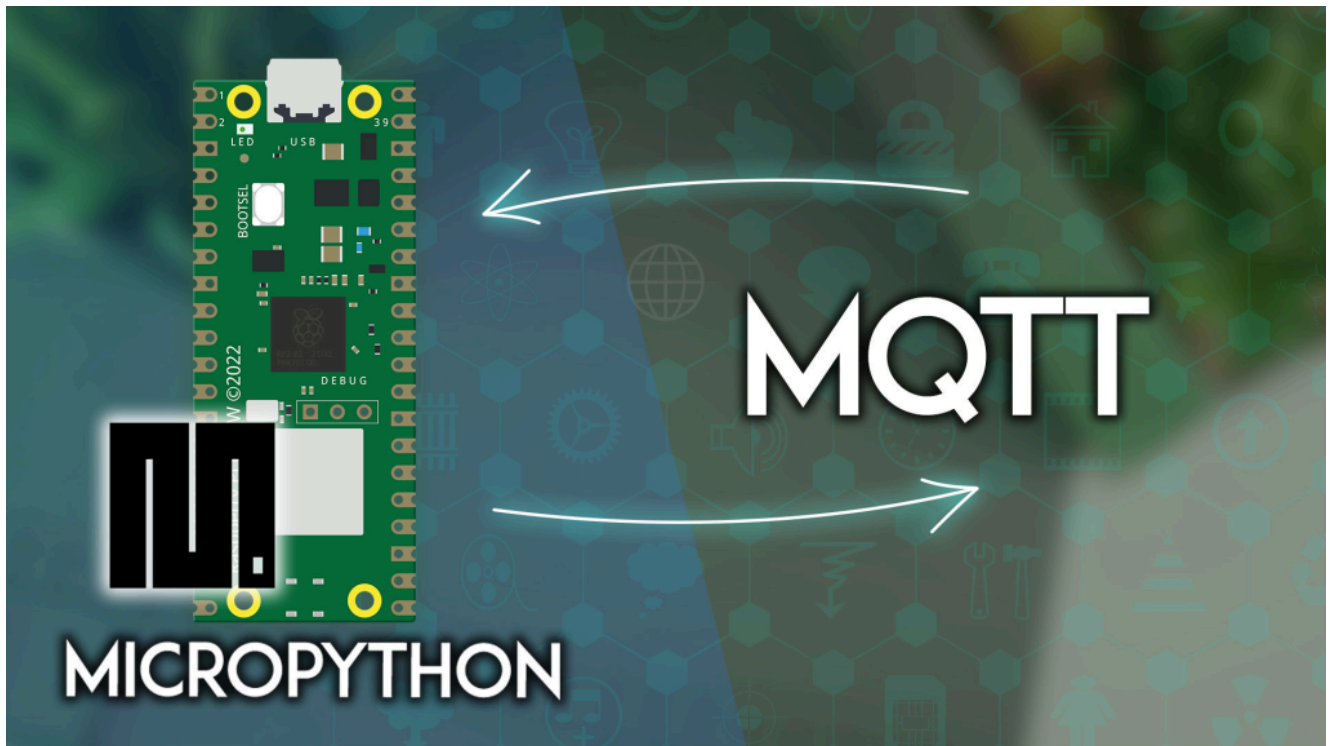


Raspberry Pi Pico W: Getting Started with MQTT (MicroPython)

This is a complete guide to using MQTT with Raspberry Pi Pico programmed with MicroPython. MQTT is a communication protocol widely used in Home Automation and IoT applications to connect multiple devices. In this tutorial, you'll learn how to choose and set up an MQTT broker and how to publish and subscribe to MQTT messages with the Raspberry Pi Pico.



New to the Raspberry Pi Pico? Get started with the Raspberry Pi Pico [here](#).

Table of Contents:

Throughout this guide we'll cover the following topics:

- [Introducing MQTT](#)
- [MQTT Basic Concepts](#)
- [Set Up the MQTT Broker](#)
- [Installing MQTT MicroPython Modules](#)

- [Creating a Configuration File](#)
- [Publishing MQTT Messages](#)
- [Subscribing to MQTT Topics](#)

Prerequisites

Before proceeding, make sure you check the following prerequisites:

MicroPython Firmware

To follow this tutorial you need MicroPython firmware installed in your Raspberry Pi Pico board. You also need an IDE to write and upload the code to your board.

The recommended MicroPython IDE for the Raspberry Pi Pico is Thonny IDE. Follow the next tutorial to learn how to install Thonny IDE, flash MicroPython firmware, and upload code to the board.

- [Programming Raspberry Pi Pico using MicroPython](#)

Parts Required

To follow the examples in this tutorial, you need the following parts:

- [Raspberry Pi Pico W](#)
- [BME280](#)
- [Breadboard](#)
- [Jumper wires](#)

You can use the preceding links or go directly to [MakerAdvisor.com/tools](https://makeradvisor.com/tools) to find all the parts for your projects at the best price!



Introducing MQTT

Already familiar with MQTT? [Jump to the next section.](#)

MQTT stands for Message Queuing Telemetry Transport. MQTT is a simple messaging protocol designed for constrained devices with low bandwidth. So, it's the perfect solution to exchange data between multiple IoT devices. MQTT communication works as a *publish* and *subscribe* system. Devices can *publish* messages on a specific topic. All devices that are *subscribed* to that topic receive the message.



Its main applications include sending messages to control outputs, reading and publishing data from sensor nodes, and much more.

MQTT Basic Concepts

In MQTT, there are a few basic concepts that you need to understand:

- Publish/Subscribe
- Messages
- Topics
- Broker

Publish/Subscribe

The first concept is the ***publish*** and ***subscribe*** system. In a publish and subscribe system, a device can publish a message on a topic or be subscribed to a particular topic to receive messages.



- For example, **Device 1** publishes on a topic;
- **Device 2** is subscribed to the same topic that **Device 1** is publishing in;
- So, **Device 2** receives the message.

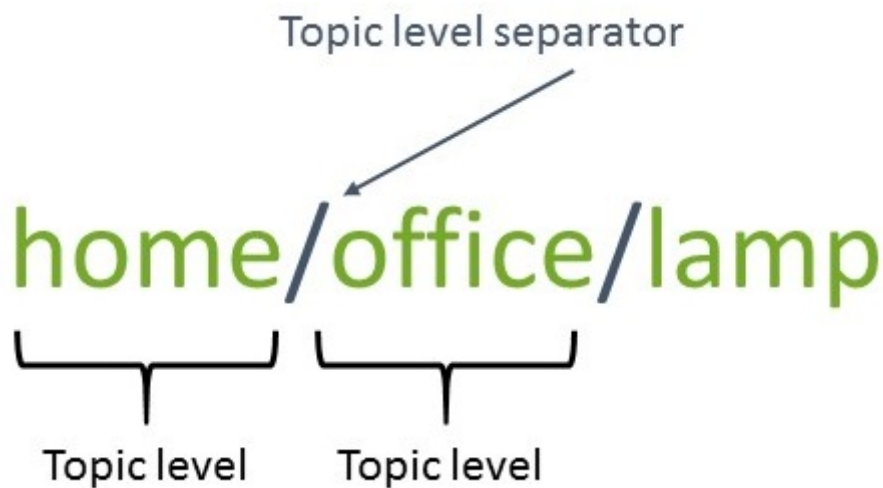
MQTT Messages

Messages are the information that you want to exchange between your devices. It can be a message like a command to control an output or data like sensor readings.

Topics

Another important concept is the *topics*. Topics are the way you register interest in incoming messages or how you specify where you want to publish the message.

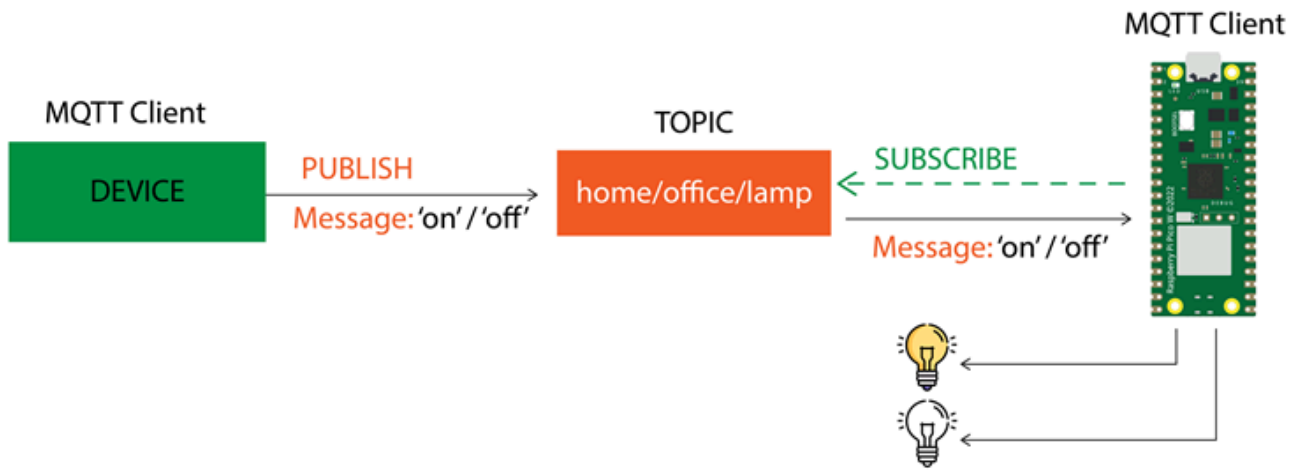
Topics are represented with strings separated by a forward slash. Each forward slash indicates a topic level. Here's an example of how you would create a topic for a lamp in your home office:



Note: topics are case-sensitive, which makes the following topics different.

`home/office/lamp`
 $\neq$
`home/office/Lamp`

If you would like to turn on a lamp in your home office using MQTT you can imagine the following scenario:



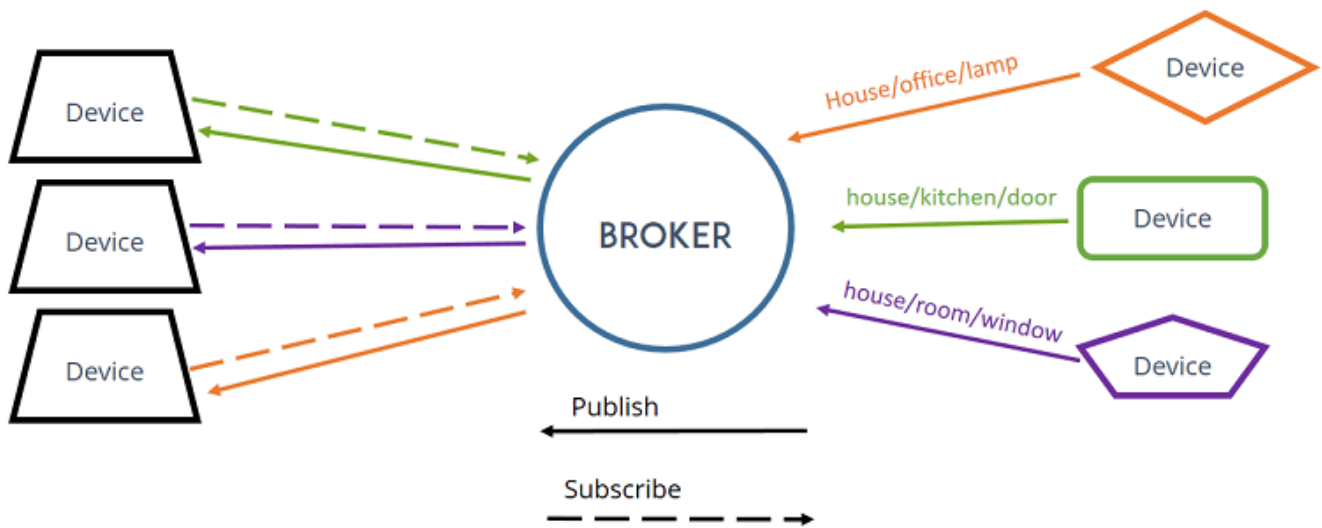
1. A device publishes on and off messages on the `home/office/lamp` topic.
2. You have a device that controls a lamp (it can be your Raspberry Pi Pico, or any other board or device). The Pico, that controls your lamp, is subscribed to that same topic: `home/office/lamp`.
3. So, when a new message is published on that topic, the Pico receives the on or off messages and turns the lamp on or off.

The device that is publishing the messages can be another microcontroller board, or a Home Automation controller platform with MQTT support like Node-RED, Home Assistant, Adafruit IO, Domoticz, or OpenHAB, for example.

Broker

Finally, another important concept is the *broker*.

The MQTT broker is responsible for receiving all messages, filtering the messages, deciding who is interested in them, and then publishing the message to all subscribed clients.



There are several brokers you can use. For example:

1. **Cloud MQTT Broker solutions:** you can use commercial MQTT broker solutions like HiveMQ, for example. You don't need to set up anything. You just need to create an account and you're ready to go (this is the one we'll use in this tutorial).
2. **Local MQTT broker:** you can install an MQTT broker locally on your computer or on your Raspberry Pi. The **Mosquitto** MQTT broker hosted on a Raspberry Pi is widely used in many hobbyist projects and it's also the solution we use more often.
3. **Cloud MQTT Broker:** as an alternative to the previous solution, you can also install MQTT broker on your own cloud server.

In summary:

- MQTT is a communication protocol very useful in Internet of Things projects;
- In MQTT, devices can publish messages on specific topics and can be subscribed to topics to receive messages;
- You need a broker when using MQTT. It receives all the messages and sends them to the subscribed devices.

Set Up the MQTT Broker

To use MQTT, you need an MQTT broker. The broker receives all MQTT messages and sends them to all subscribed clients.

MQTT Broker Solutions

There are many MQTT broker solutions you can use. Here we'll describe the ones we're more familiar with:

1. [Mosquitto MQTT broker installed on a Raspberry Pi](#): this is an open-source MQTT broker that you can install locally on your Raspberry Pi (the Raspberry Pi computer, not the Pico board). We've been using this option a lot and it always worked well for us.
2. [Mosquitto MQTT broker installed on a cloud server](#): a great alternative if you want your broker to be accessible worldwide.
3. [HiveMQ Broker](#): this is a cloud MQTT broker service. You just need to create an account and choose a plan. They provide a free plan that is suitable for most IoT hobbyist projects.

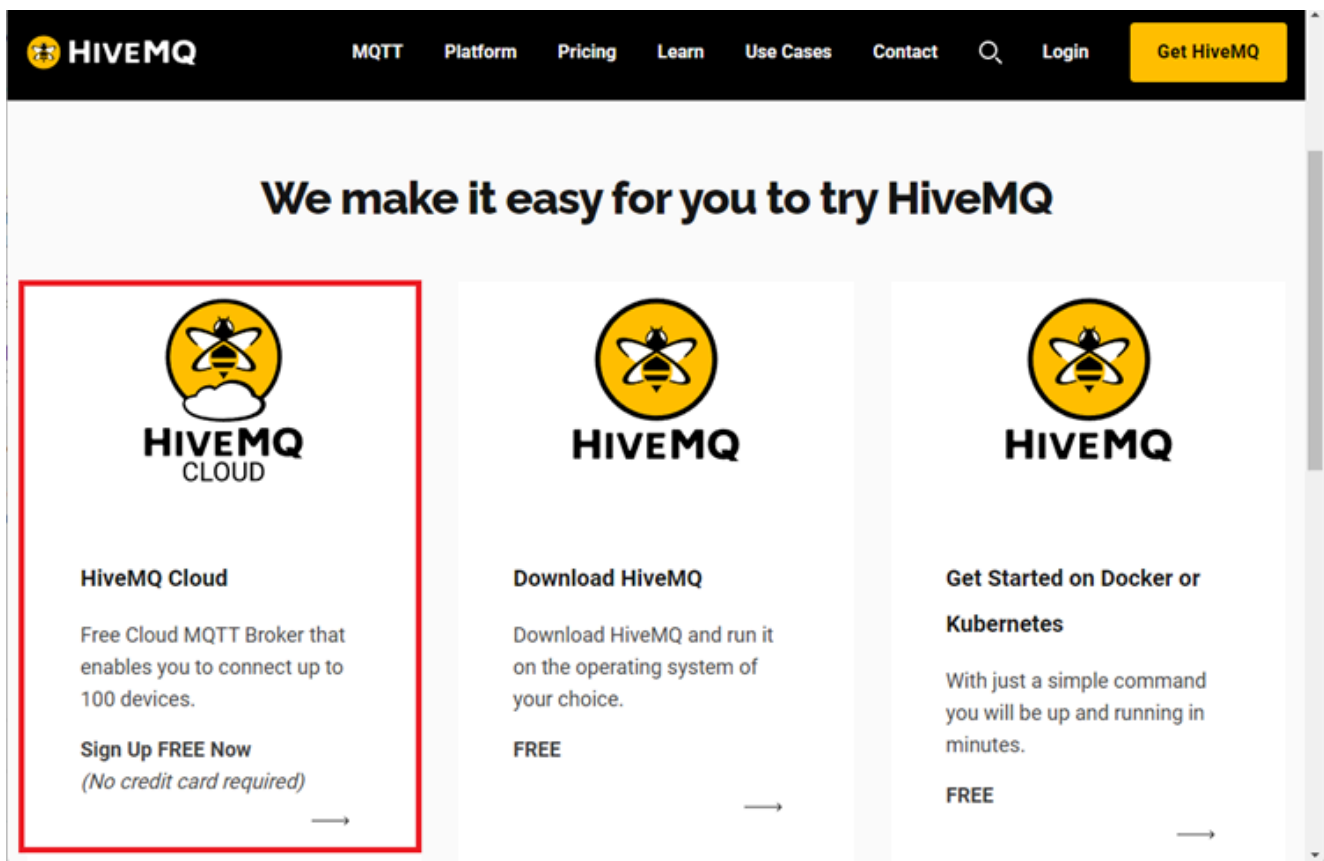
For simplicity, in this guide we'll use HiveMQ because you don't need to install or configure anything like in the Mosquitto MQTT broker. You just need to create an account, set up a cluster and you're ready to go. Alternatively, if you want to use Mosquitto broker or any other broker of your choice, don't worry. The code will be compatible with any broker as long as you fill in the right broker details.

Regardless of the MQTT broker you use, you need to have the MQTT URL, username, and password before proceeding to the following sections.

Setting Up the HiveMQ MQTT Broker

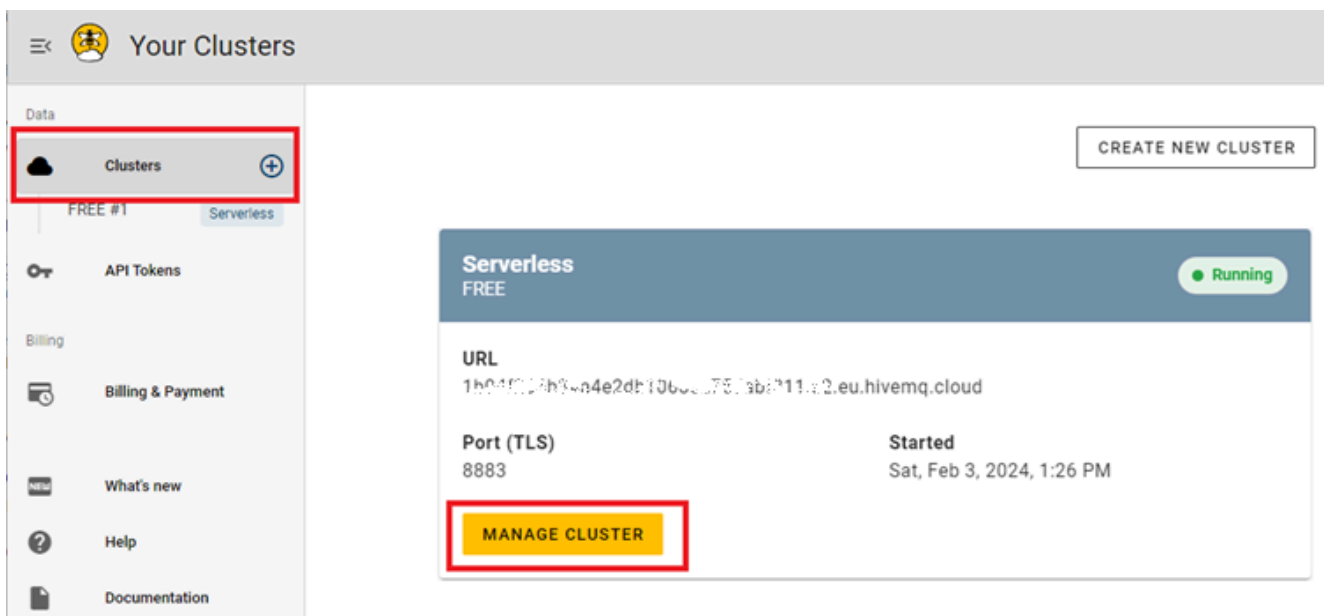
In this section, we'll show you how to set up your HiveMQ MQTT broker.

- 1) First, you need to create an account. Go to hivemq.com and click on **Start free**.
- 2) Choose the **HiveMQ Cloud plan**.



3) Login or create a new account and fill in the details to complete your profile.

4) You should have a new cluster created by default. Click on **Manage Cluster**.



5) Copy the Cluster URL to a safe place because you'll need it later.

Cluster Details

Cluster Information

Current Plan: Serverless
Current Tier: FREE

Name: 1b04f...
Cloud Provider: Azure

Connection Settings

Cluster URL: 10b04f...s2.eu.hivemq.cloud EDIT

Port: 8883

Websocket Port: 8884

TLS URI: 10b04f...s2.eu.hivemq.cloud:8883/mqtt

Websocket URI:

6) Click on the **Access Management** tab at the top.

7) Fill in the form with a username and password. You'll need to remember these details later to connect to the MQTT broker. Set the permission to *Publish and Subscribe*.

8) Finally, click **Create Credential**.

Access Management

Credentials

Currently you have not created any credentials. Fill out the following form to create an access credentials pair and limit access to your HiveMQ Cloud MQTT instance. To learn more [check out our Security Fundamentals guide](#).

Username *: SaraSantos
At least 5 characters

Password *:
At least 8 characters, 1 digit, 1 uppercase character

Confirm Password *:
Passwords must match

Permission *: Publish and Subscribe
Add permissions to limit access

CREATE CREDENTIAL

If your credentials are successfully created, your MQTT broker is set up.

Username	Permission type	Actions
SaraSantos	 Publish and Subscribe	DELETE

Make sure you have the following information before proceeding to the next section:

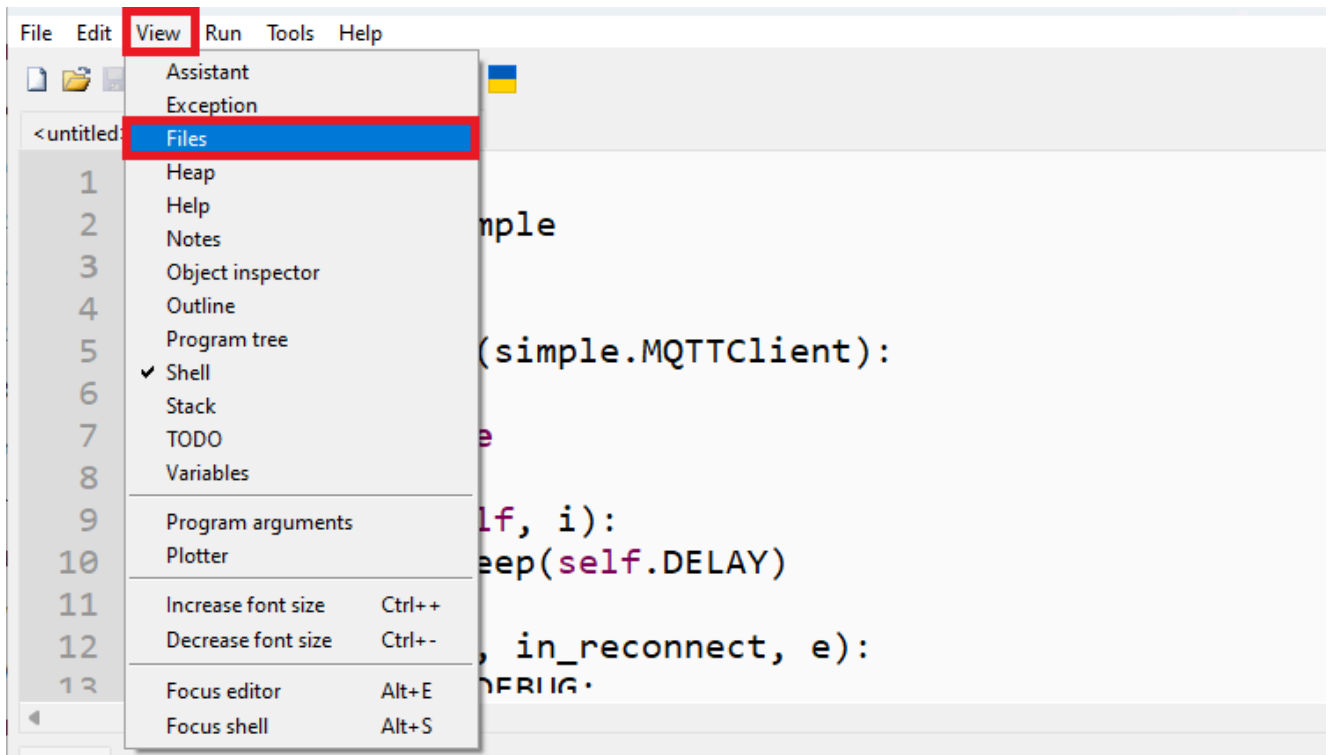
- MQTT Server:
 - Cluster URL (for HiveMQ)
 - MQTT broker IP address or URL (if you're using a different MQTT Broker)
- MQTT username
- MQTT password

Installing MQTT MicroPython Modules

To write code in MicroPython to use the MQTT communication protocol, we'll use two MQTT modules: *umqtt.simply.py*, and *umqtt.robust.py*.

Follow the next steps to upload the modules to your Raspberry Pi Pico.

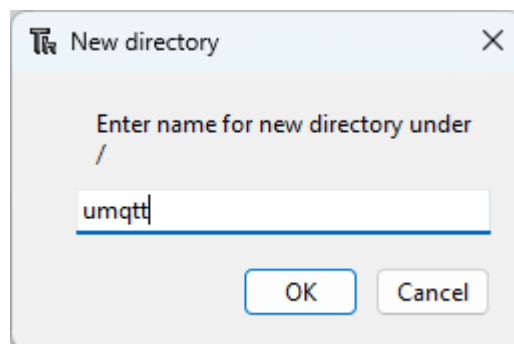
With the Raspberry Pi Pico connected to your computer and with a connection established with Thonny IDE, go to **View > Files**.



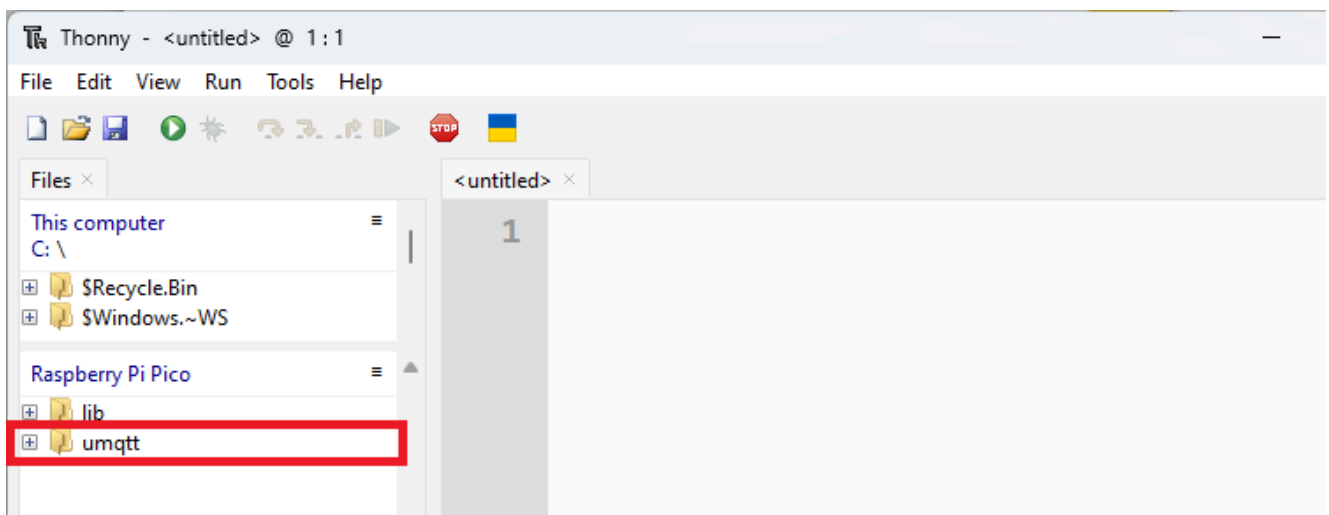
A new sidebar will show up with all the files on the Raspberry Pi Pico filesystem.

Right-click on the Raspberry Pi Pico sidebar and click on **New directory...**

This new directory should be called ***umqtt***. Click **Ok**.

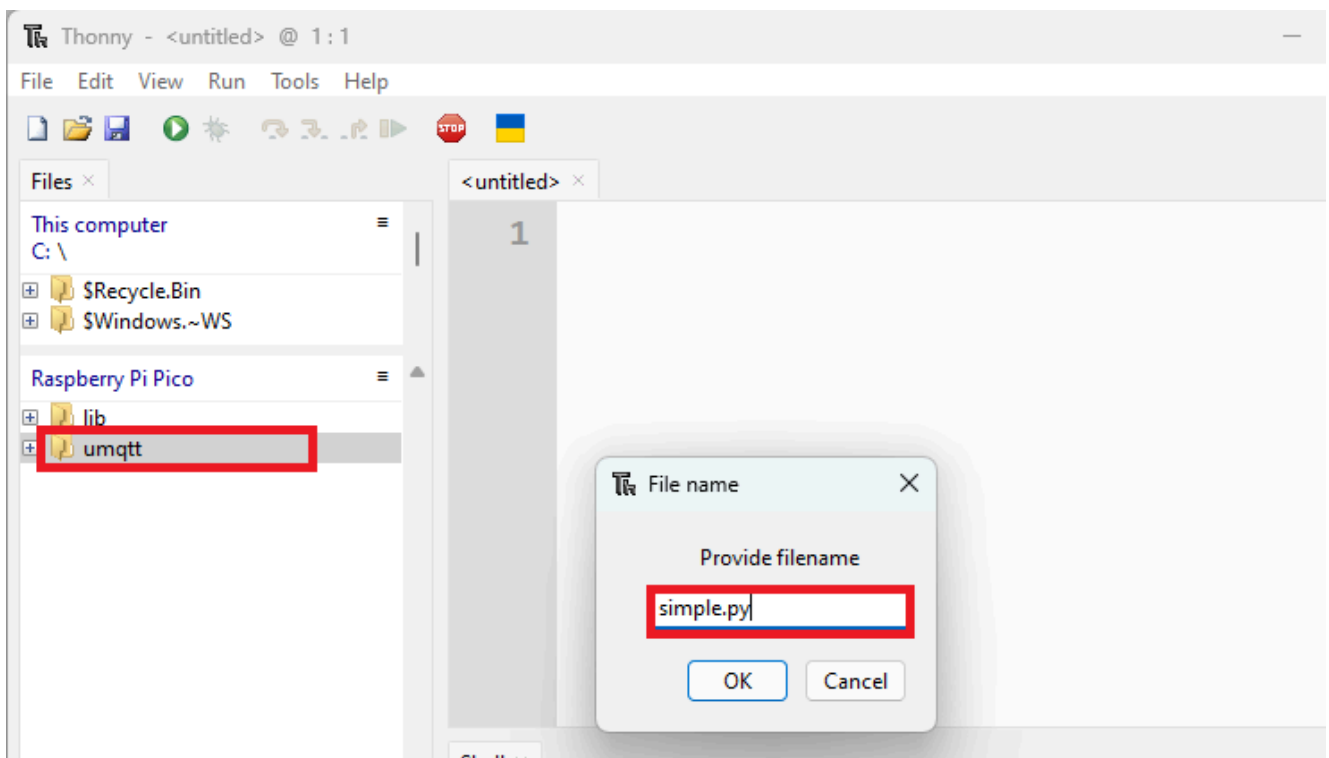


The new directory will show up on the left sidebar.



Right-click on the *umqtt* folder and select **New file...**

That new file should be called *simple.py*.



Copy the *simple.py* code into that new file. The code can be found on the link below:

- [Click here to download *simple.py*](#)

```
# forked from: https://github.com/micropython/micropython-lib/tree/master/micropython-lib/umqtt
import usocket as socket
import ustruct as struct
from ubinascii import hexlify
```

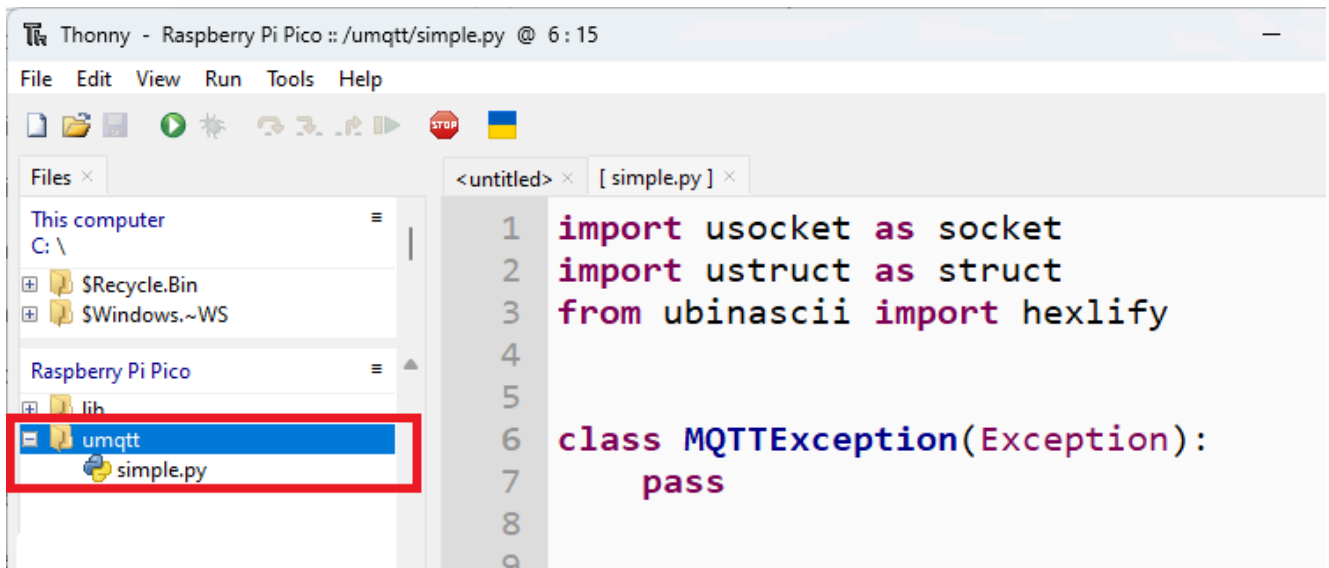
```
class MQTTException(Exception):
    pass

class MQTTClient:
    def __init__(
        self,
        client_id,
        server,
        port=0,
        user=None,
        password=None,
        keepalive=0,
        ssl=False,
        ssl_params={},
    ):
        if port == 0:
            port = 8883 if ssl else 1883
        self.client_id = client_id
        self.sock = None
        self.server = server
```

[View raw code](#)

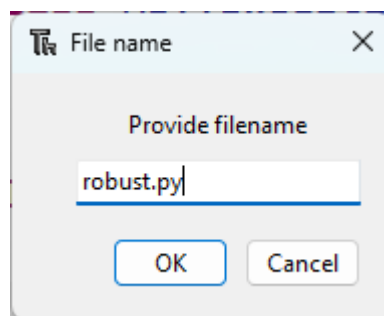
After copying the code to the *simple.py* file, save the code.

Now, if you expand the *umqtt* folder, you'll see that the *simple.py* file is there.



Right-click again on the *umqtt* folder and create another file by clicking on **New file...**

This new file should be called *robust.py*.



Copy the *robust.py* code into that new file. The code can be found on the link below:

- [Click here to download robust.py](#)

```

# forked from: https://github.com/micropython/micropython-lib/tree/master/umqtt
import utime
from . import simple

class MQTTClient(simple.MQTTClient):
    DELAY = 2
    DEBUG = False

    def delay(self, i):

```

```

        utime.sleep(self.DELAY)

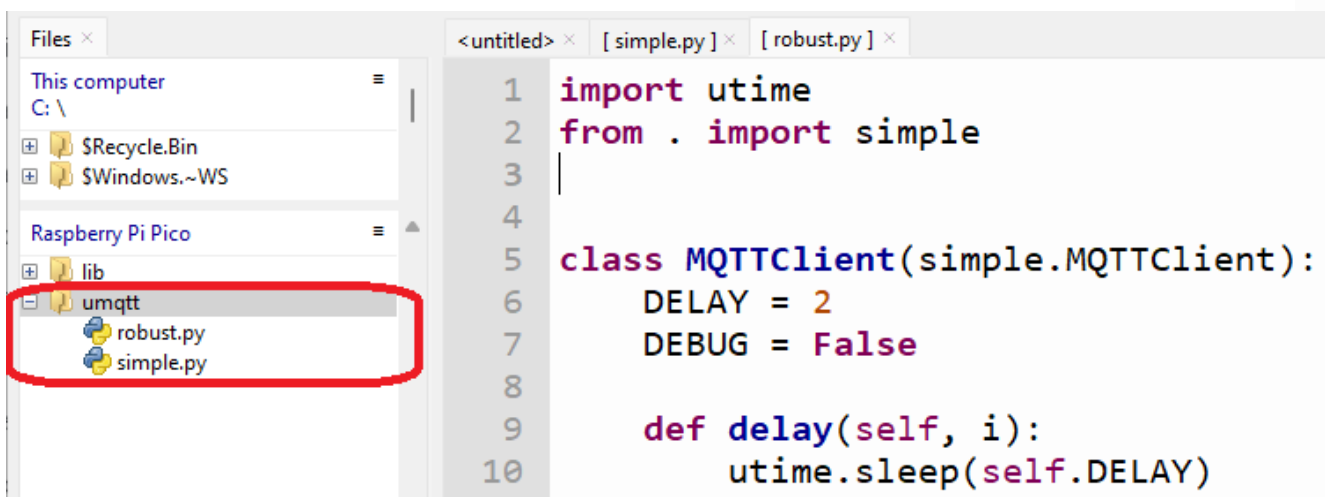
def log(self, in_reconnect, e):
    if self.DEBUG:
        if in_reconnect:
            print("mqtt reconnect: %r" % e)
        else:
            print("mqtt: %r" % e)

def reconnect(self):
    i = 0
    while 1:
        try:
            return super().connect(False)
        except OSError as e:
            self.log(True, e)
            i += 1

```

[View raw code](#)

After copying the code to the *robust.py* file, save the code. At this moment, you should have the *umqtt* folder with the *simple.py* and *robust.py* files inside.



The modules required for MQTT were successfully uploaded to the Raspberry Pi Pico.

Creating a Configuration File

We'll create a configuration file to save the SSID, password, and your MQTT broker details: URL, username, and password.

Create a new file called *config.py* on Thonny IDE and copy the following code:

```
wifi_ssid = 'REPLACE_WITH_YOUR_SSID'
wifi_password = 'REPLACE_WITH_YOUR_PASSWORD'
mqtt_server = b'MQTT_BROKER_URL'
mqtt_username = b'BROKER_USERNAME'
mqtt_password = b'BROKER_PASSWORD'
```

[View raw code](#)

Replace the variables with your own details.

Then, go to **File > Save as...** and select **Raspberry Pi Pico**. Save the file as *config.py* (overwrite any existing files with the same name). This file should be saved on the root of the Raspberry Pi Pico and **not** inside the *umqtt* folder.

Publishing MQTT Messages

In this example, you'll learn how to publish MQTT messages on a certain topic with your Raspberry Pi Pico. As an example, we'll publish temperature, humidity, and pressure readings from a BME280 sensor. Alternatively, you can use any other sensor or random values to test the project and concepts.

Before proceeding, make sure that:

- You connect a BME280 sensor to the Raspberry Pi Pico. Use GPIO 4 (SDA) and GPIO 5 (SCL);
- You must upload the *BME280.py* module to control the BME280 with your Raspberry Pi Pico — [check this tutorial](#) and install the library as indicated there.

Related content: Raspberry Pi Pico: BME280 Get Temperature, Humidity, and Pressure (MicroPython)

The following code gets sensor data from the BME280 sensor and publishes the readings on different MQTT topics.

```
# Rui Santos & Sara Santos - Random Nerd Tutorials
# Complete project details at https://RandomNerdTutorials.com/ra

from machine import Pin, I2C
from time import sleep
import network
from umqtt.simple import MQTTClient
import config
import BME280

# Constants for MQTT Topics
MQTT_TOPIC_TEMPERATURE = 'pico/temperature'
MQTT_TOPIC_PRESSURE = 'pico/pressure'
MQTT_TOPIC_HUMIDITY = 'pico/humidity'

# MQTT Parameters
MQTT_SERVER = config.mqtt_server
MQTT_PORT = 0
MQTT_USER = config.mqtt_username
MQTT_PASSWORD = config.mqtt_password
MQTT_CLIENT_ID = b"raspberrypi_picow"
MQTT_KEEPALIVE = 7200
MQTT_SSL = True
MQTT_SSL_PARAMS = {'server_hostname': MQTT_SERVER}

# Initialize I2C communication
i2c = I2C(id=0, scl=Pin(5), sda=Pin(4), freq=10000)
```

[View raw code](#)

How the Code Works

We use the following MQTT topics to publish data:

- For the temperature — pico/temperature

- For the humidity — `pico/humidity`
- For the pressure — `pico/pressure`

And the workflow to publish data is the following:

1. Connect the Pico to the internet;
2. Connect to the MQTT broker;
3. After connecting to the broker, we can continuously publish MQTT messages.

Let's take a look at how the code works.

Importing Libraries

First, we need to import the required libraries. We import the `Pin` and `I2C` from the `machine` module, and the `BME280` to interface with the BME280 sensor. The `network` module to connect to Wi-Fi and the `MQTTClient` class from `umqtt.simple` to use MQTT functions.

```
from machine import Pin, I2C
from time import sleep
import network
from umqtt.simple import MQTTClient
import config
import BME280
```

MQTT Topics

On the following variables we save the topics where we want to publish our messages (sensor readings).

```
# Constants for MQTT Topics
MQTT_TOPIC_TEMPERATURE = 'pico/temperature'
MQTT_TOPIC_PRESSURE = 'pico/pressure'
MQTT_TOPIC_HUMIDITY = 'pico/humidity'
```

MQTT Details

In the following lines we set up the MQTT parameters to connect to our MQTT broker. We import the server URL, username, and password from the ***config.py*** file.

```
# MQTT Parameters
MQTT_SERVER = config.mqtt_server
MQTT_PORT = 0
MQTT_USER = config.mqtt_username
MQTT_PASSWORD = config.mqtt_password
MQTT_CLIENT_ID = b"raspberrypi_picow"
MQTT_KEEPALIVE = 7200
MQTT_SSL = True
MQTT_SSL_PARAMS = {'server_hostname': MQTT_SERVER}
```

The `MQTT_CLIENT_ID` should be a unique ID to identify the MQTT client. You can give it whatever name you want, but it needs to be unique among the MQTT clients connected to your broker. In our case, it's `raspberrypi_picow`.

```
MQTT_CLIENT_ID = b"raspberrypi_picow"
```

BME280 Sensor

In the following lines, we initialize the BME280 sensor and create a function called `get_sensor_readings()` that gets data from the BME280 and returns the temperature, humidity, and pressure.

```
# Initialize I2C communication
i2c = I2C(id=0, scl=Pin(5), sda=Pin(4), freq=10000)

# Initialize BME280 sensor
bme = BME280.BME280(i2c=i2c, addr=0x76)

def get_sensor_readings():
    temp = bme.temperature[:-1]
```

```
hum = bme.humidity[:-1]
pres = bme.pressure[:-3]
return temp, hum, pres
```

Initialize Wi-Fi

The `initialize_wifi()` function connects the Raspberry Pi Pico to a network. You need to initialize Wi-Fi to connect to the MQTT broker and exchange messages.

```
def initialize_wifi(ssid, password):
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)

    # Connect to the network
    wlan.connect(ssid, password)

    # Wait for Wi-Fi connection
    connection_timeout = 10
    while connection_timeout > 0:
        if wlan.status() >= 3:
            break
        connection_timeout -= 1
        print('Waiting for Wi-Fi connection...')
        sleep(1)

    # Check if connection is successful
    if wlan.status() != 3:
        return False
    else:
        print('Connection successful!')
        network_info = wlan.ifconfig()
        print('IP address:', network_info[0])
        return True
```

You may also like: [Raspberry Pi Pico: Web Server \(MicroPython\)](#)

Connect to MQTT

The `connect_mqtt()` function connects to the MQTT broker using the details about the MQTT broker you've set up previously.

```
def connect_mqtt():
    try:
        client = MQTTClient(client_id=MQTT_CLIENT_ID,
                             server=MQTT_SERVER,
                             port=MQTT_PORT,
                             user=MQTT_USER,
                             password=MQTT_PASSWORD,
                             keepalive=MQTT_KEEPALIVE,
                             ssl=MQTT_SSL,
                             ssl_params=MQTT_SSL_PARAMS)

        client.connect()
        return client
    except Exception as e:
        print('Error connecting to MQTT:', e)
        raise # Re-raise the exception to see the full traceback
```



Publish MQTT Messages

The `publish_mqtt()` function publishes a message on a topic. Pass as argument the topic and the message you want to send.

```
def publish_mqtt(topic, value):
    client.publish(topic, value)
    print(topic)
    print(value)
    print("Publish Done")
```

Publishing Sensor Readings

Now that we have all functions and variables defined, we can finally connect to the internet and to the broker to start publishing messages.

First, we try to connect to Wi-Fi using the SSID and password stored on the ***config.py*** file.

```
try:
    if not initialize_wifi(config.wifi_ssid, config.wifi_password):
        print('Error connecting to the network... exiting program')
```



If we succeed in connecting to Wi-Fi, we can connect to the MQTT broker.

```
else:
    # Connect to MQTT broker, start MQTT client
    client = connect_mqtt()
```

After connecting, we get new sensor data and save it on the temperature, humidity, and pressure variables.

```
# Read sensor data
temperature, humidity, pressure = get_sensor_readings()
```

Finally, we use the `publish_mqtt()` function to publish the readings on their specific topics. The message must be a string. So, we need to convert the sensor data using the `str()` function.

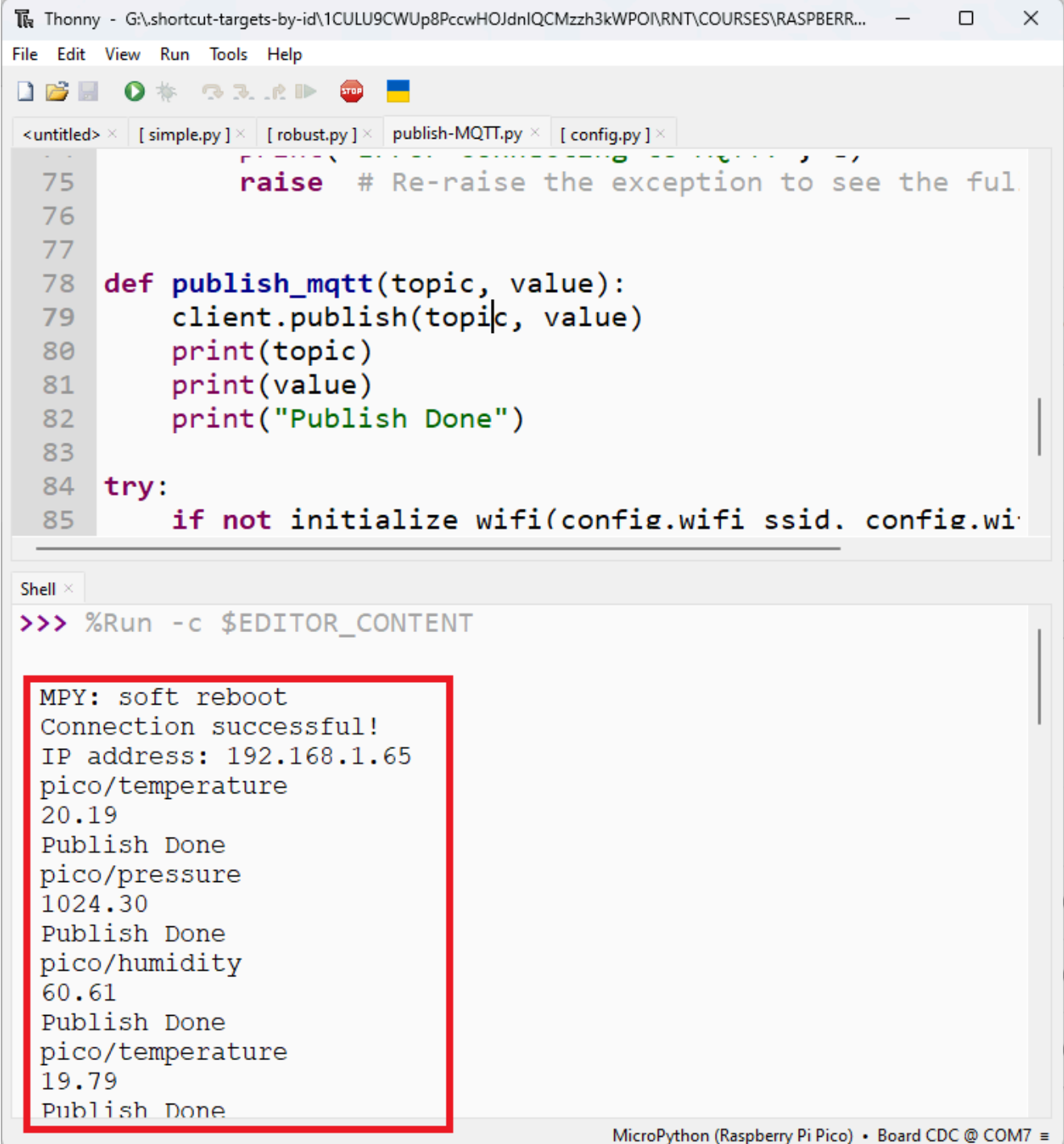
```
# Publish as MQTT payload
publish_mqtt(MQTT_TOPIC_TEMPERATURE, str(temperature))
publish_mqtt(MQTT_TOPIC_PRESSURE, str(pressure))
publish_mqtt(MQTT_TOPIC_HUMIDITY, str(humidity))
```

We publish new readings every 10 seconds.

```
sleep(10)
```

Testing the Code

Run the previous code on your Raspberry Pi Pico. It will connect to the internet and start publishing messages every 10 seconds.



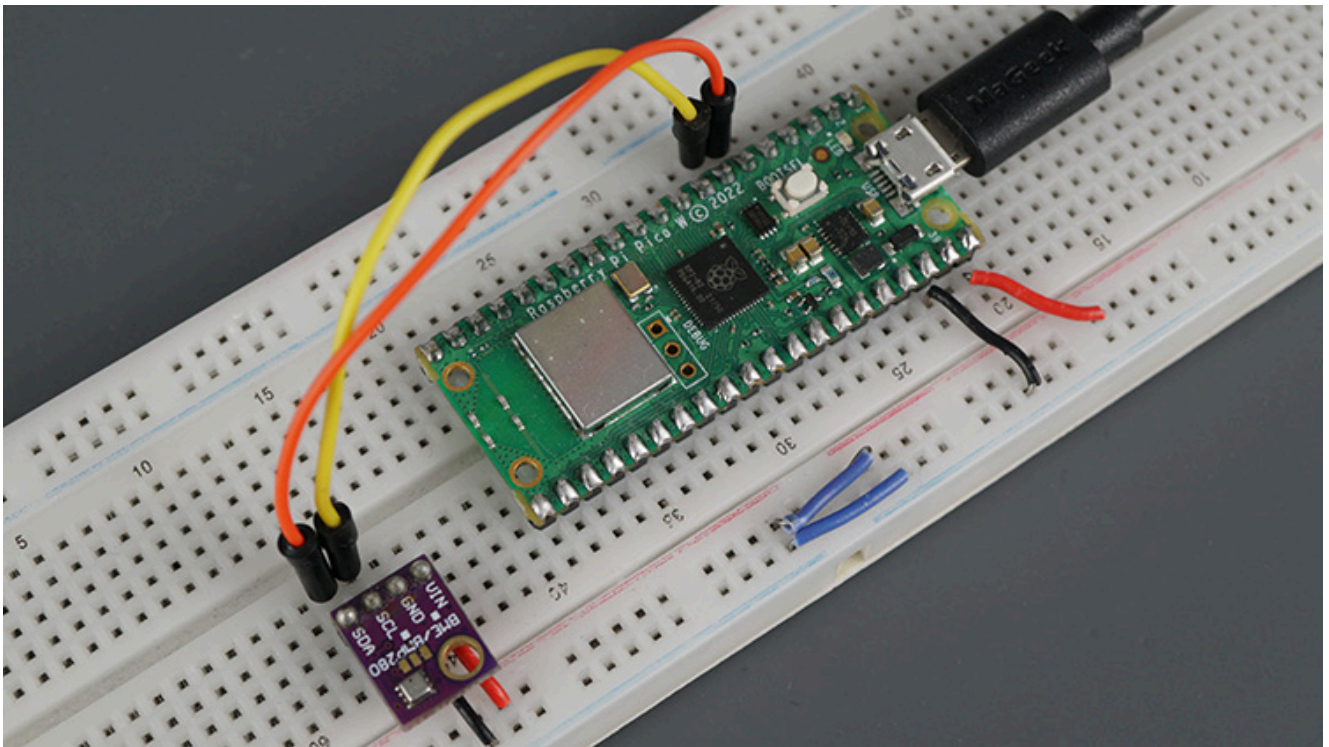
The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations and execution. The editor window displays a Python script with the following code:

```
75         raise # Re-raise the exception to see the full
76
77
78 def publish_mqtt(topic, value):
79     client.publish(topic, value)
80     print(topic)
81     print(value)
82     print("Publish Done")
83
84 try:
85     if not initialize_wifi(config.wifi_ssid, config.wifi_password):
```

The bottom panel shows the Shell window with the command `>>> %Run -c $EDITOR_CONTENT`. The output of the script is displayed below the command, enclosed in a red box:

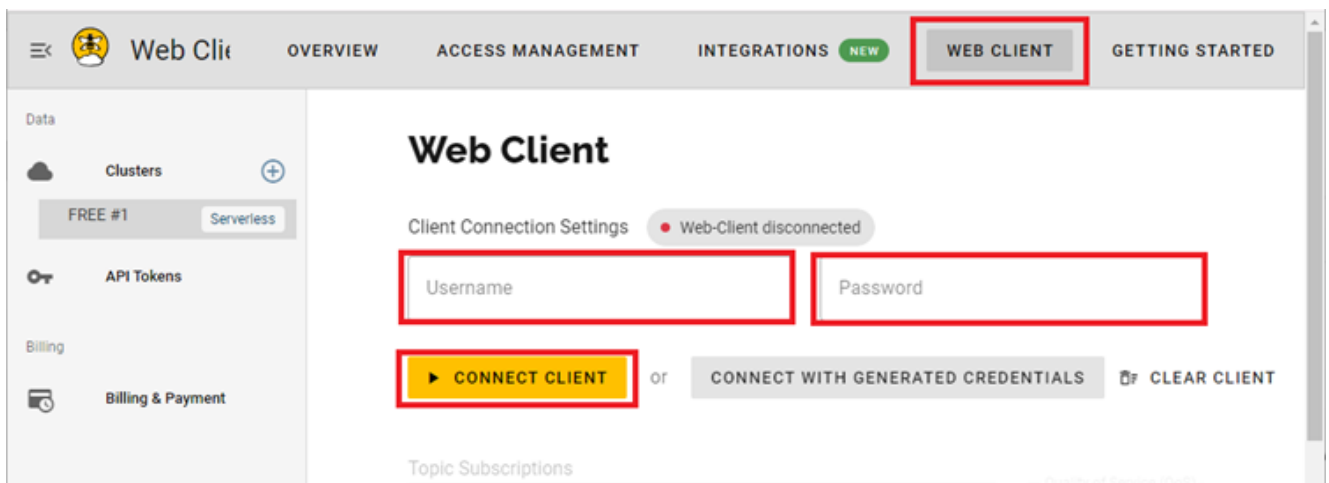
```
MPY: soft reboot
Connection successful!
IP address: 192.168.1.65
pico/temperature
20.19
Publish Done
pico/pressure
1024.30
Publish Done
pico/humidity
60.61
Publish Done
pico/temperature
19.79
Publish Done
```

The status bar at the bottom right indicates "MicroPython (Raspberry Pi Pico) • Board CDC @ COM7".



Now, we can check if we can receive the messages on an MQTT Client subscribed to those topics. HiveMQ provides a Web Client interface that allows you to subscribe and publish to topics for testing purposes.

Go to your HiveMQ cluster, and click on the **Web Client** tab. Insert your MQTT broker username and password and click **Connect Client**.



Subscribe to the topics that the Raspberry Pi Pico is publishing to. Fill in `pico/temperature` and then click on **+Subscribe**.

Web Client

Client Connection Settings

● Web-Client connected

Username

SaraSantos

Password

.....

■ DISCONNECT CLIENT

🗑️ CLEAR CLIENT

Topic Subscriptions

Topic Name

pico/temperature

Quality of Service (QoS)

0 - At most... ▼

+ SUBSCRIBE

🗑️ UNSUBSCRIBE FROM ALL

No active subscription. Add a subscription to receive new messages.

SUBSCRIBE TO ALL TOPICS (#)

Repeat the process for pico/humidity and pico/pressure .

Topic	QoS	Actions	
pico/pressure	0	↔ CHANGE COLOR	🗑️ UNSUBSCRIBE
pico/humidity	0	↔ CHANGE COLOR	🗑️ UNSUBSCRIBE
pico/temperature	0	↔ CHANGE COLOR	🗑️ UNSUBSCRIBE

After subscribing to those topics, scroll down to the bottom of the page. You should start receiving the Raspberry Pi Pico MQTT messages.

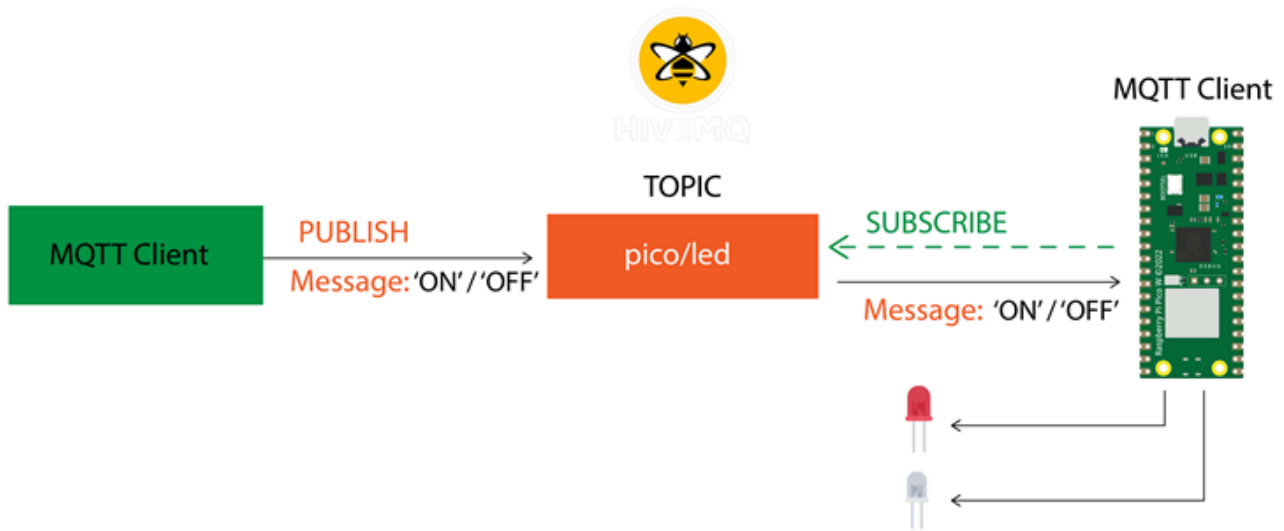
Message	Topic	QoS	Timestamp
19.78	pico/temperature	0	1706992059152
60.74	pico/humidity	0	1706992059147
1024.99	pico/pressure	0	1706992059137
60.72	pico/humidity	0	1706992049449
19.79	pico/temperature	0	1706992049442
1024.96	pico/pressure	0	1706992049416

And that's it. You successfully published MQTT messages with the Raspberry Pi Pico. Now, you can use any other MQTT Client to subscribe to those messages and receive the data. For example, having [Node-RED nodes](#) or Adafruit IO widgets subscribed to MQTT topics and displaying the data on charts and gauges.

You can also have any other microcontroller subscribed to those topics to receive the data—it can be another Raspberry Pi Pico, an ESP32, an ESP8266, or another board.

Subscribe to MQTT Topics

To show you how to subscribe to MQTT topics using the Raspberry Pi Pico we'll create a simple example, in which the Pico is subscribed to the `pico/led` topic. Then, with the Web Client from HiveMQ, we'll publish ON and OFF messages on that topic. The Pico will receive the messages and will control an LED accordingly.



Here's the workflow to subscribe to MQTT topics.

1. Connect the Raspberry Pi Pico to the internet;
2. Connect to the MQTT broker;
3. Subscribe to the MQTT topic;
4. Create and assign a callback function that will run when a message is received;
5. Create a loop that is constantly checking for new MQTT messages.

The following code subscribes to the `pico/led` topic and controls an LED according to the received message.

```
# Rui Santos & Sara Santos - Random Nerd Tutorials
# Complete project details at https://RandomNerdTutorials.com/ra

from machine import Pin
from time import sleep
import network
from umqtt.simple import MQTTClient
import config

# Define LED
led = Pin('LED', Pin.OUT)

# Constants for MQTT Topics
MQTT_TOPIC_LED = 'pico/led'
```

```
# MQTT Parameters
MQTT_SERVER = config.mqtt_server
MQTT_PORT = 0
MQTT_USER = config.mqtt_username
MQTT_PASSWORD = config.mqtt_password
MQTT_CLIENT_ID = b'raspberrypi_picow'
MQTT_KEEPALIVE = 7200
MQTT_SSL = True
MQTT_SSL_PARAMS = {'server_hostname': MQTT_SERVER}

# Init Wi-Fi Interface
def initialize_wifi(ssid, password):
```

[View raw code](#)

Let's take a quick look at the relevant sections of code for this example.

Subscribe to MQTT Topics

We create a function called `subscribe()` that accepts as arguments the MQTT client and the topic we want to subscribe to.

```
def subscribe(client, topic):
    client.subscribe(topic)
    print('Subscribe to topic:', topic)
```

Callback Function

We need to create a callback function that will run when a new message is received on a topic we are subscribed to. The topic and message arguments are automatically passed to the function when it is called.

```
def my_callback(topic, message):
    # Perform desired actions based on the subscribed topic and r
    print('Received message on topic:', topic)
    print('Response:', message)
    # Check the content of the received message
    if message == b'ON':
```

```
    print('Turning LED ON')
    led.value(1) # Turn LED ON
elif message == b'OFF':
    print('Turning LED OFF')
    led.value(0) # Turn LED OFF
else:
    print('Unknown command')
```

In this function, we check the content of the message. If it is ON , we'll turn the Raspberry Pi Pico built-in LED ON.

```
if message == b'ON':
    print('Turning LED ON')
    led.value(1) # Turn LED ON
```

If the message is OFF , it will turn the LED off.

```
elif message == b'OFF':
    print('Turning LED OFF')
    led.value(0) # Turn LED OFF
```

If we receive any other messages, we print Unknown command .

```
else:
    print('Unknown command')
```

Connecting to Wi-Fi, MQTT Broker and Checking Messages

To subscribe to the MQTT broker and constantly check for incoming messages, first we need to connect to the internet.

The following lines try to connect to the internet:

```
try:
    # Initialize Wi-Fi
    if not initialize_wifi(config.wifi_ssid, config.wifi_password):
        print('Error connecting to the network... exiting program')
```



If we succeed, we also connect to the MQTT broker.

```
client = connect_mqtt()
```

Then, assign the callback function by using the `set_callback()` and passing as argument the callback function we created previously `my_callback`.

```
client.set_callback(my_callback)
```

Finally, we can subscribe to MQTT topics. In this case, we're just subscribing to the `MQTT_TOPIC_LED`, but you can subscribe to multiple topics.

```
subscribe(client, MQTT_TOPIC_LED)
```

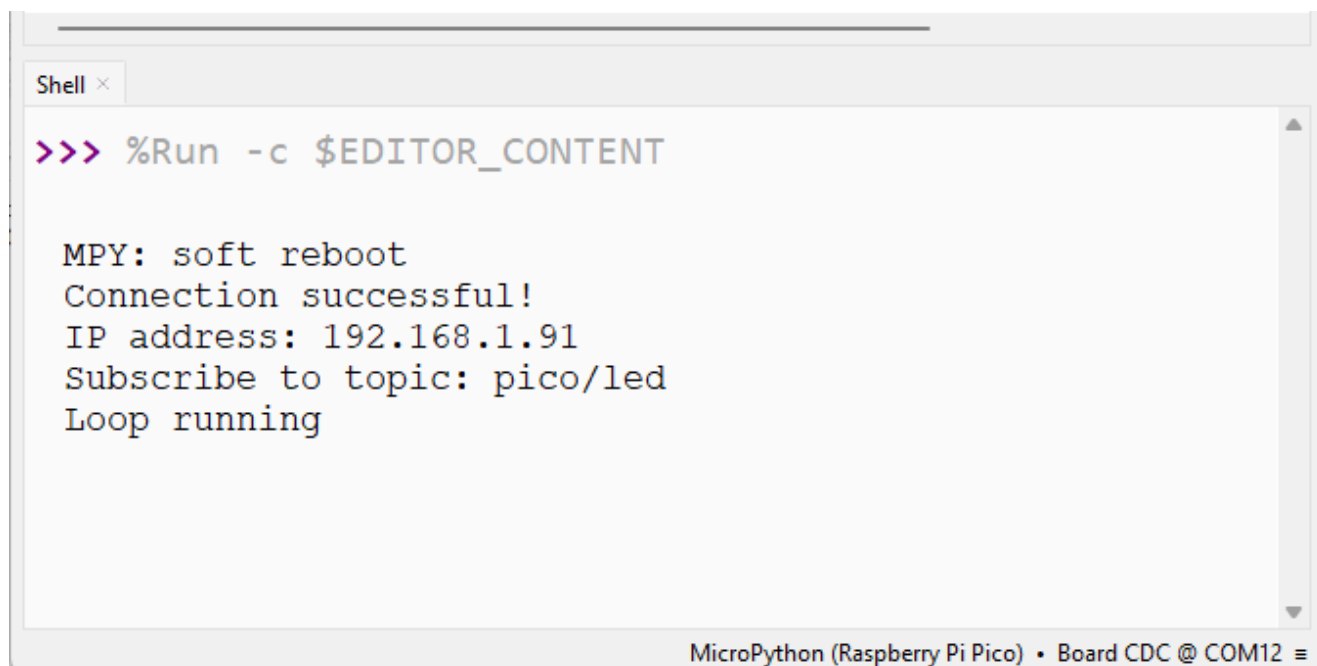
Then, create a while loop to constantly check for incoming messages using the `check_msg()` method.

```
while True:
    sleep(5)
    client.check_msg()
    print('Loop running')
```

You can also add other tasks to the loop. We check for new messages every five seconds, but you can use a smaller or longer delay time depending on how often you're expecting to receive messages.

Testing the Code

Run the previous code on your Raspberry Pi Pico. It will connect to the internet and subscribe to the MQTT topic.



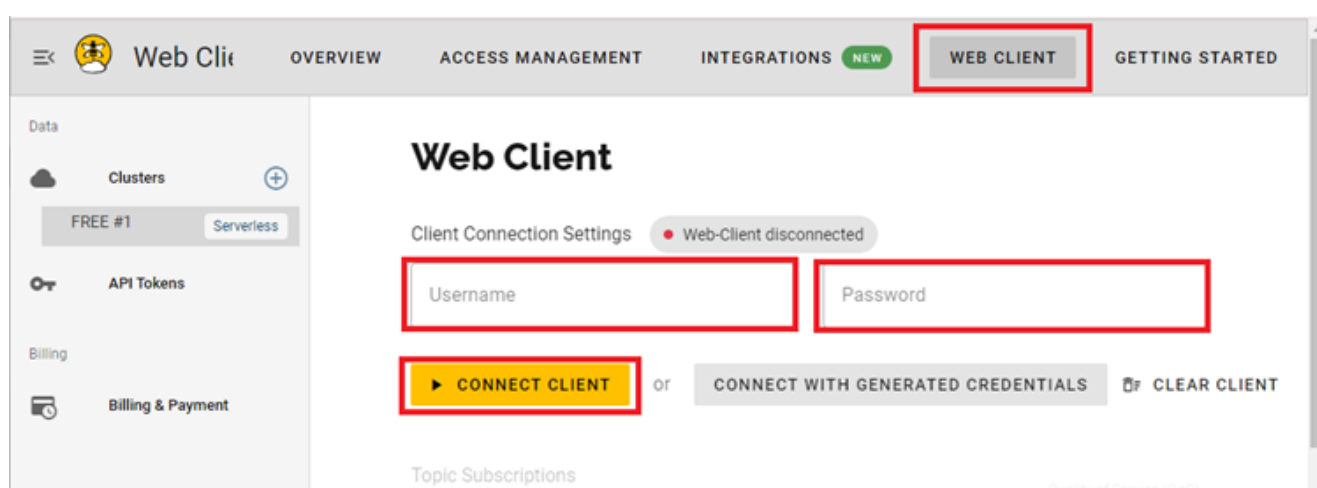
```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Connection successful!
IP address: 192.168.1.91
Subscribe to topic: pico/led
Loop running

MicroPython (Raspberry Pi Pico) • Board CDC @ COM12
```

Now, let's publish some messages on the `pico/led` topic to control the LED using HiveMQ Web Client.

Go to your HiveMQ cluster Web Client. Insert your MQTT broker username and password and click **Connect Client**.



Scroll down the page to the *Publish Message* section.

Insert `pico/led` on the **Topic Name** field, and type **ON** in the **Message** field.

Publish Message

Topic Name

pico/led

Quality of Service...

0 - At ... ▼

Message

ON

➤ PUBLISH

REMOVE ALL

i

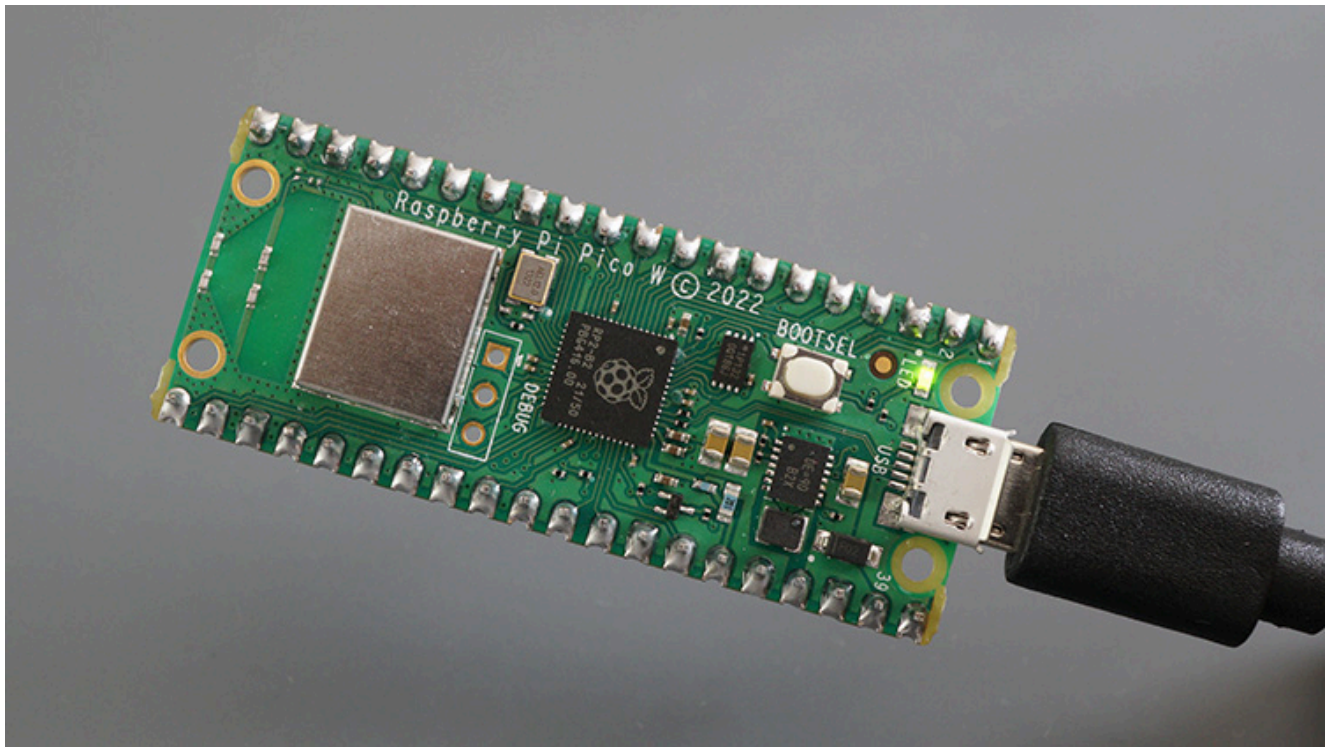
The client received no messages yet.

Finally, press the **Publish** button.

Your Raspberry Pi Pico will receive the message, and will turn the on-board LED on.

```
Shell x
Loop running
Loop running
Loop running
Loop running
Loop running
Loop running
Loop running
Loop running
Loop running
Received message on topic: b'pico/led'
Response: b'ON'
Turning LED ON
Loop running
```

MicroPython (Raspberry Pi Pico) • Board CDC @ COM7



Go back to HiveMQ Client and send an OFF message.

Publish Message

Topic Name

pico/led

Quality of Servic...
0 - At ... ▼

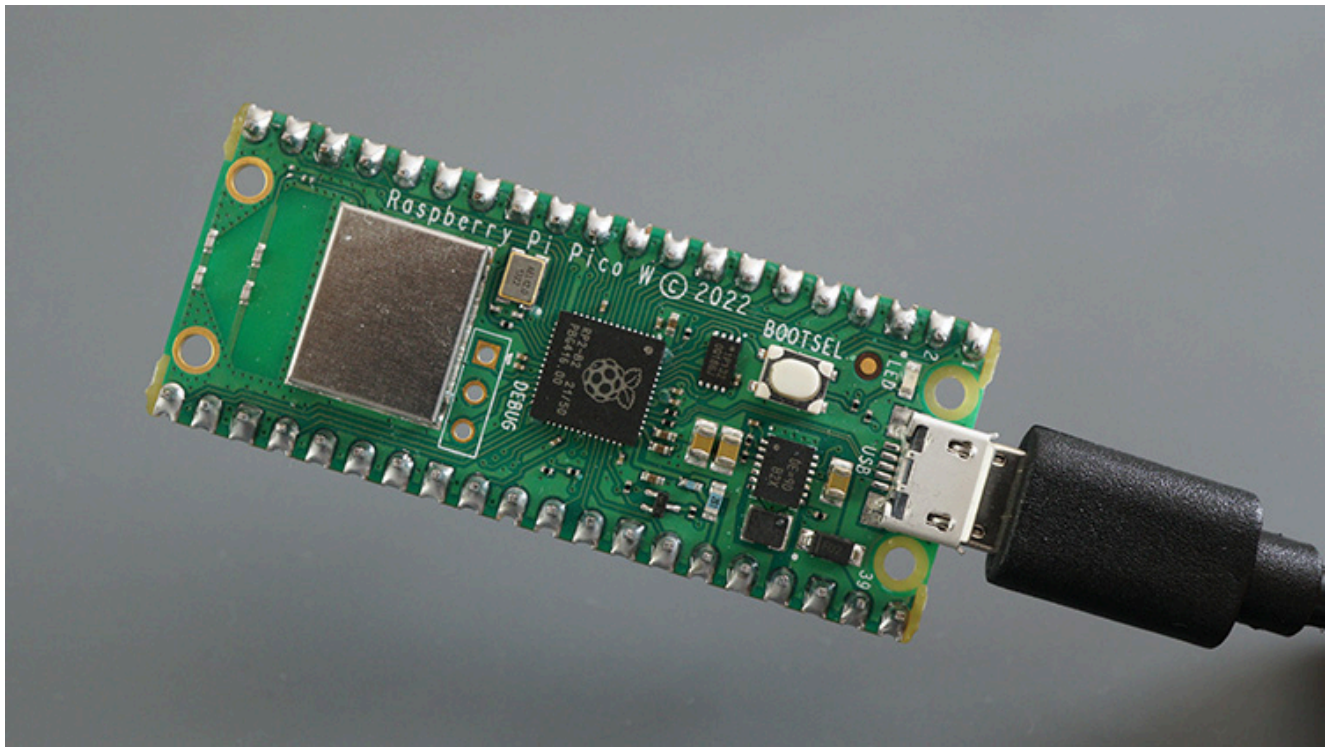
Message

OFF

➤ PUBLISH

REMOVE ALL

The Raspberry Pi Pico will receive the message and turn off the LED.



And that's it. You successfully subscribed to an MQTT topic and executed a different task according to the received message. Now, instead of HiveMQ Web Client, you can use any Home Automation or IoT platform with MQTT support that allows you to build a web interface with buttons or switches to send MQTT messages.

Wrapping Up

In this guide, you've learned all the essential concepts to start using MQTT with the Raspberry Pi Pico. We've set up an MQTT broker and published and subscribed to MQTT topics with the Pico.

We hope you've found this tutorial useful.

Learn more about the Raspberry Pi Pico with our eBook:

- [Learn Raspberry Pi Pico/Pico W with MicroPython](#)

We have other guides that you may also like:

- [Raspberry Pi Pico: Web Server \(MicroPython\)](#)
- [Raspberry Pi Pico: SSD1306 OLED Display \(MicroPython\)](#)
- [Raspberry Pi Pico and Pico W Pinout Guide: GPIOs Explained](#)
- [More Raspberry Pi Pico Tutorials and Guides](#)

If you want to use MQTT with the ESP32 or ESP8266 you can follow the next tutorials:

- [MicroPython – Getting Started with MQTT on ESP32/ESP8266](#)
- [MicroPython: MQTT – Publish BME280 Sensor Readings \(ESP32/ESP8266\)](#)
- [ESP32 MQTT – Publish and Subscribe with Arduino IDE](#)
- [ESP32 MQTT – Publish BME280 Sensor Readings \(Arduino IDE\)](#)
- [ESP8266 NodeMCU MQTT – Publish BME280 Sensor Readings \(Arduino IDE\)](#)

Thanks for reading.

**PCBWay**
PCB Fabrication & Assembly

ONLY \$5 for 10 PCBs

✓ 24-hour Build Time ✓ Quality Guaranteed

✓ Most Soldermask Colors:



[Order now](#)



www.pcbway.com



SMART HOME with Raspberry Pi, ESP32, ESP8266 [eBook]

Learn how to build a home automation system and we'll cover the following main subjects: Node-RED, Node-RED Dashboard, Raspberry Pi, ESP32, ESP8266, MQTT, and InfluxDB database [DOWNLOAD »](#)

Recommended Resources