

1 Introduction to MQTT, Docker and Arduino Simulator

Table of Contents

1 Introduction to MQTT, Docker and Arduino Simulator.....	1
1.0 Basics of IoT.....	1
1.1 Python MQTT Up.....	2
1.2 Python MQTT Up and Down.....	3
1.3 Docker and container development.....	5
1.4 Arduino MQTT Up (and Down).....	6
1.5 Lab feedback.....	8

1.0 Basics of IoT

This lab will introduce the MQTT (MQ Telemetry Transport) protocol and some free MQTT brokers one can test basic IoT principles against. We are also going to dip our toes into the Docker container technology.

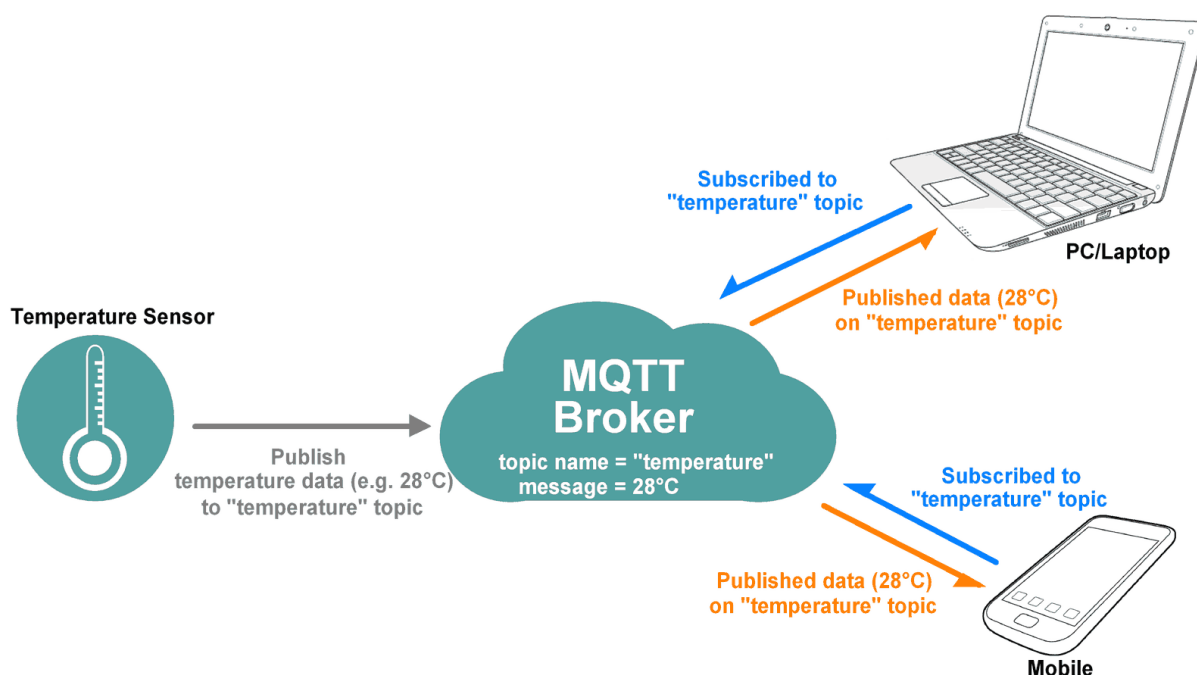


Fig 1. A temperature sensor publish its value to the topic temperature.

Requirements

We are going to use Python and the Paho MQTT Python client library from Eclipse:

<https://www.eclipse.org/paho/>, install instructions:

<https://www.eclipse.org/paho/clients/python/>. You can use some other Python library if you want, but this is the most popular one.

At end we are going to use an Arduino simulator from: <http://virtronics.com.au/> which we are going to run a MQTT Sketch (program) in to get familiar with Arduino and its principles.

To get started look thru the links in Learn about MQTT and the lab study guide I have set up. If you install the MQTT.fx application from: <https://mqttfx.jensd.de/> on your computer you can try publish and subscribe in an easy way.

Some popular open and free MQTT brokers are

- broker.hivemq.com
- test.mosquitto.org
- iot.eclipse.org

After that you have connected to the broker at the Websockets Client Showcase: <http://www.hivemq.com/demos/websocket-client/> you can try to subscribe to the Topic: “testtopic/#”

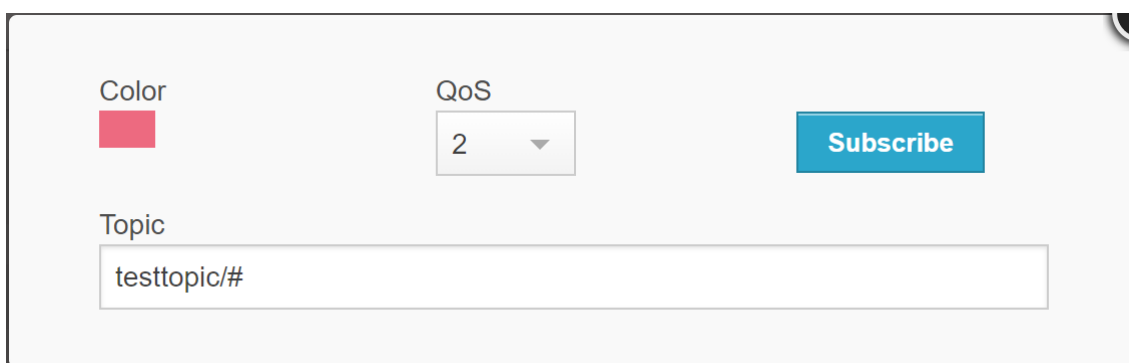


Fig 2. Websockets Client Showcase.

A stream of messages should be visible in the Messages part of the web page.

Using the bundled **mqtt_paho_test.py** example you should be able to perform the same subscription to the Topic: “testtopic/#”.

You can follow some guide like the “Beginners Guide To The Paho MQTT Python Client” at: <http://www.steves-internet-guide.com/into-mqtt-python-client/> to perform the 1.1 task. This web page is also attached in the lab.

1.1 Python MQTT Up

Task

Write a MQTT publisher (imagine this is the IoT temperature sensor in fig 1) and a subscriber in Python. The subscribers acts as clients that can connect to the broker and listen to a topic that the publisher sends.

Warning do not not subscribe to ‘#’. This will make your subscriber listen to any topic. But I know you will try it :)

The payload sent via the topic in my example contains the publisher, current date-time, a random temperature and some LoRa variables.

Output from the publisher script could look like this

```
PS C:\tmp> python .\mqtt_device1.py
Connecting to broker: broker.hivemq.com
```

```
[+] Connected to: broker.hivemq.com, port: 1883
Publishing to topic: hajo66/devices/node1/up
Sending messages...
```

Sensor data: 26° C at time: 2019-10-30 10:45:53.97

Message id: 1

Publishing to topic: hajo66/devices/node1/up, JSON payload: {"app_id": "hajo66", "dev_id": "node1", "port/channel": 2, "rssi": -34, "snr": -9, "sf": "SF7BW125", "C_F": "C", "temperature": 26, "time": "2019-10-30 10:45:53.97", "message id/counter": 1}

Sensor data: 44° C at time: 2019-10-30 10:45:58.98

Message id: 2

Publishing to topic: hajo66/devices/node1/up, JSON payload: {"app_id": "hajo66", "dev_id": "node1", "port/channel": 4, "rssi": -8, "snr": 9, "sf": "SF7BW125", "C_F": "C", "temperature": 44, "time": "2019-10-30 10:45:58.98", "message id/counter": 2}

...

Output from the subscriber script could or should look like this

```
PS C:\tmp> python .\mqtt_server1.py
Connecting to broker: broker.hivemq.com
[+] Connected to: broker.hivemq.com, port: 1883
Flags: {'session present': 0}, return code: 0
Subscribed to topic: hajo66/devices/node1/up
Waiting for messages...
```

```
Received topic: hajo66/devices/node1/up with payload: b'{"app_id": "hajo66", "dev_id":
"node1", "port/channel": 2, "rssi": -34, "snr": -9, "sf": "SF7BW125", "C_F": "C",
"temperature": 26, "time": "2019-10-30 10:45:53.97", "message id/counter": 1}', at
subscribers local time: 2019-10-30 10:45:54.02
```

```
Valid JSON: {'app_id': 'hajo66', 'dev_id': 'node1', 'port/channel': 2, 'rssi': -34, 'snr':
-9, 'sf': 'SF7BW125', 'C_F': 'C', 'temperature': 26, 'time': '2019-10-30 10:45:53.97',
'message id/counter': 1}
```

```
Pretty sensor output: Message 1 from hajo66/node1 on port/channel: 2 got temperature: -34°
C at timestamp: 2019-10-30 10:45:53.97
```

```
Received topic: hajo66/devices/node1/up with payload: b'{"app_id": "hajo66", "dev_id":
"node1", "port/channel": 4, "rssi": -8, "snr": 9, "sf": "SF7BW125", "C_F": "C",
"temperature": 44, "time": "2019-10-30 10:45:58.98", "message id/counter": 2}', at
subscribers local time: 2019-10-30 10:45:59.00
```

```
Valid JSON: {'app_id': 'hajo66', 'dev_id': 'node1', 'port/channel': 4, 'rssi': -8, 'snr':
9, 'sf': 'SF7BW125', 'C_F': 'C', 'temperature': 44, 'time': '2019-10-30 10:45:58.98',
'message id/counter': 2}
```

```
Pretty sensor output: Message 2 from hajo66/node1 on port/channel: 4 got temperature: 44° C
at timestamp: 2019-10-30 10:45:58.98
```

...

Report:

With the hints from the output above write the necessary code (a MQTT publisher and a MQTT subscriber in Python) to send a JSON payload (see example above in bold text) with a specified interval and random values for: port/channel, RSSI, SNR, temperature and the current message id/counter.

You do not need to hand in the code and or show in the lab that this solution works. It is sufficient to hand in task 1.2.

1.2 Python MQTT Up and Down

Task

Continue on the 1.1 task with new script filenames and add functionality to both the “sensor” device and the “computer” program. Use an additional topic for down bound traffic. The computer will publish on this topic and the sensor device will subscribe to this topic.

For example when the sensor device sends a message and the computer receive it, the computer should send an ACK (or any message) back to the sensor device which in turn should be able to receive it. In other words we should support both a UP and DOWN messages in MQTT.

When the sensor device have received a message you can sleep for some seconds and then send a message again. In other words they will play ping pong with each other.

Output from the device

```
PS C:\tmp> python .\mqtt_device2.py
Connecting to broker: broker.hivemq.com
[+] Connected to: broker.hivemq.com, port: 1883
Flags: {'session present': 0}, return code: 0
Subscribed to topic: hajo66/devices/node1/down
Sending And Waiting for messages...
```

```
Sensor data: 42° C at time: 2019-10-30 13:17:05.11
Publishing to topic: hajo66/devices/node1/up, JSON payload: {"app_id": "hajo66", "dev_id":
"node1", "port/channel": 6, "rssi": 5, "snr": 49, "sf": "SF7BW125", "C_F": "C",
"temperature": 42, "time": "2019-10-30 13:17:05.11", "message id/counter": 1}
```

```
Received topic: hajo66/devices/node1/down with payload: b'ACK_MSG_RECEIVED', at subscribers
local time: 2019-10-30 13:17:05.18
```

```
Sensor data: 30° C at time: 2019-10-30 13:17:10.18
Publishing to topic: hajo66/devices/node1/up, JSON payload: {"app_id": "hajo66", "dev_id":
"node1", "port/channel": 2, "rssi": -76, "snr": 29, "sf": "SF7BW125", "C_F": "C",
"temperature": 30, "time": "2019-10-30 13:17:10.18", "message id/counter": 2}
```

```
Received topic: hajo66/devices/node1/down with payload: b'ACK_MSG_RECEIVED', at subscribers
local time: 2019-10-30 13:17:10.24
```

...

Output from server

```
PS C:\tmp> python .\mqtt_server2.py
Connecting to broker: broker.hivemq.com
[+] Connected to: broker.hivemq.com, port: 1883
Flags: {'session present': 0}, return code: 0
Subscribed to topic: hajo66/devices/node1/up
Waiting for messages...
```

```
Received topic: hajo66/devices/node1/up with payload: b'{"app_id": "hajo66", "dev_id":
"node1", "port/channel": 6, "rssi": 5, "snr": 49, "sf": "SF7BW125", "C_F": "C",
"temperature": 42, "time": "2019-10-30 13:17:05.11", "message id/counter": 1}', at
subscribers local time: 2019-10-30 13:17:05.15
```

```
Sending ACK To Device: ACK_MSG_RECEIVED
Publishing to topic: hajo66/devices/node1/down, Payload: ACK_MSG_RECEIVED
Message id: 1
```

```
Received topic: hajo66/devices/node1/up with payload: b'{"app_id": "hajo66", "dev_id":
"node1", "port/channel": 2, "rssi": -76, "snr": 29, "sf": "SF7BW125", "C_F": "C",
```

```
"temperature": 30, "time": "2019-10-30 13:17:10.18", "message id/counter": 2}', at  
subscribers local time: 2019-10-30 13:17:10.21
```

```
Sending ACK To Device: ACK_MSG_RECEIVED  
Publishing to topic: hajo66/devices/node1/down, Payload: ACK_MSG_RECEIVED  
Message id: 2
```

Report:

Hand in the code and or show in the lab that your solution works. The function and output rules are exactly like 1.1 but down bound messages should work as well.

It may be fun to test a one publisher to many subscribers scenario if you got time.

Perhaps also with a mobile app as MQTT Dashboard:

<https://play.google.com/store/apps/details?id=com.app.vetru.mqttdashboard> or IoT MQTT Panel: <https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod>.

Howto: In the mobile apps, first add a connection to the same broker the publisher publish messages on. Then add a panel of text log type where you input the topic you want to listen to.

1.3 Docker and container development

Install Docker Desktop and prepare your system so you can build your own images and run containers. See the attached docs: Docker-guide-case_study_v1.pdf and Docker-remove-Cheat-Sheet.pdf.

Review the Read and Watchlist for lab 1 in Learn for essential Docker knowledge.

For the task you can follow a guide like this one: <https://www.docker.com/blog/containerized-python-development-part-1/>

Rename the bundled **Dockerfile.lin** or **Dockerfile.win** to just **Dockerfile** or add the '-f name_of_dockerfile' option to the build command to be able to try the bundled **mqtt_paho_test.py** example.

Build an image and run a Linux container

- `docker build -t mqtt-image-lin .`
- `docker run -it mqtt-image-lin`

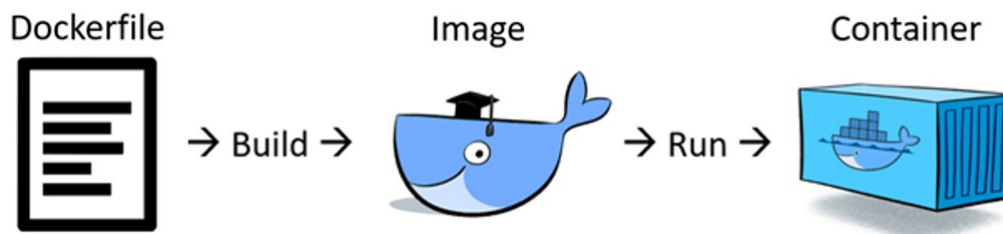


Fig 3. Shows the steps in the process to obtain a Docker container.

Build an image and run a Windows container

To run containers in Windows mode you need to change the mode in Docker Desktop first:
<https://stackoverflow.com/questions/48066994/docker-no-matching-manifest-for-windows-amd64-in-the-manifest-list-entries>

- `docker build -t mqtt-image-win .`
- `docker run -it mqtt-image-win`

If you got time you can try to dockerize your code from 1.1 or 1.2 as well.

Report:

Show with a screenshot that you got your Python code working in a Docker container.

1.4 Arduino MQTT Up (and Down)

Task

This task will unfortunately not work without real hardware and an Ethernet port. The disadvantage with real Ethernet is that one need to put in MAC address and IP-address by oneself and that limits the lab possibilities at school somewhat. This task however works as an introduction for how you can work with Arduino boards on a basic level.

Start the Virtronics Arduino Simulator application. If its not installed you can unzip the Virtronics_Arduino Simulator_v1.11.7z file from here:

<http://users.du.se/~hjo/cs/gmi28v/resources/Arduino/>.

Try some of the bundled Sketches. For example the 01.Basics\blink.ino program. Use for example Uno as hardware.

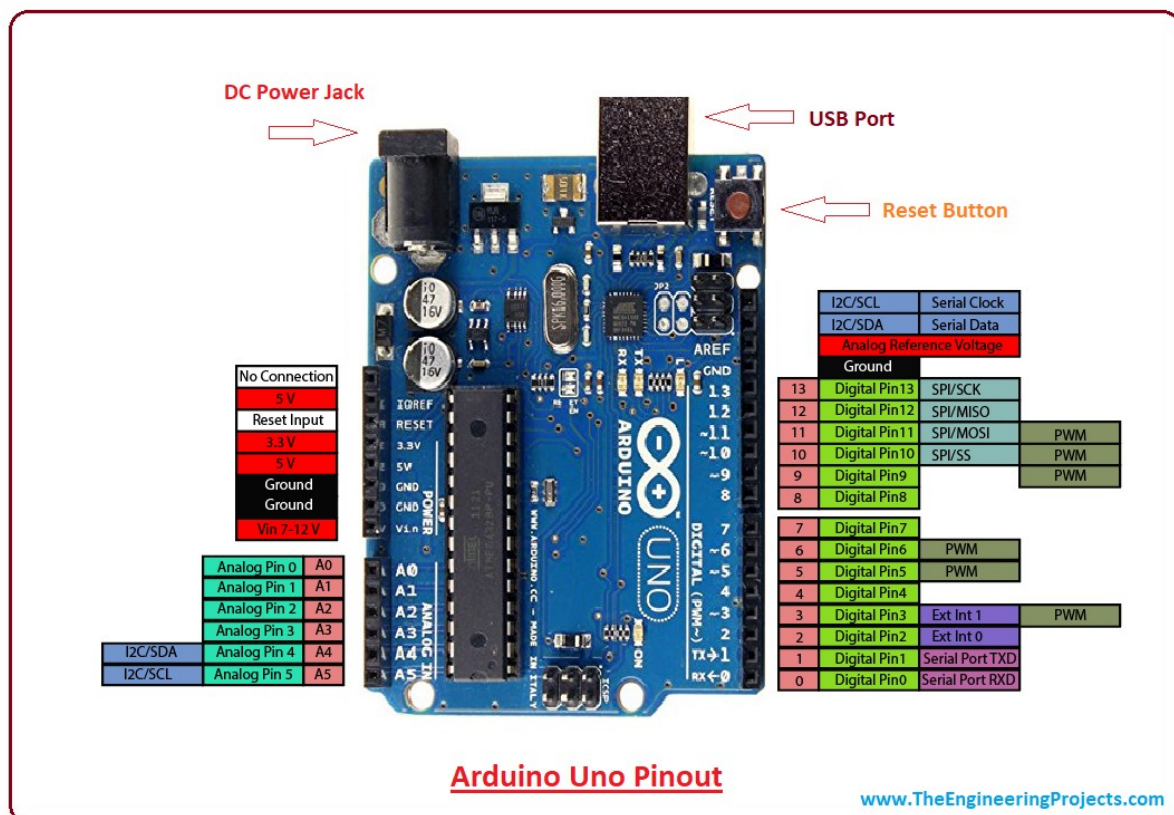


Fig 4. Arduino Uno Pinout.

You can look up the terms here: <https://www.arduino.cc/glossary/en/>
GPIO: https://en.wikipedia.org/wiki/General-purpose_input/output

Load the ReadAnalogVoltage.ino and press the Input/Output button. As you can see you can control the Analog pins value with the dialog slider controls. The current variable values are visible in the center table. You can view the value in the Serial Logger as well.

Load the DigitalReadSerial.ino and now you can control the value (1/0) by pressing the image circuit cards Digital pin.

You can modify the sketch if you press F6. Note that the simulator seems to cache files so you may not be able to edit a sketch in another editor and then open and run it in the simulator.

Note! The following will not work in simulator but you can try it anyway if you want.

Now we will implement the sensor device code in the Arduino Simulator. You must use a MQTT library as: Arduino client for MQTT: <https://pubsubclient.knolleary.net/> or similar.

Copy the Python computer code from task 1.1 or 1.2 to a new file and adjust the code (if needed) so it can subscribe to the Arduino sensor device sending MQTT messages.

Part 1

Steps in Arduino Simulator.

1. Make a folder named MQTT in the \Arduino simulator\Libraries\ folder and copy the PubSubClient.cpp and PubSubClient.h files there.
2. Load the pubsubclient-2.7\examples\mqtt_basic\mqtt_basic.ino sketch. Ignore error and warnings messages (press close) if something occur.
3. When it asks for the <functional> include navigate to the \Arduino simulator\Libraries root folder and press ok. Ignore error and warnings messages (press close) if something occur.
4. Edit the server IP address to your MQTT servers address and the topic outTopic so the program sends messages to your Python scripts topic.
 - Verify that the changes are made, sometimes you need to restart the simulator or save to a new filename.

If you want. Try some more applications as the DHT11 Temperature and Humidity Sensor.

Part 2

Steps in Arduino Simulator.

1. Download the DHT library from: <https://github.com/markruys/arduino-DHT> and use the example code or use the tutorial and example code/library from: <https://www.brainy-bits.com/dht11-tutorial/>
2. Make a DHT folder in the \Arduino simulator\Libraries\ folder and put the DHT.cpp and DHT.h file there. Then open your sketch and run it.

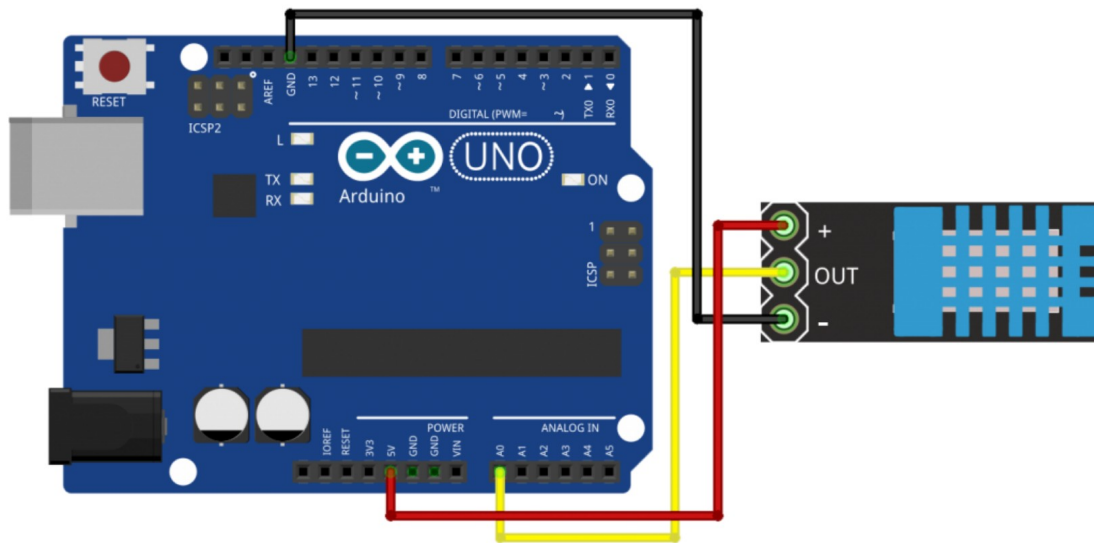


Fig 5. Arduino Uno DHT11 connections.

Part 3

Put the former 2 parts together into one sketch and send MQTT messages to your Python subscriber with Temperature and Humidity Sensor data.

Report:

None.

1.5 Lab feedback

- a) Were the labs relevant and appropriate and what about length etc?
- b) What corrections and/or improvements do you suggest for these labs?