

**ESCUELA POLITÉCNICA  
SUPERIOR DE CÓRDOBA**  
Universidad de Córdoba



## Internet de las Cosas

*Máster Universitario en Inteligencia Computacional e*

*Internet de las Cosas*

### **Práctica 4. MQTT Básico**

Carlos Checa Moreno



UNIVERSIDAD DE CÓRDOBA

# ÍNDICE GENERAL

---

<b>MQTT BÁSICO.....</b>	<b>3</b>
-------------------------	----------

4A – CONTROL MEDIANTE MQTT.....	12
---------------------------------	----

# ÍNDICE DE FIGURAS

---

<b>FIGURA 1:</b> DISPOSITIVOS COLOCADOS .....	4
<b>FIGURA 2:</b> MQTT BROKER ON.....	5
<b>FIGURA 3:</b> IP DEL BROKER .....	5
<b>FIGURA 4:</b> CONEXIÓN A MQTT SERVER.....	6
<b>FIGURA 5:</b> CÓDIGO MODIFICADO DE CLIENT1 .....	8
<b>FIGURA 6:</b> MAIN MODIFICADO DE CLIENT1 .....	9
<b>FIGURA 7:</b> SBC1 PUBLICANDO TEMPERATURA EN SENSOR/TEMP1.....	10
<b>FIGURA 8:</b> SBC2 SUSCRITO A /SENSOR/TEMP1 .....	11
<b>FIGURA 9:</b> MESSAGES RECIBIDOS EN EL TOPIC /SENSOR/TEMP1 .....	12
<b>FIGURA 10:</b> SIMULACIÓN T < 20 -> CERRAR VENTANA .....	13
<b>FIGURA 11:</b> CLIENTES Y 1 TOPIC DEL BROKER .....	14
<b>FIGURA 12:</b> CLIENTES Y 2 TOPICS DEL BROKER .....	15
<b>FIGURA 13:</b> SIMULACIÓN. LED ENCENDIDO POR VENTANA CERRADA .....	17

## Práctica 4

# MQTT BÁSICO

---

Se puede integrar sistemas con el paradigma Publicador-Suscriptor utilizando el protocolo MQTT en Cisco Packet Tracer.

Creamos un esquemático nuevo en blanco. **File → New (Ctrl + N)**

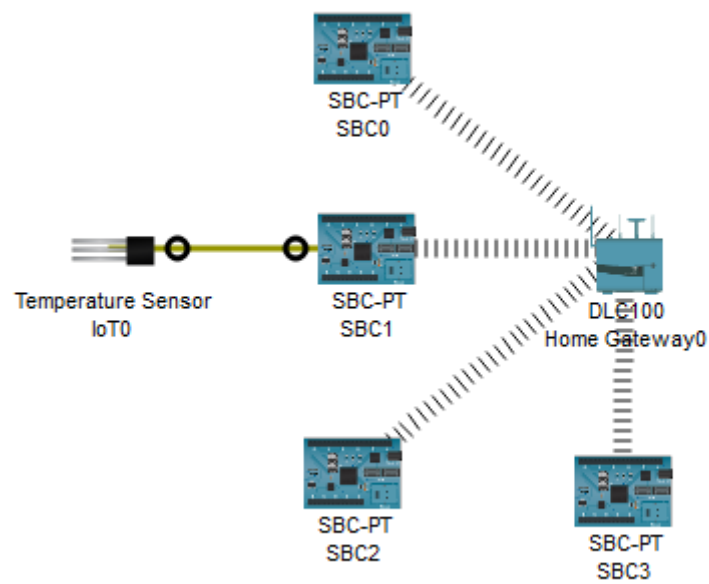
1) Incorporar los siguientes dispositivos y elementos:

- SBC: **Components → Boards → SBC Board**
- Home Gateway: **Network Devices → Wireless Devices → Home Gateway**
- Temperatura: **Components -> Sensors -> Temperature Sensor**

2) Interconectar el sensor de Temperatura con el SBC1 **Connections -> IoT Custom Cable**

de la siguiente manera:

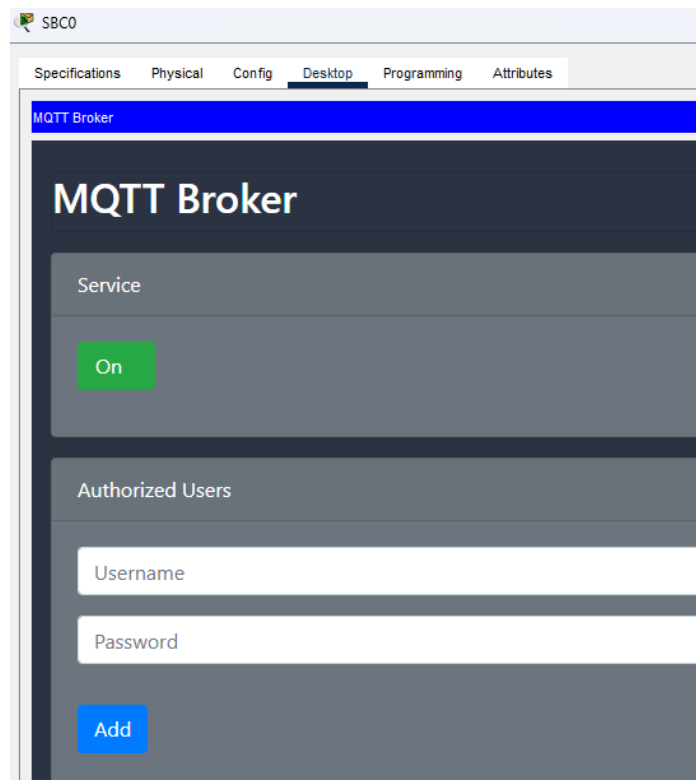
- Temperatura Pin A0 → SBC Pin D0



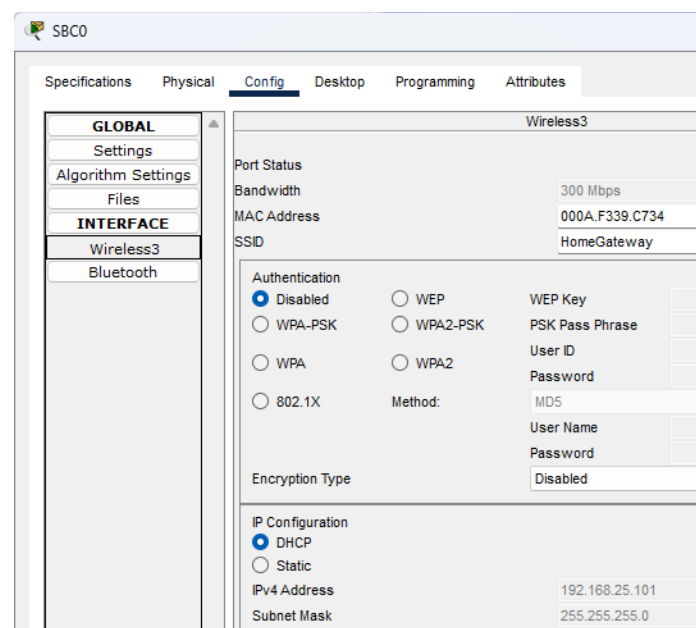
**Figura 1:** Dispositivos colocados

3) Diseñar el código de programación del SBC (en Python):

- Abrir el cuadro de diálogo del SBC0 → Programming.
  - Pulsar el botón New.
  - Introducir un nombre de proyecto: Broker0
  - Seleccionar Global Script Project: MQTT Broker – (Python)
  - Pulsar Install to Desktop
  - Pulsar Run
  - Entrar en Desktop → MQTT Broker y activar el Service a On.
  - Anotar la IP del Broker (mirar en la pestaña Config → Wireless3 → IP Configuration → IPv4 Address).



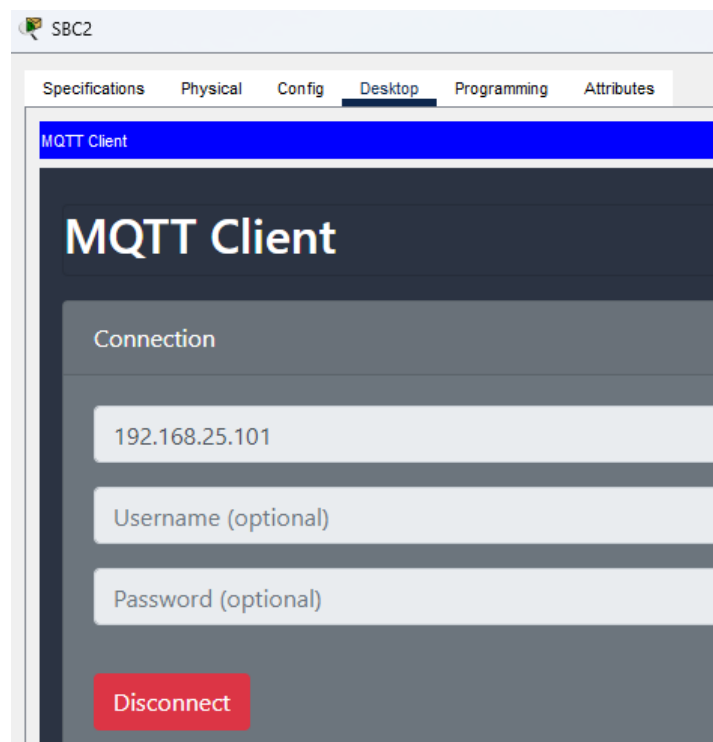
**Figura 2: MQTT Broker ON**



**Figura 3: IP del Broker**

- Abrir el cuadro de diálogo del SBC2 → Programming.
  - Pulsar el botón New.
  - Introducir un nombre de proyecto: Client2

- Seleccionar Global Script Project: MQTT Client – (Python)
- Pulsar Install to Desktop
- Pulsar Run
- Abrir la pestaña Desktop → MQTT Client
  - En Connection → Broker Address, poner la IP del Broker.
  - Pulsar el botón Connect.



**Figura 4:** Conexión a MQTT Server

- Abrir el cuadro de diálogo del SBC3 → Programming.
  - Pulsar el botón New.
  - Introducir un nombre de proyecto: Client3
  - Seleccionar Global Script Project: MQTT Client – (Python)
  - Pulsar Install to Desktop
  - Pulsar Run
  - Abrir la pestaña Desktop → MQTT Client
    - En Connection → Broker Address, poner la IP del Broker.

- Pulsar el botón Connect.

- Abrir el cuadro de diálogo del SBC1 → Programming.
  - Pulsar el botón New.
  - Introducir un nombre de proyecto: Client1
  - Seleccionar Global Script Project: MQTT Client – (Python)
  - Incluir el siguiente código:

```
from gpio import *
```

```
def publishTemperature():
```

```
    value = ( analogRead(0) * (200.0 / 1023.0) ) - 100.0
```

```
    topic = "/sensor/temp1"
```

```
    payload = str("%.2f C"% (value,))
```

```
    qos = "0"
```

```
    mqttclient.init()
```

```
    mqttclient.publish(topic, payload, qos)
```

```
    CLI.exit()
```

```
def autoConnect():
```

```
    brokerIP = ""
```

```
    user = ""
```

```
    password = ""
```

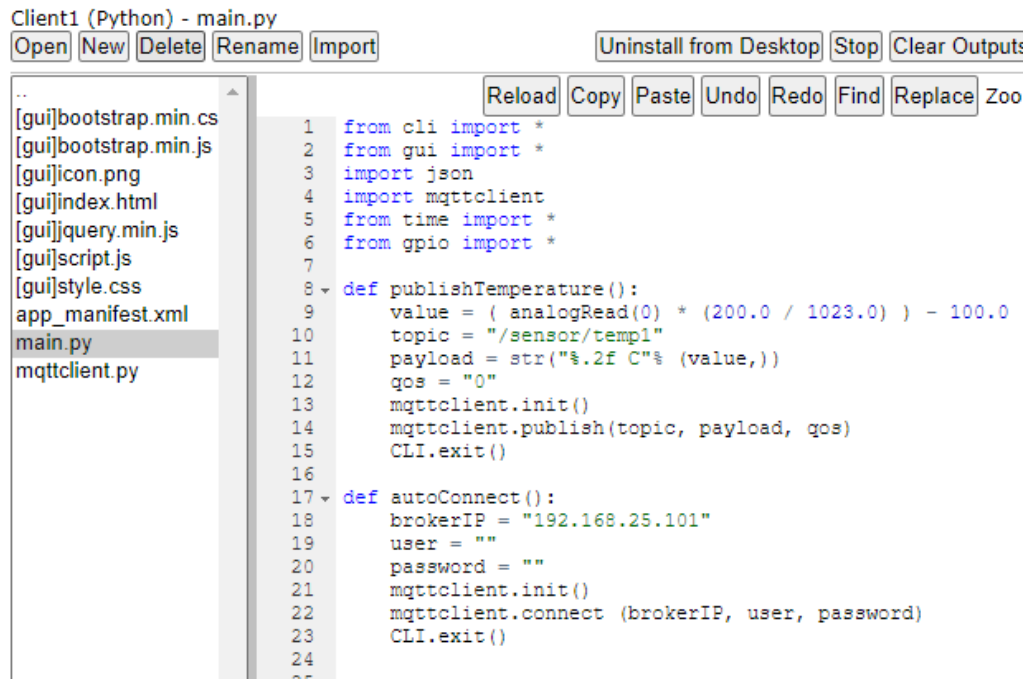
```
    mqttclient.init()
```

```
    mqttclient.connect (brokerIP, user, password)
```

```
    CLI.exit()
```

- En brokerIP poner la dirección IP del Broker0
-





```

Client1 (Python) - main.py
Open New Delete Rename Import Uninstall from Desktop Stop Clear Outputs
Reload Copy Paste Undo Redo Find Replace Zoo

..
[gui]bootstrap.min.cs
[gui]bootstrap.min.js
[gui]icon.png
[gui]index.html
[gui]jquery.min.js
[gui]script.js
[gui]style.css
app_manifest.xml
main.py
mqttclient.py

1 from cli import *
2 from gui import *
3 import json
4 import mqttclient
5 from time import *
6 from gpio import *
7
8 def publishTemperature():
9     value = ( analogRead(0) * (200.0 / 1023.0) ) - 100.0
10    topic = "/sensor/temp1"
11    payload = str("%.2f C"% (value,))
12    qos = "0"
13    mqttclient.init()
14    mqttclient.publish(topic, payload, qos)
15    CLI.exit()
16
17 def autoConnect():
18     brokerIP = "192.168.25.101"
19     user = ""
20     password = ""
21     mqttclient.init()
22     mqttclient.connect (brokerIP, user, password)
23     CLI.exit()
24
25

```

**Figura 5:** Código modificado de Client1

- En la función main() Incluir el siguiente código al inicio:

pinMode (0, IN)

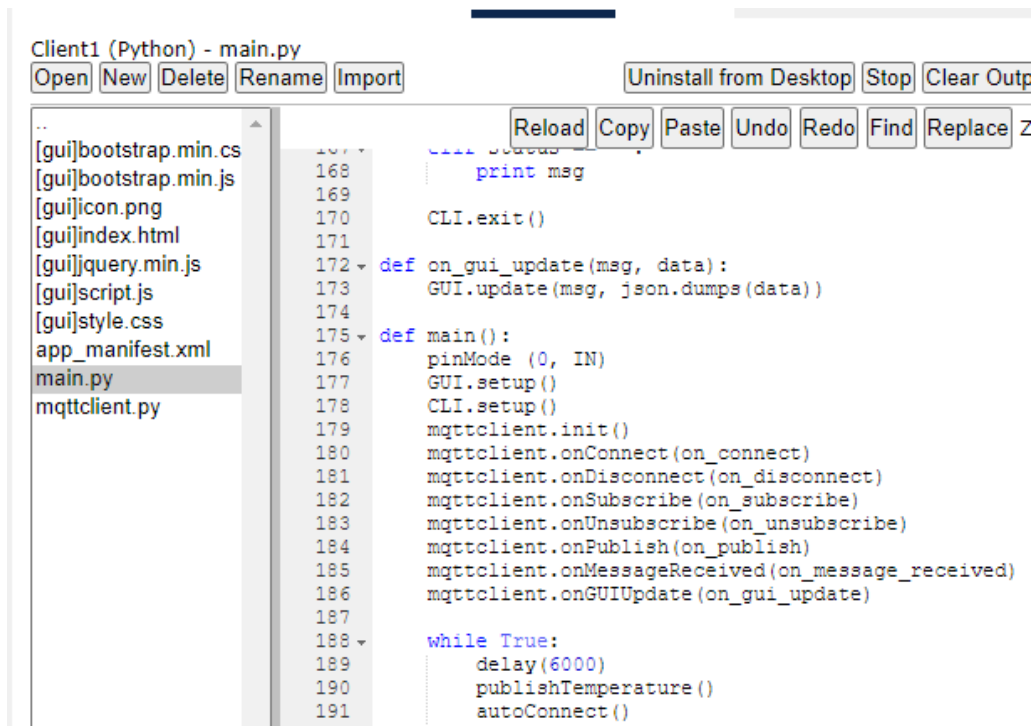
- Y sustituir el bucle final por:

while True:

    delay(6000)

    publishTemperature()

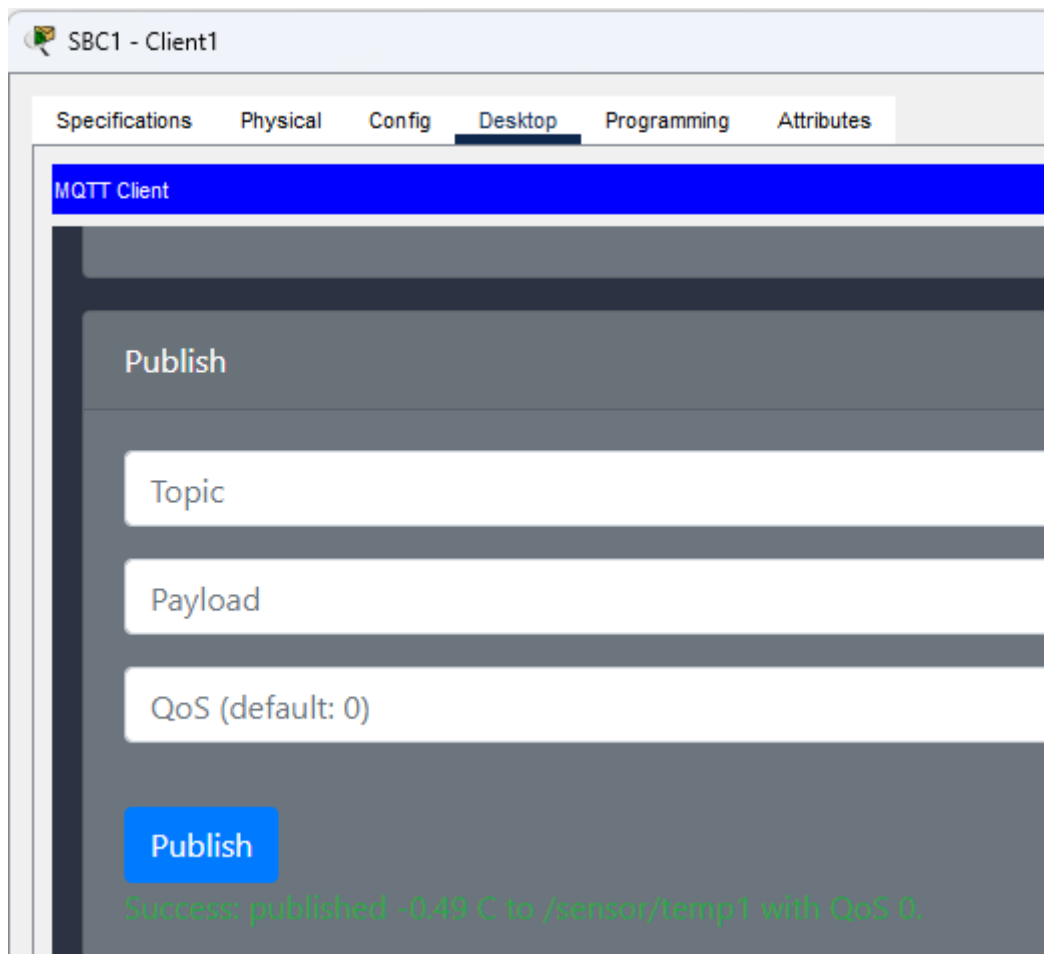
    autoConnect()



**Figura 6:** Main modificado de Client1

- Pulsar Install to Desktop
- Pulsar Run
- Abrir la pestaña Desktop → MQTT Client
  - En Connection → Broker Address, poner la IP del Broker.
  - Pulsar el botón Connect.

El cliente MQTT del SBC1 se registra automáticamente en el bróker MQTT del SBC0 mediante el código incluido (en autoConnect) y envía datos en el topic “/sensor/temp1”.

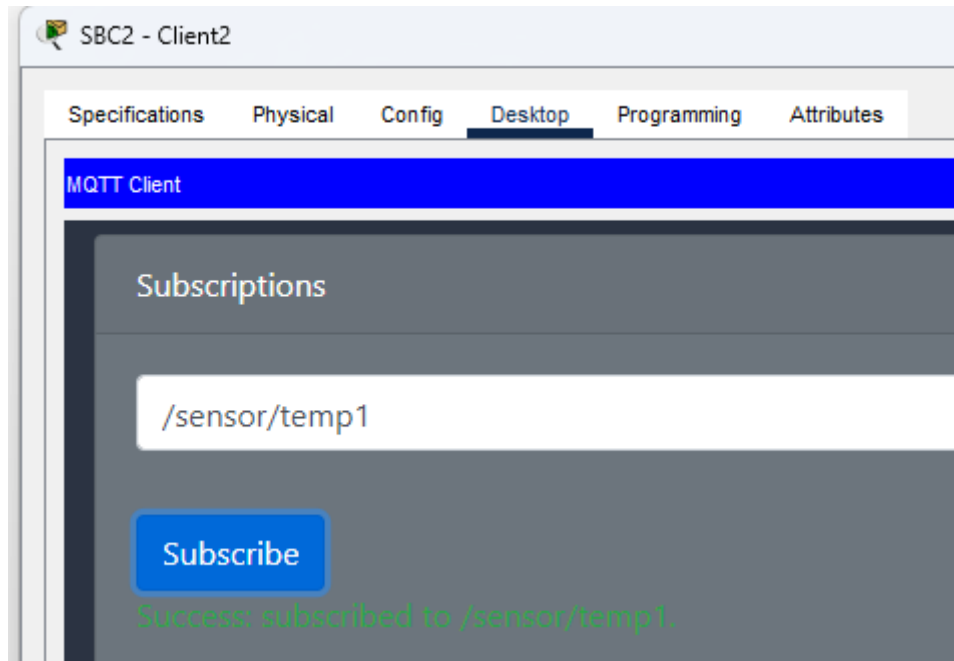


**Figura 7:** SBC1 Publicando temperatura en sensor/temp1

El resto de clientes MQTT deben conectarse y registrarse en el bróker:

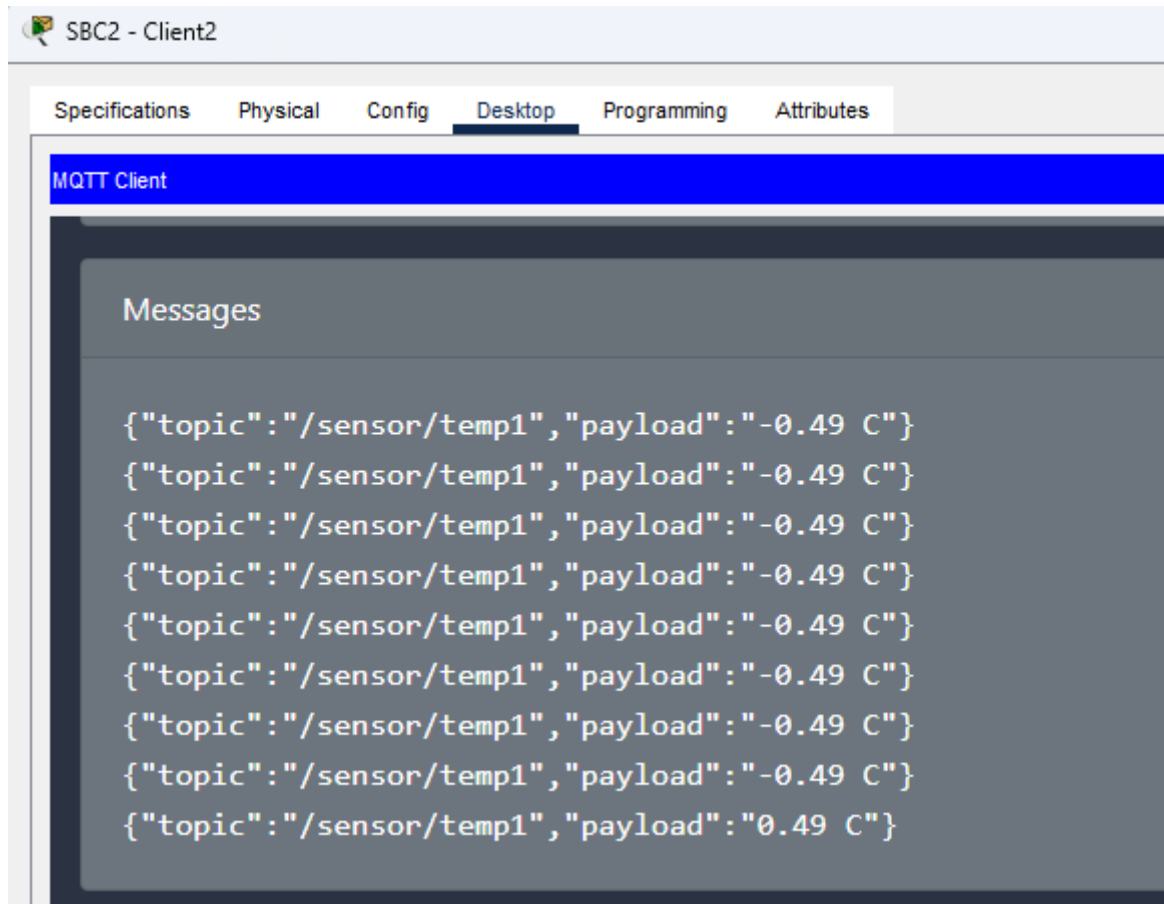
- Entrar en SBC2 → Desktop → MQTT Client
  - En Connection → Broker Address poner la IP del Broker.
    - Pulsar el botón Connect.
  - Una vez conectado, aparecen nuevos campos. En Subscriptions → Topic poner:
    - /sensor/temp1
    - Pulsar el botón Subscribe
- Entrar en SBC3 → Desktop → MQTT Client
  - En Connection → Broker Address poner la IP del Broker.
    - Pulsar el botón Connect.

- En Subscriptions → Topic poner:
  - /sensor/temp2
  - Pulsar el botón Subscribe



**Figura 8:** SBC2 suscrito a /sensor/temp1

En el apartado Messages se pueden ver los mensajes que se van recibiendo. En SBC2, se reciben y se muestran. En SBC3, no se muestran porque no está suscrito al topic que publica SBC1.



**Figura 9:** Messages recibidos en el topic /sensor/temp1

## 4a – Control mediante MQTT

- a) Diseñar un sistema basado en MQTT para controlar la apertura y cierre de una ventana conectada al SBC2 en función de la temperatura medida por el sensor conectado a SBC1.

Para este ejercicio he conectado mediante el cable de IoT una ventana al SBC2 en el puerto D0 de ambos dispositivos. Después he modificado el código que se ejecuta al recibir un mensaje en el SBC2, de tal forma que si la temperatura es mayor que 20, se abre la ventana y si es menor, se cierra.

```

def on_message_received(status, msg, packet):
    if status == "Success" or status == "Error":
        print status + ": " + msg
    elif status == "":
        print msg

    # Gestion de temperatura en mensaje recibido
    try:
        message_parts = msg.split(" ")
        if len(message_parts) > 1:
            temperature_str = message_parts[1].split()[0] # Obtener
            temperatura
            temperature = float(temperature_str)

            if temperature < 20:
                print("Temperatura menor de 20 C. Cerrando
                ventana.")
                customWrite(0, "0")
            else:
                print("Temperature mayor de 20 C. Abriendo
                ventana.")
                customWrite(0, "1")

    except Exception as e:
        print "Error processing message: " + str(e)

    CLI.exit()

```

También es necesario incluir el pinMode(0, OUT) en el main.

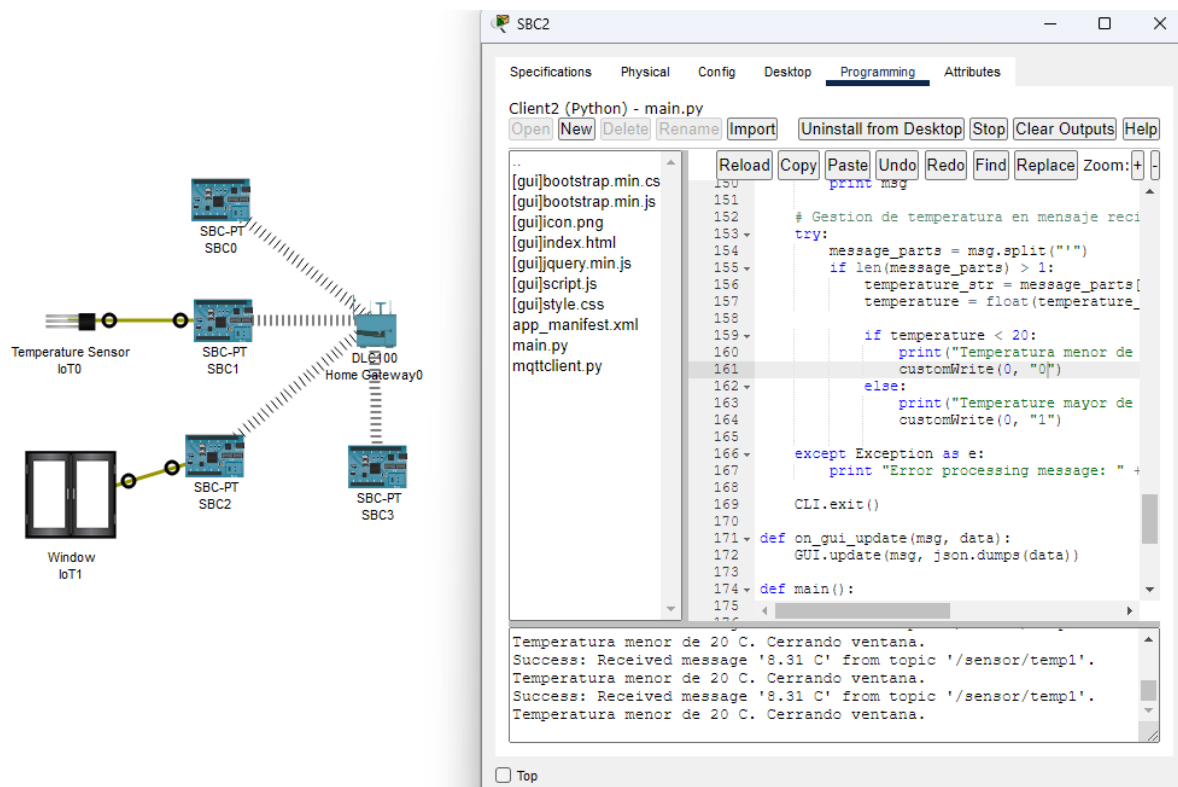
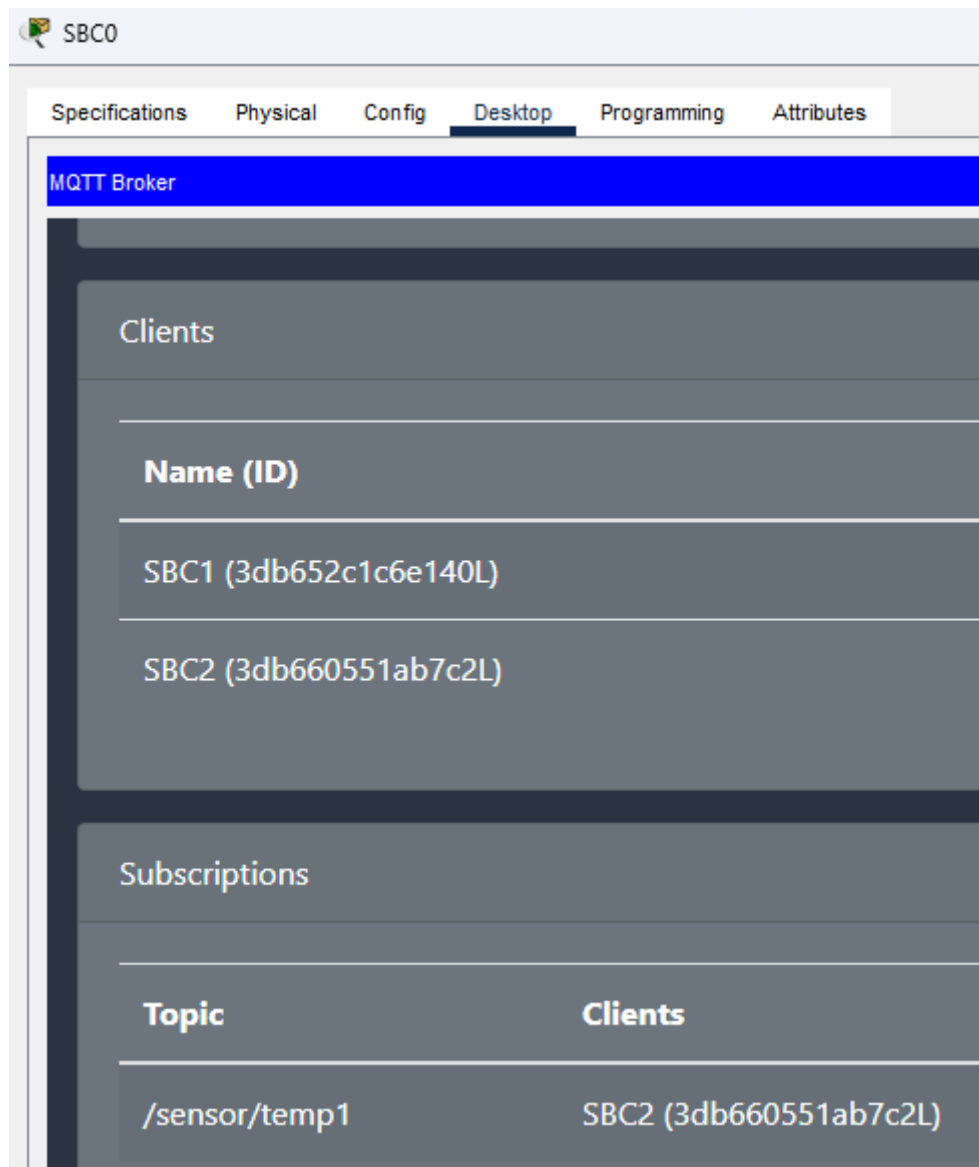


Figura 10: Simulación  $t < 20$  -> cerrar ventana



**Figura 11:** Clientes y 1 Topic del Broker

- b) Incorporarle al sistema anterior, un LED al SBC3 que se activará cuando la ventana se cierre y se mantendrá el LED encendido durante 2 segundos.

Para este sistema he creado un nuevo Topic "/devices/window1" al cual el SBC2 se encargará de publicar el estado de la ventana cuando esté cerrada. Para esta tarea he añadido el siguiente código al SBC2:

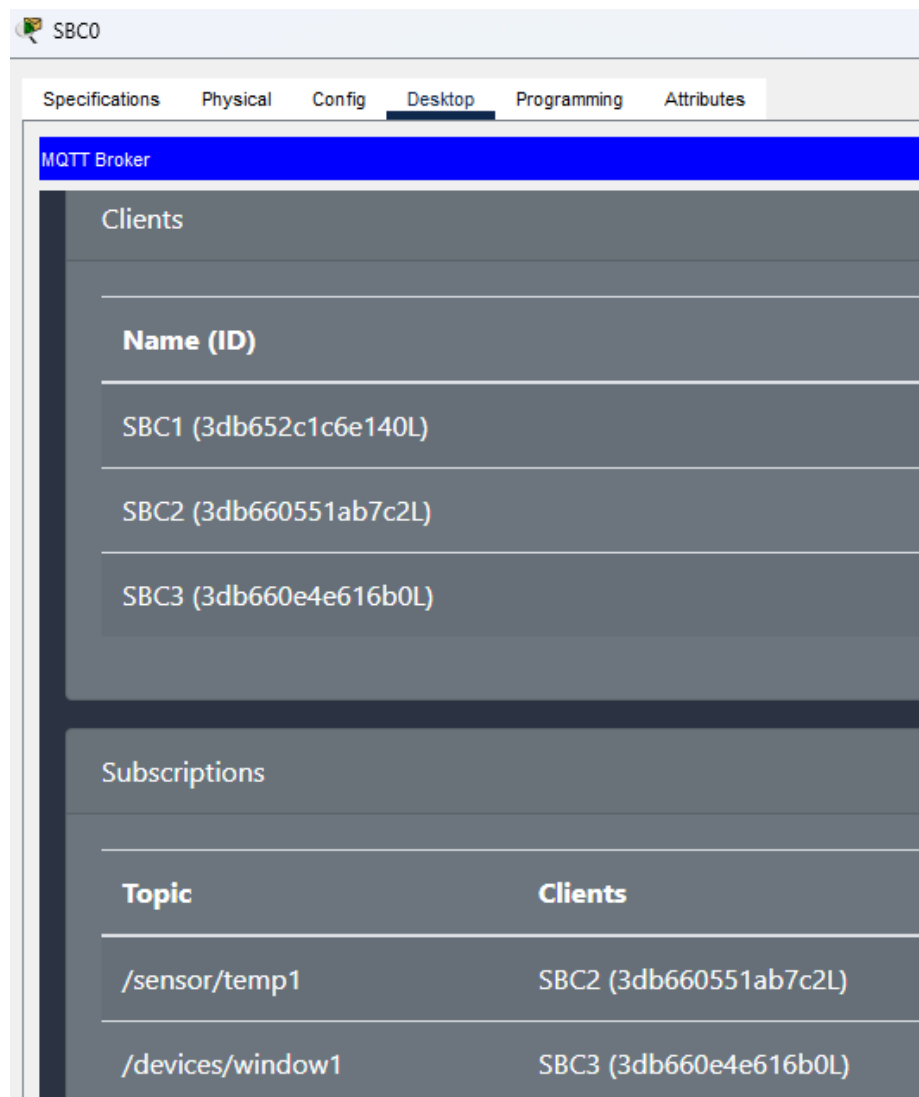
```
def publishWindowState():
    topic = "/devices/window1"
    payload = "cerrada"
    qos = "0"
    mqttclient.init()
    mqttclient.publish(topic, payload, qos)
```

```
CLI.exit()
```

Y llamo a `publishWindowState` cada vez que el SBC2 cierra o intenta cerrar la ventana (“intenta” pues puede que ya esté cerrada).

```
if temperature < 20:
    print("Temperatura menor de 20 C. Cerrando
    ventana.")
    customWrite(0, "0")
    publishWindowState()
else:
    print("Temperature mayor de 20 C. Abriendo
    ventana.")
    customWrite(0, "1")
```

En la figura 11 podemos ver como tenemos el nuevo Topic del cual es cliente el SBC3.



**Figura 12:** Clientes y 2 Topics del Broker



A continuación, en el SBC3 he tenido que modificar el código para que cada vez que reciba “cerrada” del Topic “/devices/window1” encienda el LED durante 2 segundos.

Como se hizo anteriormente, en `on_message_received` se analiza el mensaje recibido, y si contiene el estado “cerrada”, enciende el LED inmediatamente con `analogWrite(0, 1023)` y registra el tiempo en que se activó el LED. Este estado se mantiene en `led_state`, que guarda si el LED está activo (`is_active`) y el momento en que se activó (`last_change`).

Luego, en el bucle principal, la función `LED_activate` se ejecuta constantemente. Si el LED está activo y han pasado 2 segundos desde su activación, la función apaga el LED con `analogWrite(0, 0)` y actualiza el estado del LED a inactivo. De esta manera, el LED se enciende durante 2 segundos y luego se apaga automáticamente sin bloquear el flujo del programa.

```
# Estado del LED y temporizador
led_state = {
    "is_active": False,
    "last_change": 0,
}

def LED_activate():
    # Maneja la activación y desactivación del LED.
    global led_state
    current_time = time()

    if led_state["is_active"]:
        # Si el LED está activo y han pasado 2 segundos, apagar LED
        if current_time - led_state["last_change"] >= 2:
            print("Han pasado 2 segundos. Apagando LED.")
            analogWrite(0, 0) # Apaga el LED
            led_state["is_active"] = False
    else:
        # Si el LED no está activo, no hacer nada
        pass

def on_message_received(status, msg, packet):
    if status == "Success" or status == "Error":
        print(status + ": " + msg)
    elif status == "":
        print(msg)

    # Gestión de estado de ventana
    try:
        message_parts = msg.split(" ")
        if len(message_parts) > 1:
            window_state = message_parts[1].split()[0] # Obtener estado
            ventana

            if window_state == "cerrada":
                print("Ventana cerrada, activando LED.")
                analogWrite(0, 1023) # Enciende el LED inmediatamente
```

```

        led_state["is_active"] = True # Marca el LED como activo
        led_state["last_change"] = time() # Registra el tiempo
    actual

    except Exception as e:
        print("Error processing message: " + str(e))

def main():
    autoConnect()
    pinMode(0, OUT)
    GUI.setup()
    CLI.setup()
    mqttclient.init()
    mqttclient.onConnect(on_connect)
    mqttclient.onDisconnect(on_disconnect)
    mqttclient.onSubscribe(on_subscribe)
    mqttclient.onUnsubscribe(on_unsubscribe)
    mqttclient.onPublish(on_publish)
    mqttclient.onMessageReceived(on_message_received)
    mqttclient.onGUIUpdate(on_gui_update)

    while True:
        LED_activate()
        delay(100)

```

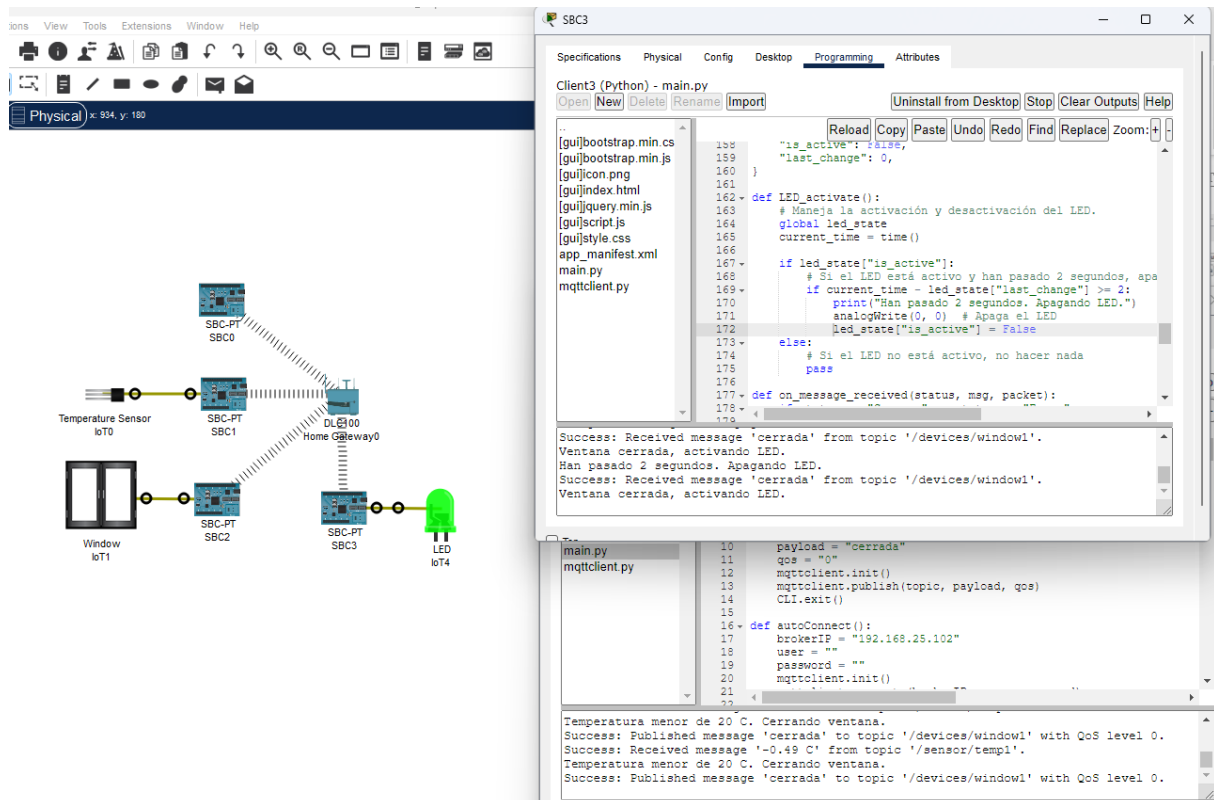


Figura 13: Simulación. LED encendido por ventana cerrada