

# Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones



Desarrollo de aplicaciones para plataformas ubicuas IoT

PRÁCTICAS MOSQUITTO MQTT

**Estudiante**

Elmer José Muñoz Zúñiga

**Código**

100616020291

**Profesor:**

Ing. Julian Andrés Bolaños

Popayán, Cauca

2021

# Índice

<b>1. Prácticas Mosquitto MQTT</b>	<b>3</b>
1.1. Practica 1 . . . . .	3
1.2. Practica 2 . . . . .	4
<b>2. Ejercicio de clase - 13 de Abril</b>	<b>5</b>
<b>3. Script Publicación</b>	<b>6</b>
<b>4. Script con comodín \$</b>	<b>8</b>
<b>5. Script con implementación SQL</b>	<b>9</b>
<b>6. Practica de Python</b>	<b>10</b>

# 1. Prácticas Mosquitto MQTT

## 1.1. Practica 1

Después de la correcta instalación de mosquitto si se quiere cambiar el puerto de escucha para su correcto funcionamiento, se debe acceder al archivo mosquitto.conf, donde es necesario seguir los siguientes pasos: Abrir con un editor en modo administrador el archivo mosquitto.conf y realizar el ajuste “ejm cambio de puerto” como se ve en la figura 1.

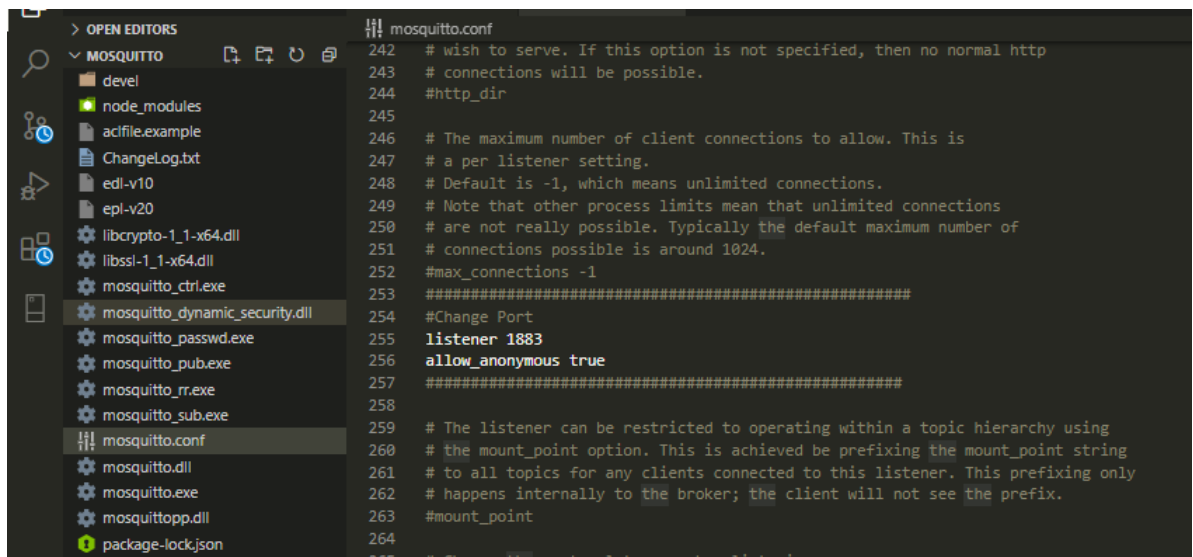


Figura 1: Archivo mosquitto.conf

Se comprueba si el Servicio ya se encuentra instalado y ejecutándose, ya que se debería poder ejecutar: “net stop mosquitto”, como se muestra en la figura 2

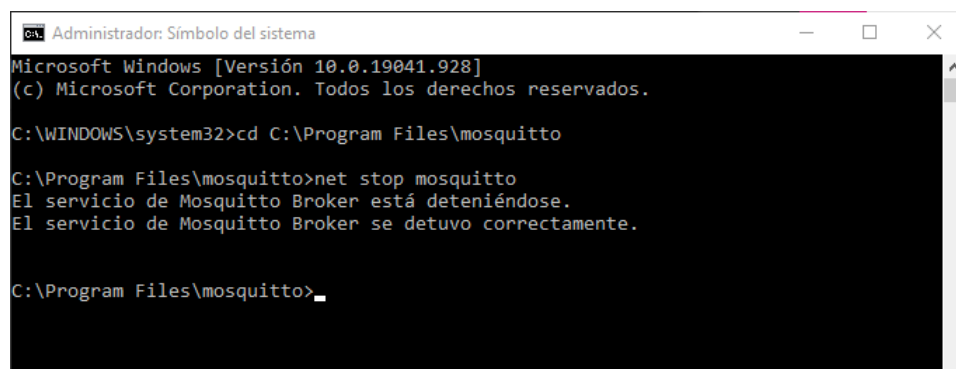


Figura 2: Net stop mosquitto

Y posteriormente “net start mosquitto”, como se muestra en la figura 3

```
C:\Program Files\mosquitto>net start mosquitto

El servicio de Mosquitto Broker se ha iniciado correctamente.

C:\Program Files\mosquitto>
```

Figura 3: Net start mosquitto

Finalmente comprobamos si las reglas del firewall están establecidas para escuchar el puerto configurado, al ejecutar el comando “netstat -a” debería aparecer como se muestra a continuación:

```
Administrador: Símbolo del sistema - netstat -a

C:\Program Files\mosquitto>netstat -a

Conexiones activas

Proto  Dirección local      Dirección remota      Estado
TCP    0.0.0.0:80            LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:135           LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:443           LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:445           LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:1688          LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:1883          LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:3306          LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:5040          LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:27036         LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:49664         LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:49665         LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:49666         LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:49667         LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:49668         LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:49691         LAPTOP-05G8R99B:0     LISTENING
TCP    0.0.0.0:50128         LAPTOP-05G8R99B:0     LISTENING
TCP    10.0.0.1:139          LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:3213        LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:5354        LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:5354        LAPTOP-05G8R99B:49669 ESTABLISHED
TCP    127.0.0.1:5354        LAPTOP-05G8R99B:49674 ESTABLISHED
TCP    127.0.0.1:5480        LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:6463        LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:15292       LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:15393       LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:16494       LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:19876       LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:23116       LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:23119       LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:23119       LAPTOP-05G8R99B:53522 TIME_WAIT
TCP    127.0.0.1:27015       LAPTOP-05G8R99B:0     LISTENING
TCP    127.0.0.1:27015       LAPTOP-05G8R99B:49755 ESTABLISHED
TCP    127.0.0.1:27017       LAPTOP-05G8R99B:0     LISTENING
```

Figura 4: Net stat -a

## 1.2. Practica 2

Teniendo el puerto donde Mosquitto se está ejecutando, se realiza una subscripción a un topic de la siguiente manera Dirigir una ventana cmd a la carpeta raíz de mosquito y ejecutar el siguiente comando: “mosquitto.sub -h 127.0.0.1 -p 1883 -t ”casa” -v”, significa que el cliente MQTT se subscribirá al topic Casa, en este caso vemos que la ventana presentada se encuentra vacía y en espera, significa que el programa está trabajando correctamente, como se muestra en la figura 5.

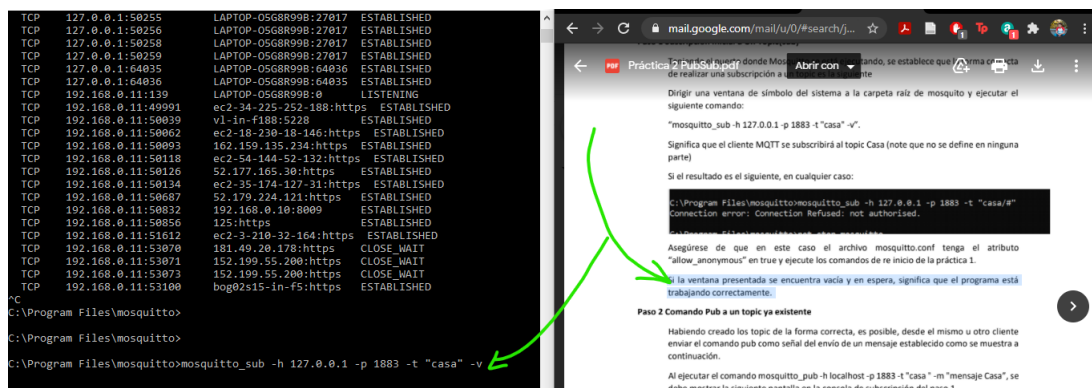


Figura 5: Mosquitto sub

Habiendo creado los topic de la forma correcta, es posible, desde otra ventana cmd enviar el comando pub como señal del envío de un mensaje establecido como se muestra en la figura 6.

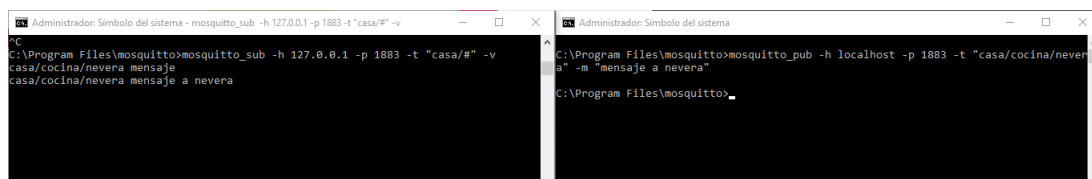


Figura 6: Comprobación de solución

## 2. Ejercicio de clase - 13 de Abril

A continuación se presentan los respectivos comprobantes de la realización de la practica de clase del 13 de Abril.

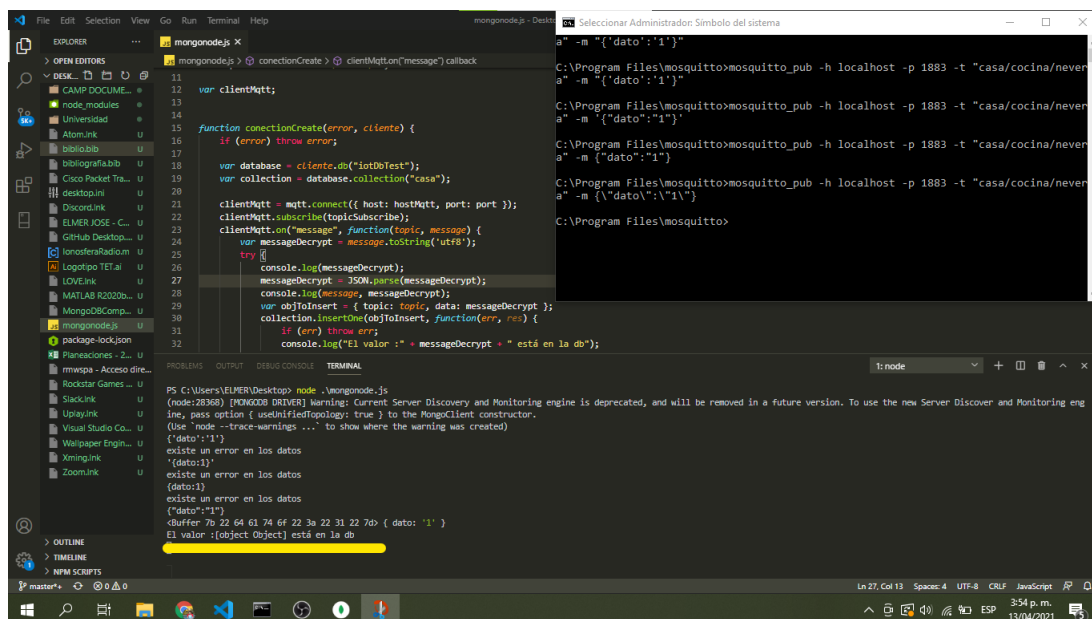


Figura 7: Código y Prueba de funcionamiento

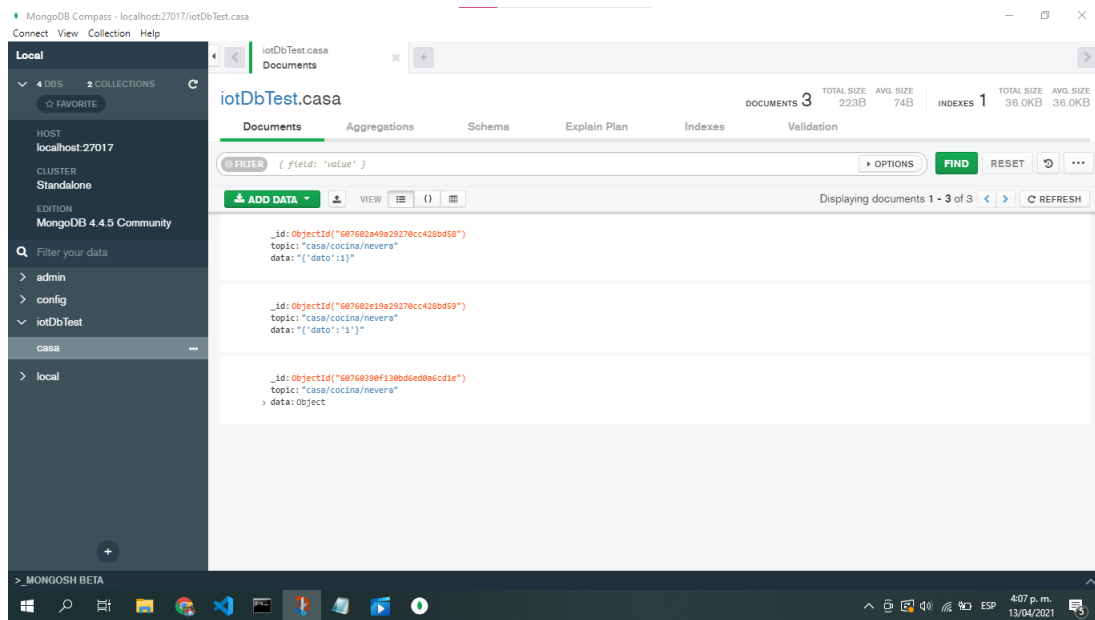


Figura 8: Comprobación en MongoDB

### 3. Script Publicación

Para este ejercicio es necesario tener en funcionamiento el código de clase del literal anterior llamado "mongonode.js". En la figura 9 podemos notar el funcionamiento del ejercicio indicado donde se corren los códigos "mongonode.js" y "publicacion.js" al mismo tiempo.

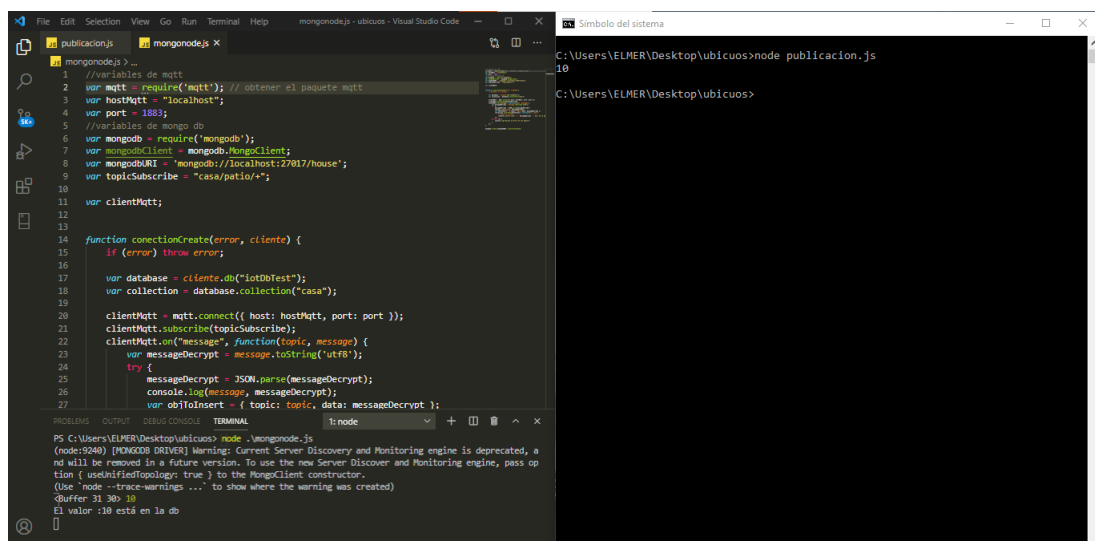


Figura 9: Funcionamiento Mongonode.js y publicacion.js

Con ello podemos observar la utilización correcta del publish, y por ende, la realización correcta del ejercicio, guardando en MongoDB el dato publicado, como lo vemos en la figura 10.

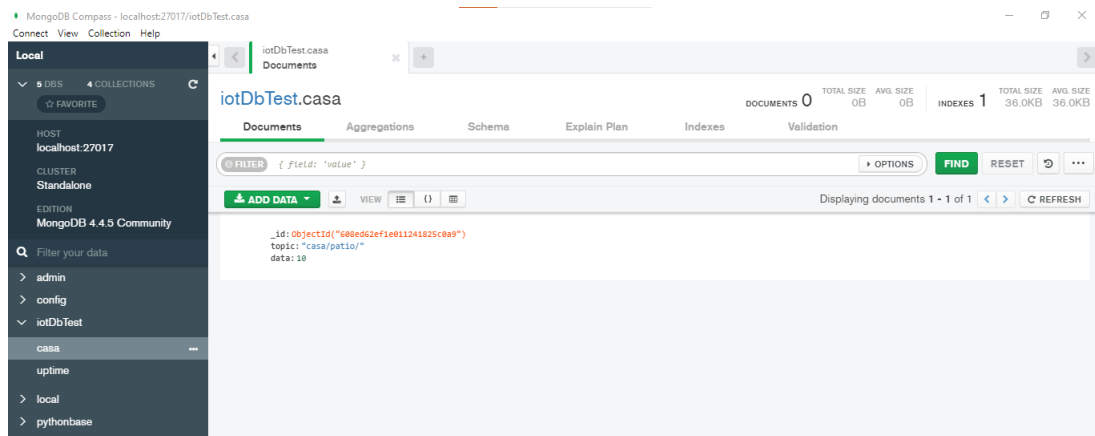


Figura 10: Funcionamiento MongoDB

Finalmente podemos denotar el código realizado para el ejercicio llamado "publicación", donde a grandes rasgos se suscribe al topic "casa/patio/" y posteriormente se prosigue a crear una publicación de un dato, en este caso el numero 10, donde cabe resaltar que se debe convertir a String, como se ve en la figura 11

```

1  publicacion.js x  mongonode.js
2  1 var mqtt = require('mqtt')
3  2 var hostMqtt = "localhost";
4  3 var port = 1883;
5  4 var client = mqtt.connect({ host: hostMqtt, port: port })
6  5
7  6 client.on('connect', function() {
8  7   client.subscribe('casa/patio/*', function(err) {
9  8     if (!err) {
10  9       n = retorna();
11  10      client.publish('casa/patio/', n, 20000)
12  11     }
13  12   })
14  13 })
15  14
16  15 client.on('message', function(topic, message) {
17  16   console.log(message.toString())
18  17   client.end()
19  18 })
20  19
21  20 function retorna() {
22  21   num = 10
23  22   return num.toString()
24  23 }
25  24

```

Figura 11: Código publicacion.js

Por medio de este script se pueden realizar varias practicas, como publicar varios números al azar y ser guardados en MongoDB, como se puede observar en la figura 12

```

1 var mqtt = require('mqtt');
2 var hostMqtt = "localhost";
3 var port = 1883;
4 var client = mqtt.connect({ host: hostMqtt, port: port });
5
6 client.on('connect', function() {
7   client.subscribe('casa/patio+', function(err) {
8     if (err) {
9       console.log(message.toString());
10      client.end();
11    }
12    for (let i = 0; i < 2; i++) {
13      n = retorno();
14      client.publish('casa/patio/', n, 20000);
15    }
16  });
17
18 client.on('message', function(topic, message) {
19   console.log(message.toString());
20   client.end();
21 })
22
23 function retorno() {
24   var num = Math.floor(Math.random() * (54 - 1)) + 1;
25   return num.toString();
26 }

```

```

C:\Users\ELMER\Desktop\ubicuos>node publicacion.js
10
C:\Users\ELMER\Desktop\ubicuos>node publicacion.js
1
14
C:\Users\ELMER\Desktop\ubicuos>

```

```

(Use 'node --trace-warnings...' to show where the warning was created)
<Buffer 31 30> 10
El valor :10 está en la db
<Buffer 31 30> 10
El valor :10 está en la db
<Buffer 31 30> 10
El valor :10 está en la db
<Buffer 31> 1
El valor :1 está en la db
El valor :14 está en la db

```

Figura 12: Variación código publicacion.js

## 4. Script con comodín \$

Para esta practica se debe suscribir a un nuevo topic usando el comodín \$, este tipo de topic son usados para obtener estadísticas del broker. Para este caso se utilizara la suscripción a "\$SYS/broker/uptime", el cual corresponde al tiempo en segundos en el cual el broker ha estado online, como se ve en la figura 13.

```

1 //variables de mqtt
2
3 var mqtt = require('mqtt'); // obtener el paquete mqtt
4 var hostMqtt = "localhost";
5 var port = 1883;
6 //variables de mongo db
7 var mongodb = require('mongodb');
8 var MongoClient = mongodb.MongoClient;
9 var mongoDbURI = "mongodb://localhost:27017/house";
10 var topicSubscribe = "casa/cocina/"; //Con el signo + le decimos que vemos a escuchar todo lo de la cocina
11 var topicSubscribe1 = "$SYS/broker/uptime";
12
13 var clientMqtt;
14
15
16 function connectionCreate(error, cliente) {
17   if (error) throw error;
18   var database = cliente.db("lotDbTest");
19   var collection = database.collection("uptime");
20 }

```

```

El valor :114789 seconds está en la db
114791 seconds
El valor :114791 seconds está en la db
114802 seconds
El valor :114802 seconds está en la db
114813 seconds
El valor :114813 seconds está en la db
114824 seconds
El valor :114824 seconds está en la db
114835 seconds
El valor :114835 seconds está en la db

```

Figura 13: Código con nuevo topic \$SYS

podemos ver que se crea una nueva colección llamada uptime donde se van a guardar los datos estadísticos solicitados, como se ve en las figuras 14 y 15



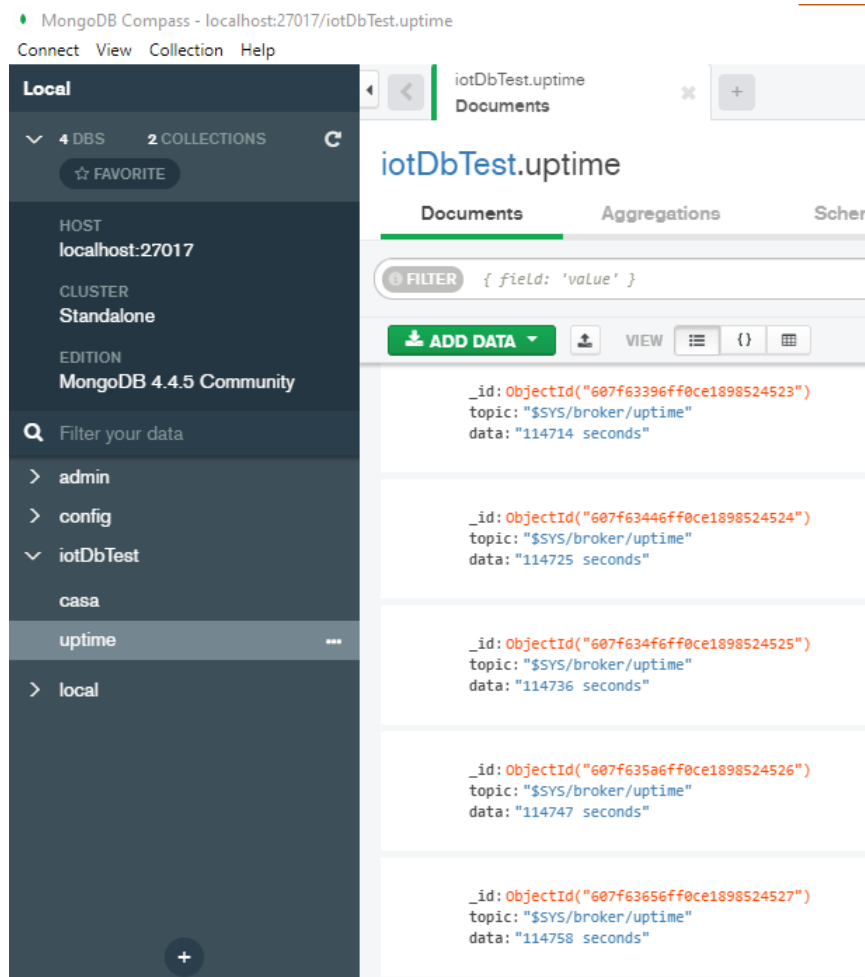


Figura 14: Comprobación MongoDB

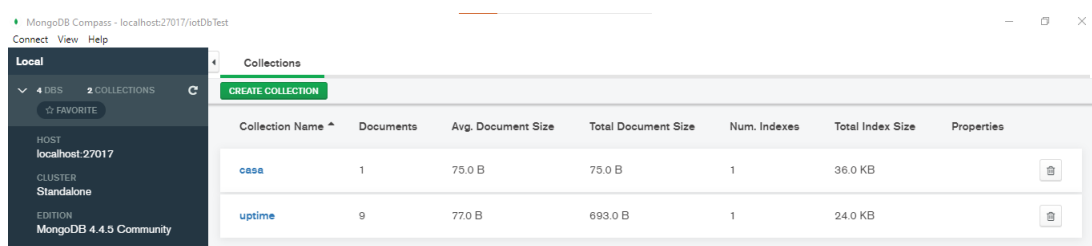


Figura 15: Nueva colección creada

## 5. Script con implementación SQL

En la siguiente implementación se debe cambiar la base de datos a una Mysql para ello, se debe crear una base de datos y una tabla dentro mediante xampp y su servidor apache, en el código "MongoDB" se debe crear un objeto de conexión hacia la base de datos SQL, como se ve en las figuras 16 y 17

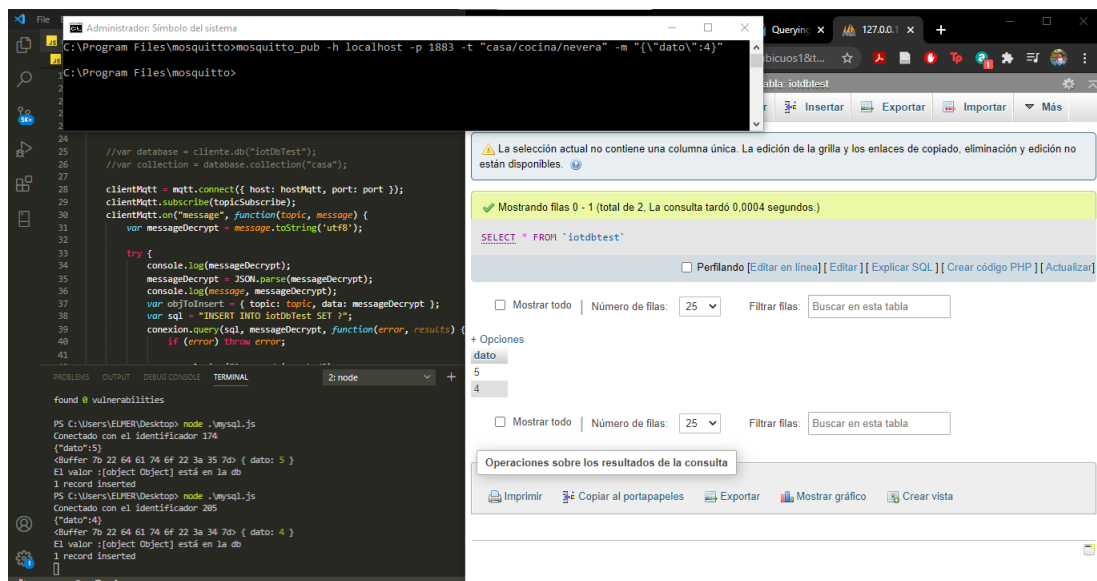


Figura 16: Funcionamiento del código y comprobación de guardado en MySQL

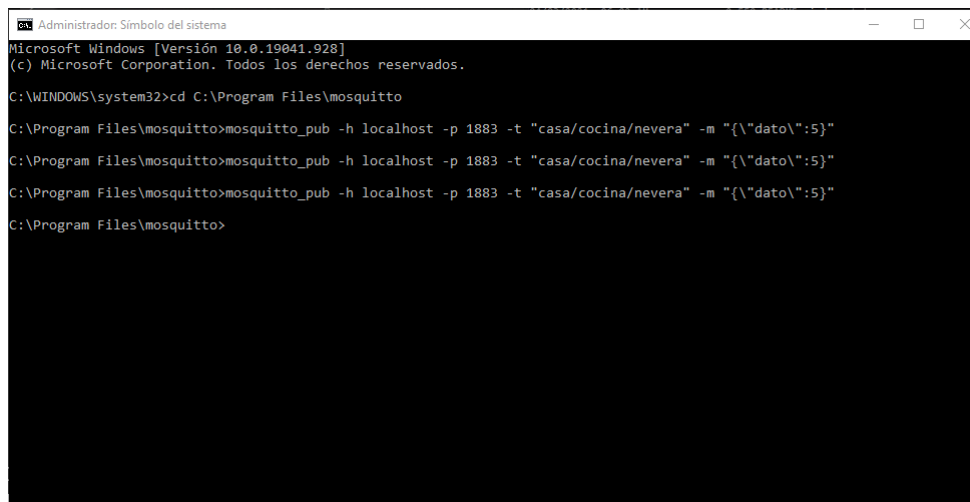


Figura 17: Publicación desde cmd

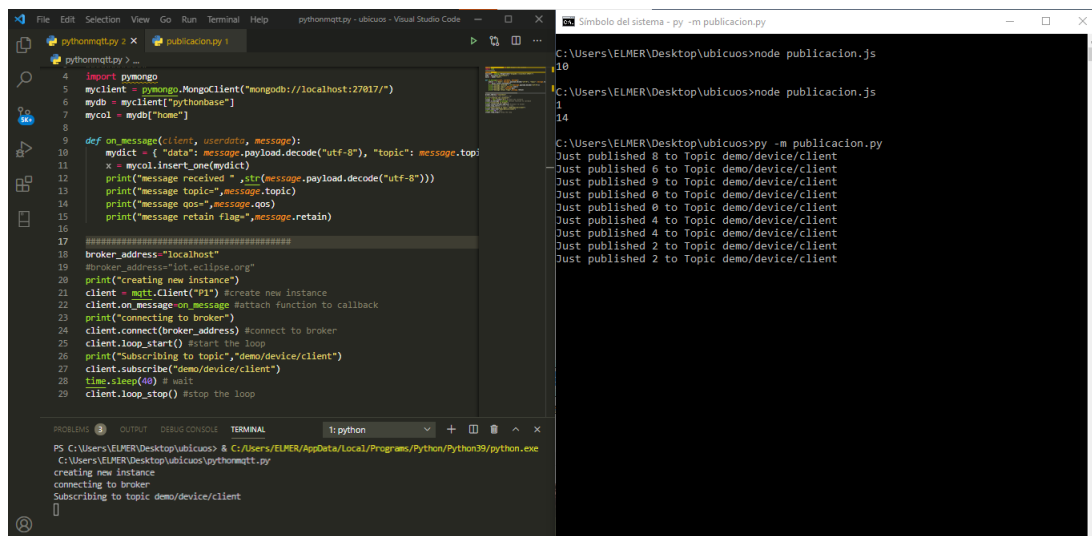
## 6. Practica de Python

Se debe implementar el código brindado en el taller, como se indica en el la figura 18.

```
pythonmqtt.py 2 x publicacion.py 1
pythonmqtt.py > ...
5 *****
6 import pymongo
7 myclient = pymongo.MongoClient("mongodb://localhost:27017/")
8 mydb = myclient["pythonbase"]
9 mycol = mydb["home"]
10
11 def on_message(client, userdata, message):
12     mydict = { "data": message.payload.decode("utf-8"), "topic": message.topic }
13     x = mycol.insert_one(mydict)
14     print("message received " ,str(message.payload.decode("utf-8")))
15     print("message topic=",message.topic)
16     print("message qos=",message.qos)
17     print("message retain flag=",message.retain)
18
19 #####
20 broker_address="localhost"
21 #broker_address="iot.eclipse.org"
22 print("creating new instance")
23 client = mqtt.Client("P1") #create new instance
24 client.on_message=on_message #attach function to callback
25 print("connecting to broker")
26 client.connect(broker_address) #connect to broker
27 client.loop_start() #start the loop
28 print("Subscribing to topic:", "demo/device/client")
29 client.subscribe("demo/device/client")
30 time.sleep(40) # wait
31 client.loop_stop() #stop the loop
```

Figura 18: Código de la practica a implementar

Una vez hecho esto se debe incluir el envío de mensajes Publish, comprobando su funcionamiento como se ve en la figura 19.



```
File Edit Selection View Go Run Terminal Help pythonmqtt.py - ubiucos - Visual Studio Code
pythonmqtt.py 2 x publicacion.py 1
4 import pymongo
5 myclient = pymongo.MongoClient("mongodb://localhost:27017/")
6 mydb = myclient["pythonbase"]
7 mycol = mydb["home"]
8
9 def on_message(client, userdata, message):
10     mydict = { "data": message.payload.decode("utf-8"), "topic": message.topic }
11     x = mycol.insert_one(mydict)
12     print("message received " ,str(message.payload.decode("utf-8")))
13     print("message topic=",message.topic)
14     print("message qos=",message.qos)
15     print("message retain flag=",message.retain)
16
17 #####
18 broker_address="localhost"
19 #broker_address="iot.eclipse.org"
20 print("creating new instance")
21 client = mqtt.Client("P1") #create new instance
22 client.on_message=on_message #attach function to callback
23 print("connecting to broker")
24 client.connect(broker_address) #connect to broker
25 client.loop_start() #start the loop
26 print("Subscribing to topic:", "demo/device/client")
27 client.subscribe("demo/device/client")
28 time.sleep(40) # wait
29 client.loop_stop() #stop the loop

C:\Users\ELMER\Desktop\ubiucos>node publicacion.js
10
C:\Users\ELMER\Desktop\ubiucos>node publicacion.js
1
14
C:\Users\ELMER\Desktop\ubiucos>py -m publicacion.py
Just published 8 to Topic demo/device/client
Just published 6 to Topic demo/device/client
Just published 9 to Topic demo/device/client
Just published 0 to Topic demo/device/client
Just published 0 to Topic demo/device/client
Just published 4 to Topic demo/device/client
Just published 4 to Topic demo/device/client
Just published 2 to Topic demo/device/client
Just published 2 to Topic demo/device/client

PS C:\Users\ELMER\Desktop\ubiucos> & C:\Users\ELMER\AppData\Local\Programs\Python\Python39\python.exe
C:\Users\ELMER\Desktop\ubiucos\pythonmqtt.py
creating new instance
connecting to broker
Subscribing to topic demo/device/client
```

Figura 19: Funcionamiento de pythonmqtt.py y publicación.py

Esto se logra mediante el Script "publicación.py" donde se conecta al broker, se utiliza la misma instancia del código anteriormente mencionado y finalmente se publica en el topic "demo/device/client" establecido previamente, como se ve en la figura 20



```
pythonmqtt.py 2  publicacion.py 1 X
publicacion.py > ...
1  import paho.mqtt.client as mqtt
2  from random import randrange
3  import time
4
5  mqttBroker = "localhost"
6  client = mqtt.Client("P1")
7  client.connect(mqttBroker)
8
9  while True:
10     randomNumber = randrange(10)
11     client.publish("demo/device/client", randomNumber)
12     print("Just published " + str(randomNumber) + " to Topic demo/device/client")
13     time.sleep(1)
```

Figura 20: Código publicación.py