry Pi	13
Conexión Raspberry Pi con la Malla opción librería PainlessMeshBoost	17
Conexión de Raspberry Pi con la malla opción MQTT	21
Guardar datos provenientes de la malla en una base de datos en la Raspberry Pi	24
Configuración de módulo 'LoRa/GPS HAT' instalado en la Raspberry Pi para envío de mensajes mediante LoRa:	32
Configuración del entorno objeto semántico en la Raspberry Pi:	35
Instalación de sensores a los nodos de la malla:	38

Índice de figuras

Figura 1	Página oficial descarga IDE Arduino	5
Figura 2	Aceptar términos de licencia	6
Figura 3	Instalación drivers	6
Figura 4	Ubicación de instalación	7
Figura 5	Finalización de instalación del IDE	7
Figura 6	Administrar Bibliotecas	8
Figura 7	Búsqueda de instalación librería	9
Figura 8	Instalación librerías adicionales requeridas por librería Painless Mesh	9
Figura 9	Ubicación ejemplo StartHere	10
Figura 10	Sketch startHere	11
Figura 11	Elección de la placa	12
Figura 12	Velocidad de carga	¡Error! Marcador no definido.
Figura 13	Puertos COM	12
Figura 14	Comando para actualizar paquetes	13
Figura 15	Comando para actualizar sistema operativo	13
Figura 16	Instalación bróker MQTT	13
Figura 17	Comando para ver el estado del servicio	13
Figura 18	Resultado del comando para ver el servicio	13
Figura 19	Comando de suscripción al bróker MQTT	13
Figura 20	Comando para publicación en el bróker MQTT	14
Figura 21	Mensaje recibido al suscribirse al tópico específico	14
Figura 22	Publicación rechazada en bróker MQTT	14
Figura 23	Suscripción rechaza en bróker MQTT	14
Figura 24	Puerto utilizado por Mosquitto	14
Figura 25	Ruta archivo de configuración Mosquitto	14
Figura 26	Archivo de configuración sin modificar	15
Figura 27	Archivo de configuración modificado	15
Figura 28	Comando para reiniciar Mosquitto	15
Figura 29	Comando para comprobar el estado del servicio	15
Figura 30	Estado del servicio Mosquitto	16
Figura 31	puerto usado por Mosquitto	16
Figura 32	descarga librería PainlessMeshBoots	17
Figura 33	instalación librerías Boost desde ventana comandos Raspberry Pi	17
Figura 34	directorio raíz librería Painlessmesh	17
Figura 35	detección del compilador y generación del entorno de compilación en el directorio actual	18
Figura 36	generación del ejecutable e instalación de la librería Painlessmesh en la Raspberry Pi	18
Figura 37	configuración nodo que permite la conexión entre la malla y la Raspberry Pi	19
Figura 38	Subir Código a la placa	19
Figura 39	interacción malla con Raspberry Pi	20
Figura 40	Ejecución ejemplo LoRa parámetro "sender"	31
Figura 41	Ejecución ejemplo LoRa parámetro "res"	31
Figura 42	Pines LoRa/GPS HAT	32

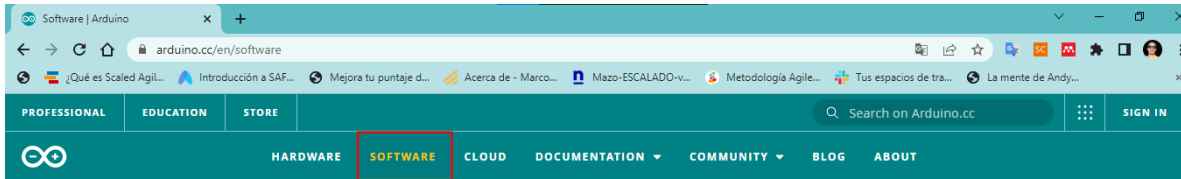
Figura 43 Pines Raspberry Py	33
Figura 44 correspondencia pines modulo RFM con pines GPIO RPi	33
Figura 45 Cableado Pines LoRa HAT y Raspberry Pi	34
Figura 46 correspondencia pines modulo SX1276 con pines GPIO RPi	34
Figura 47 ejecución script	35
Figura 48 permisos archivo utils.py	35
Figura 49 Líneas de código para agregar	35
Figura 50 carga del objeto	36
Figura 51 crear ejecutables	36
Figura 52 ejecutables creados a partir de JSON objeto	36
Figura 53 mini-protoboard	37
Figura 54 LM35	37
Figura 55 YL69	38
Figura 56 Fotorresistencia LDR	39
Figura 57 pines usados ESP32	39
Figura 58 conexión sensores con ESP32 1 pin GND, 2 pin Vcc	40

Anexo B

En este anexo se presenta una serie de pasos los cuales no pueden ser mostrados en la monografía, los cuales son de importancia para comprender el proceso y funcionamiento realizado en la prueba de concepto.

Instalación IDE Arduino

Descargamos el IDE de la página oficial de Arduino <https://www.arduino.cc/en/software> des de la ventana software en nuestro caso la versión 1.8.16 para Windows 10 Figura 1



Downloads



Figura 1 Página oficial descarga IDE Arduino

Con el archivo de instalación descargados se procede a la instalación de este, damos doble clic sobre el archivo, donde se desplegará una ventana que solicita aceptar los términos de licencias para continuar con la instalación Figura 2.

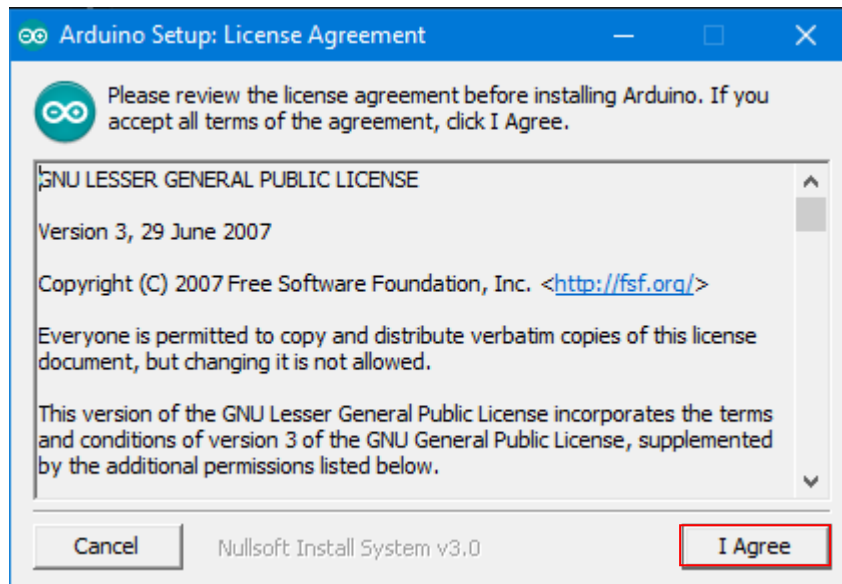


Figura 2 Aceptar términos de licencia

Aceptados los términos de licencia se procede con la instalación de los diferentes drivers para el correcto reconocimiento y funcionamiento de las placas que se conecten a nuestra computadora Figura 3.

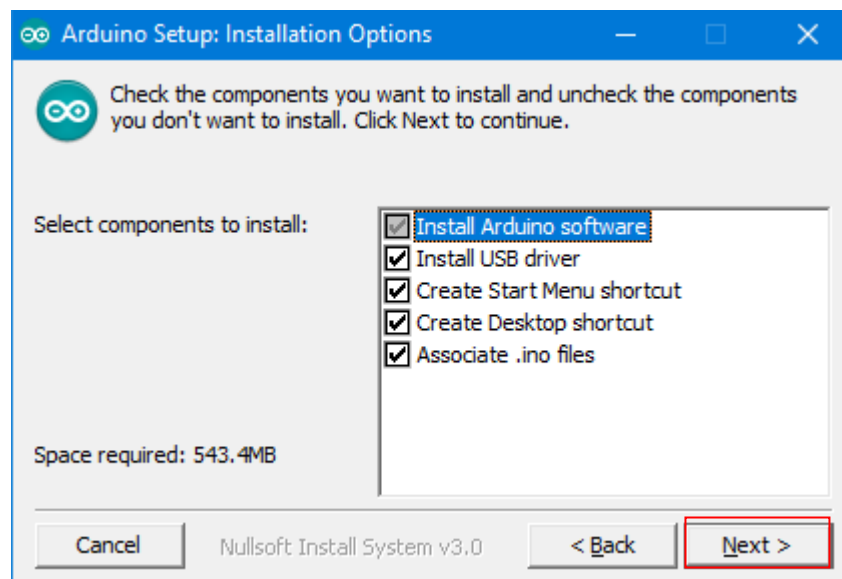


Figura 3 Instalación drivers

Teniendo instalados los drivers se procede a seleccionar la carpeta de instalación se puede dejar la ubicación que viene predeterminada y solo presionamos en instalar Figura 4

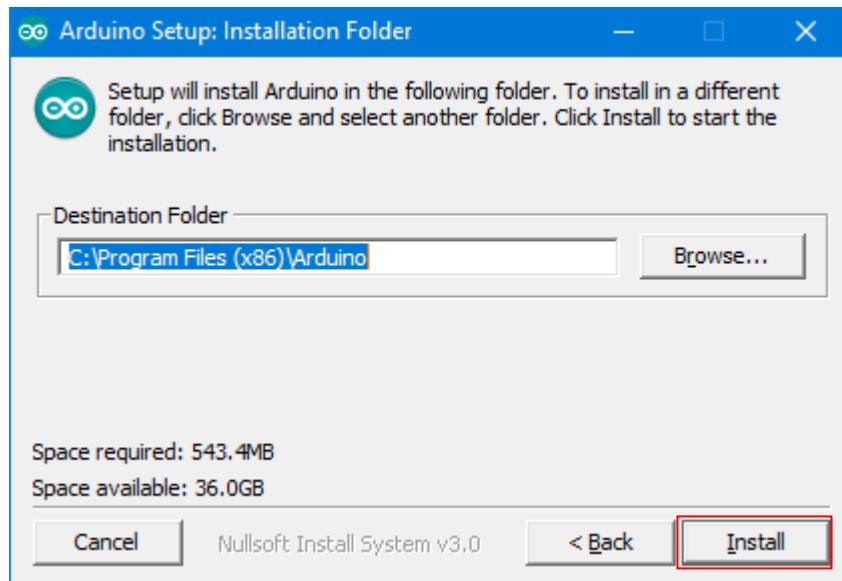


Figura 4 Ubicación de instalación

Terminado el proceso anterior cerramos la ventana de instalación Figura 5.

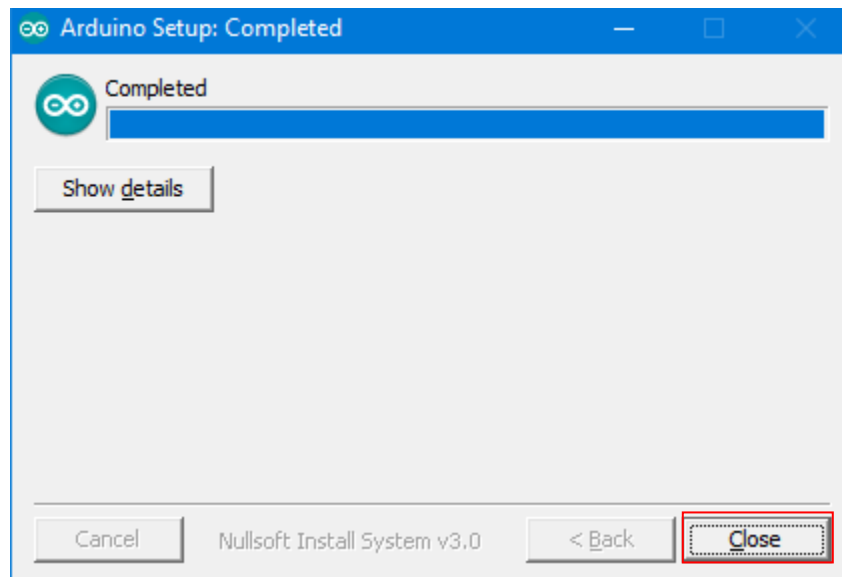


Figura 5 Finalización de instalación del IDE

Configuración entorno de desarrollo IDE Arduino

Para el uso del entorno de desarrollo se hace necesario la instalación de diferentes librerías, en nuestro caso la librería Painless Mesh. Para la instalación de la librería se hace uso de la pestaña “Herramientas” de la barra superior, que al darle clic nos despliega una serie de opciones, hacemos uso de la opción Administrar Bibliotecas Figura 6

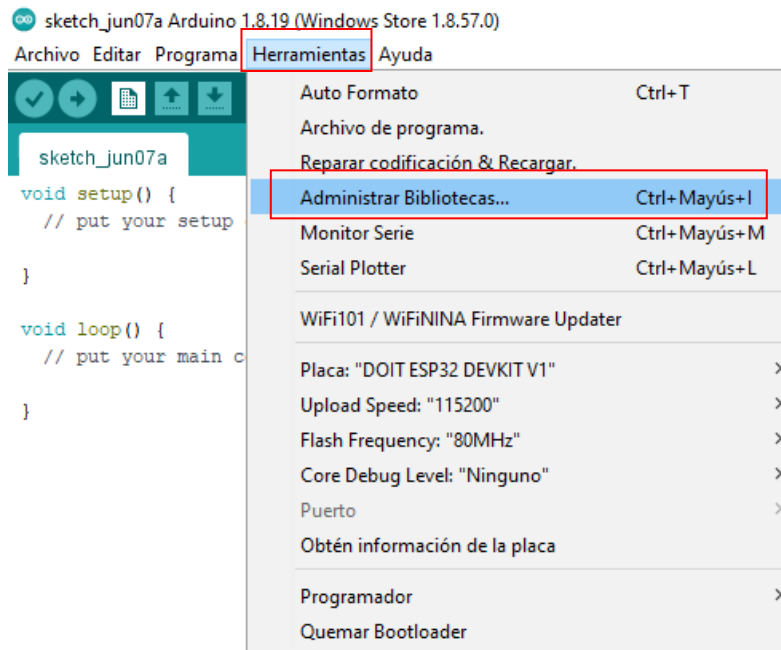


Figura 6 Administrar Bibliotecas

Al dar clic en la opción “Administrar Bibliotecas” se desplegará una pantalla donde debemos ingresar el nombre de la librería a instalar, este buscará las librerías disponibles donde debemos seleccionar la librería requerida en nuestro caso Painless Mesh, también se debe seleccionar la versión (1.4.9) y por último proceder con la instalación Figura 7.

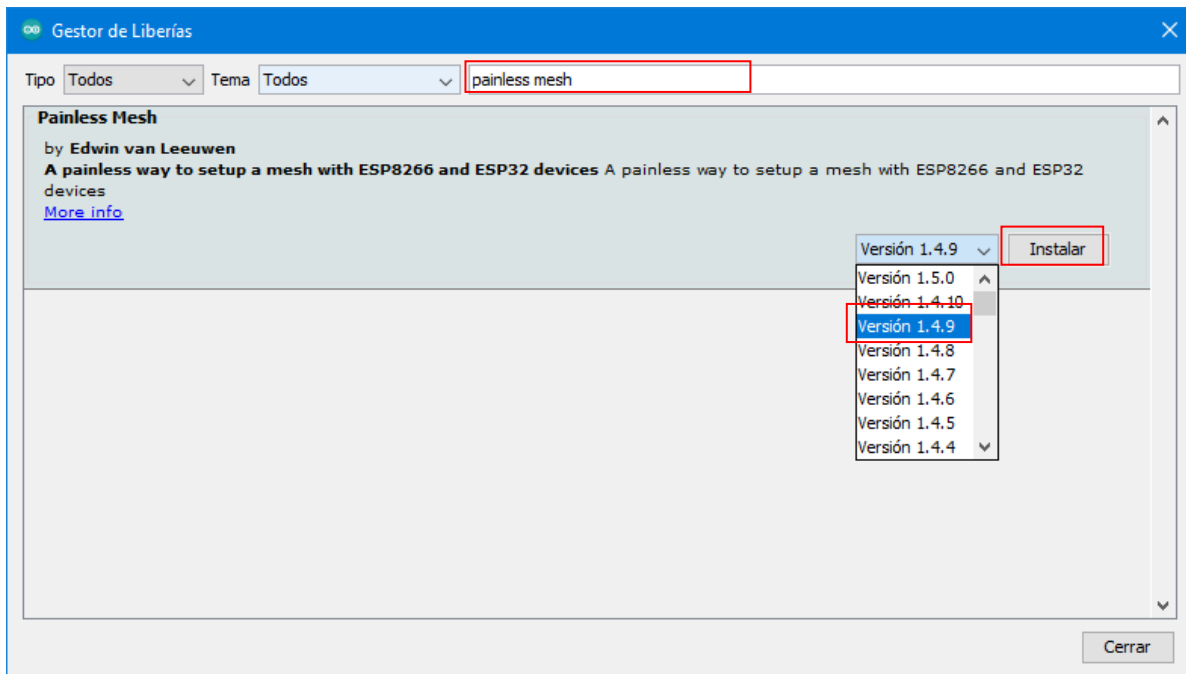


Figura 7 Búsqueda de instalación librería

Durante el proceso de instalación aparece una ventana con una sugerencia de instalación de librerías adicionales a las cuales daremos instalar todo Figura 8.

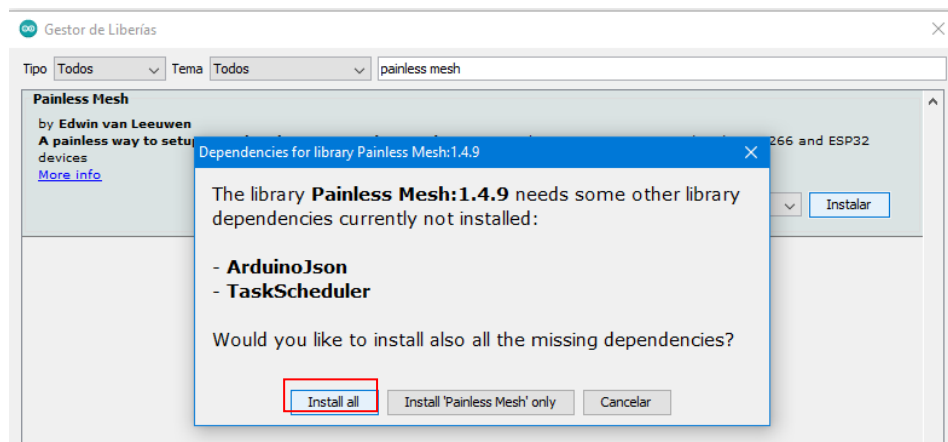


Figura 8 Instalación librerías adicionales requeridas por librería Painless Mesh

Ajustes IDE para creación de la malla con las placas ESP32

Teniendo las configuraciones previas se procede a cargar el código proporcionado en los ejemplos de la librería Painless Mesh, la prueba se realizó a partir del ejemplo StartHere Figura 12

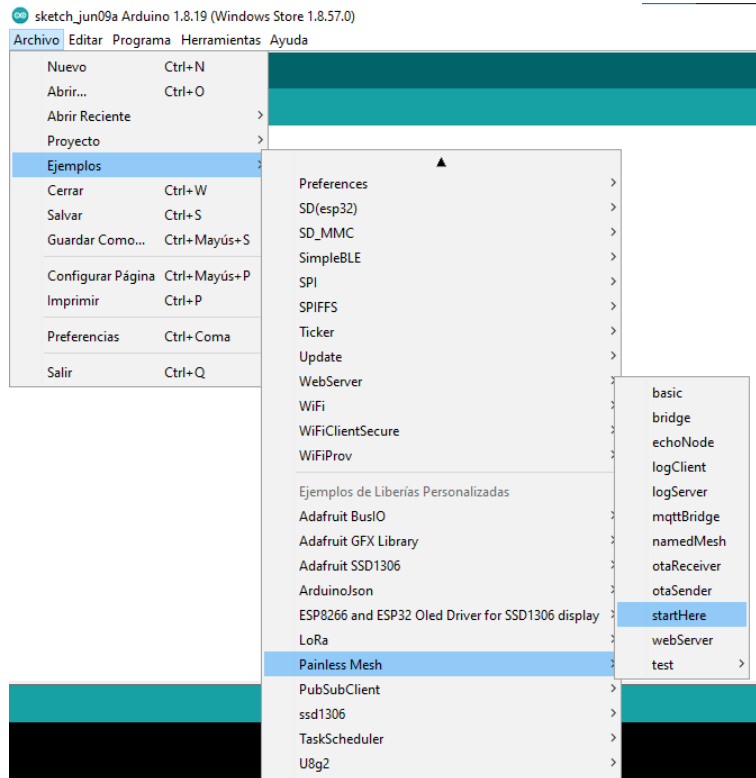
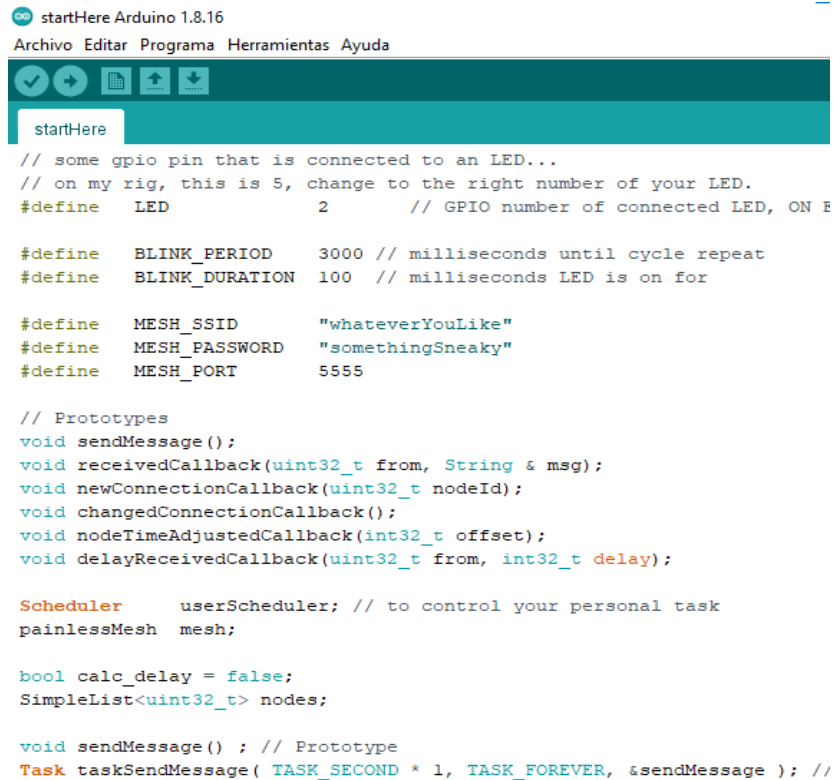


Figura 9 Ubicación ejemplo StartHere

Sketch usado para la creación de la malla



```
startHere Arduino 1.8.16
Archivo Editar Programa Herramientas Ayuda

// some gpio pin that is connected to an LED...
// on my rig, this is 5, change to the right number of your LED.
#define LED 2 // GPIO number of connected LED, ON F

#define BLINK_PERIOD 3000 // milliseconds until cycle repeat
#define BLINK_DURATION 100 // milliseconds LED is on for

#define MESH_SSID "whateverYouLike"
#define MESH_PASSWORD "somethingSneaky"
#define MESH_PORT 5555

// Prototypes
void sendMessage();
void receivedCallback(uint32_t from, String & msg);
void newConnectionCallback(uint32_t nodeId);
void changedConnectionCallback();
void nodeTimeAdjustedCallback(int32_t offset);
void delayReceivedCallback(uint32_t from, int32_t delay);

Scheduler userScheduler; // to control your personal task
painlessMesh mesh;

bool calc_delay = false;
SimpleList<uint32_t> nodes;

void sendMessage() ; // Prototype
Task taskSendMessage( TASK_SECOND * 1, TASK_FOREVER, &sendMessage ); //
```

Figura 10 Sketch startHere

En la pestaña de Herramientas debemos configurar la placa con la que vamos a trabajar, la velocidad de carga y el puerto en el que se encuentra nuestra placa conectada, en nuestro caso se trabajó con la placa DOIT ESP32 DEVKIT V1 Figura 9, una velocidad de carga de 115200 Figura 10 y el puerto COM Figura 11

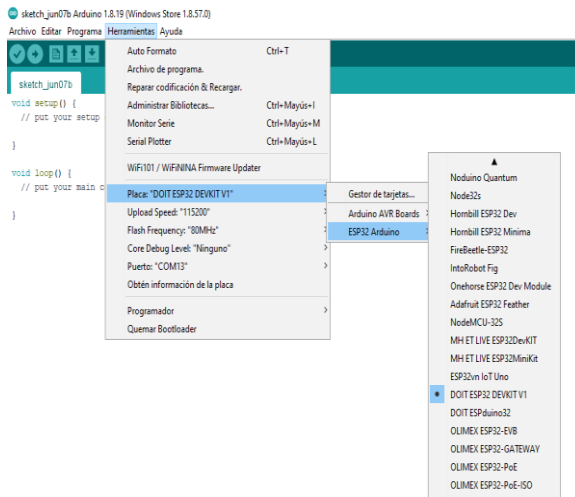


Figura 11 Elección de la placa

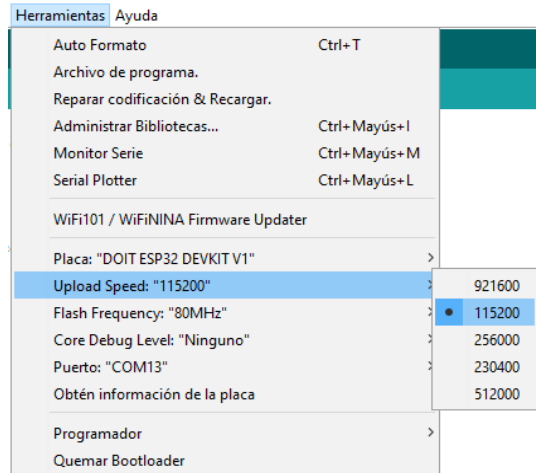


Figura 12 Velocidad de carga

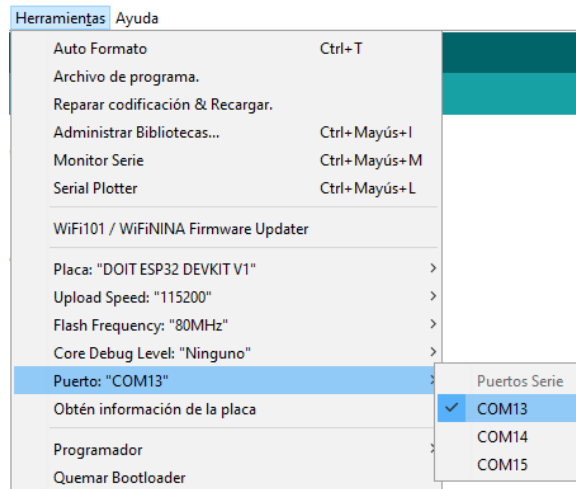


Figura 13 Puertos COM

NOTA: El puerto puede variar dependiendo del equipo donde se ha conectado la placa.

Configuración del bróker MQTT en la Raspberry Pi

La Raspberry Pi es capaz de funcionar como un bróker MQTT, esto es posible después de realizar una serie de configuraciones, pero antes de iniciar se debe actualizar el sistema operativo de la Raspberry Pi, para realizarlo primero se actualizan los repositorios del sistema (ver Figura 14) y posteriormente se actualizan los paquetes instalados (ver Figura 15).

```
pi@raspberrypi:~ $ sudo apt-get update
```

Figura 14 Comando para actualizar paquetes

```
pi@raspberrypi:~ $ sudo apt-get upgrade
```

Figura 15 Comando para actualizar sistema operativo

Luego de terminar el proceso de actualización, es posible iniciar las configuraciones para utilizar el bróker MQTT dentro de la Raspberry Pi, empezando con la instalación del bróker MQTT, en este caso el bróker elegido es Mosquitto debido a que se encuentra disponible dentro de los repositorios de Raspberry Pi OS. Para instalar Mosquitto se utiliza el siguiente comando (ver Figura 16)

```
pi@raspberrypi:~ $ sudo apt-get install mosquitto mosquitto-clients
```

Figura 16 Instalación bróker MQTT

Posterior a la instalación de Mosquitto debemos comprobar el estado del servicio con el siguiente comando (ver Figura 17)

```
pi@raspberrypi:~ $ sudo systemctl status mosquitto.service
```

Figura 17 Comando para ver el estado del servicio

Si todo ha salido bien el resultado del comando debe ser el siguiente (ver Figura 18)

```
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2022-06-10 11:21:08 -05; 3h 50min ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Process: 2566 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Process: 2567 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Process: 2568 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
   Process: 2569 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
   Main PID: 2570 (mosquitto)
     Tasks: 1 (limit: 1598)
           CPU: 5.421s
   CGroup: /system.slice/mosquitto.service
           └─2570 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

jun 10 11:21:08 raspberrypi systemd[1]: Starting Mosquitto MQTT Broker...
jun 10 11:21:08 raspberrypi systemd[1]: Started Mosquitto MQTT Broker.
```

Figura 18 Resultado del comando para ver el servicio

En este momento el bróker está habilitado para ser usado. Para probar su funcionamiento debemos abrir dos terminales, una que tenga el rol de suscriptor y otro que va a funcionar como publicador, para esto utilizamos los siguientes comandos (ver Figura 19 y Figura 20)

```
pi@raspberrypi:~ $ mosquitto_sub -h localhost -t "topicoPrueba"
```

Figura 19 Comando de suscripción al bróker MQTT

```
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t "topicoPrueba" -m "Hola mundo"
```

Figura 20 Comando para publicación en el bróker MQTT

Una vez se publique el mensaje el resultado del suscriptor debe ser el mostrado en la Figura 21

```
pi@raspberrypi:~ $ mosquitto_sub -h localhost -t "topicoPrueba"  
Hola mundo
```

Figura 21 Mensaje recibido al suscribirse al tópic específico

En este momento el bróker MQTT solo puede ser utilizado de manera local y no puede ser utilizado por dispositivos o servicios externos, esto se puede comprobar cuando se ejecuta el comando para publicar o suscribir utilizando la dirección IP de la Raspberry Pi en lugar de 'localhost' (ver Figura 22 y Figura 23)

```
pi@raspberrypi:~ $ mosquitto_pub -h 192.168.0.16 -t "topicoPrueba" -m "Hola mundo"  
Error: Connection refused
```

Figura 22 Publicación rechazada en bróker MQTT

```
pi@raspberrypi:~ $ mosquitto_sub -h 192.168.0.16 -t "topicoPrueba"  
Error: Connection refused
```

Figura 23 Suscripción rechaza en bróker MQTT

El mensaje de error obtenido es causado por el bloqueo del puerto que utiliza Mosquitto, dicho puerto es el 1883, esto se comprueba mediante el siguiente comando (ver Figura 24)

```
pi@raspberrypi:~ $ sudo lsof -i TCP:1883  
COMMAND  PID  USER  FD  TYPE  DEVICE  SIZE/OFF  NODE  NAME  
mosquitto 2551 mosquitto  5u  IPv4  22135      0t0  TCP  localhost:1883 (LISTEN)  
mosquitto 2551 mosquitto  6u  IPv6  22136      0t0  TCP  localhost:1883 (LISTEN)
```

Figura 24 Puerto utilizado por Mosquitto

Para abrir el puerto utilizado por Mosquitto debemos modificar su archivo de configuración denominado mosquitto.conf. Hay que aclarar que para modificar el archivo debemos tener permisos de superusuario, una vez se cuenta con los permisos adecuados, se utiliza un editor de texto con el siguiente comando (ver Figura 25)

```
pi@raspberrypi:~ $ sudo nvim /etc/mosquitto/mosquitto.conf
```

Figura 25 Ruta archivo de configuración Mosquitto

El archivo de configuración debe verse de la siguiente manera (ver Figura 26)

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log
```

Figura 26 Archivo de configuración sin modificar

Una vez allí debemos agregar dos líneas (ver Figura 27)

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

allow_anonymous true
listener 1883
```

Figura 27 Archivo de configuración modificado

Las últimas dos líneas son las líneas agregadas las cuales hacen referencia a que se permiten conexiones anónimas y que el puerto 1883 empieza a escuchar. Posterior a este paso debemos reiniciar el servicio relacionado con Mosquitto para que los cambios se hagan efectivos con el comando mostrado a continuación (ver Figura 28)

```
pi@raspberrypi:~ $ sudo systemctl restart mosquitto.service
```

Figura 28 Comando para reiniciar Mosquitto

Una vez el servicio ha sido reiniciado debemos verificar el estado, para comprobar que los cambios no afectaron el servicio con el siguiente comando (ver Figura 29)

```
pi@raspberrypi:~ $ sudo systemctl status mosquitto.service
```

Figura 29 Comando para comprobar el estado del servicio

El resultado del comando anterior debe ser el siguiente (ver Figura 30)


```
● mosquitto.service - Mosquitto MQTT Broker
  Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
  Active: active (running) since Fri 2022-06-10 11:21:08 -05; 3h 50min ago
    Docs: man:mosquitto.conf(5)
          man:mosquitto(8)
  Process: 2566 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/S)
  Process: 2567 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, status=0/S)
  Process: 2568 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/S)
  Process: 2569 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/S)
 Main PID: 2570 (mosquitto)
   Tasks: 1 (limit: 1598)
    CPU: 5.421s
  CGroup: /system.slice/mosquitto.service
          └─2570 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

jun 10 11:21:08 raspberrypi systemd[1]: Starting Mosquitto MQTT Broker...
jun 10 11:21:08 raspberrypi systemd[1]: Started Mosquitto MQTT Broker.
```

Figura 30 Estado del servicio Mosquitto

Una vez se realizan estos pasos comprobamos que el puerto 1883 que utiliza Mosquitto esté abierto con el siguiente comando

```
pi@raspberrypi:~ $ sudo lsof -i TCP:1883
COMMAND  PID    USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
mosquitto 2570  mosquitto  5u  IPv4  23010      0t0  TCP *:1883 (LISTEN)
mosquitto 2570  mosquitto  6u  IPv6  23011      0t0  TCP *:1883 (LISTEN)
```

Figura 31 puerto usado por Mosquitto

Efectivamente se comprueba que el puerto está abierto y podemos comprobar que se puede utilizar el bróker MQTT desde servicios y dispositivos externos mediante los siguientes comandos

```
pi@raspberrypi:~ $ mosquitto_pub -h 192.168.101.5 -t "topicoPrueba" -m "Hola mundo"
```

```
pi@raspberrypi:~ $ mosquitto_sub -h 192.168.101.5 -t "topicoPrueba"
Hola mundo
```

Conexión Raspberry Pi con la Malla opción librería PainlessMeshBoost

Se debe descargar la librería PainlessMeshBoost desde el repositorio

<https://gitlab.com/painlessMesh/painlessmeshboost> en la Raspberry Pi como se ve en la

Figura 16

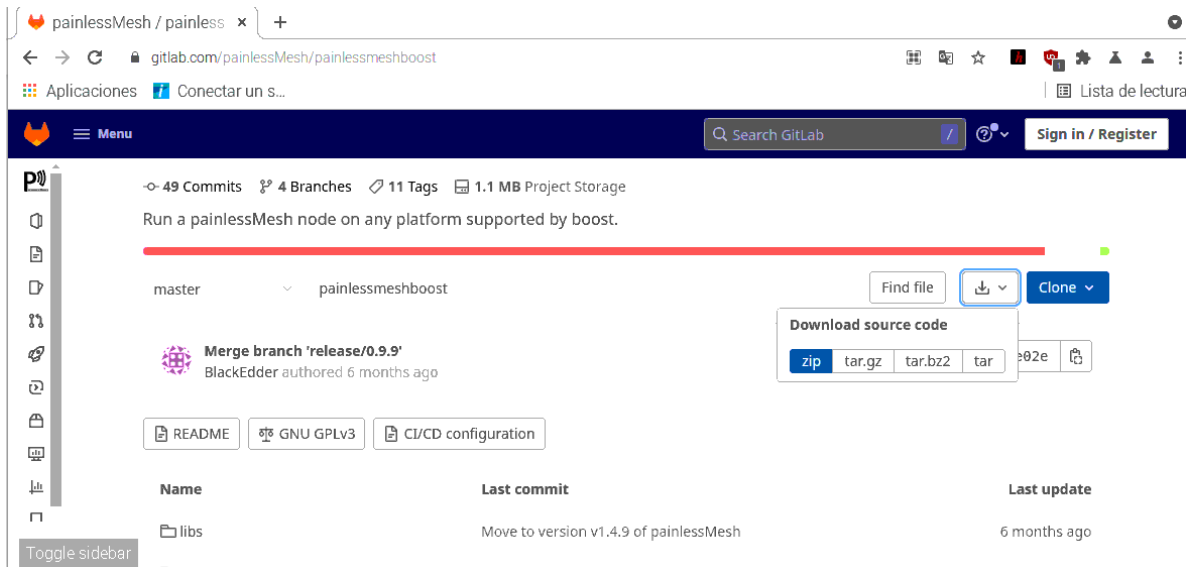


Figura 32 descarga librería PainlessMeshBoots

PainlessMeshBoots depende de una serie de librerías de Boost en sistemas operativos derivados de Debian las cuales pueden ser instaladas desde la ventana de comandos Figura 17 con la orden `sudo apt-get install libboost-system-dev libboost-thread-dev libboost-regex-dev libboost-program-options-dev libboost-chrono-dev libboost-date-time-dev libboost-atomic-dev libboost-filesystem-dev`

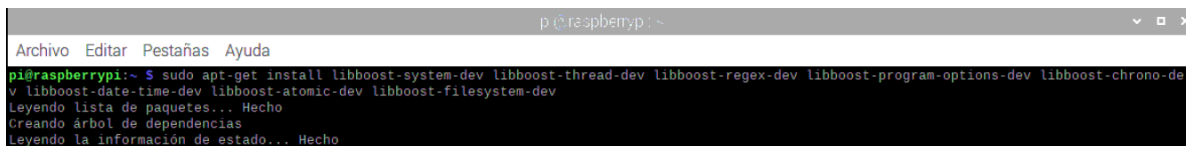


Figura 33 instalación librerías Boost desde ventana comandos Raspberry Pi

Con las librerías necesarias instaladas se proceda a la instalación de la librería PainlessMeshBoots en la Raspberry Pi accediendo desde la ventana de comandos a la carpeta raíz de la librería descargada Figura 18, donde se ejecuta los comandos para preparar el entorno de compilación Figura 19, crear el ejecutable de la librería e instalación Figura 20

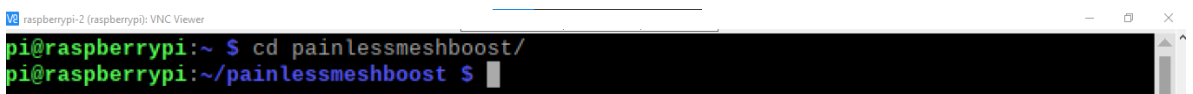


Figura 34 directorio raíz librería Painlessmesh

```

pi@raspberrypi:~/painlessmeshboost $ cmake .
-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Boost: /usr/lib/arm-linux-gnueabi/cmake/Boost-1.74.0/BoostConfig.cmake (found suitable version "1.74.0", minimum required is "1.42.0") found components: filesystem system thread regex program_options date_time
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/painlessmeshboost

```

Figura 35 detección del compilador y generación del entorno de compilación en el directorio actual

```

pi@raspberrypi:~ $ cd painlessmeshboost/
pi@raspberrypi:~/painlessmeshboost $ make
Scanning dependencies of target painlessMeshBoost
[ 25%] Building CXX object CMakeFiles/painlessMeshBoost.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/painlessMeshBoost.dir/src/fake_serial.cpp.o
[ 75%] Building CXX object CMakeFiles/painlessMeshBoost.dir/libs/painlessMesh/src/scheduler.cpp.o
[100%] Linking CXX executable bin/painlessMeshBoost
[100%] Built target painlessMeshBoost
pi@raspberrypi:~/painlessmeshboost $ sudo make install
[100%] Built target painlessMeshBoost
Install the project...
-- Install configuration: ""
-- Installing: /usr/local/bin/painlessMeshBoost
pi@raspberrypi:~/painlessmeshboost $

```

Figura 36 generación del ejecutable e instalación de la librería Painlessmesh en la Raspberry Pi

Con la librería PainlessMeshBoots instalada en la Raspberry Pi se procede a la configuración de uno de los nodos de la malla con el sketch “bridge” proporcionado en los ejemplos de la librería painlessmesh donde se debe configurar la misma estación de wifi que se encuentra conectada la Raspberry Pi Figura 21

```
bridge Arduino 1.8.19 (Windows Store 1.8.57.0)
Archivo Editar Programa Herramientas Ayuda
Subir
bridge $
// https://gitlab.com/painlessMesh/painlessMesh/wikis/bridge-between-mesh-and-another-network
//*****
#include "painlessMesh.h"

#define MESH_PREFIX "whateverYouLike"
#define MESH_PASSWORD "somethingSneaky"
#define MESH_PORT 5555

#define STATION_SSID "Agro-IoT"
#define STATION_PASSWORD "gPdKg9ig"
#define STATION_PORT 5555
uint8_t station_ip[4] = {192,168,0,102}; // IP of the se

// prototypes
void receivedCallback( uint32_t from, String smsg );

painlessMesh mesh;

void setup() {
  Serial.begin(115200);
  mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION ); // set before init() so that you can see startup messages

  // Channel set to 6. Make sure to use the same channel for your mesh and for you other
  // network (STATION_SSID)
  mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, WIFI_AP_STA, 6 );
  // Setup over the air update support
  mesh.initOTAReceive("bridge");
}
```

Credenciales wifi
IP correspondiente a la Raspberry Pi

Figura 37 configuración nodo que permite la conexión entre la malla y la Raspberry Pi

Con el código configurado se sube a la placa ESP32 Figura 25, donde por medio de la Herramienta Monitor serial del ide se puede observar su funcionamiento Figura 26

```
bridge Arduino 1.8.19 (Windows Store 1.8.57.0)
Archivo Editar Programa Herramientas Ayuda
Subir
bridge $
// https://gitlab.com/painlessMesh/painlessMesh/wikis/bridge-between-mesh-and-another-network
//*****
#include "painlessMesh.h"
```

Figura 38 Subir Código a la placa

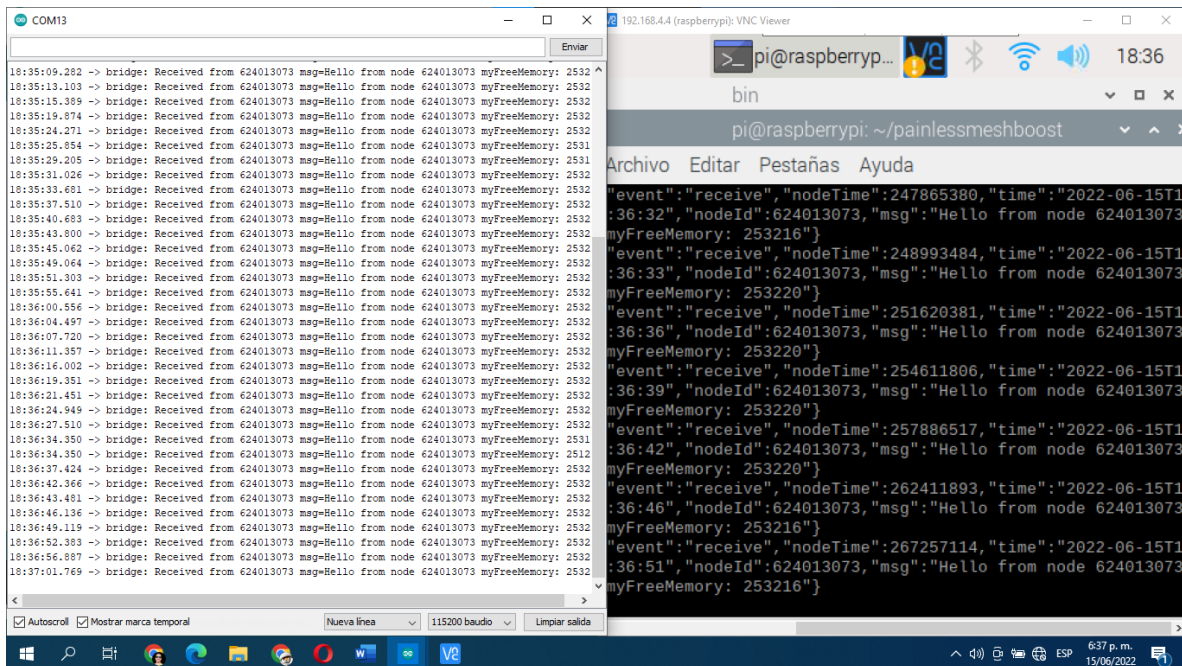
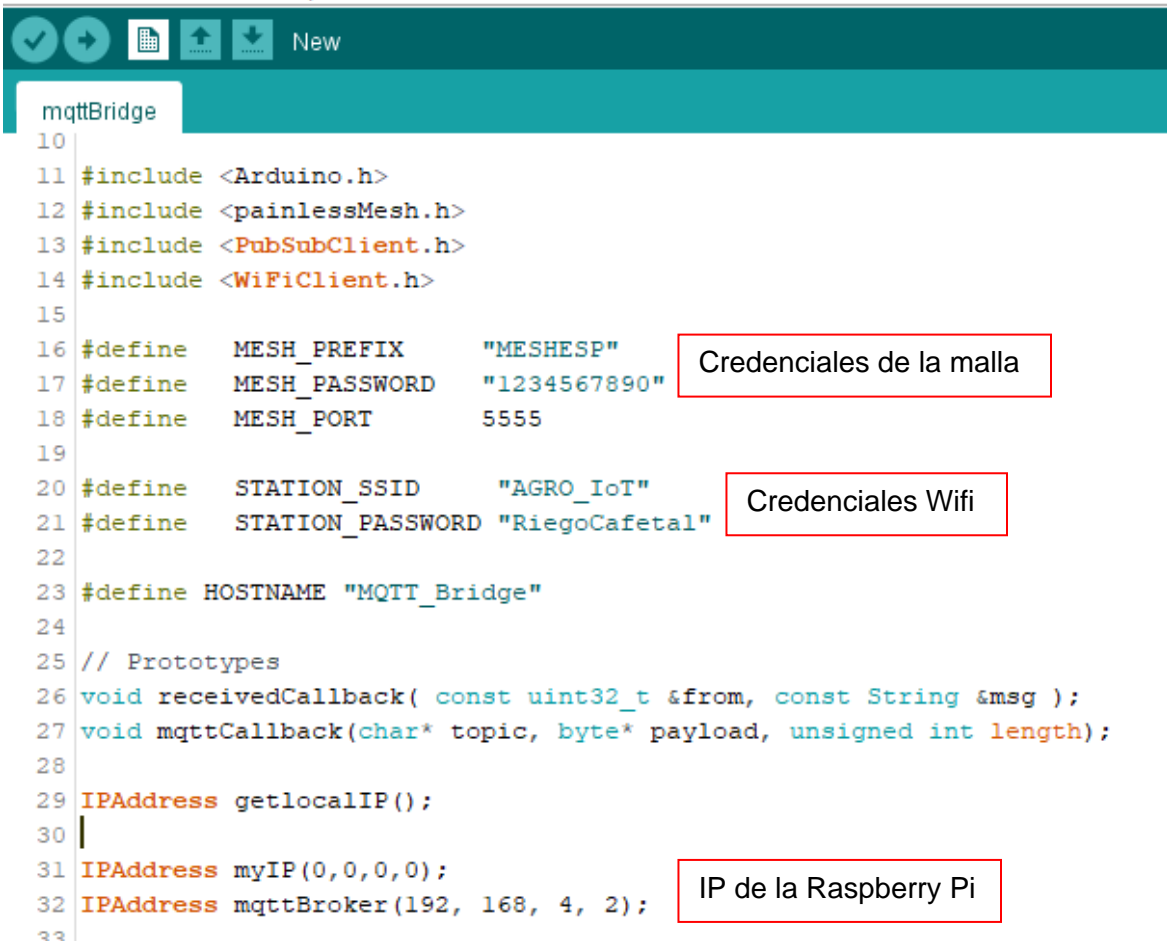


Figura 39 interacción malla con Raspberry Pi

Nota: las credenciales Wifi que son configuradas en el sketch “bridge” del nodo puente y la dirección IP del servidor pueden cambiar de acuerdo con el punto de acceso al que se conecten los dispositivos.

Conexión de Raspberry Pi con la malla opción MQTT

Posterior a la configuración del bróker MQTT dentro de la Raspberry Pi, se procede a realizar el envío y recepción de mensajes desde la malla a dicho bróker, esto se logra gracias al sketch de la librería PainlessMesh que tiene por nombre “mqttBridge”, el cual es modificado para que realice la conexión a la malla y a la misma red Wifi donde se conecta las Raspberry Pi como se puede ver en la Figura X



```
mqttBridge
10
11 #include <Arduino.h>
12 #include <painlessMesh.h>
13 #include <PubSubClient.h>
14 #include <WiFiClient.h>
15
16 #define MESH_PREFIX "MESHESP"
17 #define MESH_PASSWORD "1234567890"
18 #define MESH_PORT 5555
19
20 #define STATION_SSID "AGRO_IoT"
21 #define STATION_PASSWORD "RiegoCafetal"
22
23 #define HOSTNAME "MQTT_Bridge"
24
25 // Prototypes
26 void receivedCallback( const uint32_t &from, const String &msg );
27 void mqttCallback(char* topic, byte* payload, unsigned int length);
28
29 IPAddress getLocalIP();
30
31 IPAddress myIP(0,0,0,0);
32 IPAddress mqttBroker(192, 168, 4, 2);
33
```

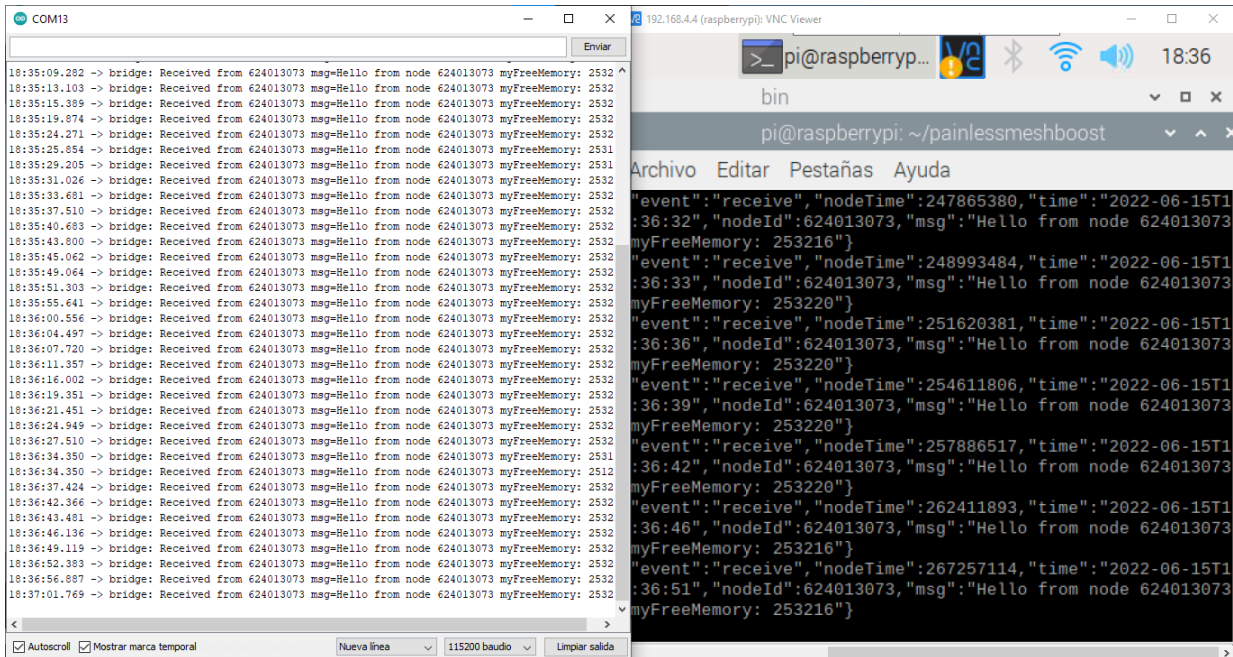
Se procede a subir el código a uno de los nodos que va a funcionar como puente entre la malla de sensores y la Raspberry Pi, para comprobar que está funcionando ingresamos al monitor serial proporcionado por el entorno Arduino

```
COM13
18:35:09.282 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:13.103 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:15.389 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:19.874 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:24.271 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:25.854 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2531
18:35:29.205 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2531
18:35:31.026 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:33.681 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:37.510 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:40.683 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:43.800 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:45.062 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:49.064 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:51.303 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:35:55.641 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:00.556 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:04.497 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:07.720 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:11.357 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:16.002 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:19.351 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:21.451 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:24.949 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:27.510 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:34.350 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2531
18:36:34.350 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2512
18:36:37.424 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:42.366 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:43.481 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:46.136 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:49.119 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:52.383 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:36:56.887 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
18:37:01.769 -> bridge: Received from 624013073 msg=Hello from node 624013073 myFreeMemory: 2532
```

Como se puede observar el puente está recibiendo mensajes por parte de la malla donde se identifica el id del nodo y su mensaje, ahora se procede a comprobar que se estén publicando estos mensajes en el bróker MQTT con el tópic "ainlessMesh/from/#", utilizando el comando para suscribirse a dicho tópic

```
pi@raspberrypi:~ $ mosquitto_sub -h 192.168.0.14 -t "ainlessMesh/from/#"
```

Al ejecutar el comando debemos obtener en la terminal de la Raspberry Pi los mensajes que el puente MQTT muestra en el monitor serial, como se observa a continuación



En este paso aseguramos que la malla y la Raspberry Pi están integradas.

Guardar datos provenientes de la malla en una base de datos en la Raspberry Pi

Una vez se reciban los mensajes de la malla en la Raspberry Pi, la configuración de los nodos de la malla debe cambiar, más específicamente en la construcción de los mensajes que se envían, esta modificación se realiza dentro del código llamado 'StartHere'. La construcción de los mensajes se realiza en la función 'makeMessage' la cual recibe como único parámetro el id del nodo, además de este dato se añade dentro de la función un valor numérico aleatorio que simula los datos provenientes de un sensor de temperatura, con esto se crea una cadena de tipo String estructurada de manera que se asemeje a un objeto tipo JSON, que permita la transformación de dicha cadena a un objeto JSON dentro de la Raspberry Pi y en consecuencia realizar la inserción de los datos contenidos en el mensaje a la base de datos. La función makeMessage está especificada en la figura X.

```
String makeMessage(uint32_t nodeId) {
    String msg = "{\"nodeid\":\" + String(nodeId);
    msg += "\",\"temperature\":\" + String(random(150)) + "\"";
    return msg;
}

void sendMessage() {
    String msg = makeMessage(mesh.getNodeId());
    mesh.sendBroadcast(msg);

    if (calc_delay) {
        SimpleList<uint32_t>::iterator node = nodes.begin();
        while (node != nodes.end()) {
            mesh.startDelayMeas(*node);
            node++;
        }
        calc_delay = false;
    }
}
```

Una vez construido el mensaje que se va a enviar al puente MQTT, monitor serial de este debe recibir el siguiente mensaje de la malla, especificando el id del nodo que envía el mensaje y valor simulado de temperatura

```
10:29:41.731 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":19}
10:29:43.524 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":6}
10:29:47.380 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":138}
10:29:52.163 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":16}
10:29:54.973 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":66}
10:29:59.539 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":25}
10:30:03.533 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":64}
10:30:05.697 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":102}
10:30:08.471 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":121}
10:30:10.465 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":28}
10:30:13.557 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":117}
10:30:17.768 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":133}
10:30:22.343 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":99}
10:30:24.055 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":130}
10:30:27.382 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":141}
10:30:31.226 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":70}
10:30:34.772 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":91}
```

Autoscroll Show timestamp Newline 115200 baud Clear output

Al correr el comando para suscribirme el tópic (ver Figura X) se obtiene que el mensaje estructurado como JSON está llegando a la Raspberry Pi para su debido manejo


```
10:49:27.839 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":103}
{"nodeid":624433613,"temperature":99}
10:49:31.141 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":85}
{"nodeid":624433613,"temperature":94}
10:49:35.396 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":35}
{"nodeid":624433613,"temperature":109}
10:49:40.197 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":99}
{"nodeid":624433613,"temperature":86}
10:49:42.611 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":94}
{"nodeid":624433613,"temperature":65}
10:49:44.038 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":89}
{"nodeid":624433613,"temperature":28}
10:49:45.427 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":109}
{"nodeid":624433613,"temperature":89}
10:49:47.002 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":86}
{"nodeid":624433613,"temperature":64}
10:49:50.691 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":65}
{"nodeid":624433613,"temperature":89}
10:49:55.550 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":89}
{"nodeid":624433613,"temperature":28}
10:50:00.349 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":28}
{"nodeid":624433613,"temperature":122}
10:50:03.596 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":64}
{"nodeid":624433613,"temperature":87}
10:50:05.286 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":122}
{"nodeid":624433613,"temperature":81}
10:50:09.335 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":87}
{"nodeid":624433613,"temperature":81}
10:50:11.776 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":81}
{"nodeid":624433613,"temperature":40}
10:50:13.846 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":40}
{"nodeid":624433613,"temperature":113}
10:50:16.995 -> bridge: Received from 624433613 msg={"nodeid":624433613,"temperature":113}
```

Al tener el mensaje con los datos llegando a la Raspberry Pi se deben realizar un conjunto de configuraciones para lograr guardar esta información en un base de datos, los cuales son mostrados a continuación:

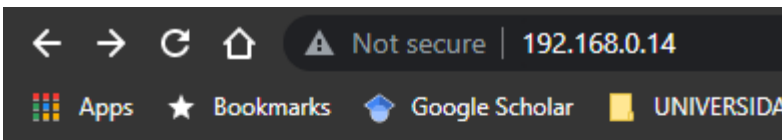
Para un manejo más eficiente del motor de base de datos es aconsejable una interfaz web para la administración de este, por ello debemos empezar instalando un servidor web, el recomendado es Apache y para instalarlo ejecutamos el siguiente comando

```
pi@raspberrypi:~ $ sudo apt-get install apache2 -y
```

Para comprobar que el servidor está activo, debemos primero saber la dirección IP de la Raspberry Pi e ingresar a la dirección IP en el navegador web


```
pi@raspberrypi:~ $ hostname -I
192.168.0.14
```

El resultado en el navegador web debe ser el siguiente



Index of /

[Name](#) [Last modified](#) [Size](#) [Description](#)

 [phpmyadmin/](#) 2020-10-15 13:08 -

Apache/2.4.53 (Raspbian) Server at 192.168.0.14 Port 80

Después de tener el servidor web funcionando se procede a instalar el servidor de bases de datos MySQL con el siguiente comando

```
pi@raspberrypi:~ $ sudo apt-get install mariadb-server php-mysql -y
```

Una vez instalado reiniciamos el servidor web Apache con el siguiente comando

```
pi@raspberrypi:~ $ sudo systemctl restart apache2.service
```

Posterior a la instalación del servidor de bases de datos debemos crear un usuario con su respectiva contraseña, además de la asignación de todos los permisos con los siguientes comandos

```
pi@raspberrypi:~ $ sudo mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 133
Server version: 10.5.15-MariaDB-0+deb11u1 Raspbian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create user admin@localhost identified by agroiot;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'agroiot' at line 1
MariaDB [(none)]> create user admin@localhost identified by 'agroiot';
Query OK, 0 rows affected (0.006 sec)

MariaDB [(none)]> grant all privileges on *.* to admin@localhost;
Query OK, 0 rows affected (0.007 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.003 sec)

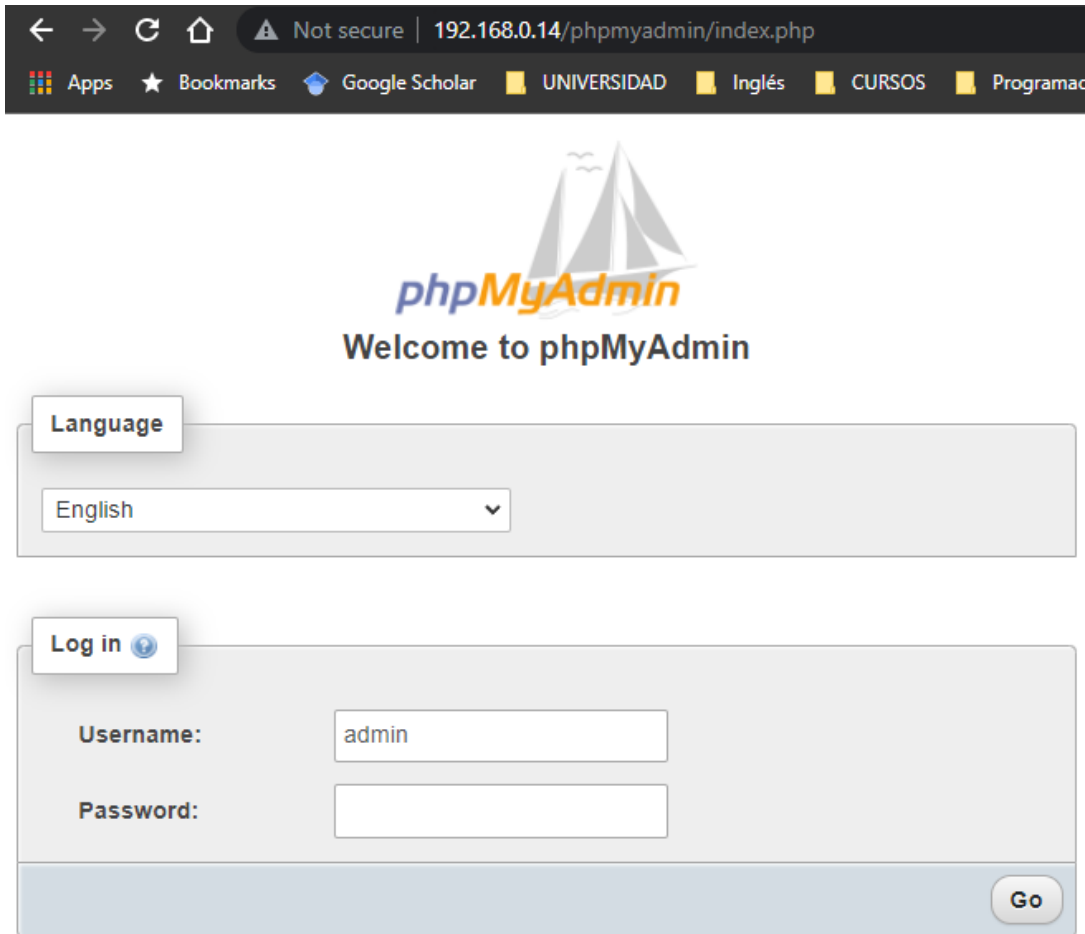
MariaDB [(none)]> exit;
Bye
```

Se puede notar que el usuario creado es 'admin' y la contraseña es 'agroiot'.

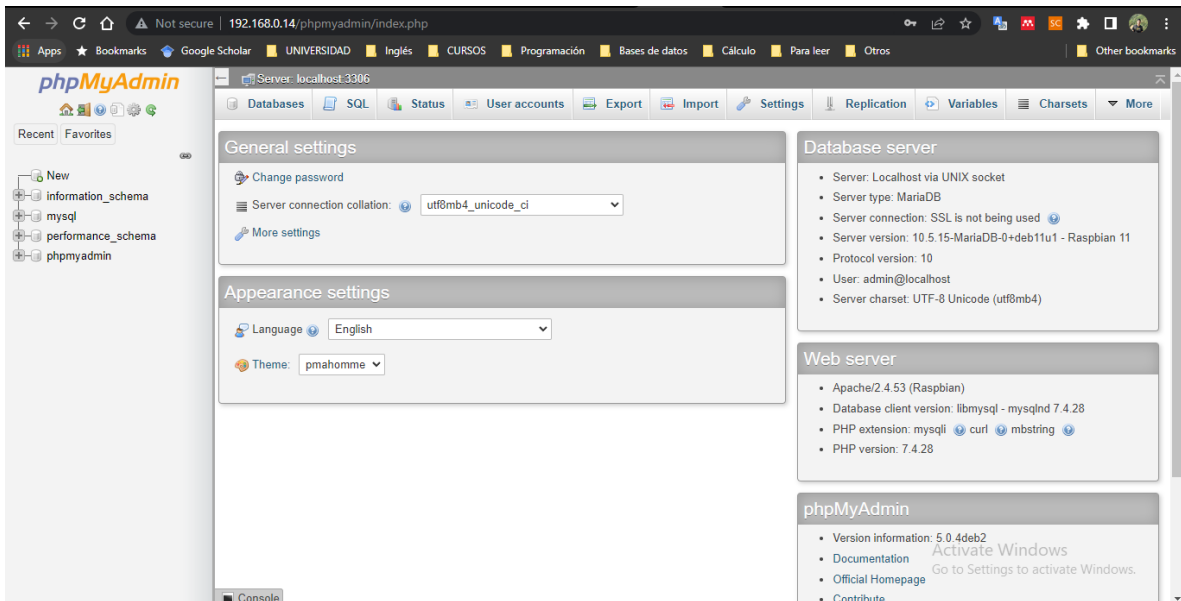
Por último, instalamos la aplicación web 'phpMyAdmin' para la administración del servidor de base de datos con el siguiente comando

```
pi@raspberrypi:~ $ sudo apt-get install phpmyadmin -y
```

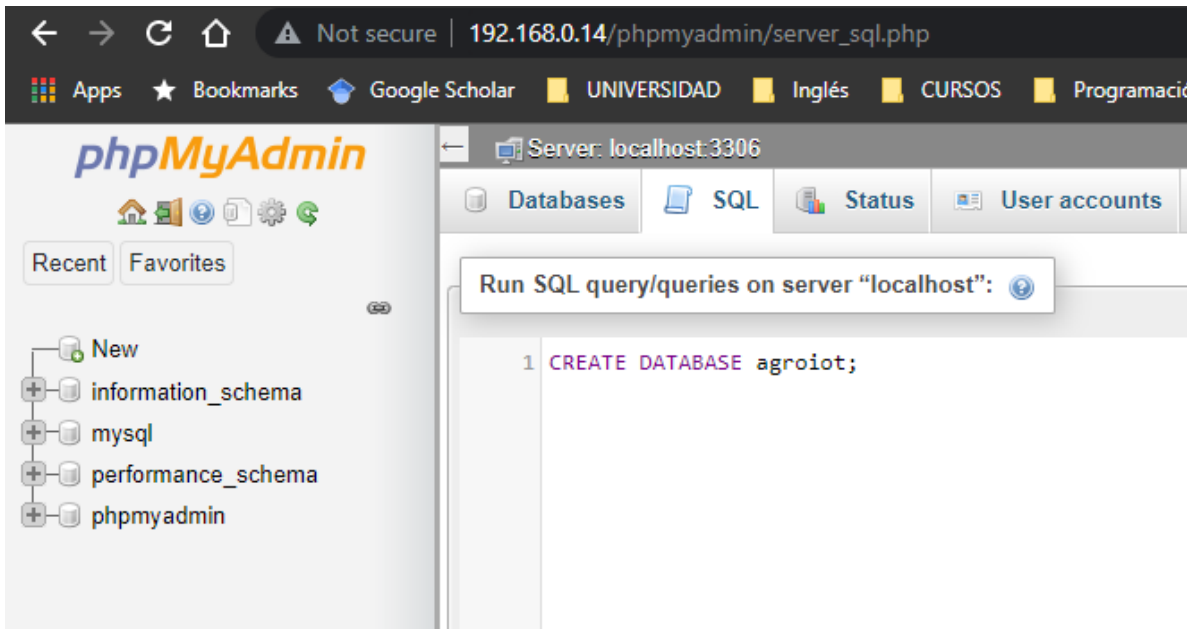
Una vez instalado comprobamos que está funcionando, ingresando a la dirección IP de la Raspberry Pi agregando */phpmyadmin*



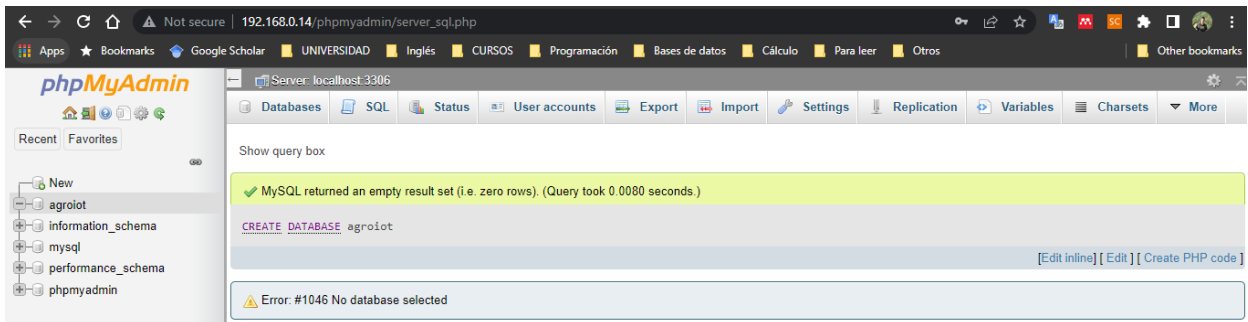
Una vez nos encontramos con esta interfaz debemos ingresar las credenciales definidas anteriormente para ingresar al administrador de bases de datos gráfico



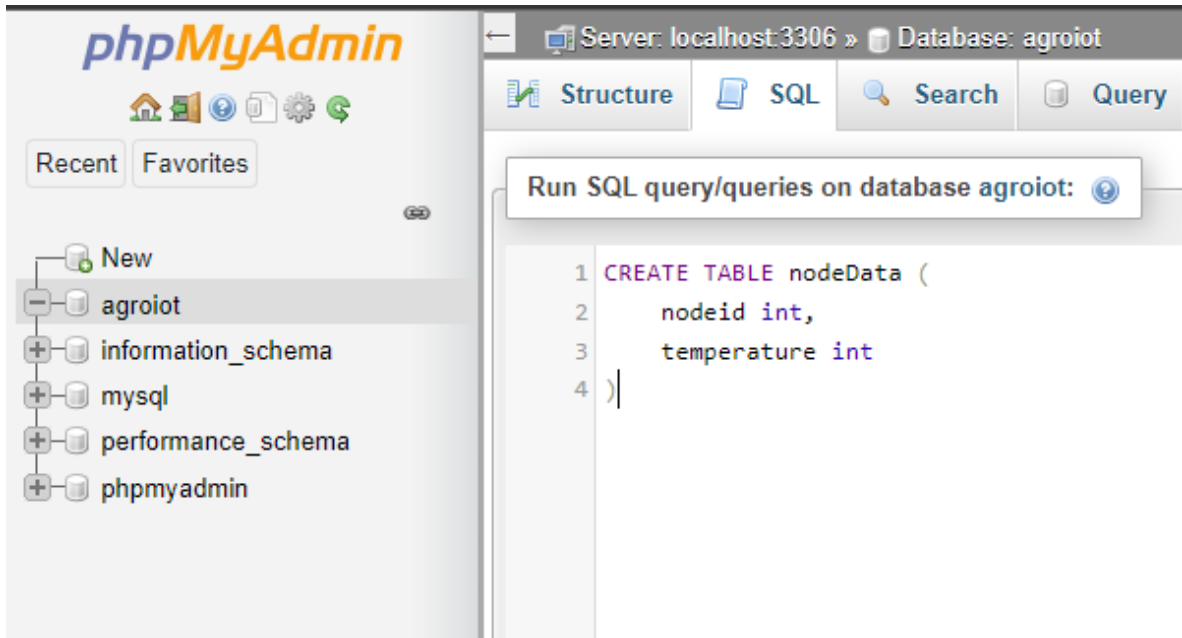
Estando en el administrador ingresamos a la pestaña denominada 'SQL' donde se ejecutará la consulta para crear la base datos donde se guardará la información de la malla.



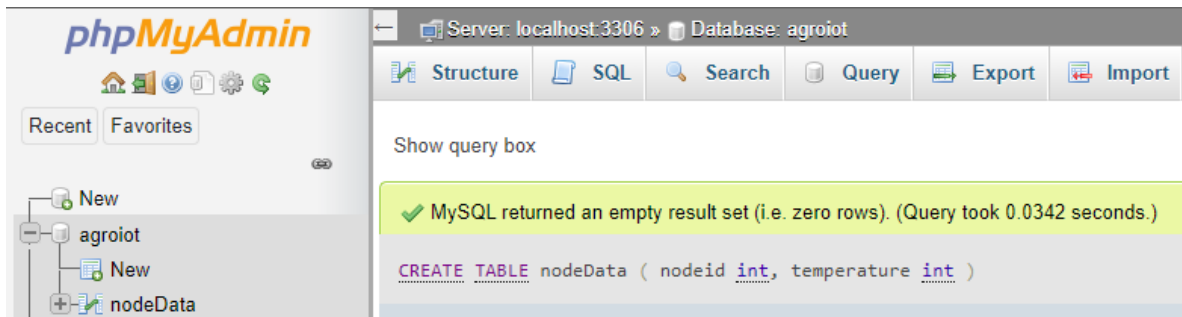
Al ejecutar la consulta debemos obtener el siguiente resultado



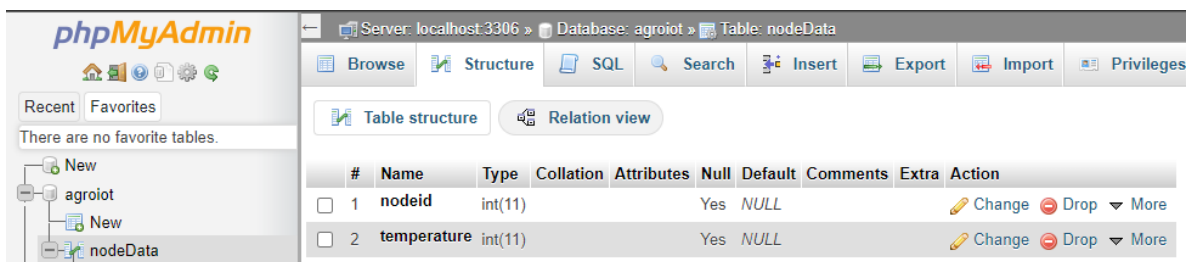
Se evidencia que en el costado izquierdo ha sido creada la base de datos con nombre 'agroiot', a la cual ingresamos haciendo click sobre ella, para posteriormente crear la tabla que almacenará la información de la malla con la siguiente consulta



El resultado posterior a la ejecución debe ser el siguiente



Comprobamos la estructura de la tabla presionando click sobre la misma



Al tener lista la base de datos y la tabla donde se va a guardar la información proveniente de la malla, el siguiente paso consiste en recibir el mensaje del bróker MQTT, transformarlo en un objeto JSON y por último realizar la inserción de la información contenida en el JSON a la base de datos, para realizarlo utilizamos el siguiente código escrito en Python v3.7

```

import paho.mqtt.client as mqtt
import json
import mysql.connector
from time import time

MQTT_HOST = '192.168.0.14'
MQTT_PORT = 1883
MQTT_CLIENT_ID = 'Python MQTT client'
TOPIC = 'painlessMesh/from/#'

mydb = mysql.connector.connect(
    host='localhost',
    database='agroiot',
    user='admin',
    password='agroiot'
)

def on_connect(mqtt_client, user_data, flags, conn_result):
    mqtt_client.subscribe(TOPIC)

def on_message(mqtt_client, user_data, message):
    payload = message.payload.decode('utf-8')
    print(payload)
    data = json.loads(payload)
    insertdb(data["nodeid"], data["temperature"])

def insertdb(nodeid, temperature):
    mycursor = mydb.cursor()

    sql = "INSERT INTO nodeData (nodeid, temperature) VALUES (%s, %s)"
    val = (nodeid, temperature)

    mycursor.execute(sql, val)

    mydb.commit()

def main():
    mqtt_client = mqtt.Client(MQTT_CLIENT_ID)

    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    mqtt_client.connect(MQTT_HOST, MQTT_PORT)
    mqtt_client.loop_forever()

main()

```

Al ejecutar el script obtenemos los siguientes resultados

nodeid	temperature
624433613	62
624433613	24
624433613	31
624433613	109
624433613	125
624433613	96
624433613	111
624433613	100
624433613	30
624433613	40
624433613	19

Configuración de módulo ‘LoRa/GPS HAT’ instalado en la Raspberry Pi para envío de mensajes mediante LoRa:

En esta etapa se realizaron 3 pruebas que nos permitieran la interacción entre dos Raspberry Pi con la expansión del módulo LoRa/GPS HAT y una prueba realizada mediante cableado de pines:

El uso del LoRa/GPS HAT superpuesto en la Raspberry pi se realizan dos pruebas las cuales consistieron:

Uso librería rpi-lora-tranceiver-master

Para el uso de esta librería se sigue una serie de pasos especificados en el [Manual](#) proporcionado por Dragino específicamente en el apartado del ejemplo 3, donde nos proporciona el repositorio para descargar la librería y los pasos necesarios para su instalación en la Raspberry Pi, este proceso se debe ejecutar en dos Raspberry Pi donde una será el emisor y otra el receptor cada una con su respectivo modulo LoRa/GPS HAT superpuesto figura RPI+HAT, después de instalar la librería se procede con la ejecución en cada una de las raspberry pi donde se debe especificar su función emisor o receptor por medio de los parámetros “sender” Figura 39 y “res” Figura 40 *respectivamente*

```

raspberrypi@raspberrypi:~/rpi-lora-tranceiver-master/dragino_lora_app $ ./dragino_lora_app sender
SX1276 detected, starting.
Send packets at SF7 on 915.000000 Mhz.

```

Figura 40 Ejecución ejemplo LoRa parámetro "sender"

```

raspberrypi@raspberrypi:~/rpi-lora-tranceiver-master/dragino_lora_app $ ./dragino_lora_app res
SX1276 detected, starting.
Listening at SF7 on 915.000000 Mhz.
.....

```

Figura 41 Ejecución ejemplo LoRa parámetro "res"

Uso librerías para módulos LoRa SX1276 y RFM96

Para la segunda prueba se buscó trabajar directamente sobre el módulo LoRa que se encuentra en el HAT, mediante la librería de Python dedicada al módulo específico, en nuestro caso se debió explorar las librerías para el módulo SX1276 y el módulo RFM96. Para el manejo de estas se debe mapear los pines correspondientes del módulo LoRa (ver Figura 41) que se conectan a la Raspberry Pi mediante los puertos o pines GPIO (ver Figura 42) correspondientes, seguido se ejecutaron los códigos ejemplos que proporciona la librería con resultados no muy favorables que serán descritos en la sección de resultados.

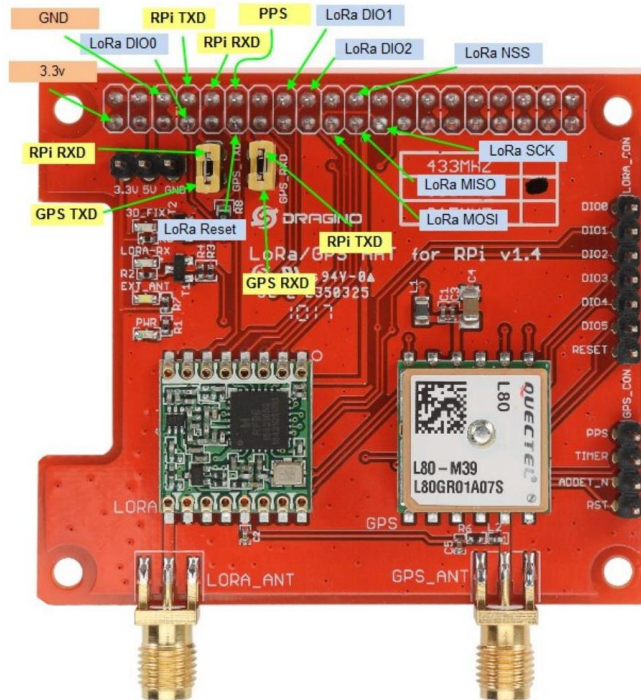


Figura 42 Pines LoRa/GPS HAT


```

raspberrypi@raspberrypi:~$ gpio readall
-----P1 3B-----
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 3.3v | | | 1 | 2 | | | 5v | | |
| 3 | 9 | SDA,1 | IN | 1 | 3 | 4 | | | 5v | | |
| 4 | 7 | SCL,1 | IN | 1 | 5 | 6 | | | 0v | | |
| 4 | 7 | GPIO,7 | IN | 0 | 7 | 8 | 1 | IN | TxD | 15 | 14 |
| | | 0v | | | 9 | 10 | 1 | IN | RxD | 16 | 15 |
| 17 | 0 | GPIO,0 | IN | 1 | 11 | 12 | 0 | IN | GPIO,1 | 1 | 18 |
| 27 | 2 | GPIO,2 | IN | 0 | 13 | 14 | | | 0v | | |
| 22 | 3 | GPIO,3 | IN | 0 | 15 | 16 | 0 | IN | GPIO,4 | 4 | 23 |
| | | 3.3v | | | 17 | 18 | 0 | IN | GPIO,5 | 5 | 24 |
| 10 | 12 | MOSI | ALT0 | 0 | 19 | 20 | | | 0v | | |
| 9 | 13 | MISO | ALT0 | 0 | 21 | 22 | 0 | IN | GPIO,6 | 6 | 25 |
| 11 | 14 | SCLK | ALT0 | 0 | 23 | 24 | 1 | OUT | CE0 | 10 | 8 |
| | | 0v | | | 25 | 26 | 1 | OUT | CE1 | 11 | 7 |
| 0 | 30 | SDA,0 | IN | 1 | 27 | 28 | 1 | IN | SCL,0 | 31 | 1 |
| 5 | 21 | GPIO,21 | IN | 1 | 29 | 30 | | | 0v | | |
| 6 | 22 | GPIO,22 | IN | 1 | 31 | 32 | 0 | IN | GPIO,26 | 26 | 12 |
| 13 | 23 | GPIO,23 | IN | 0 | 33 | 34 | | | 0v | | |
| 19 | 24 | GPIO,24 | IN | 0 | 35 | 36 | 0 | IN | GPIO,27 | 27 | 16 |
| 26 | 25 | GPIO,25 | IN | 0 | 37 | 38 | 0 | IN | GPIO,28 | 28 | 20 |
| | | 0v | | | 39 | 40 | 0 | IN | GPIO,29 | 29 | 21 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
-----P1 3B-----

```

Figura 43 Pines Raspberry Pi

Cableado

Para realizar el uso del módulo RFM95 instalado en el LoRa/GPS HAT se hace uso de mapeo de pines especificado en la documentación de la librería [pyLoraRFM9x](#) Figura 44 y sus diferentes pasos para su configuración en instalación.

RFM module pin	Raspberry Pi GPIO pin
MISO	MISO
MOSI	MOSI
NSS/CS	CE1
CLK	SCK
RESET	GPIO 25
DIO0	GPIO 5
3.3V	3.3V
GND	GND

Figura 44 correspondencia pines modulo RFM con pines GPIO RPi

Para la prueba realizada mediante el cableado de pines entre el LoRa/GPS HAT y la Raspberry Pi (ver Figura 43) se hizo necesario conocer los pines correspondientes del módulo LoRa del HAT (ver Figura 41) y los pines GPIO de la Raspberry Pi (ver Figura 42) que permitiera un acceso específico o directo a los pines del módulo en mención.

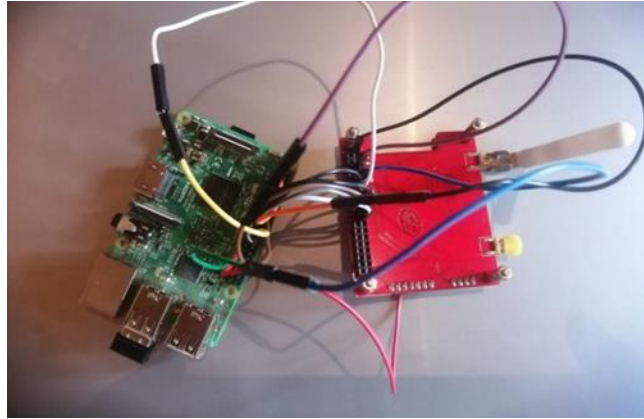


Figura 45 Cableado Pines LoRa HAT y Raspberry Pi

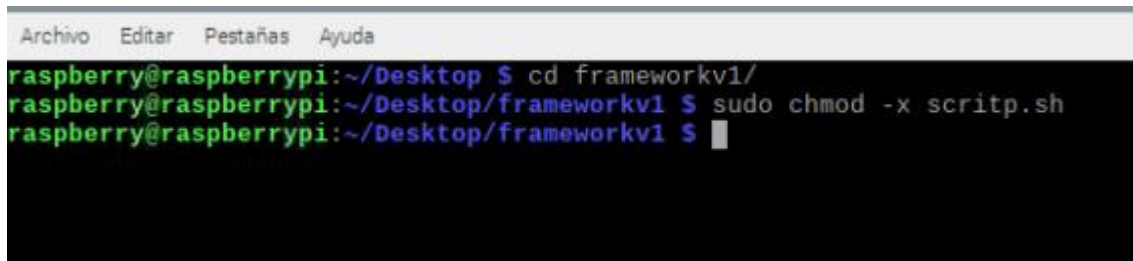
Para el uso de del modulo SX1276 de LoRa se hace necesario un mapeo de cableado como se especifica en la documentación de la librería [pyLoraRFM9x](#) Figura 45 y sus diferentes pasos para su configuración en instalación.

Ra-02 LoRa BOARD2	RaspPi GPIO
MOSI	GPIO10
MISO	GPIO 9
SCK (SCLK)	GPIO11
NSS	GPIO7 (CE1)
DIO0 (IRQ)	GPIO23
DIO1	GPIO24
DIO2	GPIO25
DIO3	GPIO5
RST (restablecer)	GPIO6
DIRIGIÓ	GPIO19

Figura 46 correspondencia pines modulo SX1276 con pines GPIO RPi

Configuración del entorno objeto semántico en la Raspberry Pi:

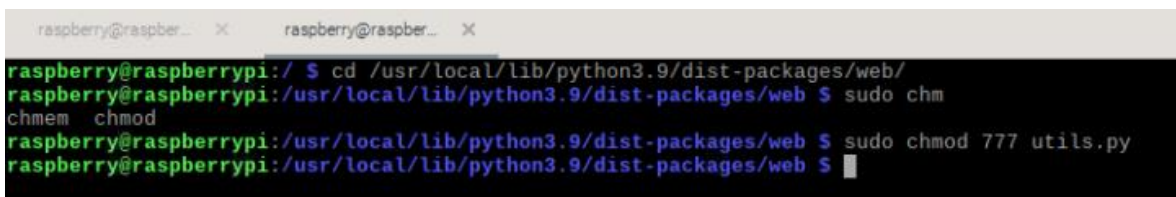
Inicialmente ejecutar el script.sh que se encuentra en el directorio raíz del framework el cual permite la descarga de librerías y dependencias al cual se le deben dar permisos de ejecución Figura 46



```
Archivo  Editar  Pestañas  Ayuda
raspberrypi@raspberrypi:~/Desktop $ cd frameworkv1/
raspberrypi@raspberrypi:~/Desktop/frameworkv1 $ sudo chmod -x scrip.sh
raspberrypi@raspberrypi:~/Desktop/frameworkv1 $
```

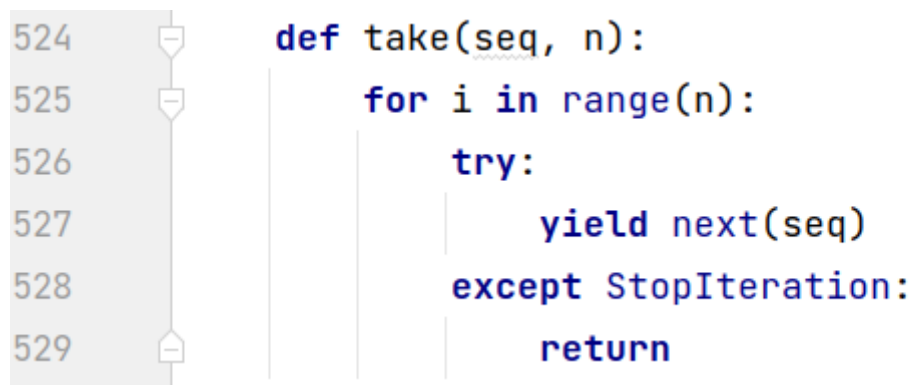
Figura 47 ejecución script

Ejecución del archivo web.py donde al ejecutarlo presenta un error el cual se debe solucionar modificando el archivo utils.py el cual se encuentra en la ruta /usr/local/lib/python3.7/dist-packages/web/utils.py al cual se le deben proporcionar todos los permisos Figura 47 para modificarlo como lo muestra la Figura 48



```
raspberrypi@raspberrypi: / $ cd /usr/local/lib/python3.9/dist-packages/web/
raspberrypi@raspberrypi: /usr/local/lib/python3.9/dist-packages/web $ sudo chm
chmem  chmod
raspberrypi@raspberrypi: /usr/local/lib/python3.9/dist-packages/web $ sudo chmod 777 utils.py
raspberrypi@raspberrypi: /usr/local/lib/python3.9/dist-packages/web $
```

Figura 48 permisos archivo utils.py



```
524
525
526
527
528
529
def take(seq, n):
    for i in range(n):
        try:
            yield next(seq)
        except StopIteration:
            return
```

Figura 49 Líneas de código para agregar

Mapeo objeto semántico denominado regulador de riego mediante objeto JSON en el archivo Riego.py donde se tiene la transmisión de datos mediante el protocolo MQTT de la malla a la Raspberry Pi como un sensor, la transmisión de datos de la Raspberry Pi a la malla mediante el protocolo MQTT como actuador, un monitoreo mediante protocolo MQTT de la malla como sensor y un pronóstico de las variables agroclimáticas mapeado como actuador.

Con el objeto semántico mapeado se procede a la creación de los ejecutables donde se debe cargar el objeto ejecutando el módulo Main.py Figura 49 y posteriormente ejecutar el módulo Riego.py Figura 50

```
raspberrypi@raspberrypi:~/Desktop/frameworkv1 $ python3 Main.py
```

Figura 50 carga del objeto

```
raspberrypi@raspberrypi:~/Desktop/frameworkv1 $ cd JSON\ Objetos/
raspberrypi@raspberrypi:~/Desktop/frameworkv1/JSON Objetos $ python3 Riego.py
{"Conceptos": [{"coffee plantation"}, {"lugares": null, "feed": {"id": "197346", "title": "Regulador de Riego", "Private": false, "tags": ["Entidad Cafetal", "E
ntity coffee plantation", "Funcionalidad de regulacion de riego", "Coffe Plantation", "Irrigation Regulation Functionality", "Irrigation Regulator"}, "descrip
tion": "Es un servicio que permite monitorear los niveles de humedad, temperatura y radiaci\u00f3n solar para iniciar el riego en el cultivo.", "feed": "https
://api.xively.com/v2/feeds/708637323.json", "auto_feed_url": null, "status": 0, "updated": "12/03/2020 20:38:32", "created": "07/07/2015 22:03:46", "creator":
"https://personal.xively.com/users/manzamb", "version": null, "website": null, "datastreams": [{"datastream_format": "string", "feedid": null, "id": "MQTTx
", "current_value": null, "at": "12/03/2020 20:38:32", "max_value": "0", "min_value": "0", "tags": ["Actuador", "Actuator", "MQTTx", "Caracteristica tx mqtt",
"Entidad Cafetal", "Entity Coffee Plantation", "Feature tx mqtt", "Funcionalidad de medida de encendido o apagado", "MQTTx", "On Off Functionality"], "unit":
{"symbol": "string", "label": "string", "unitType": 0}, "datapoints": null}, {"datastream_format": "string", "feedid": null, "id": "MQTTx", "current_value":
null, "at": "12/03/ 20:38:32", "max_value": "0.0", "min_value": "0.0", "tags": ["Caracteristica rx mqtt", "Entidad cafetal", "Entity Coffee Plantation", "Fe
ature rx mqtt", "Funcionalidad de notificaci\u00f3n de datos de la maya", "Sensor", "sensor de datos de la maya", "mesh data Notification Functionality", "mes
h data Sensor", "sensor de datos"], "unit": {"symbol": "string", "label": "string", "unitType": 0}, "datapoints": null}, {"datastream_format": "string", "feed
id": null, "id": "mqttmonitoreo", "current_value": null, "at": "12/03/2020 20:38:32", "max_value": "0", "min_value": "0", "tags": ["Caracteristica mqtt monito
reo", "Entidad Cafetal", "Entity Coffee Plantation", "Feature mqtt Monitoring", "Funcionalidad de monitorear la malla de sensores", "Sensor", "sensor de estado
de la malla", "Sensor mesh monitoring Functionality", "Mesh Condition Sensor", "sensor de estado"], "unit": {"symbol": "string", "label": "string", "unitType
": 0}, "datapoints": null}, {"datastream_format": "string", "feedid": null, "id": "prediccion", "current_value": null, "at": "12/03/2020 20:38:32", "max_value
": "0", "min_value": "0", "tags": ["Actuador", "Actuator", "Caracteristica Prediccion", "Entidad Cafetal", "Entity Coffe Plantation", "Fan", "Feature predicti
on", "Funcinalidad predecir datos por falla en la malla", "On Off Functionality", "Prediccion", "Prediction", "fan"], "unit": {"symbol": "string", "label": "s
tring", "unitType": 0}, "datapoints": null}, {"location": {"name": null, "domain": 0, "lat": "2.4471309", "lon": "-76.5981505", "ele": "4", "exposure": 0, "di
```

Figura 51 crear ejecutables

Ejecutados los módulos anteriores nos deben aparecer los siguientes archivos Figura 51

```
↑ /home/raspberrypi/Desktop/frameworkv1/Escenario/ArchivosObjeto/Ejecutables
Nombre
┆ get_mqttmonitoreo.py
┆ get_MQTTx.py
┆ get_MQTTx.py
┆ get_prediccion.py
┆ off_mqttmonitoreo.py
┆ off_MQTTx.py
┆ off_MQTTx.py
┆ off_prediccion.py
┆ on_mqttmonitoreo.py
┆ on_MQTTx.py
┆ on_MQTTx.py
┆ on_prediccion.py
┆ set_clean.py
┆ set_mqttmonitoreo.py
┆ set_MQTTx.py
┆ set_MQTTx.py
┆ set_prediccion.py
```

Figura 52 ejecutables creados a partir de JSON objeto

Instalación de sensores a los nodos de la malla:

Para este proceso se hizo necesario el uso de una mini-protoboard Figura 52 la cual permitiera realizar la conexión de los pines de los sensores con los puertos de la ESP32



Figura 53 mini-protoboard

Identificación de los pines análogos, de alimentación y GND de los sensores de temperatura (LM35) Figura 52, sensor de humedad del suelo (YL69) Figura 53 y sensor de luz solar (fotorresistencia LDR) Figura 54.

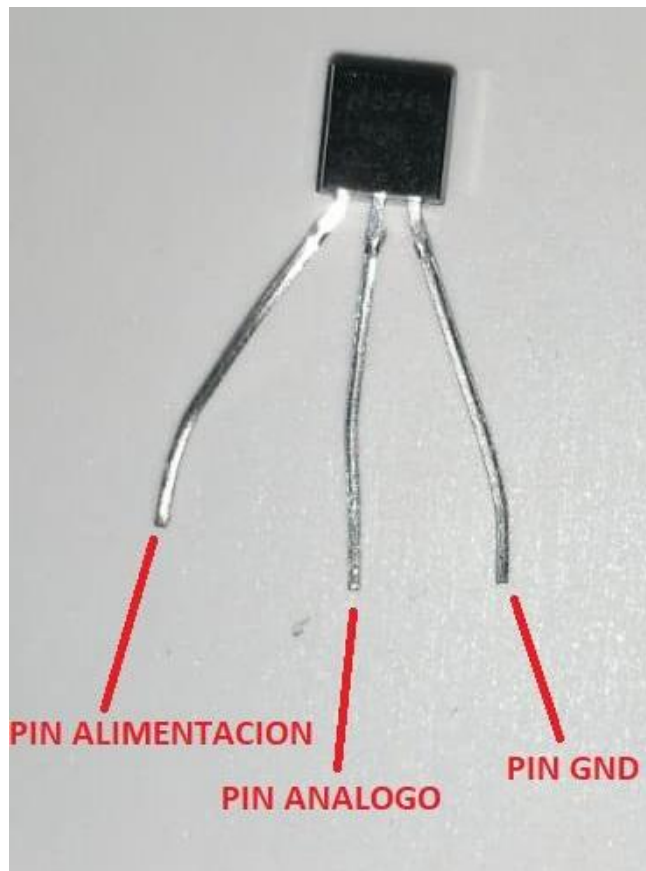


Figura 54 LM35

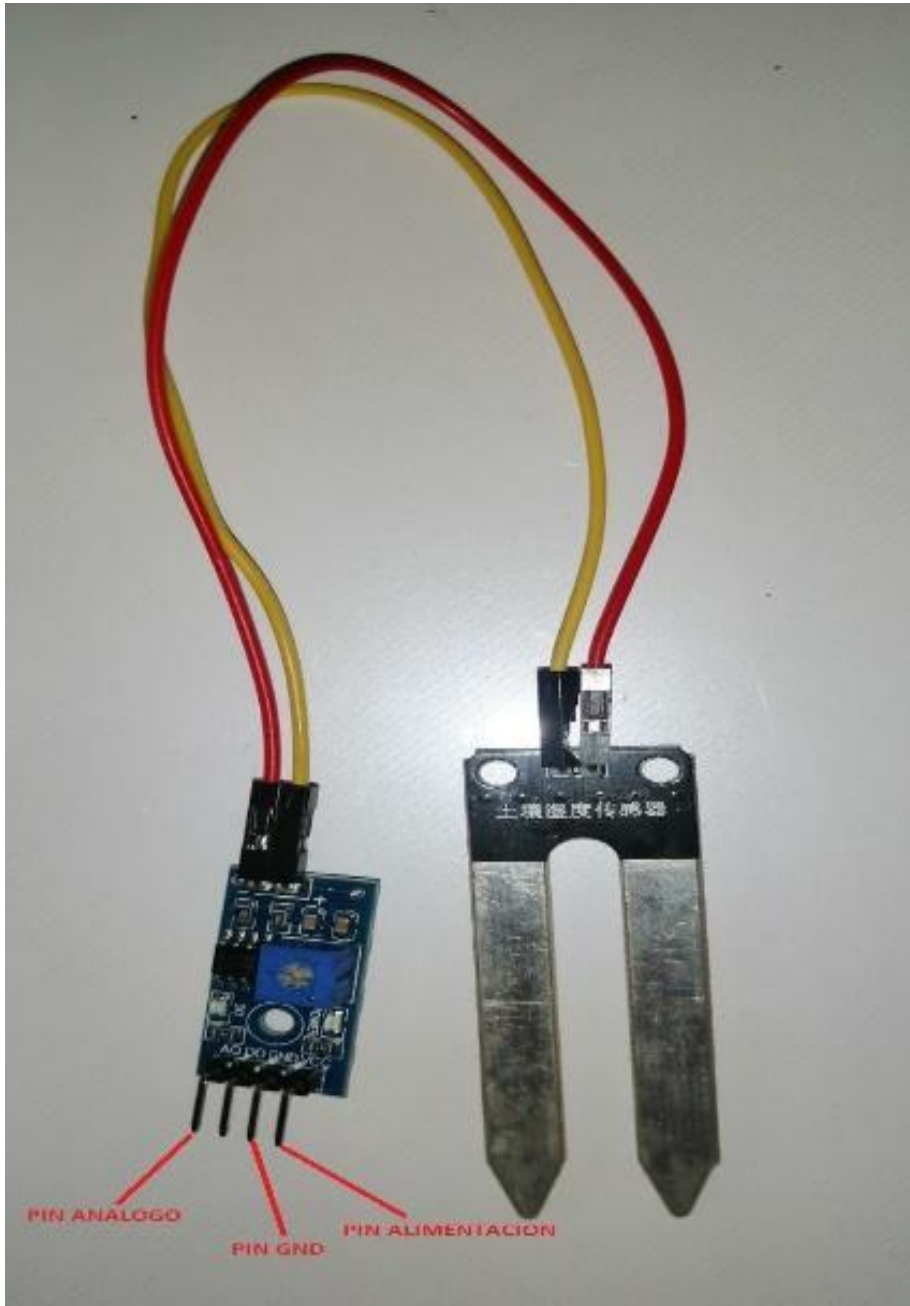


Figura 55 YL69

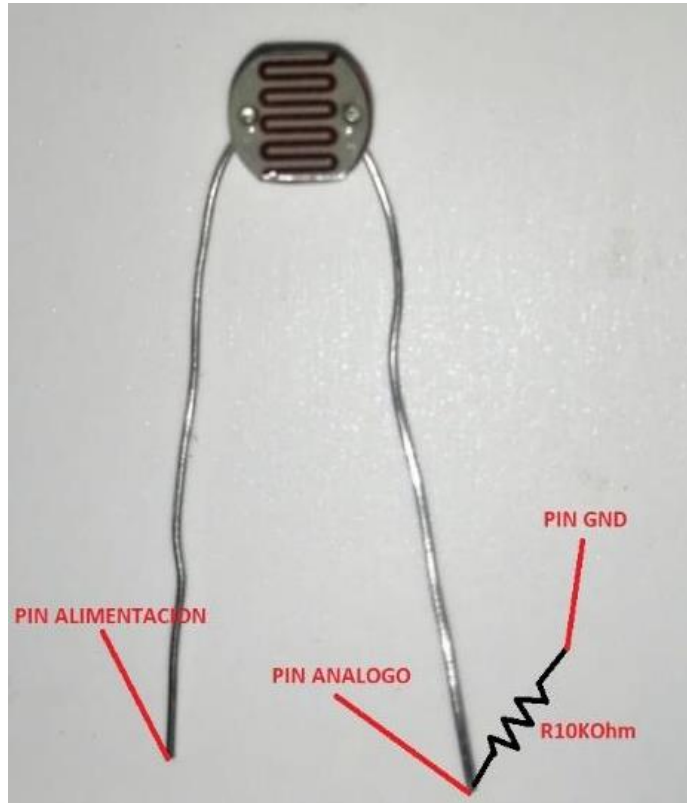


Figura 56 Fotorresistencia LDR

Pines usados de la ESP32 Figura 55, los pines analógicos habilitados para recibir datos, corresponde a los pines ADC1 habilitados para recibir señales analógicas, los pines ADC2 no podrán ser usados mientras el módulo wifi de la placa esté en funcionamiento, el pin GND es usado para las conexiones a tierra requerida por los nodos y el pin Vcc es usado para la alimentación de estos

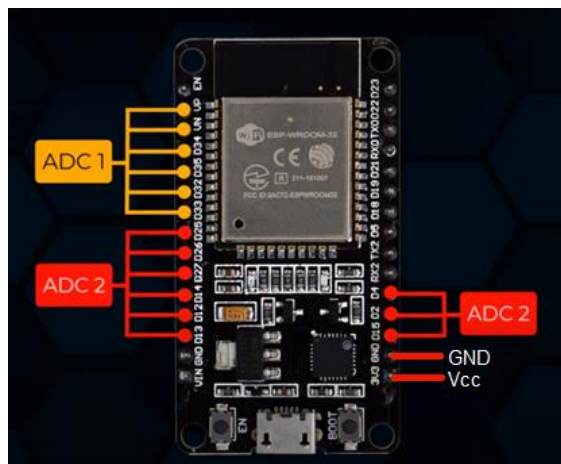


Figura 57 pines usados ESP32

En la Tabla 1 se presenta la correspondencia de los pines análogos ESP32 con los pines análogo de cada sensor

Tabla 1 correspondencia de pines

Pin análogo sensor	Pin análogo ESP32
LM35	D32
YL69	D34
Fotorresistencia LDR	D35

Para el caso de los pines GND y Vcc se realiza un cableado hacia la mini-protoboard para la conexión a tierra y alimentación de los sensores respectivamente y obtener el armado del nodo con todos sus sensores

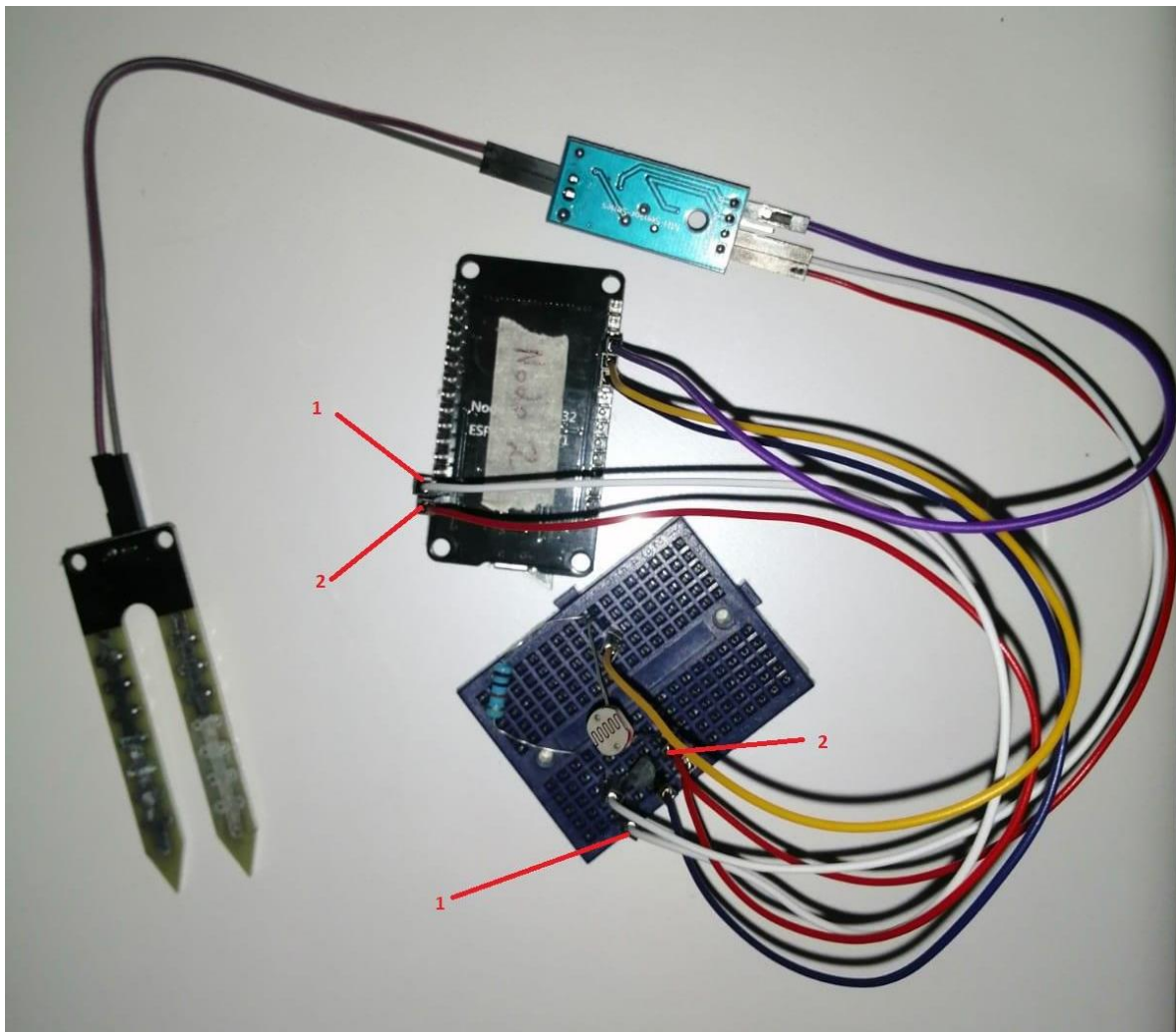


Figura 58 conexión sensores con ESP32 1 pin GND, 2 pin Vcc

Nota: para conexión de la fotorresistencia con el pin GND se hace necesario el uso de una resistencia de 10 K Ohm