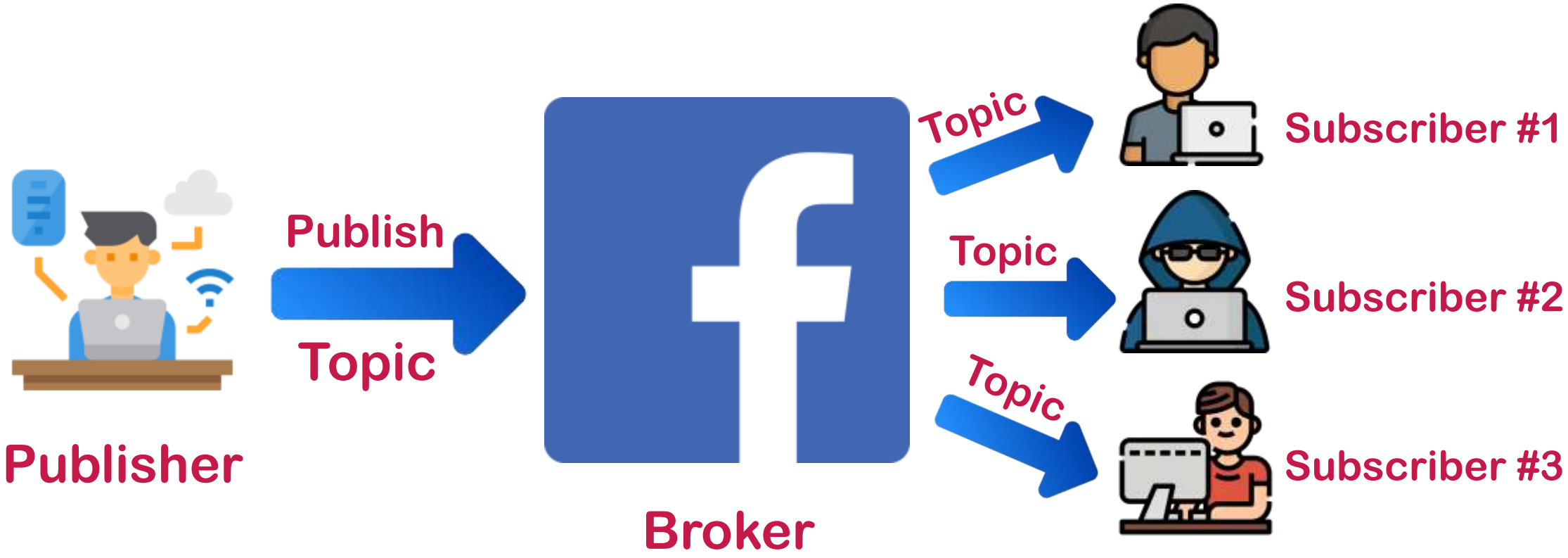
Internet of Things

MQTT for IoT Messaging

IoT Team, BFCAI

MQTT: Informal Introduction

- An admin (publisher) can publish a new post (topic) on a Facebook page.
- Facebook (broker) will send that topic to subscribers who liked the page.

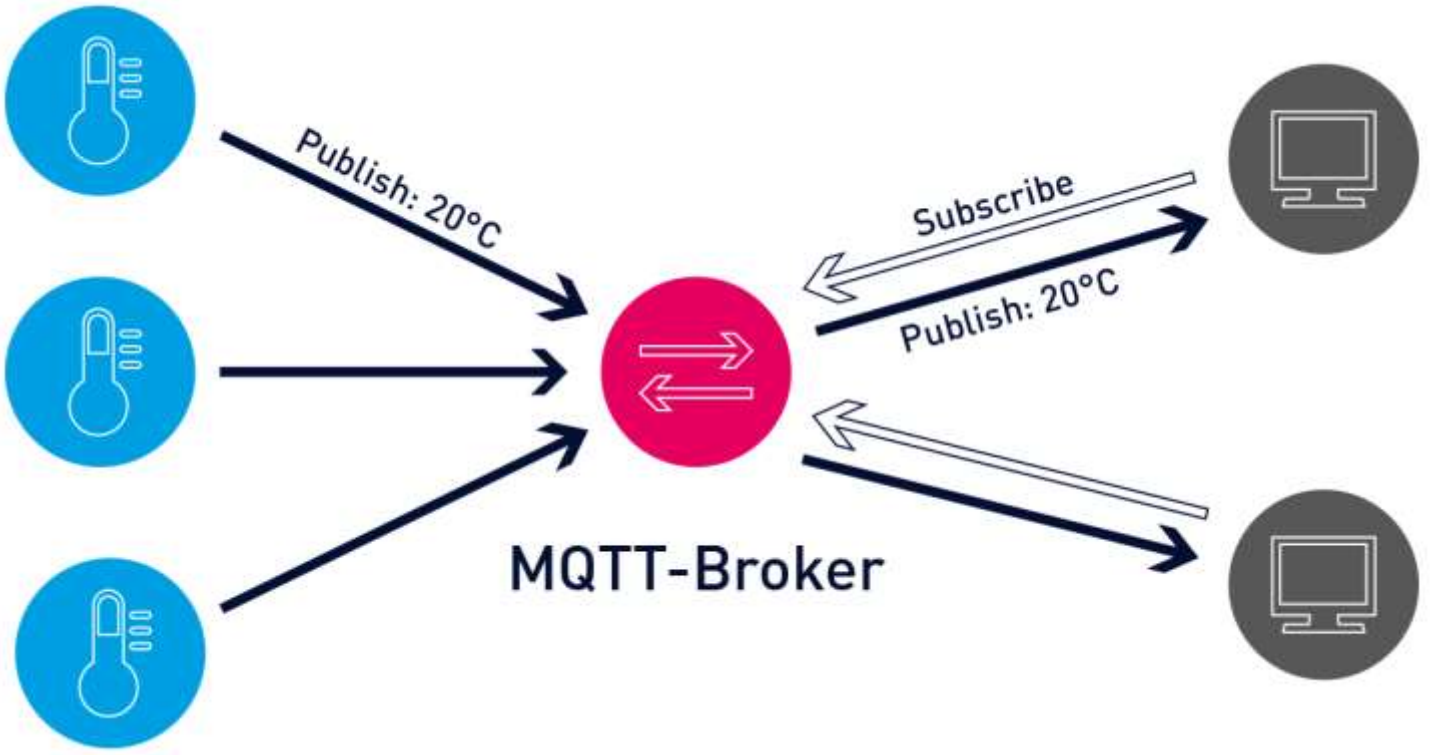


MQTT: Formal Introduction

- MQTT stands for Message Queuing Telemetry Transport.
- It is a messaging protocol designed for easy implementation.
- It is a lightweight communication protocol with minimal packet overhead.
- It is generally used for communication between IoT devices.
- MQTT is designed especially for the Internet of things (IoT).
- MQTT is more and more becoming the standard messaging protocol for IoT messaging.
- MQTT was developed by IBM in 1999.
- MQTT is a publish/subscribe protocol.

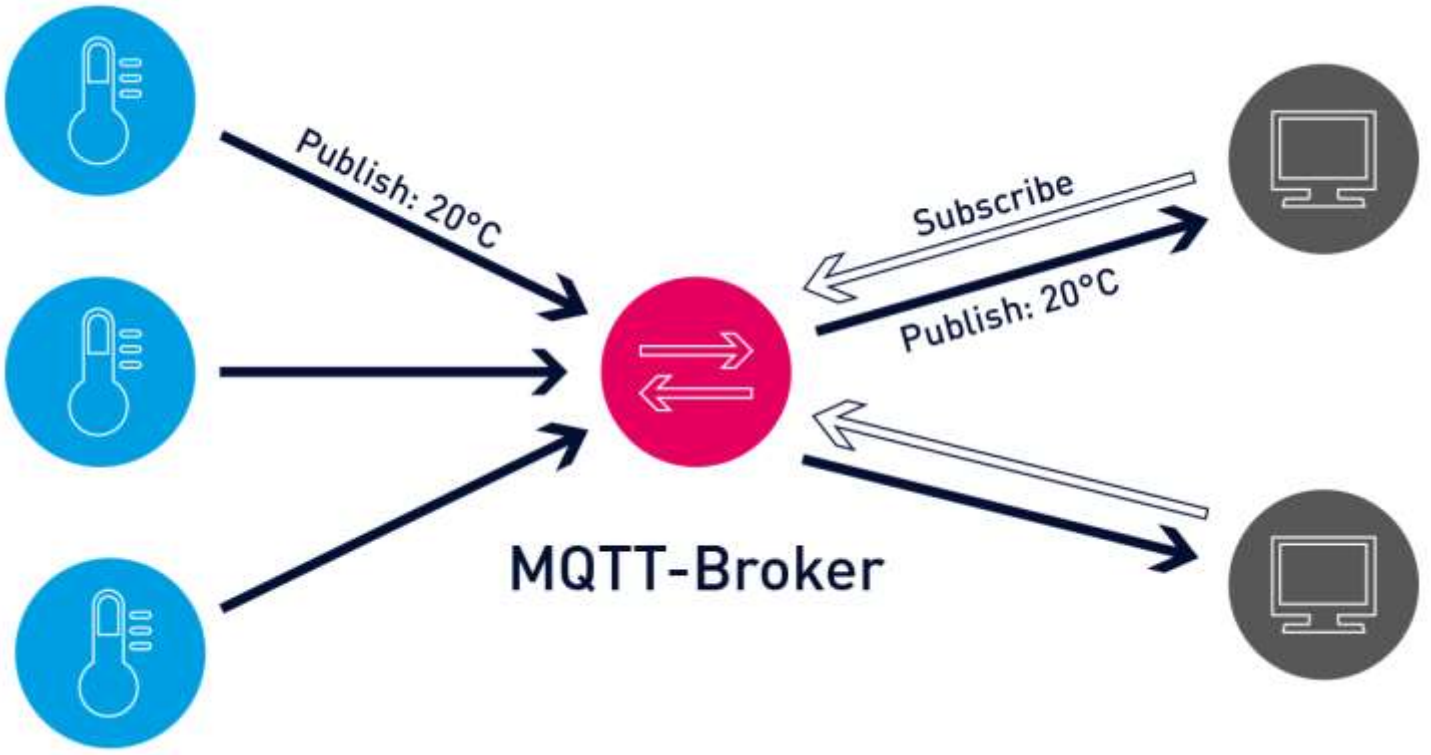
MQTT: Broker

- In MQTT, the clients (such as sensors, machines, and applications) do not directly communicate with each other but via a broker.
- Broker is a intermediary device connects various publishers and subscribers by managing and routing the data.



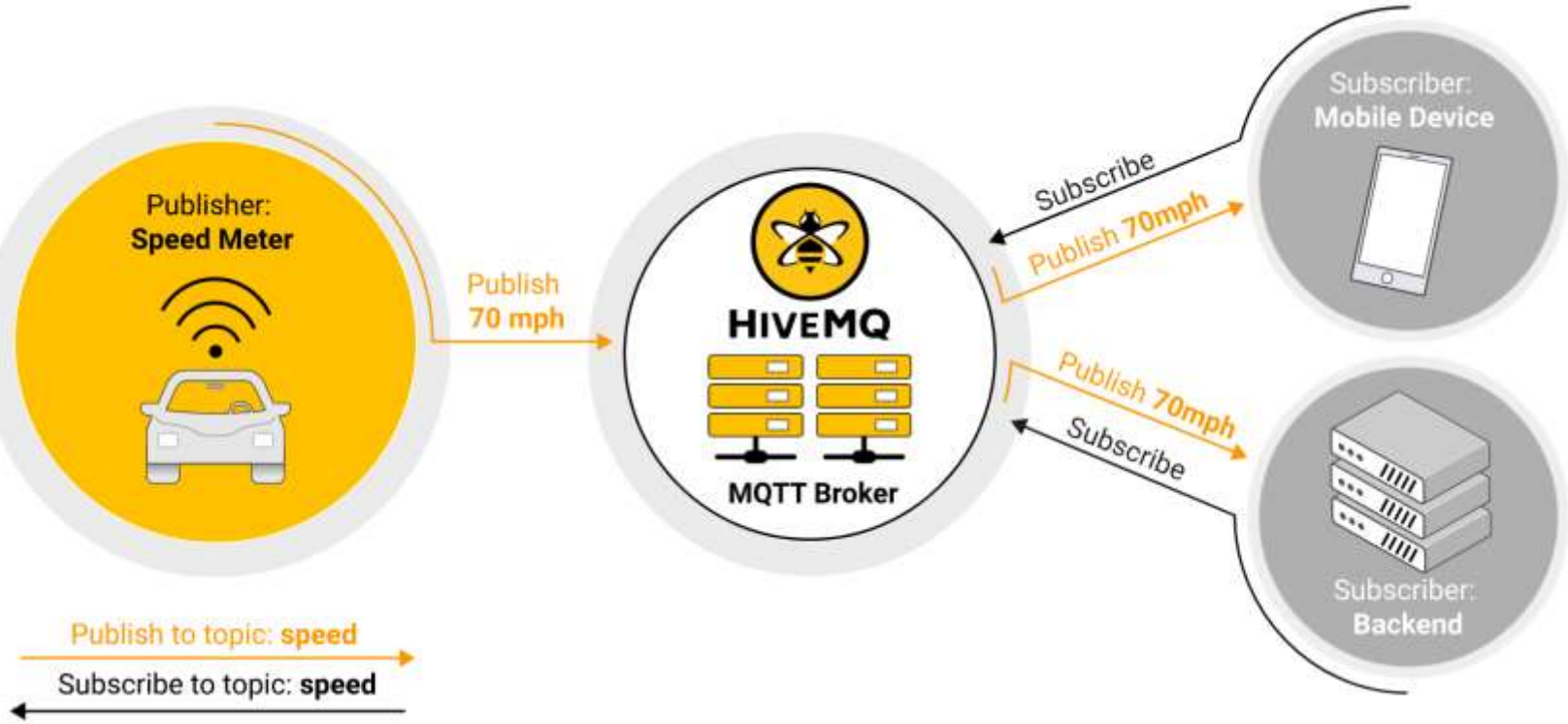
MQTT: Broker

- And just as functioning of the heart is critical for the human body, a reliable and performant MQTT broker is critical for IoT operations.
- The MQTT broker receives the data from the senders, filters the data packets, and forwards them to the receiving clients.



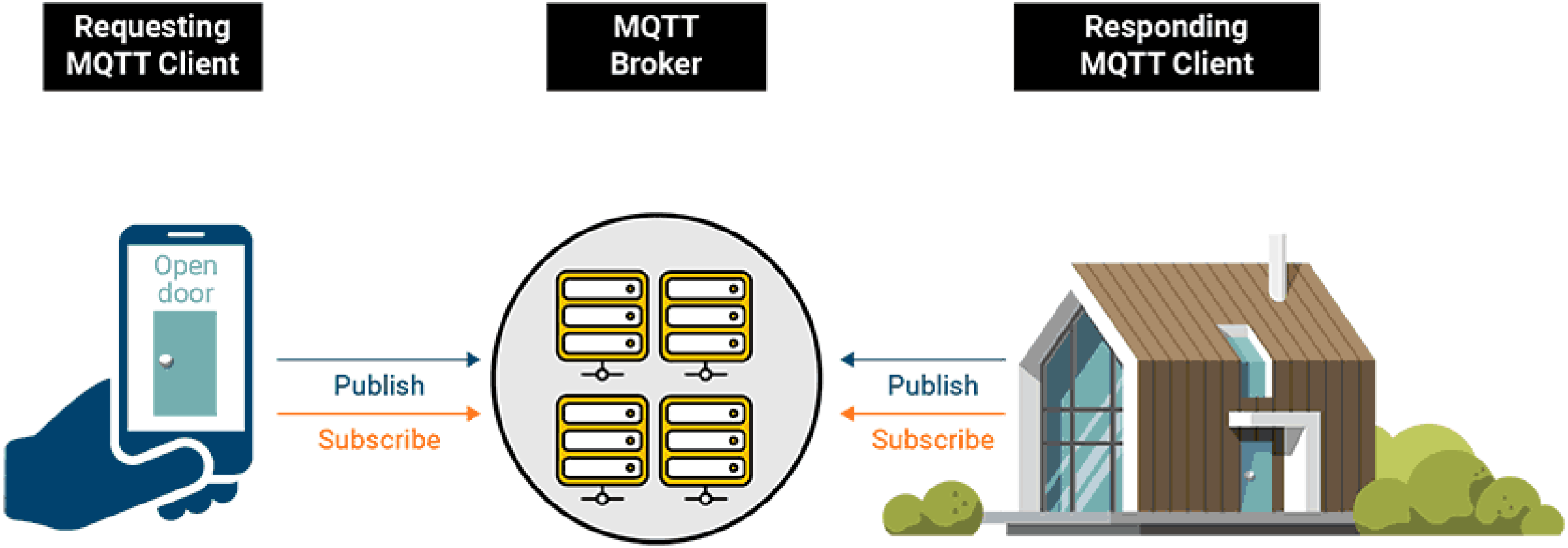
MQTT: Publishers & Subscribers

- Clients sending data are called **publishers**.
- Clients who receive data are called **subscribers**.
- In a publish and subscribe system, a device can publish a message on a topic, or it can be subscribed to a particular topic to receive messages



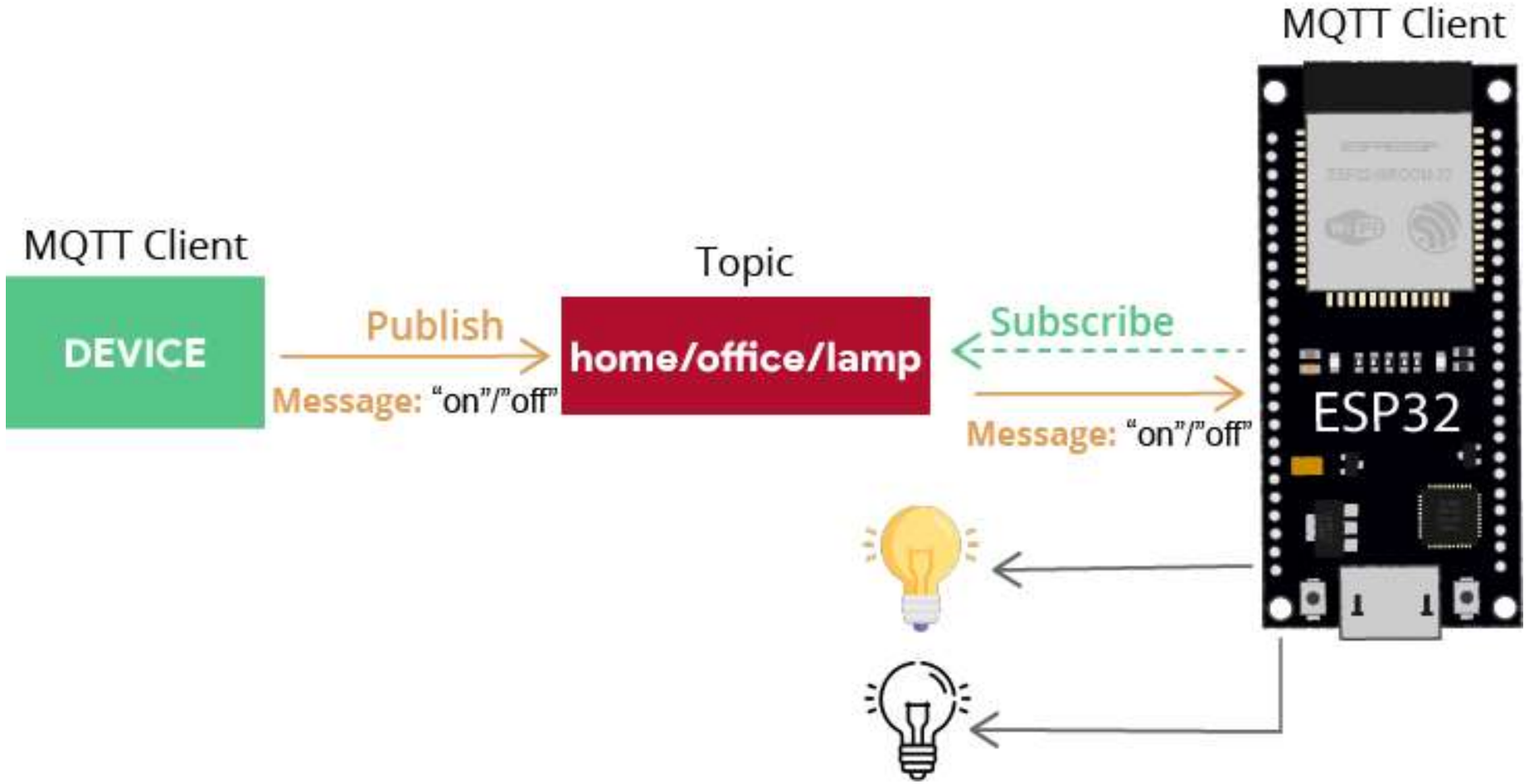
MQTT: Publishers & Subscribers

- An MQTT system enables receiving clients to become publishers as well.



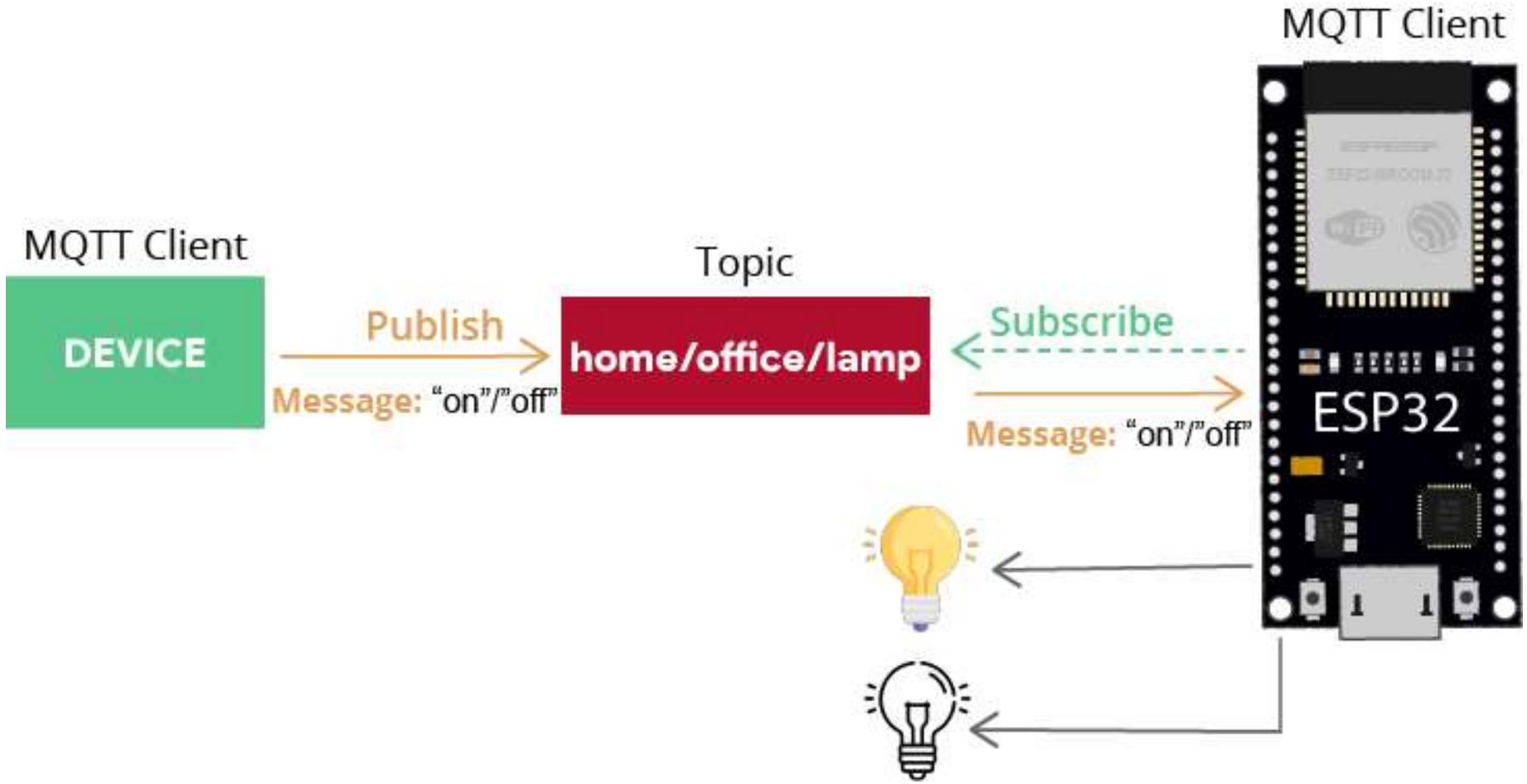
MQTT: Messages

- Messages are the data that you want to exchange between your devices.
- For example, a message can be a command or data like sensor readings.



MQTT: Topics

- A topic is the way you register interest for incoming messages or how you specify where you want to publish the message.



MQTT: Functionality

- First, the publisher sends the data collected to the broker on a particular topic, which is similar to a channel for data transmission.
- Please note that a topic can have several subtopics too.
- For example, in an application where you send the temperature data from a sensor connected to your fridge, the topic will look something like this:

`Kitchen/Fridge`

- The main topic is the `kitchen`, and the `Fridge` is the subtopic.
- The message will be Temperature:14 on the given topic.

MQTT: Functionality

- The subscribers listen to the topic.
- So, if the subscriber is listening to the `Kitchen` topic, it will have access to all the subtopics that are a part of this topic.

`Kitchen/Fridge`

- The primary function of the broker is to manage all the available topics and route the information according to the type of client, namely publishers and subscribers.
- Note that both the publishers and subscribers are referred to as clients.
- A client can be a publisher, subscriber, or both.

MQTT: Air Quality Monitoring System

- **Publishers:** Devices or machines are responsible for sending the collected data to the brokers.

  If you have an air quality monitoring system that monitors the CO_2 levels in the air every 30 seconds, the device will be set to publish the CO_2 concentration values every 30 seconds.
- **Subscribers:** Devices receive the requested sensor data from the brokers.

  An air purifier can be a subscriber of our air quality monitoring system.

  It receives the CO_2 concentration values every 30 seconds, and when it crosses a threshold value, the purifier automatically turns on.
- **Broker:** This intermediary device connects various publishers and subscribers by managing and routing the data.

MQTT: Features

Lightweight and Efficient

- MQTT clients are tiny, and they require minimal resources to operate.
- So, even microcontrollers such as ESP8266 can be used as a client as long as they have an active connection to a network.

Bidirectional Communication Protocol

- This means a device can be a publisher and a subscriber at the same time.
- This also allows easy broadcasting of messages to several devices at once.

MQTT: Features

Highly Secure

- MQTT makes it easy to encrypt messages.
- The standard unsecured port is 1883.
- The default secured MQTT broker port is 8883.
- The use of ACLs (Access Control Lists) allows restriction of subscriptions and publishing of clients.

MQTT: Features

Highly Scalable

- There is no worry about maintaining clients' addresses or IDs.
- It is effortless to expand the MQTT network.
- The only things required are the broker's IP address and the topic name.

Reliability

- MQTT is highly reliable when it comes to message delivery.
- MQTT comes with three predefined quality of service:

  QoS 0: At most once

  QoS 1: At least once

  QoS 2: Exactly once

MQTT: Quality of Service (QoS)

- MQTT provides three Quality of Service (QoS) levels for individual message delivery.
- MQTT QoS is an agreement between the message sender and receiver that defines the level of delivery guarantee for a specific message.

QoS Level	Meaning	# Messages Delivered
Level 0	The message will be delivered at most once, but maybe not at all.	0 or 1
Level 1	The message will be delivered at least once, but perhaps more.	1 or more
Level 2	The message will be delivered exactly once.	1

MQTT: QoS Level 0

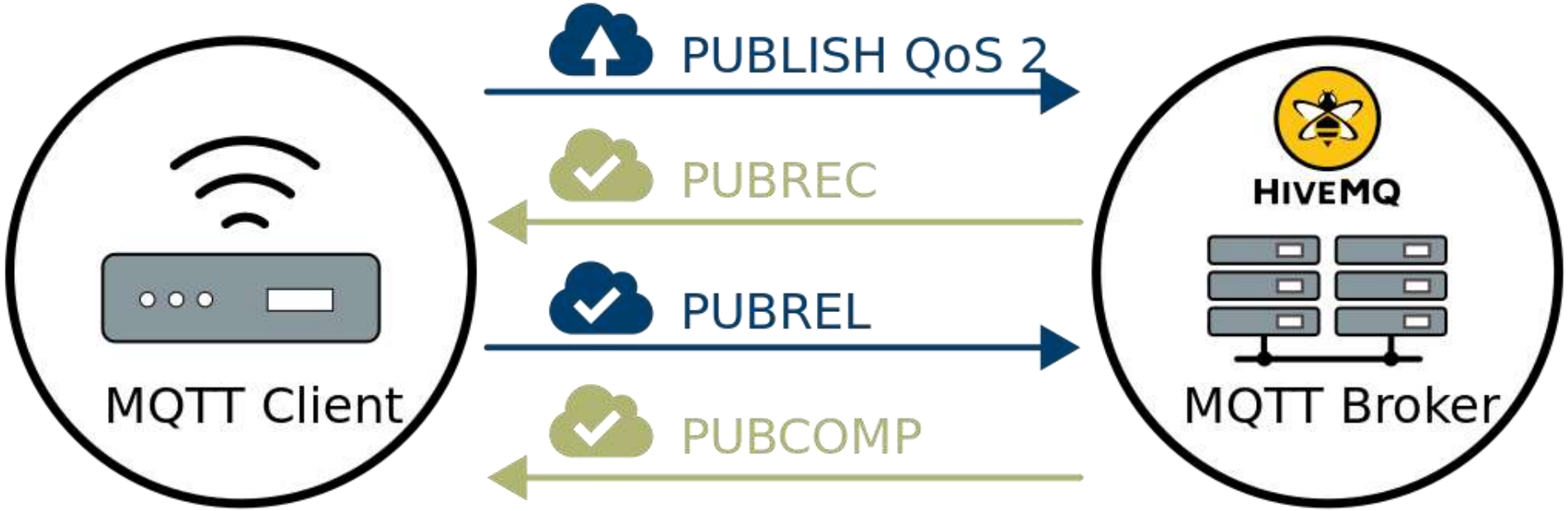
- In QoS Level 0 (Fire and Forget Level), messages are sent without any confirmation from the receiver.
- This means it is technically possible for a message to get lost, given an unreliable connection.

MQTT: QoS Level 1

- In QoS Level 1, the receiver must send a confirmation (**PUBACK**) to let the sender know that the message was received.
- However, it is possible that the receiver gets a message multiple times.
- This QoS level ensures that a message makes it from sender to receiver but does not ensure that it is received exactly once.

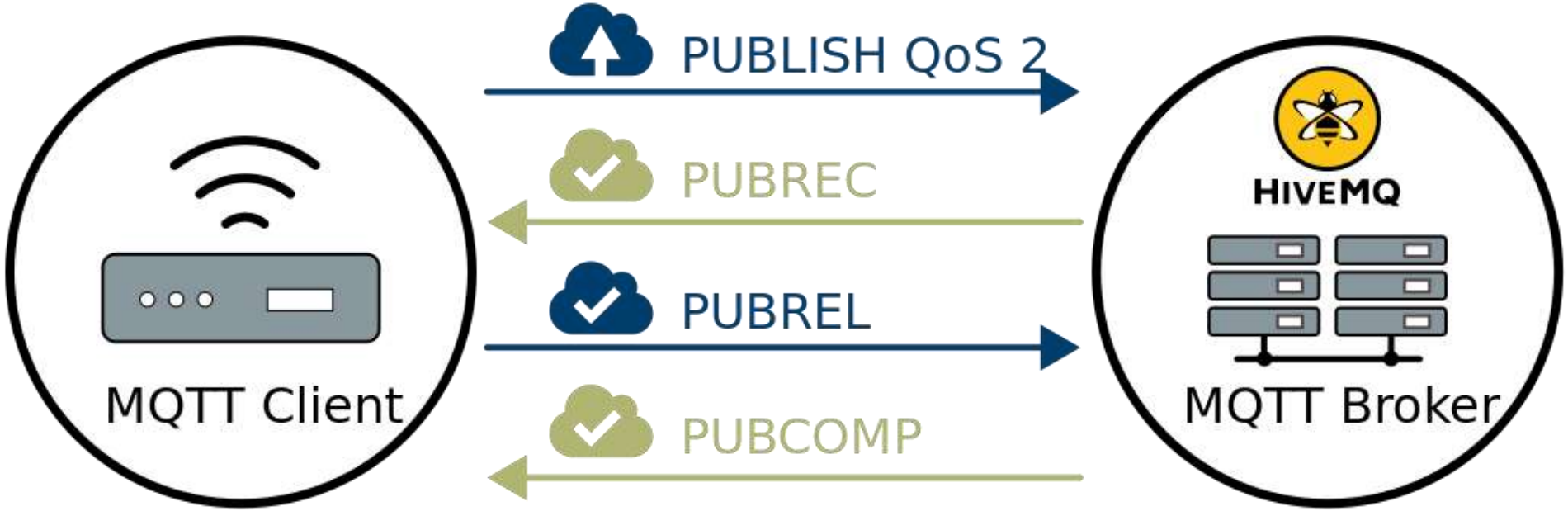
MQTT: QoS Level 2

- QoS level 2 uses a four-step communication process to ensure a message is sent exactly once only.
- QoS 2 offers the highest level of service in MQTT, ensuring that each message is delivered exactly once to the intended receiver.
- It involves a four-step handshake between the sender and receiver.



MQTT: QoS Level 2 – Explanation

- When a receiver gets a QoS 2 PUBLISH packet from a sender, it replies to the sender with a **PUBREC** packet that acknowledges the publisher.
- If the sender does not get a **PUBREC** packet from receiver, it sends the packet again with a duplicate flag until it receives an acknowledgement.

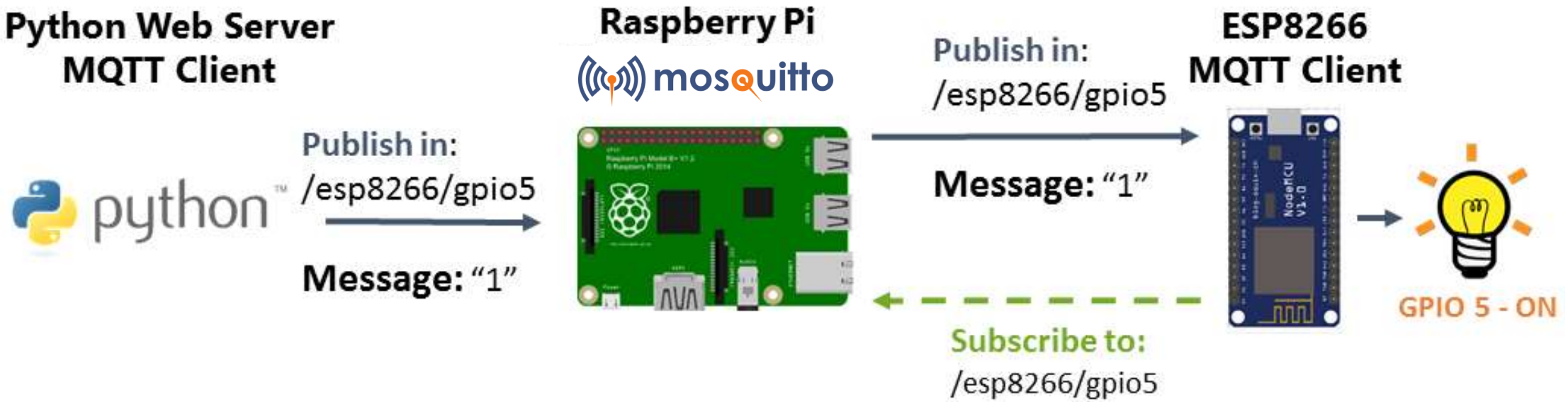


Mosquitto Broker

- Mosquitto is a popular and open-source message broker that implements the MQTT protocol.
- Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.
- The broker receives all messages from the clients, filters the messages, determines who is subscribed to the topic, and then sends the message to these subscribed clients.

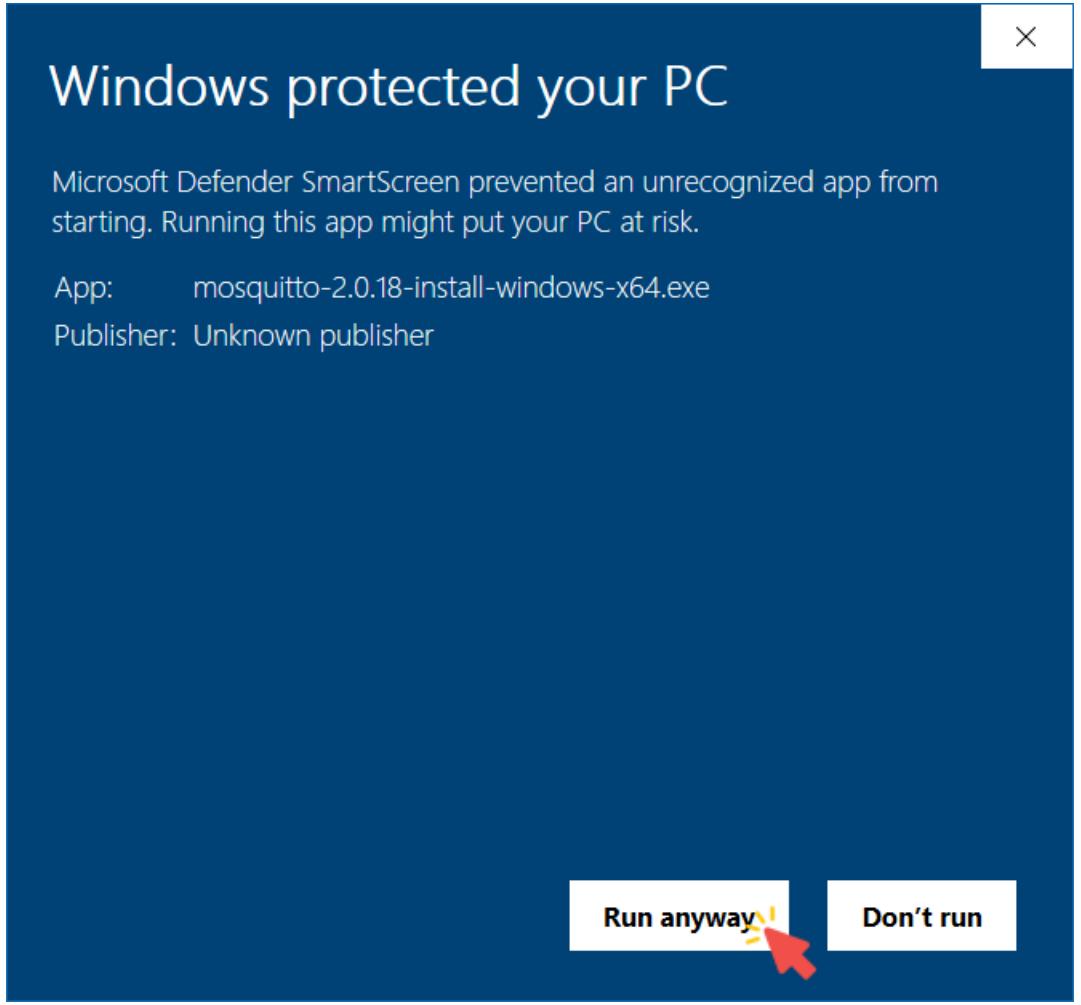
Mosquitto Broker

- In the world of IoT, where devices need to communicate efficiently, Mosquitto's ability to handle multiple connections and deliver messages in real-time is very useful.
- Mosquitto MQTT can run on various operating systems, including Linux, Windows, macOS, and even on Raspberry Pi.



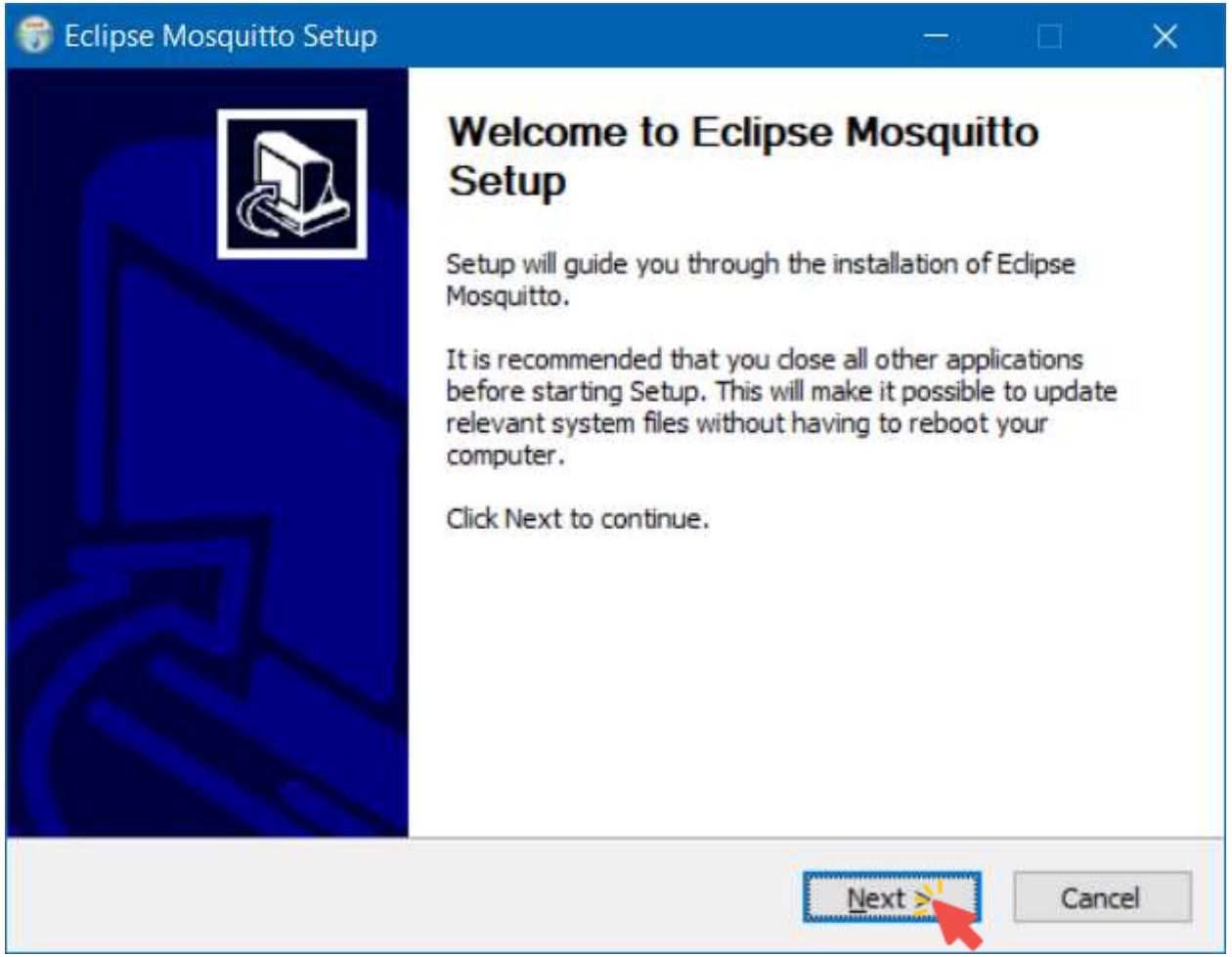
Mosquitto Broker: Installation on Windows

- If this message appears, click "Run anyway".



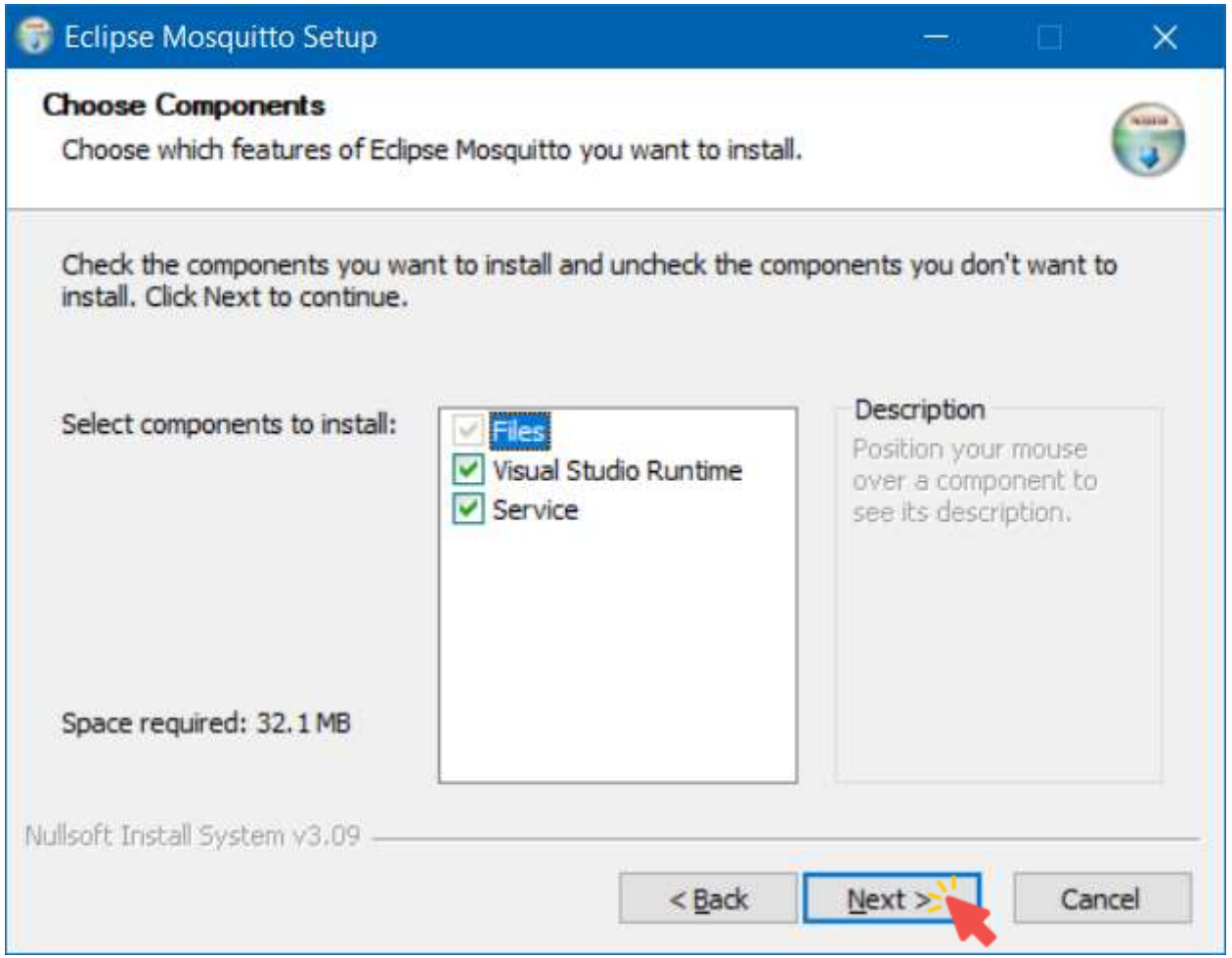
Mosquitto Broker: Installation on Windows

- Click Next.



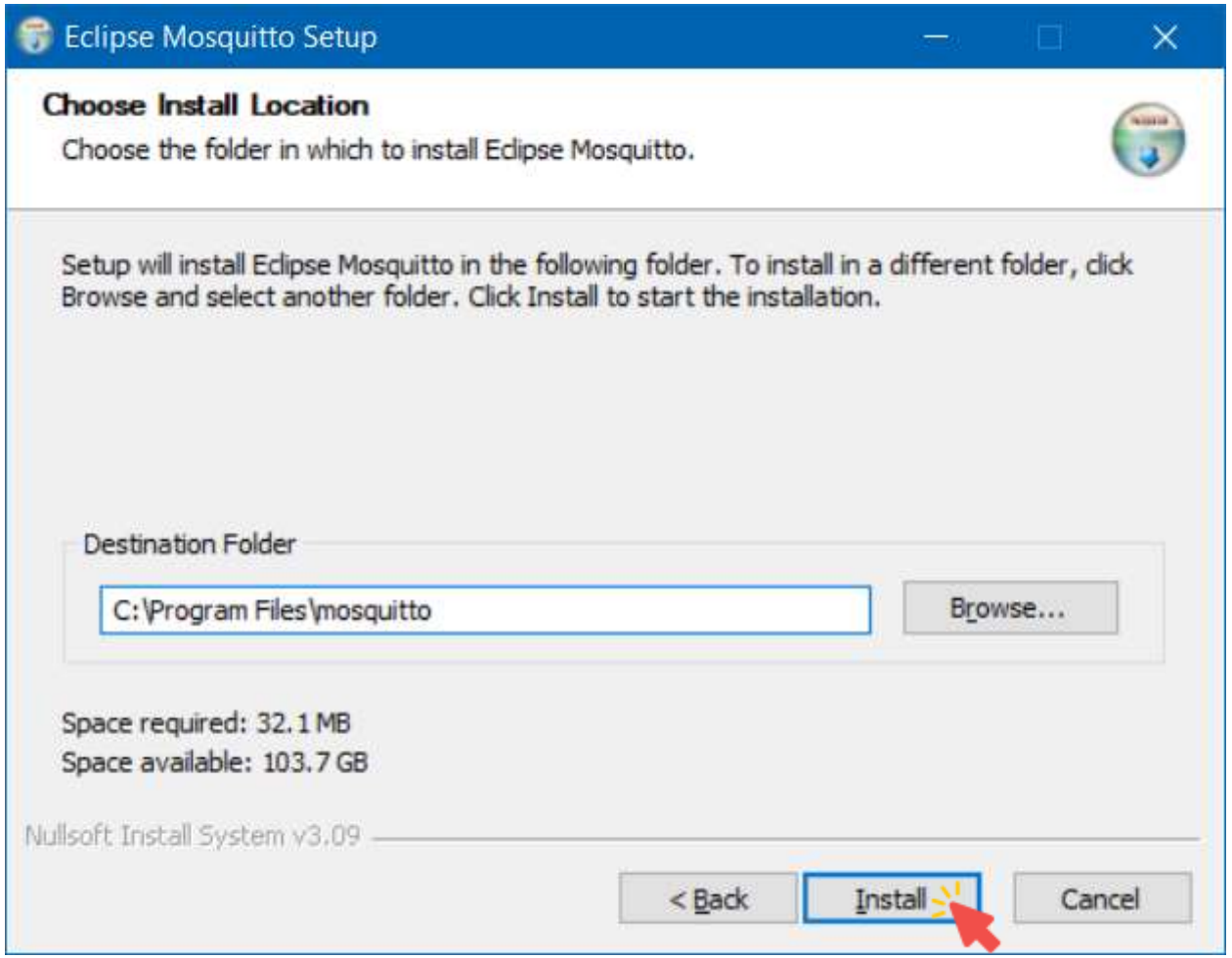
Mosquitto Broker: Installation on Windows

- Click Next.



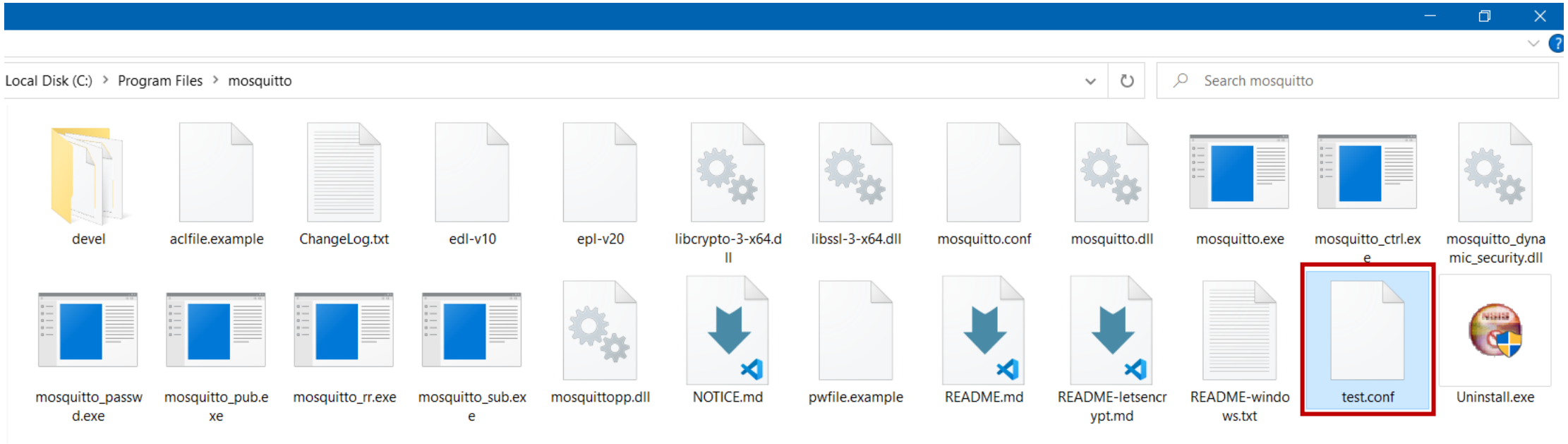
Mosquitto Broker: Installation on Windows

- Choose the installation path, and click Install.



Mosquitto Broker: Unauthenticated Access Configurations

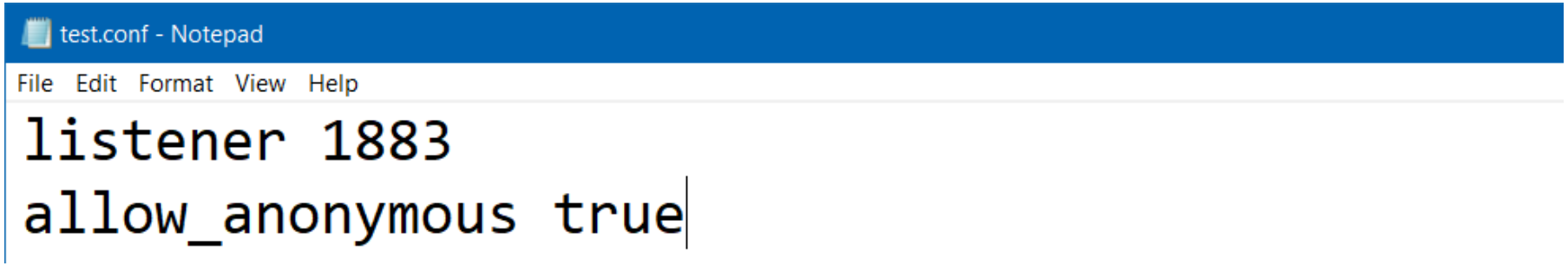
- Create a text file named `test.conf` under the Mosquitto folder (`C:\Program Files\mosquitto`).



Mosquitto Broker: Unauthenticated Access Configurations

- Open the created file, and write the following commands:

```
listener 1883
allow_anonymous true
```

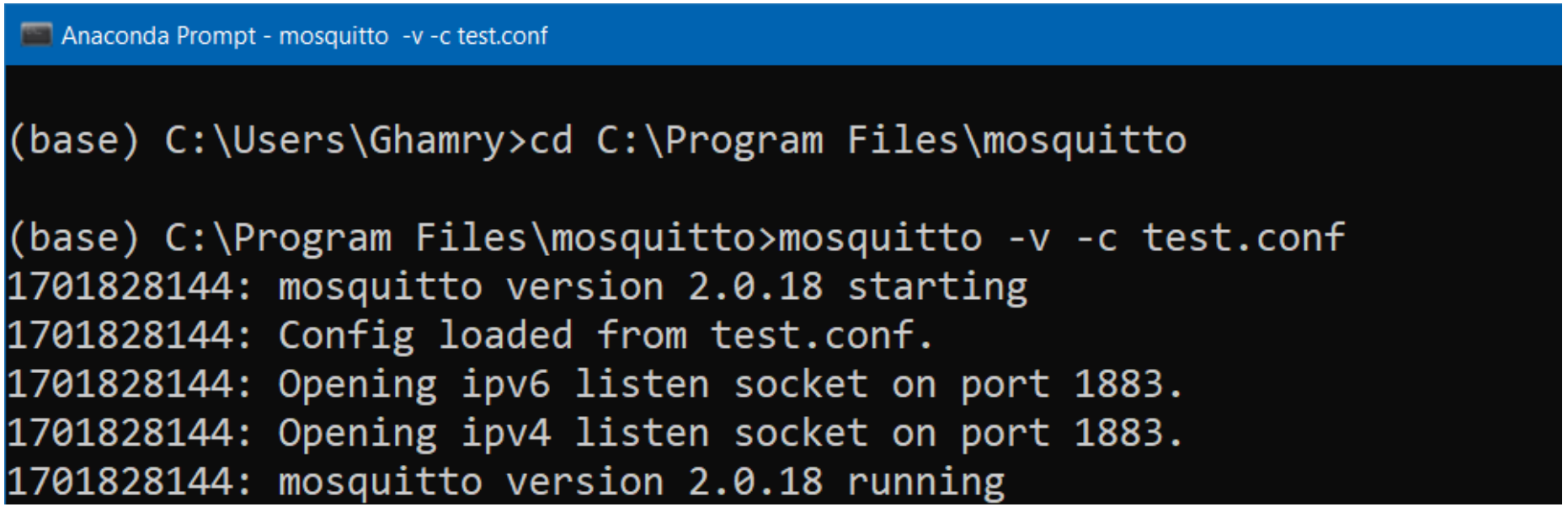


- MQTT clients typically connect to the broker on port 1883, which is the default port for unencrypted MQTT communication.
- When `allow_anonymous` is set to true, clients can connect without providing a username or password.

Mosquitto Broker: Starting the Broker

- Open CMD window and write the following commands:

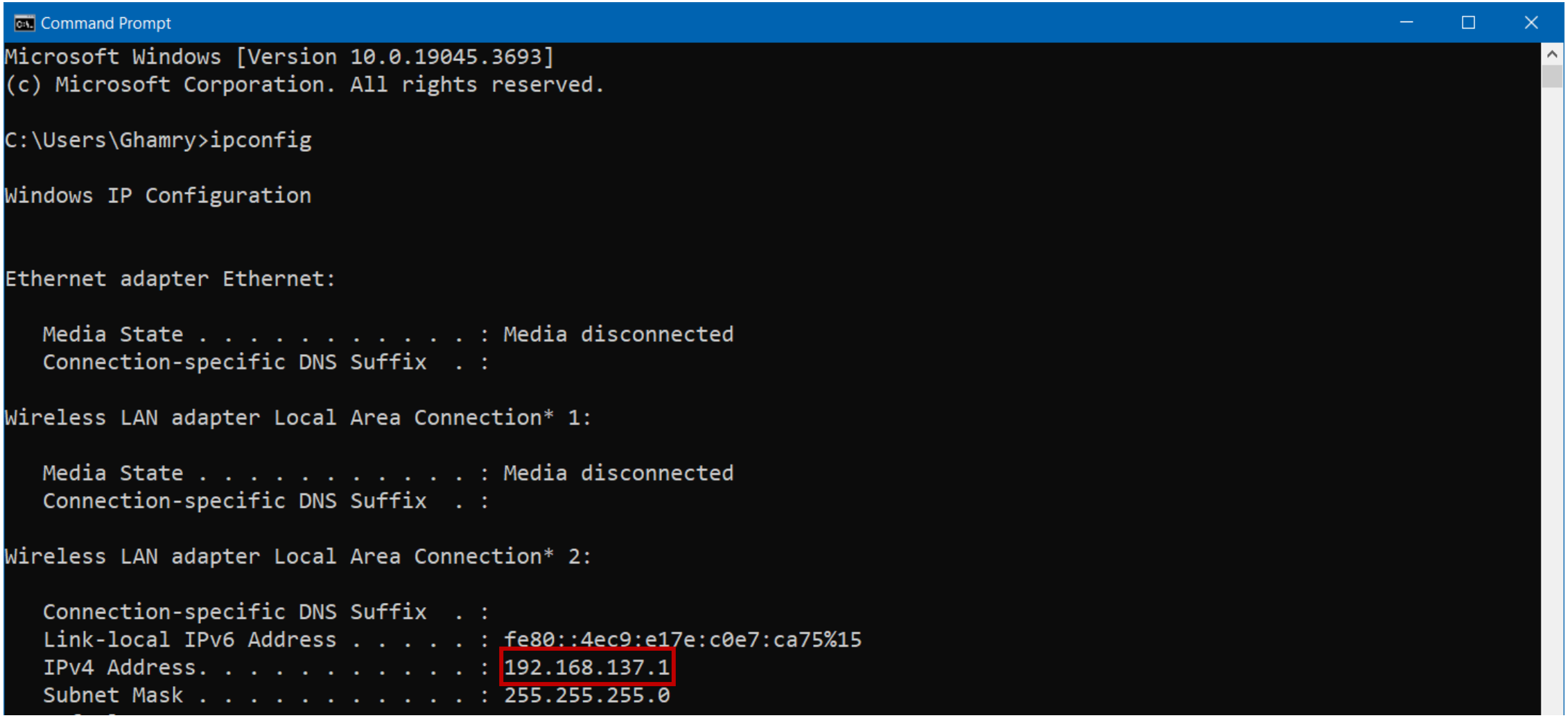
```
cd C:\Program Files\mosquitto
mosquitto -c test.conf -v
```



- `-c test.conf`: Specifies a configuration file for the Mosquitto broker.
- `-v`: Enables verbose mode to provide additional information and logging.

Mosquitto Broker: Getting Broker IP

- Open CMD window and write `ipconfig` to get the broker IP.



Paho MQTT Python Library

- The Paho Python Client provides a client class with support for MQTT.
- It provides a simple API for working with MQTT, allowing developers to easily integrate MQTT functionality into their Python applications.
- "Paho" means "communicate with everyone".
- You can install the Paho Python Client using the following `pip` command:

```
pip install paho-mqtt==1.6.1
```

Anaconda Prompt

```
(base) C:\Users\Ghamry>pip install paho-mqtt==1.6.1
Collecting paho-mqtt==1.6.1
  Downloading paho-mqtt-1.6.1.tar.gz (99 kB)
     ---------------------------------------- 99.4/99.4 kB 632.1 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Successfully built paho-mqtt
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.6.1
```

Paho MQTT Python Library: Simple Publisher and Subscriber

Python App
(Publisher)

Broker

Python App
(Subscriber)

Paho MQTT Python Library: Python Publisher App

```python
# Import the necessary modules
import paho.mqtt.client as mqtt
from time import sleep

# MQTT broker address
broker_ip = "192.168.137.1"

# MQTT broker port
port = 1883

# MQTT topic to which the publisher will publish messages
topic = "home/led"

# Quality of Service (QoS)
qos = 0

# Create an MQTT client instance with the name "publisher"
client = mqtt.Client("publisher")

# Connect to the MQTT broker using the specified IP address and port
client.connect(broker_ip, port)

# Infinite loop to continuously publish messages
while True:
    # Message to be published
    message = "Turn On"

    # Publish the message to the specified topic
    client.publish(topic, message, qos)

    # Print a message indicating that the message has been published
    print("Published message:", message)

    # Wait for 2 seconds before publishing the next message
    sleep(2)

# Disconnect from the MQTT broker
client.disconnect()
```

Paho MQTT Python Library: Python Subscriber App

```python
# Import the necessary modules
import paho.mqtt.client as mqtt

# MQTT broker address
broker_address = "192.168.137.1"

# MQTT broker port
port = 1883

# MQTT topic to which the subscriber will subscribe
topic = "home/led"

# Quality of Service (QoS)
qos = 0

# Callback function to handle incoming messages
def on_message(client, userdata, message):
    print("Received message:", message.payload.decode())

# Create an MQTT client instance with the name "subscriber"
client = mqtt.Client("subscriber")

# Connect to the MQTT broker using the specified IP address and port
client.connect(broker_address, port)

# Subscribe to the specified topic
client.subscribe(topic, qos)

# Set the callback function for incoming messages
client.on_message = on_message

# Start the MQTT client loop to receive messages
client.loop_forever()
```

Paho MQTT Python Library: Publisher & Subscriber – Output

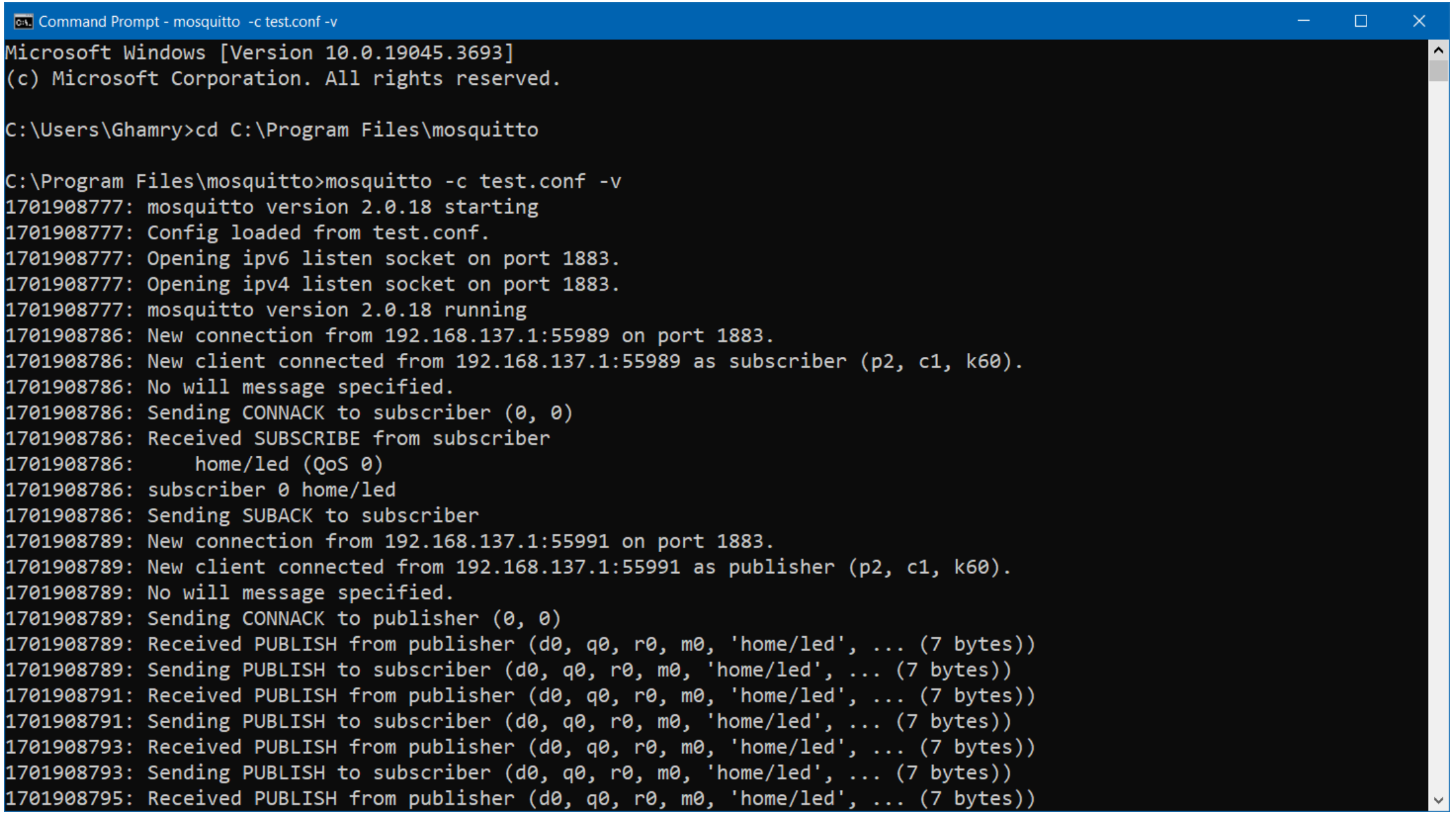
```
Published message: Turn On
Published message: Turn On
Published message: Turn On
Published message: Turn On
Published message: Turn On
Published message: Turn On
Published message: Turn On
```

Python Publisher App


```
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
```

Python Subscriber App

Paho MQTT Python Library: Mosquitto – Output



PubSubClient Library

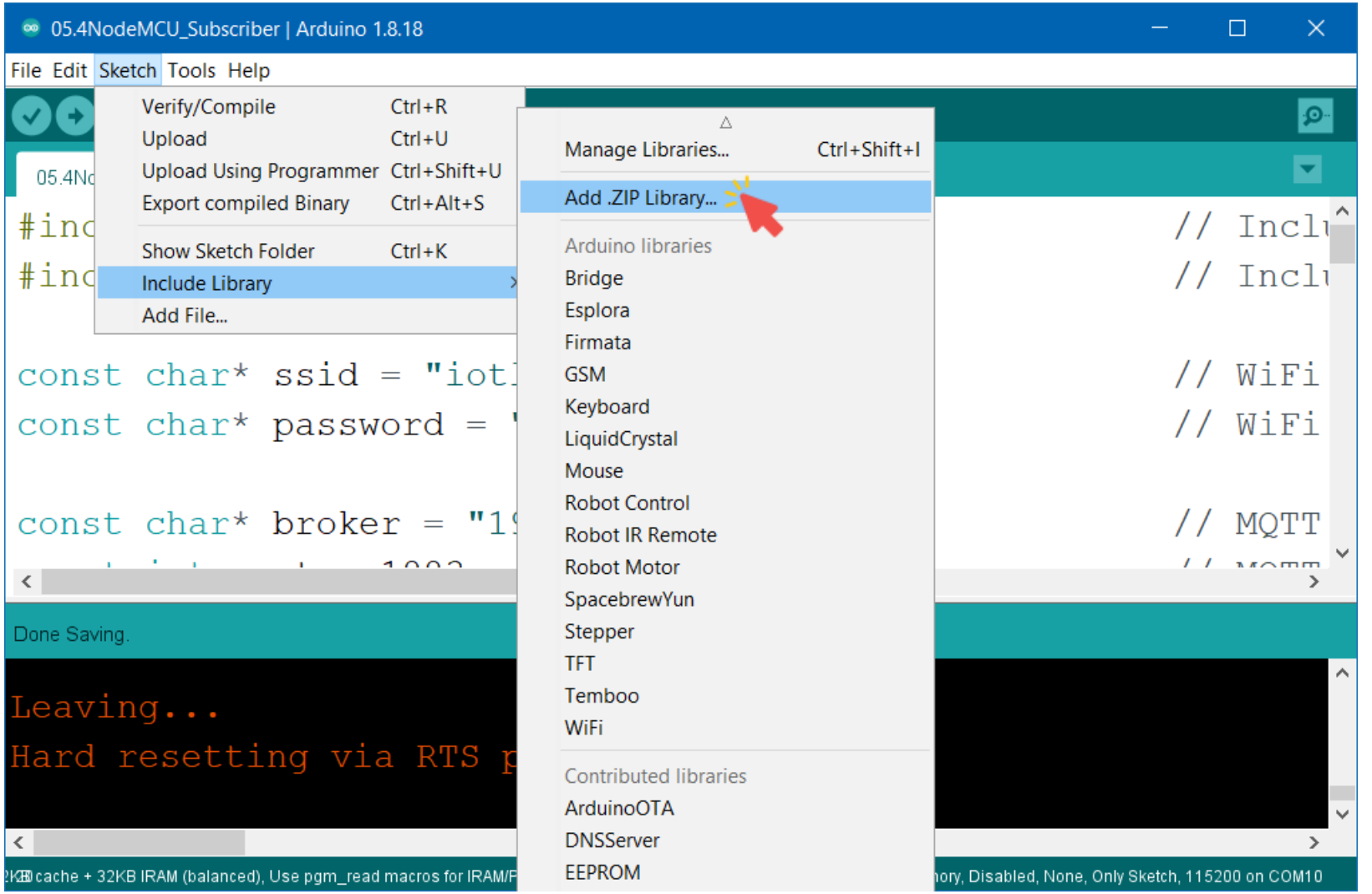
- The `PubSubClient` library provides a client for doing simple publish/subscribe messaging with a server that supports MQTT.
- The library can be installed into the Arduino IDE.

NodeMCU

ESP32

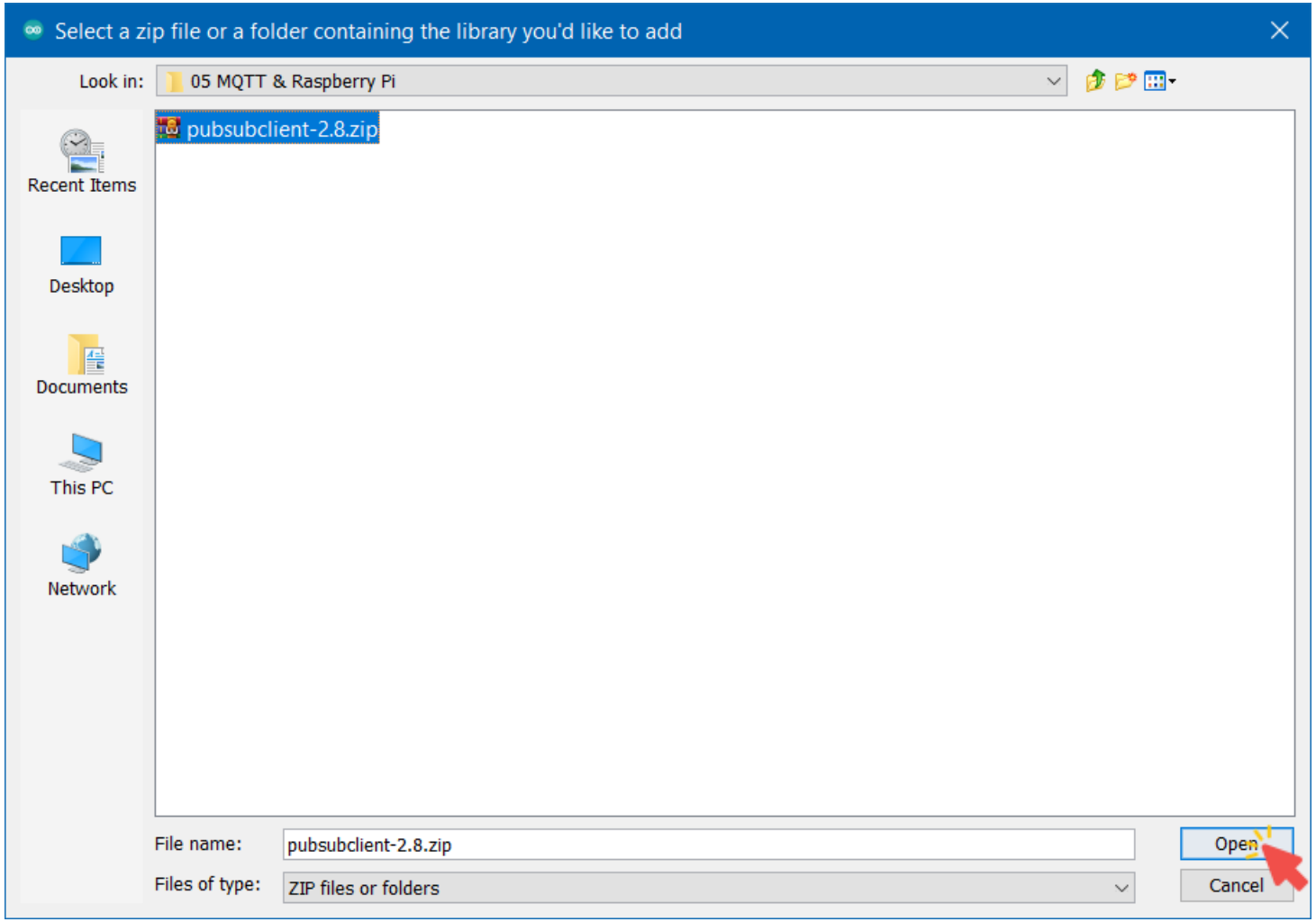
PubSubClient Library: Installation on Arduino IDE

- Open Sketch → Include Library → Add .ZIP Library.

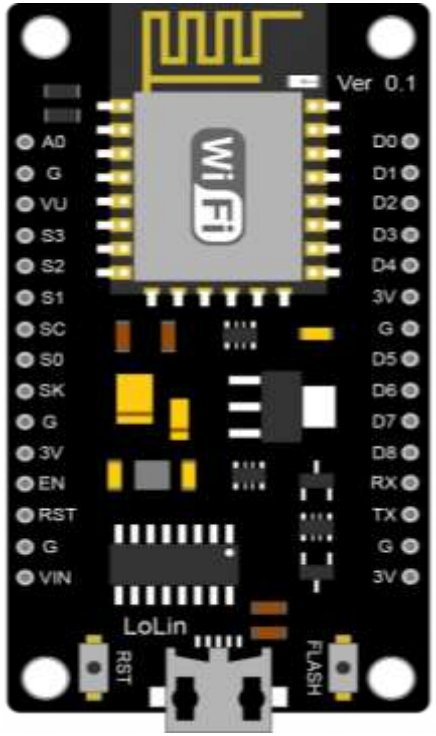


PubSubClient Library: Installation on Arduino IDE

- Choose the library file named `pubsubclient-2.8.zip`, and click Open.



NodeMCU as Publisher



NodeMCU (Publisher)

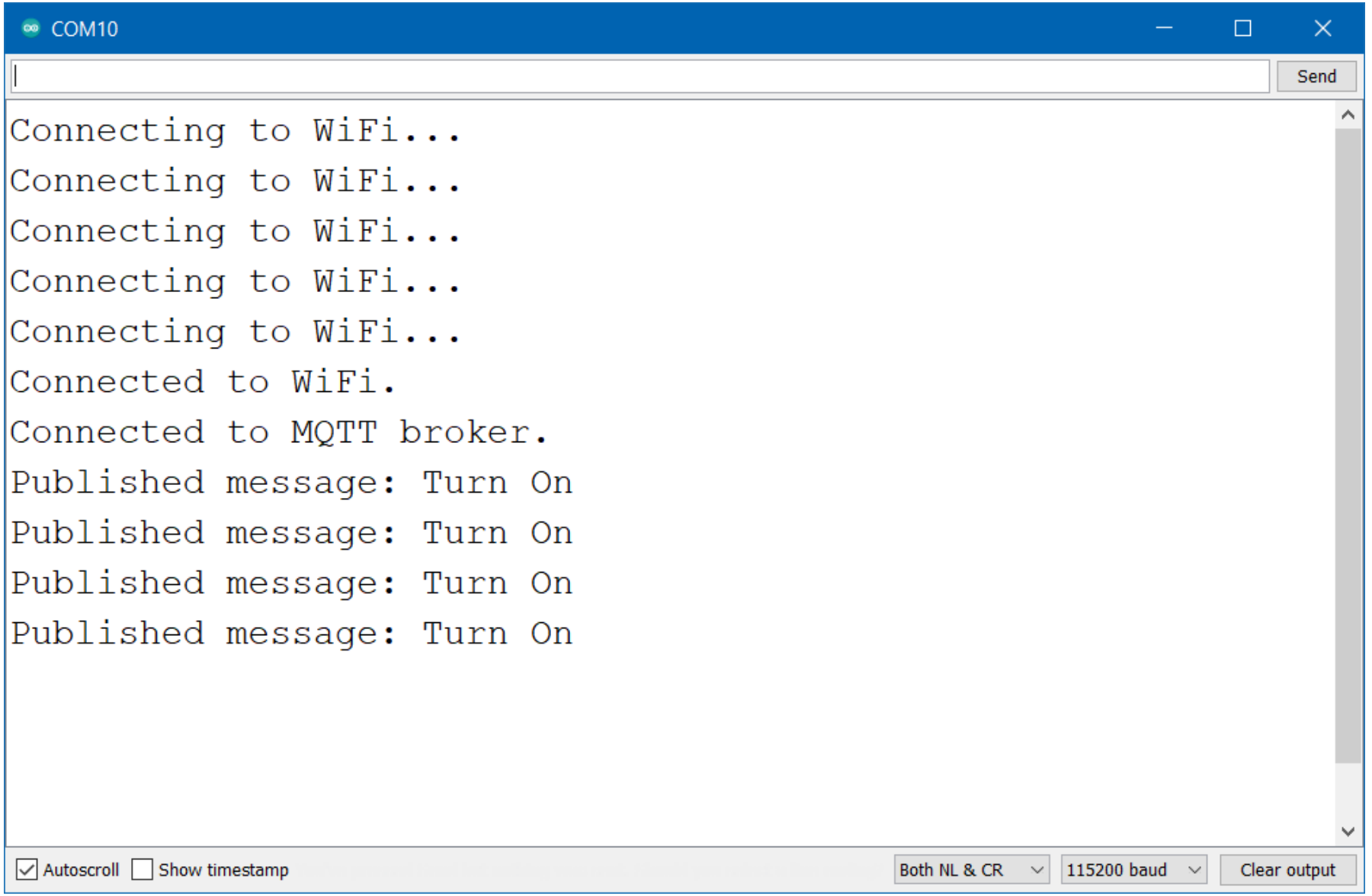
Broker

Python App (Subscriber)

NodeMCU as Publisher: Code

```
#include <ESP8266WiFi.h>                          // Include the WiFi library
#include <PubSubClient.h>                         // Include the MQTT library

const char* ssid = "iotlab";                      // WiFi SSID
const char* password = "hostiotlab";              // WiFi Password

const char* broker = "192.168.137.1";             // MQTT broker address
const int port = 1883;                            // MQTT broker port
const char* topic = "home/led";                   // MQTT topic name

WiFiClient espClient;                             // Create an object of the WiFiClient class
PubSubClient client(espClient);                   // Create an MQTT client instance

void setup() {
  Serial.begin(115200);                           // Initialize serial communication at baudrate of 115200

  WiFi.begin(ssid, password);                     // Attempt to connect to the Wi-Fi network
  while (WiFi.status() != WL_CONNECTED) {         // Wait until the NodeMCU is successfully connected
    delay(1000);                                  // Wait 1 second before rechecking Wi-Fi connection status
    Serial.println("Connecting to WiFi...");      // A message indicating an attempt to connect to Wi-Fi
  }
  Serial.println("Connected to WiFi.");           // A message indicating a successful connection

  client.setServer(broker, port);                 // Connect to the MQTT broker
  client.connect("NodeMCU_Publisher");            // Connect to MQTT broker with the name "NodeMCU_Publisher"
  Serial.println("Connected to MQTT broker.");    // Successful connection to MQTT broker
}

void loop() {
  const char* message = "Turn On";                // The message to be published
  client.publish(topic, message);                 // Publish the message to the specified topic
  Serial.print("Published message: ");            // A message prefix
  Serial.println(message);                        // Print the published message

  delay(1000);                                    // Short delay to avoid rapid publishing
}
```

NodeMCU as Publisher: NodeMCU Output



NodeMCU as Publisher: Python Output

jupyter 05.2Python_Subscriber Last Checkpoint: 4 months ago

File Edit View Run Kernel Settings Help

Code

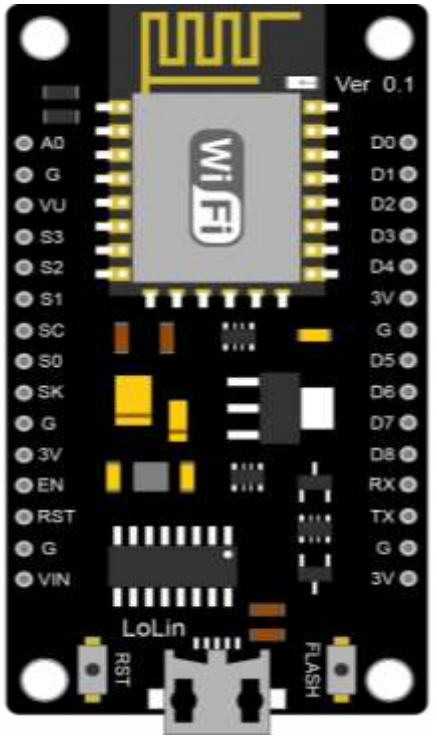
```
# Subscribe to the specified topic
client.subscribe(topic, qos)

# Set the callback function for incoming messages
client.on_message = on_message

# Start the MQTT client loop to receive messages
client.loop_forever()
```

```
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
Received message: Turn On
```

NodeMCU as Subscriber

Python App (Publisher)

Broker



NodeMCU (Subscriber)

NodeMCU as Subscriber: Code

```
#include <ESP8266WiFi.h>                                  // Include the WiFi library
#include <PubSubClient.h>                                 // Include the MQTT library

const char* ssid = "iotlab";                              // WiFi SSID
const char* password = "hostiotlab";                      // WiFi Password

const char* broker = "192.168.137.1";                     // MQTT broker address
const int port = 1883;                                    // MQTT broker port
const char* topic = "home/led";                           // MQTT topic name

WiFiClient espClient;                                     // Create an object of the WiFiClient class
PubSubClient client(espClient);                           // Create an MQTT client instance

// Callback function to handle incoming MQTT messages
void on_message(char* topic, byte* message, unsigned int length) {
  Serial.print("Message received: ");                     // A message prefix

  for (int i = 0; i < length; i++)                        // Loop through the message bytes
    Serial.print((char)message[i]);                       // Print each character to the Serial Monitor

  Serial.println();                                       // Move to a new line after printing the message
}
```

NodeMCU as Subscriber: Code

```
void setup() {
  Serial.begin(115200);                          // Initialize serial communication at baudrate of 115200

  WiFi.begin(ssid, password);                    // Attempt to connect to the Wi-Fi network
  while (WiFi.status() != WL_CONNECTED) {        // Wait until the NodeMCU is successfully connected
    delay(1000);                                 // Wait 1 second before rechecking Wi-Fi connection status
    Serial.println("Connecting to WiFi...");     // A message indicating an attempt to connect to Wi-Fi
  }
  Serial.println("Connected to WiFi.");          // A message indicating a successful connection

  client.setServer(broker, port);                // Connect to the MQTT broker
  client.setCallback(on_message);                // Set callback function for incoming messages

  client.connect("NodeMCU_Subscriber");          // Connect to MQTT broker with the name "NodeMCU_Subscriber"
  Serial.println("Connected to MQTT broker.");   // Successful connection to MQTT broker

  client.subscribe(topic);                       // Subscribe to the specified topic
}

void loop() {
  client.loop();                                 // Start MQTT client loop to receive messages
}
```

NodeMCU as Subscriber: Python Output

jupyter 05.3Python_Publisher Last Checkpoint: 4 months ago

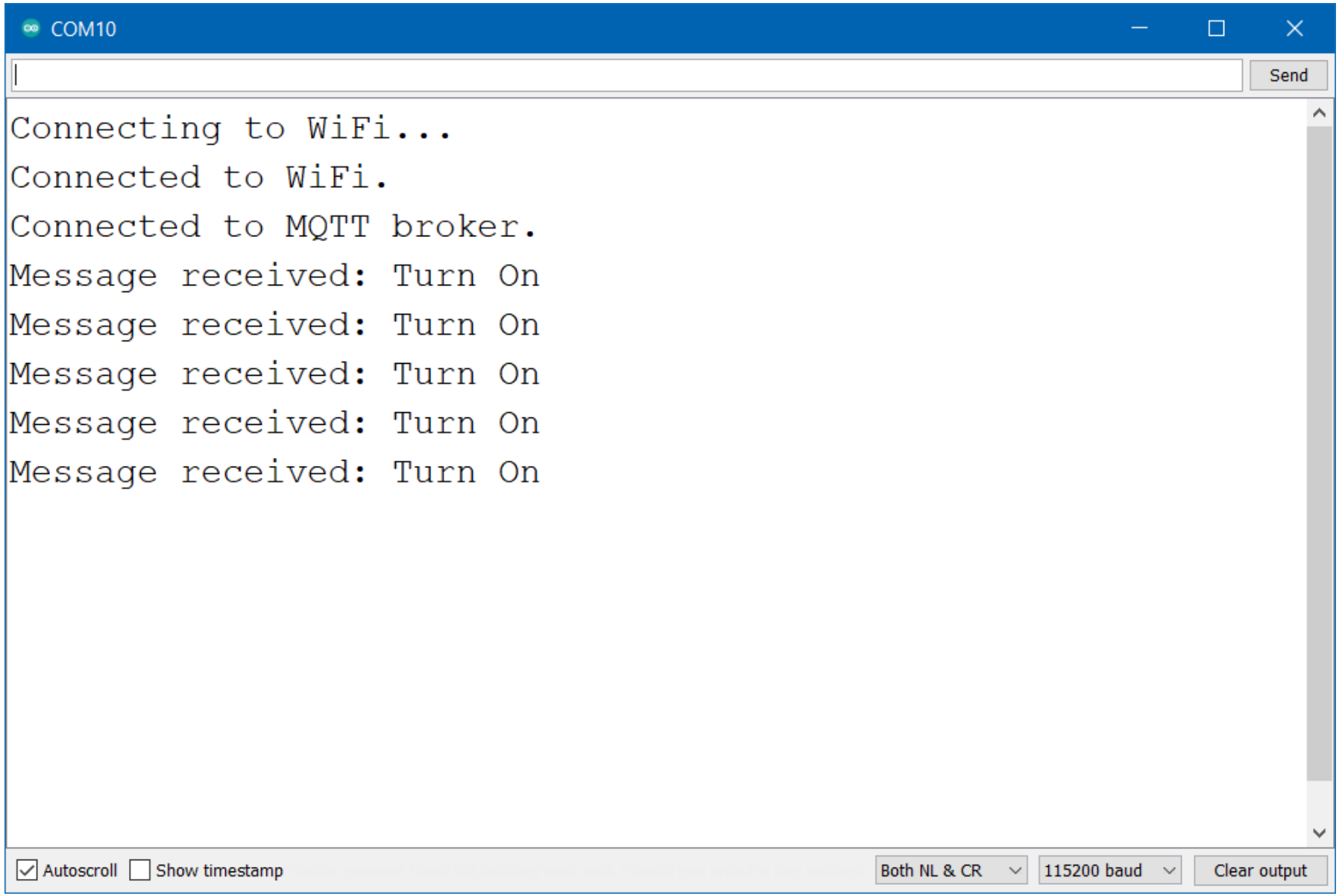
File Edit View Run Kernel Settings Help

Code

```
    message = "Turn On"

    # Print a message indicating that the message has been published
    print("Published message:", message)

    # Wait for 2 seconds before publishing the next message
    sleep(2)

# Disconnect from the MQTT broker
client.disconnect()
```

```
Published message: Turn On
Published message: Turn On
Published message: Turn On
Published message: Turn On
Published message: Turn On
```

NodeMCU as Subscriber: NodeMCU Output



References and Tutorials

- MQTT Broker Introduction
- What is MQTT Quality of Service (QoS)
- Mosquitto MQTT Broker: Pros/Cons, and Tutorial
- Raspberry Pi Publishing MQTT Messages to ESP8266
- How to Use MQTT in Python with Paho Client
- How to Use the Paho MQTT Client in Python with Examples
- Arduino Client for MQTT
- Arduino PubSubClient - MQTT Client Library