GUIDELINE FOR RND PROJECT

1. **Docker**

   Link: https://www.docker.com/

   - This is an open platform for developing, shipping, and running applications.
   - **Step 1:** git clone https://github.com/questdb/questdb-mock-power-sensor mock-sensor
   - **Step 2:** docker network create tutorial

2. **MQTT Eclipse**

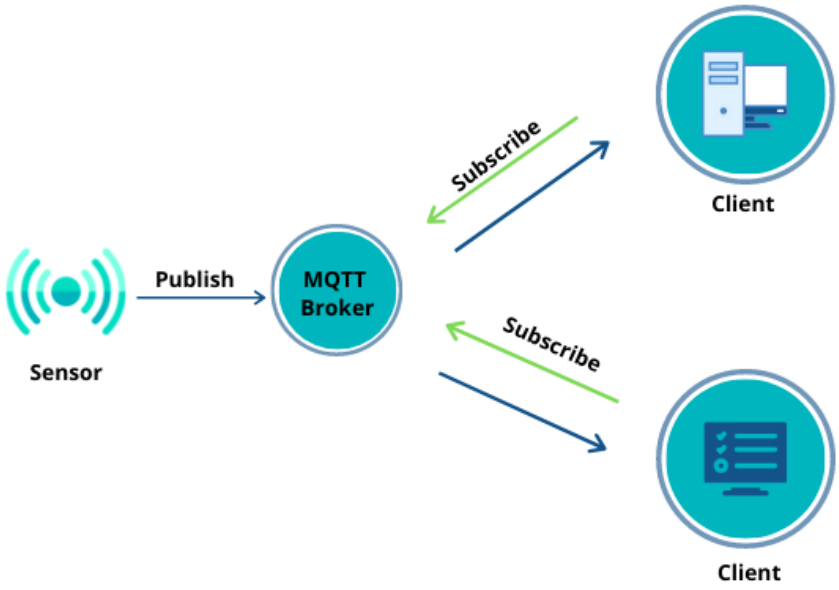
   Link: https://mosquitto.org/download/ (for both window and mac)

   https://mosquitto.org/download/#:~:text=mosquitto%2D2.0.20%2Dinstall%2Dwindows%2Dx64.exe

   - **Step 3:** Override the mosquitto file by conf/mosquitto/mosquitto.conf to fix unauthenticated clients allowed by default config for Mosquitto.
   - **Step 4:** after setting up, dashboard appear: Docker engine stopped, then go to: `C:\Users\<username>\AppData\Roaming\Docker\settings.json,` and set "wslEngineEnabled": true (or just restart the application)
   - **Step 5: Start MQTT Eclipse using Docker**
   - docker run --rm -dit -p 1883:1883 -p 9001:9001 ^-v "%cd%\conf\mosquitto\mosquitto.conf\mosquitto.conf:/mosquitto/config/mosquitto.conf" ^--network=tutorial --name=mosquitto eclipse-mosquitto (for window)
   - docker run --rm -dit -p 1883:1883 -p 9001:9001 \ -v "$(pwd)"/conf/mosquitto/mosquitto.conf:/mosquitto/config/mosquitto

.conf \ --network=tutorial --name=mosquitto eclipse-mosquitto (for mac)

⇨ "%cd%\conf\mosquitto\mosquitto.conf\mosquitto.conf:/mosquitto/config/mosquitto.conf" is the link contain file conf we cloned before



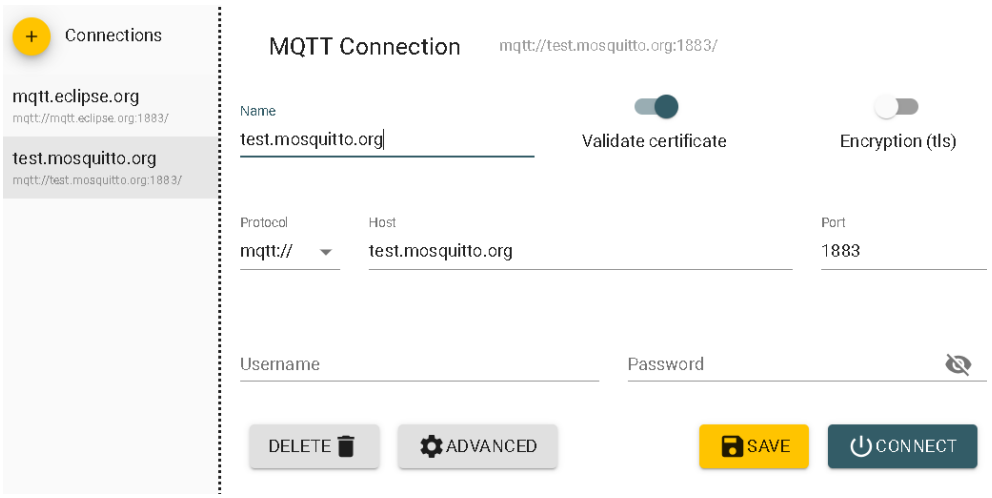
Simple communication

3. **MQTT explorer**

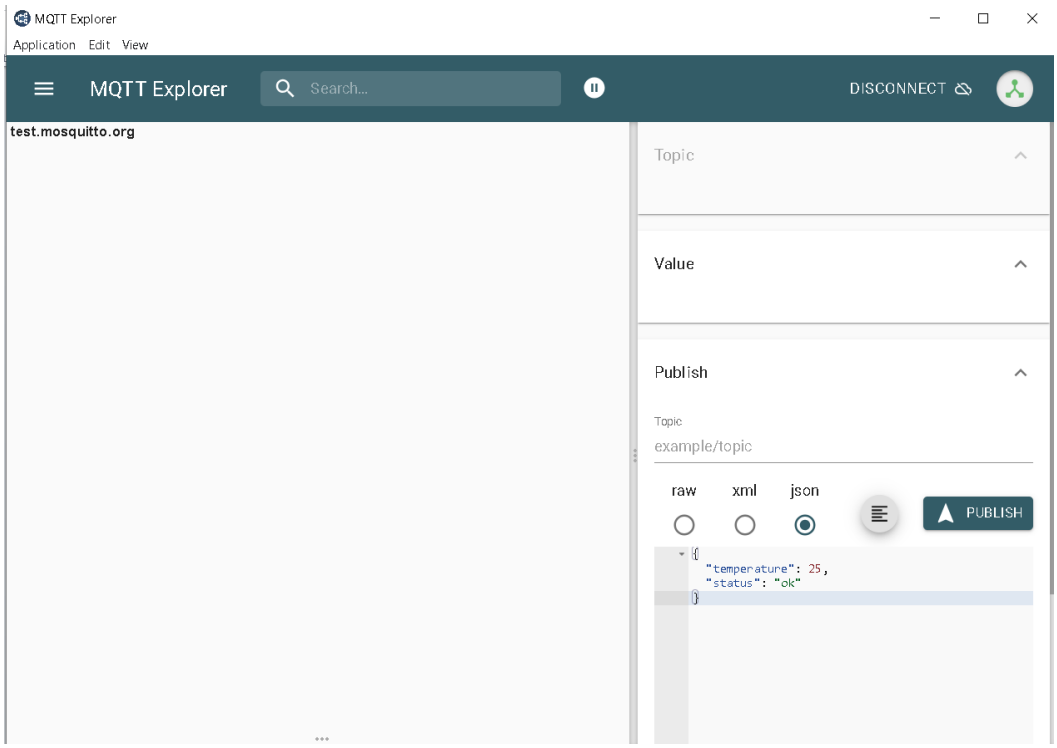
Link: https://mqtt-explorer.com/

- This is a comprehensive MQTT client.
- Working with devices/services on broker.

⇨ **This is a tool for interacting with MQTT broker(mosquitto eclipse). In other terms, it allows user to message, monitor message traffic, visualize the mqtt's structure, publish and subcribe to messages.**



MQTT Explorer connection Dashboard



MQTT Explorer Dashboard

- Set up Host address as the machine adress (Ipv4 address in this tutorial as my computer address)

4. Go

- **Link: https://go.dev/dl/**

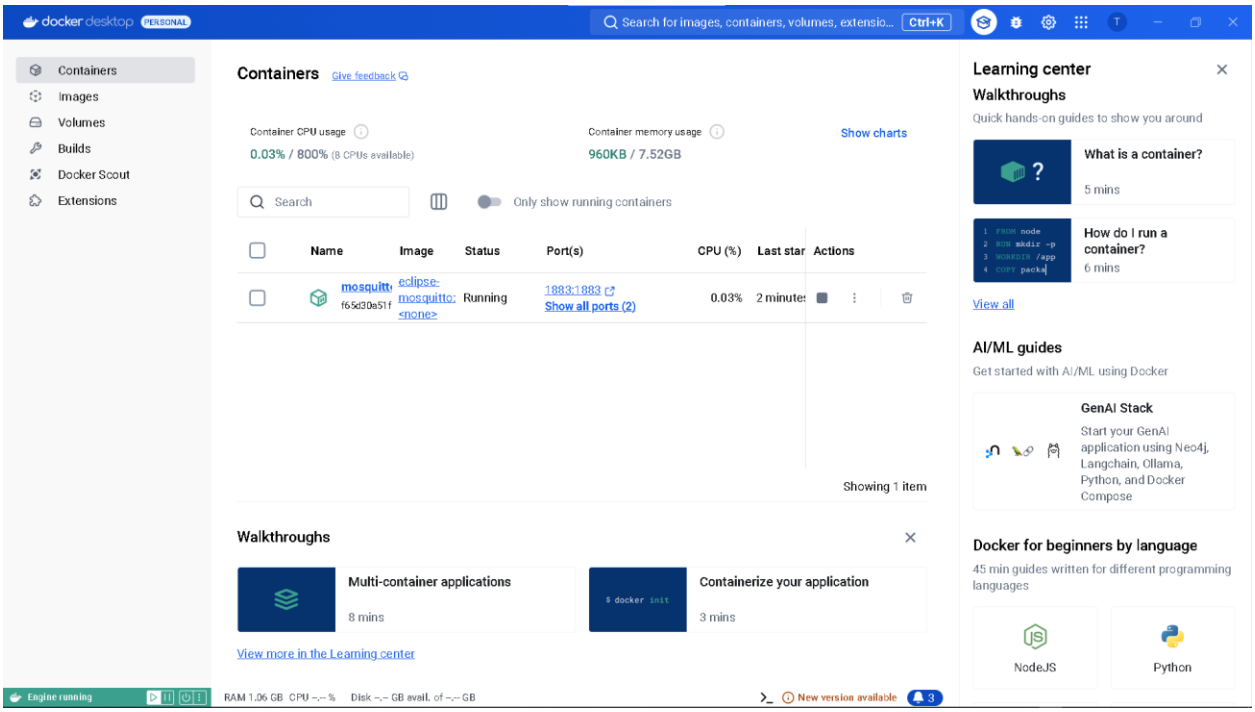
- This is an open-source programming language supported by Google.
- In this project, it used for continuously publishes sensor data to the MQTT broker, ensuring a stable flow of time-series data.

  ⇨ Allow QuestDB to later store and Grafana to visualize the data.

- **Step 6:** cd script

  go get

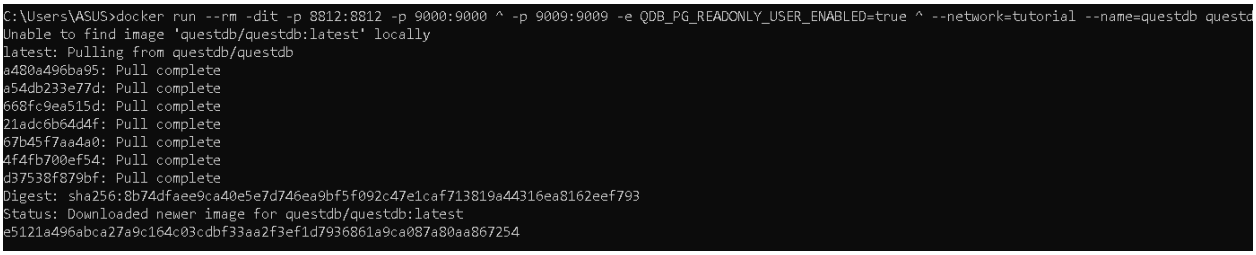
  go run ./main.go (Tunneling data into mosquitto)



5. QuestDB

- This is a time-series database.
- Has SQL coding.
- This database is designed for horizontal scaling, enabling to distribute data and queries across multiple nodes for increased performance and availability.

- **Step 7: start QuestDB using docker**



Command Prompt

- docker run --rm -dit -p 8812:8812 -p 9000:9000 ^ -p 9009:9009 -e QDB_PG_READONLY_USER_ENABLED=true ^ --network=tutorial --name=questdb questdb/questdb **(for window)**
- docker run --rm -dit -p 8812:8812 -p 9000:9000 \ -p 9009:9009 -e QDB_PG_READONLY_USER_ENABLED=true \ --network=tutorial --name=questdb questdb/questdb **(for mac)**
- Check on http://localhost:9000/ to access **questDB** platform

6. Telegraf

- Telegraf transfers data between Mosquitto and QuestDB using **QuestDB's ILP (Influx Line Protocol)** interface(this can hadle large volumes of data sent from telegraf).
- **Step 8: Start telegraf container using Docker**
- docker run --rm -it ^ -v C:\Users\ASUS\mock-sensor\conf\telegraf\telegraf.conf:/etc/telegraf/telegraf.conf ^ --network=tutorial --name=telegraf telegraf **(for window)**
- docker run --rm -it \ -v "$(pwd)"/conf/telegraf/telegraf.conf:/etc/telegraf/telegraf.conf \ --network=tutorial --name=telegraf telegraf**(for mac)**

```
C:\Users\ASUS>docker run --rm -it ^ -v C:\Users\ASUS\mock-sensor\conf\telegraf\telegraf.conf:/etc/telegraf/telegraf.conf ^ --network=tutorial --nam
e=telegraf telegraf
2024-10-26T18:31:38Z I! Loading config: /etc/telegraf/telegraf.conf
2024-10-26T18:31:38Z I! Starting Telegraf 1.32.1 brought to you by InfluxData the makers of InfluxDB
2024-10-26T18:31:38Z I! Available plugins: 235 inputs, 9 aggregators, 32 processors, 26 parsers, 62 outputs, 6 secret-stores
2024-10-26T18:31:38Z I! Loaded inputs: mqtt_consumer
2024-10-26T18:31:38Z I! Loaded aggregators:
2024-10-26T18:31:38Z I! Loaded processors:
2024-10-26T18:31:38Z I! Loaded secretstores:
2024-10-26T18:31:38Z I! Loaded outputs: socket_writer
2024-10-26T18:31:38Z I! Tags enabled: host=df9123bb9ddc
2024-10-26T18:31:38Z I! [agent] Config: Interval:1s, Quiet:false, Hostname:"df9123bb9ddc", Flush Interval:10s
2024-10-26T18:31:38Z I! [inputs.mqtt_consumer] Connected [tcp://mosquitto:1883]
```

Command Prompt

	Name	Image	Status	Port(s)	CPU (%)	Last star	Actions
☐	mosquitt f65d30a51f	eclipse-mosquitto: <none>	Running	1883:1883 Show all ports (2)	0.03%	1 hour ag	■ ⋮ 🗑
☐	questdb e5121a496a	questdb/qu <none>	Running	8812:8812 Show all ports (3)	21.55%	36 minut	■ ⋮ 🗑
☐	telegraf df9123bb9c	telegraf: <none>	Running		0.06%	1 minute	■ ⋮ 🗑

Docker Dashboard

- Now the sensor data is automatically tunneled into QuestDB.

```
# Configuration for Telegraf agent
[agent]
  ## Default data collection interval for all inputs
  interval = "1s"

[[inputs.mqtt_consumer]]
  servers = ["tcp://mosquitto:1883"]
  topics = ["sensor"]
  data_format = "json"
  client_id = "telegraf"
  data_type= "string"
  tag_keys = [
    "country",
    "status"
  ]

# Write results to QuestDB
[[outputs.socket_writer]]
  # Write metrics to a local QuestDB instance over TCP
  address = "tcp://questdb:9009"
[[outputs.file]]
  files = ["stdout"]
```

- **Step 9: Change data_format to “json” in telegraf.conf file**
- ***tag_keys:** key will be added as tag

7. **Grafana**

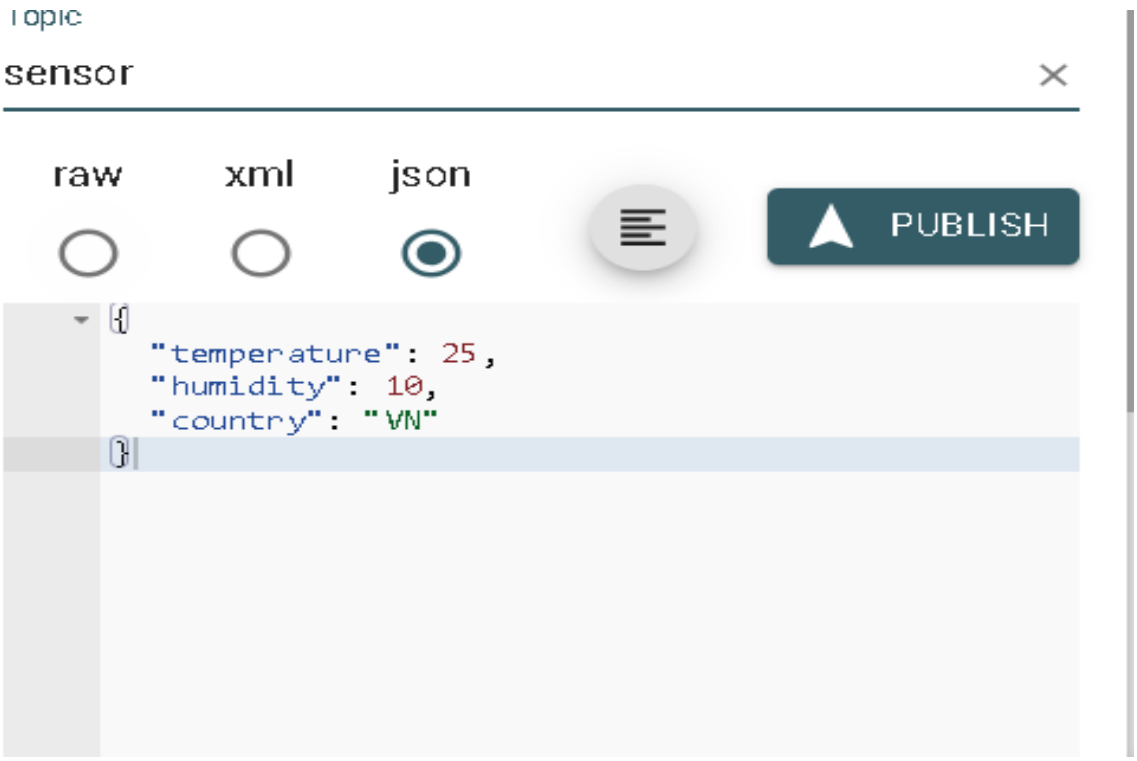
- **Step 10: Run Grafana via Docker**
- docker run --rm -dit -p 3000:3000 ^ --network=tutorial --name=grafana grafana/grafana **(for window)**
- **docker run --rm -dit -p 3000:3000 \ --network=tutorial --name=grafana grafana/grafana (for mac)**
- **Step 11:** login at **http://localhost:3000/login** using **Username:** admin

Password: admin

- **Step 12:** choose Connections/data sources -> add data source -> scroll down to the end and click on find more. Search Questdb -> install -> add new data source
- **Step 13: Config questdb-questdb-datasource**
    - **Server address:** host.docker.internal
    - **Server port:** 8812
    - **Username:**user
    - **Password:**quest
    - **TLS/SSL mode:**disable

8. **Publish testing**

On testing, I send a json package with sensor topic from mqtt explorer as a client to mqtt eclipse broker. The data then will be sent to questDB via telegraf.



JSON file

- **Step 14: Finish on dashboard, move to code and paste SQL:**

SELECT

date_trunc('minute', timestamp) AS minute,

AVG(temperature) AS avg_temperature,

AVG(humidity) AS avg_humidity,

AVG(uv_value) AS uv_value,

AVG(totalBee) AS Bee

FROM mqtt_consumer

WHERE place = 'Dong Nai'

GROUP BY minute

ORDER BY minute ASC

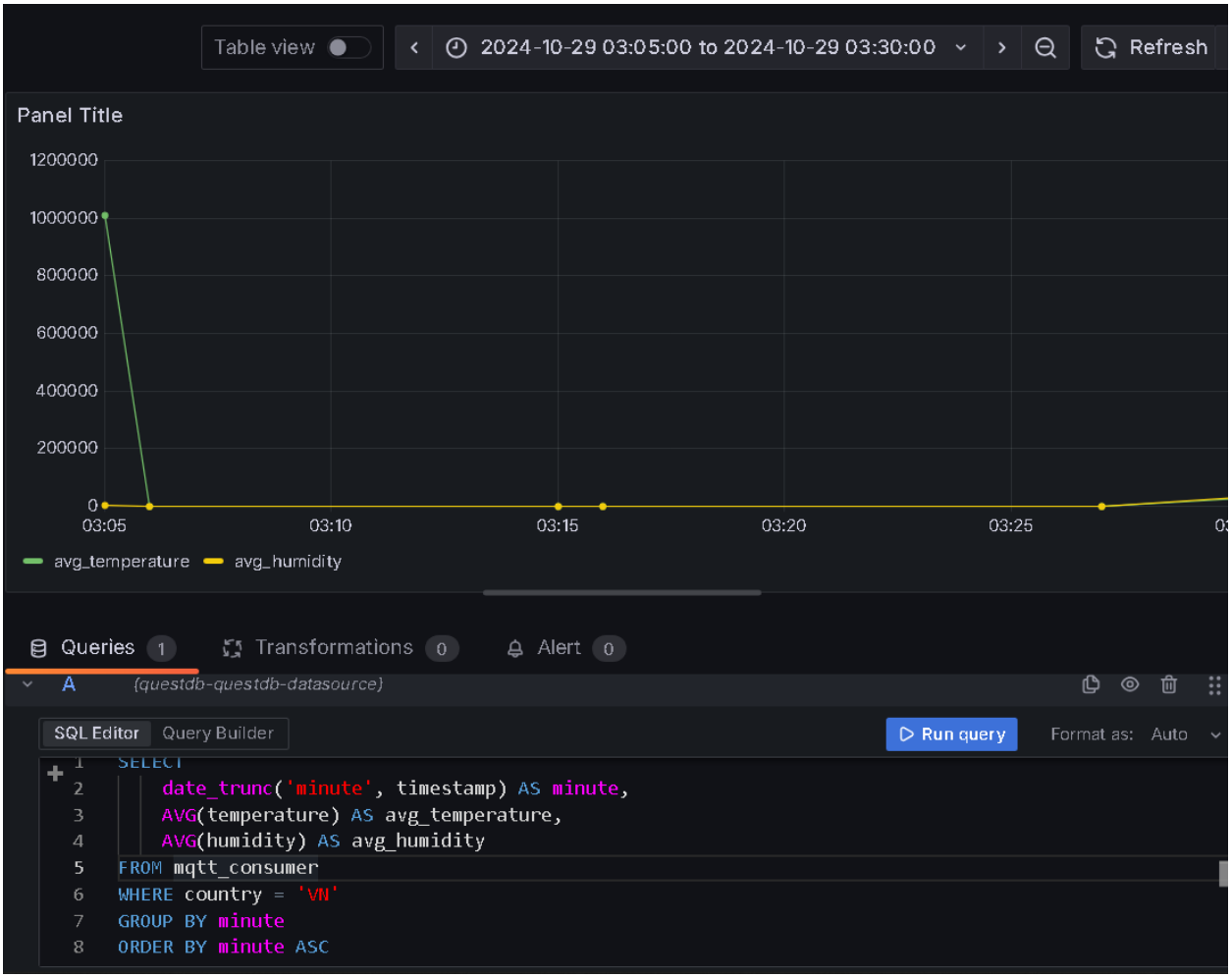
***

```
SELECT
    date_trunc('minute', timestamp) AS minute,
    AVG(temperature) AS avg_temperature,
    AVG(humidity) AS avg_humidity
FROM mqtt_consumer
WHERE country = 'VN'
GROUP BY minute
ORDER BY minute ASC
```

```
SELECT
avg(uv_value),
avg(uv_index),
avg(sound) as sound,
avg(weight) as weight,
avg(temperature_inside) as temperature_inside,
avg(humidity_inside) as humidity_inside,
avg(temperature_outside) as temperature_outside,
avg(humidity_outside) as humidity_outside,
date_trunc('minute', vn_time) AS minute
```

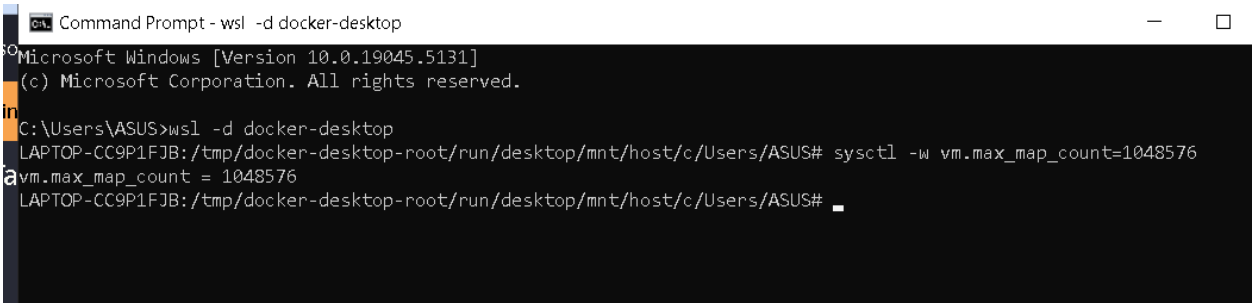
```
from "R&D_database.csv"
WHERE place = 'Dong Nai'
GROUP BY minute
ORDER BY minute ASC
```

minute	avg_temperature	avg_humidity
2024-10-29 03:05:00	1011210	2020
2024-10-29 03:06:00	10	10
2024-10-29 03:15:00	12	12
2024-10-29 03:16:00	19.7	25
2024-10-29 03:27:00	252	325
2024-10-29 03:30:00	25200	32225
2024-10-29 03:32:00	252030	3222115



Grafana Visualization Dashboard

Increase vm.max_map_count limit:



Edit Grafana etc/grafana.ini file on Docker:

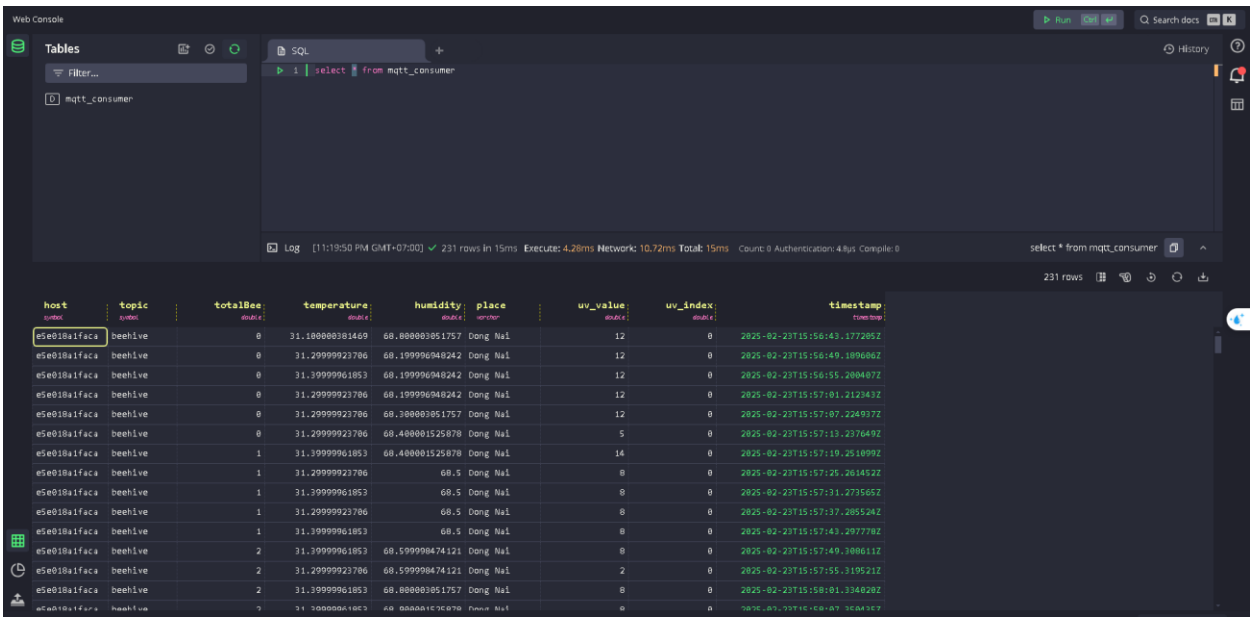
[auth.anonymous]

Enabled = true

org_name = Main Org.

org_role = Viewer

allow_embedding = true



Link:
https://drive.google.com/drive/folders/1DXZFzizVYcct8oy0EXTtlIBykglTZiCj?usp=sharing

Part 2

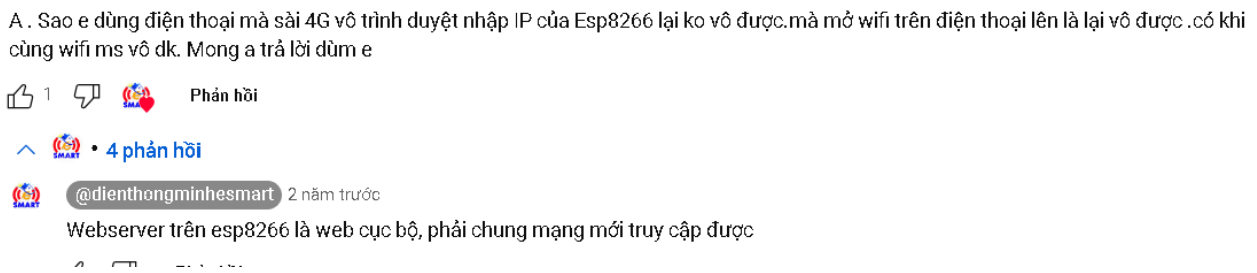
Esp8266 connect wifi with 3 mode:

Accession point mode(phát wifi): WiFi.mode(WIFI_AP);

Station mode(kết nối wifi): WiFi.mode(WIFI_STA);

Accession point and Station mode(vừa phát vừa thu): WiFi.mode(WIFI_AP_STA);

Turn off wifi mode: WiFi.mode(WIFI_OFF);

- On this project we focus on the station mode. In this mode, we need to collect data for: user name/ password/ ip gateway / subnet mask (of Access Point)
- 

- Check SSID is the current connected wifi on cmd: Netsh WLAN show interfaces
- Check password of the SSID: netsh wlan show profile name= "Tên Wi-Fi" key=clear

- Demo connect esp8266 to wifi AP:

sketch_nov10a.ino

```
#include <ESP8266WiFi.h>

const char* ssid = "Nhung";
const char* password = "khimvanhung";


void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
WiFi.mode(WIFI_STA);

WiFi.begin(ssid,password);

while(WiFi.status() !=WL_CONNECTED){
  delay(500);
  Serial.print(".");
}
Serial.println("WiFi connected");
Serial.print("IP HOST: ");
Serial.println(WiFi.localIP());
}

void loop(){

}
```

Output Serial Monitor

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM4')

```
17:59:19.252 -> ......WiFi connected
17:59:21.812 -> IP HOST: 192.168.1.138
```

- **Config for esp wifi(thiết lập cấu hình wifi): WiFi.config(staticip, gateway, subnet, dns1, dns2);**

- **Staticip:** static ip we want to assign for esp(Ip tĩnh muốn gán cho esp)
- **Gateway**: IP of gateway(router) to connect the outside network(ip của gateway của router để kết nối với mạng bên ngoài)
- **Subnet**: subnet for ip range of local network(subnet xác định phạm vi ip của mạng nội bộ)
- **Dns1, dns2**: tham số tùy chọn của máy chủ phân giải tên miền

- **Use cmd**

```
Serial.println(WiFi.gatewayIP());

Serial.println(WiFi.subnetMask());
```

- **To check the current AP gateway and subnet mask.**

```
Serial.println(WiFi.gatewayIP());
Serial.println(WiFi.subnetMask());
}
```

Serial Monitor ×

(Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM4')

```
7.900 ->  ?n??$?!?□??□OCE??□□OCAM4?<□□ClX.......WiFi connected
2.364 -> IP HOST: 192.168.1.138
2.396 -> 192.168.1.1
2.441 -> 255.255.255.0
```

-

- ****note:** the staticip should have the same subnet as AP(192.168.1.XXX)

- **Manage the connection:**

WiFi.status();

0 : WL_IDLE_STATUS	Đang thay đổi trạng thái
1 : WL_NO_SSID_AVAIL	SSID không được tìm thấy
3 : WL_CONNECTED	Kết nối thành công
4 : WL_CONNECT_FAILED	Sai mật khẩu kết nối
6 : WL_DISCONNECTED	Chưa thiết lập chế độ STA

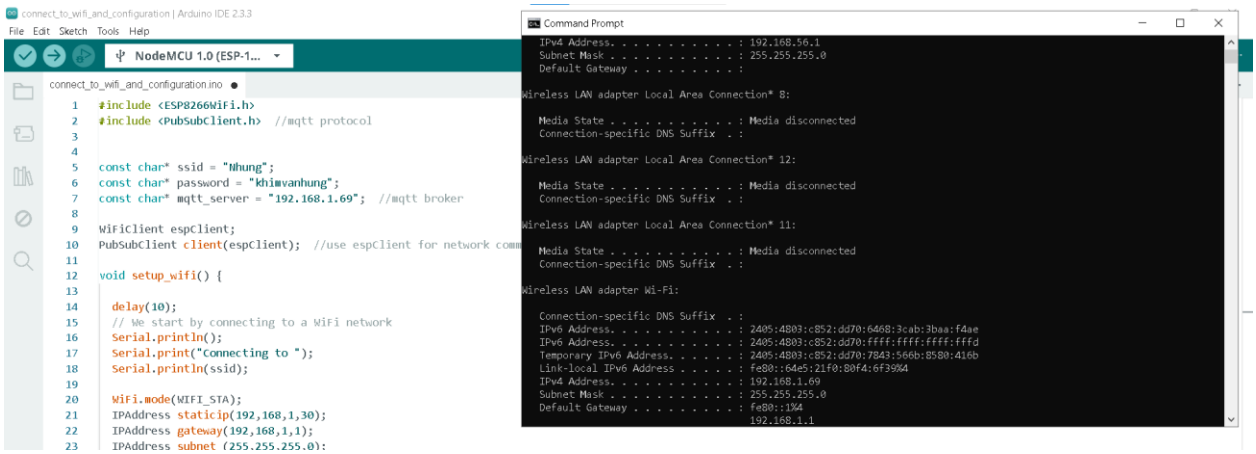
```
23  }
24  Serial.println("WiFi connected");
25  Serial.print("IP HOST: ");
26  Serial.println(WiFi.localIP());
27  Serial.println(WiFi.gatewayIP());
28  Serial.println(WiFi.subnetMask());
29  Serial.println(WiFi.status());
30  }
31
32  void loop(){
```

Output Serial Monitor ×

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM4')

```
18:51:22.296 -> 0?~?4?!?□??□_CE??□□OCAM4?,□C|X.............WiFi connected
18:51:29.701 -> IP HOST: 192.168.1.30
18:51:29.701 -> 192.168.1.1
18:51:29.743 -> 255.255.255.0
18:51:29.743 -> 3
```

- **Check if wifi is connected: WiFi.isConnected():** true if connected, false if disconnected
- **Disconnect and Reconnected after that:** WiFi.reconnect(); -> esp have this in-build function
- **Take the ip address the esp connected:** WiFi.localIP();
- **Mqtt_server:** mqtt broker



- **Fail to connect:**

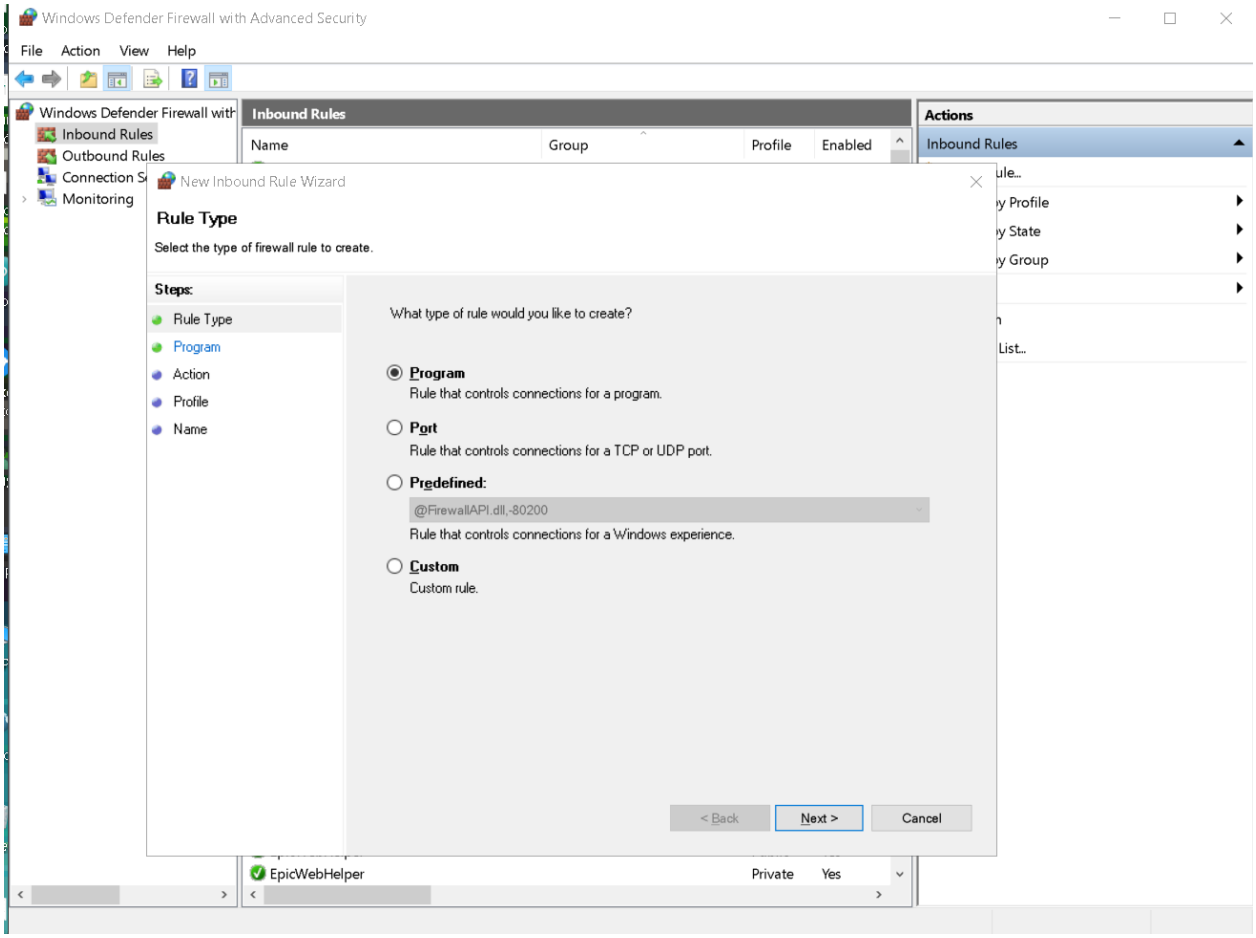
```
41
42  void callback(char* topic, byte* payload, unsigned int length) {
43  ..
44
45  }
46
47  void reconnect() {
48    // Loop until we're reconnected
49    while (!client.connected()) {
50        Serial.print("Attempting MQTT connection...");
51
52    if (client.connect("ESP8266Client")) {
53      Serial.println("connected");
54     } else {
55      Serial.print("failed, rc=");
56      Serial.print(client.state());
57      Serial.println(" try again in 5 seconds");
58      delay(5000);
59    }
60    }
61  }
```

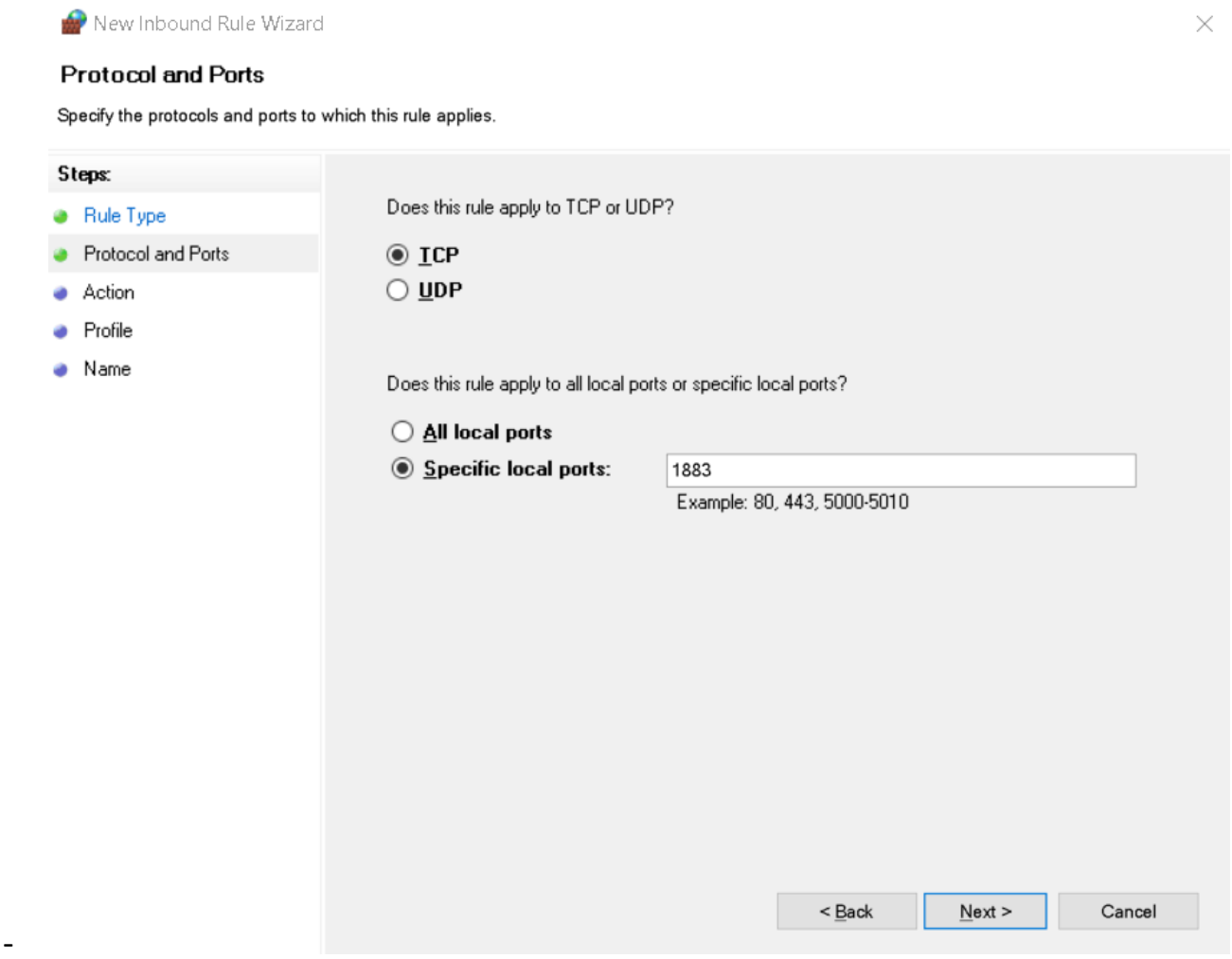
Output Serial Monitor ×

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM4')

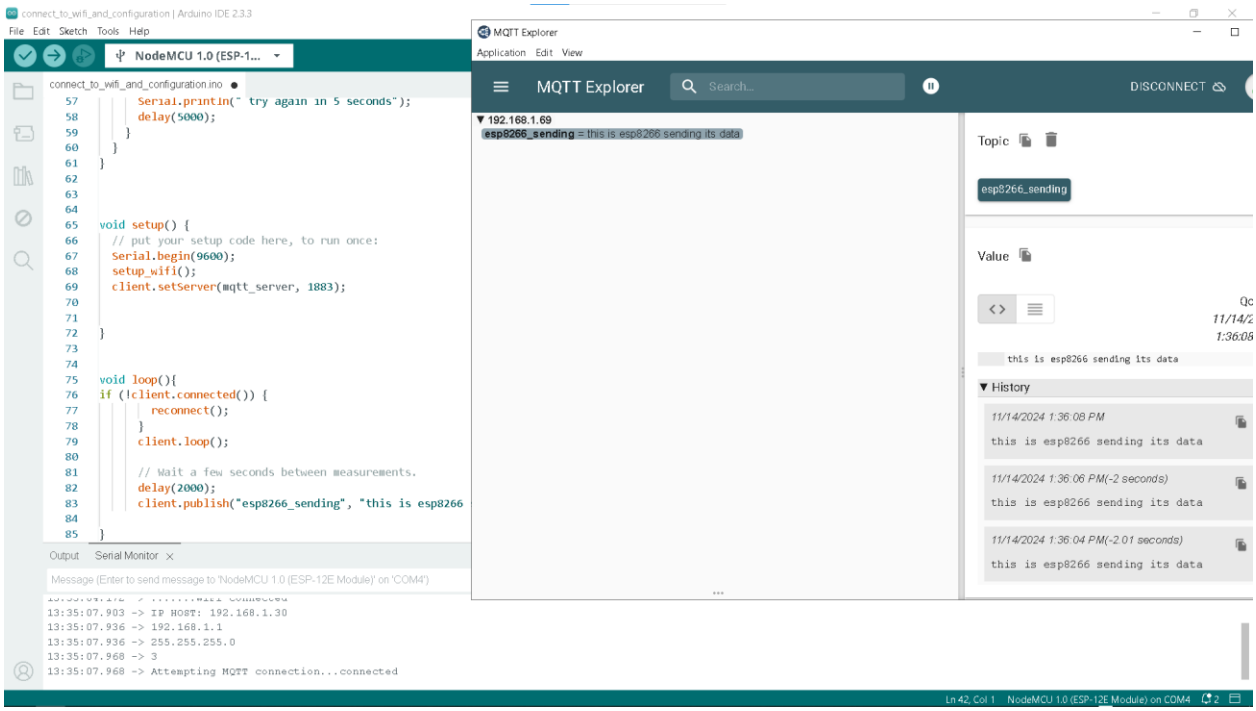
```
13:31:41.083 -> Attempting MQTT connection...failed, rc=-2 try again in 5 seconds
```

- Go to window defender/advance setting/inbound rules/new rules/ports


Windows Defender Firewall with Advanced Security
File Action View Help
Windows Defender Firewall with
Inbound Rules
Outbound Rules
Connection S
Monitoring
Inbound Rules
Name
Group
Profile
Enabled
Actions
Inbound Rules
New Inbound Rule Wizard
Rule Type
Select the type of firewall rule to create.
Steps:
Rule Type
Program
Action
Profile
Name
What type of rule would you like to create?
Program
Rule that controls connections for a program.
Port
Rule that controls connections for a TCP or UDP port.
Predefined:
@FirewallAPI.dll,-80200
Rule that controls connections for a Windows experience.
Custom
Custom rule.
< Back
Next >
Cancel
EpicWebHelper
Private
Yes


New Inbound Rule Wizard
Protocol and Ports
Specify the protocols and ports to which this rule applies.
Steps:
Rule Type
Protocol and Ports
Action
Profile
Name
Does this rule apply to TCP or UDP?
TCP
UDP
Does this rule apply to all local ports or specific local ports?
All local ports
Specific local ports:
1883
Example: 80, 443, 5000-5010
< Back
Next >
Cancel

- **Finish sending data from esp8266 to device(mqtt explorer):**



Subscribe:

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
      Serial.print("Attempting MQTT connection...");

  if (client.connect("ESP8266Client")) {
    Serial.println("connected");
    client.subscribe("device/led");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '1') {
    Serial.print("esp8266 receive message 1");
  }

}
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

}
```

Output Serial Monitor ×

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM4')

```
14:01:04.322 -> 255.255.255.0
14:01:04.354 -> 3
14:01:04.354 -> Attempting MQTT connection...connected
14:01:32.296 -> Message arrived [device/led] 1
14:01:32.328 -> esp8266 receive message 1
```

Part 3: send data from esp8266 to questdb

```
#include <Arduino_JSON.h>
```

Include library

```
JSONVar data;
```

Define json variable

```
data["temperature"] = getTemp("c");
data["humidity"] = getTemp("h");
data["place"] = "Dong Nai";
String jsonString = JSON.stringify(data);


client.publish("beehive", jsonString.c_str(), true);
```

Publish the converted data

Adjust refreshing time of Grafana dashboard on webapp:

```
                    <iframe src="http://localhost:3000/d-solo/bedrbmhdvpm9se/new-dashboard?orgId=1&refresh=1m&timezone=browser&panelId=1&__feature.dashboardSceneSolo" width="750" height="400" ></iframe>
```

➔ **With refresh = 1m equal to refreshing time will be 1 minute**

Questdb sql:

```
SELECT host, topic, uv_value, place, uv_index, totalBee, sound,
weight, temperature_inside, humidity_inside, temperature_outside,
humidity_outside, timestamp, timestamp + 25200000000L AS vn_time

from mqtt_consumer
```