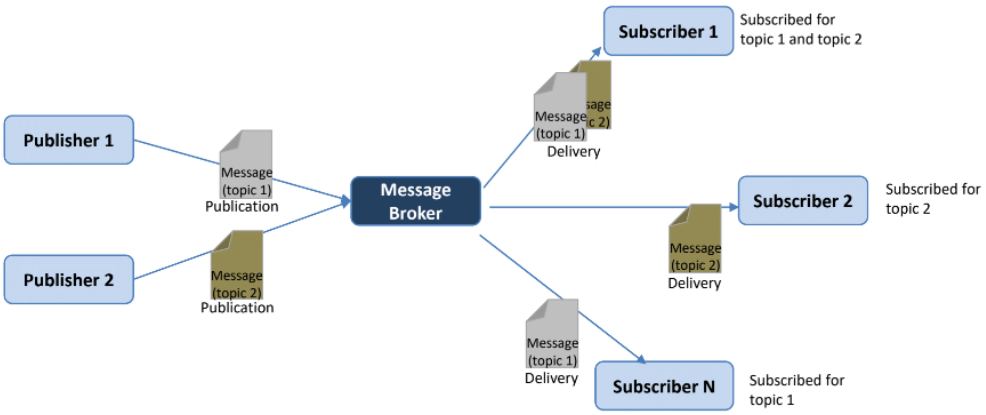
MQTT

August 26, 2019

1 Introduction



Example of MQTT

We could briefly resume the structure of the MQTT communication paradigm in this way, there are 3 type of actors:***Publisher***,***Subscriber***,***Broker***.
The ***Publisher*** is the actor that wants to send messages tagged by a *topic* while the ***Subsciber*** is the actor that wants to receive messages that belong to variable number of topic. The ***Broker*** is the actor in the middle: it receives the messages from all the publisher and forwards each of them to the suscriber according to the *topic*. Here below you can find the examples for the implementation of a publisher and a subscriber

[1]:
```
import paho.mqtt.client as PahoMQTT
import time

class MyPublisher:
    def __init__(self, clientID,broker):
        self.clientID = clientID

        # create an instance of paho.mqtt.client
        self._paho_mqtt = PahoMQTT.Client(self.clientID, False)
        # register the callback
        self._paho_mqtt.on_connect = self.myOnConnect
```

```
        self.messageBroker = broker

    def start (self):
        #manage connection to broker
        self._paho_mqtt.connect(self.messageBroker, 1883)
        self._paho_mqtt.loop_start()

    def stop (self):
        self._paho_mqtt.loop_stop()
        self._paho_mqtt.disconnect()

    def myPublish(self, topic, message):
        # publish a message with a certain topic
        self._paho_mqtt.publish(topic, message, 2)

    def myOnConnect (self, paho_mqtt, userdata, flags, rc):
        print ("Connected to %s with result code: %d" % (self.messageBroker,␣
↪rc))
```

[ ]:
```
class MySubscriber:
        def __init__(self, clientID,topic,broker):
            self.clientID = clientID
            # create an instance of paho.mqtt.client
            self._paho_mqtt = PahoMQTT.Client(clientID, False)

            # register the callback
            self._paho_mqtt.on_connect = self.myOnConnect
            self._paho_mqtt.on_message = self.myOnMessageReceived

            self.topic = topic
            self.messageBroker = broker


        def start (self):
            #manage connection to broker
            self._paho_mqtt.connect(self.messageBroker, 1883)
            self._paho_mqtt.loop_start()
            # subscribe for a topic
            self._paho_mqtt.subscribe(self.topic, 2)

        def stop (self):
            self._paho_mqtt.unsubscribe(self.topic)
            self._paho_mqtt.loop_stop()
            self._paho_mqtt.disconnect()

        def myOnConnect (self, paho_mqtt, userdata, flags, rc):
```

```
            print ("Connected to %s with result code: %d" % (self.
↪messageBroker, rc))

        def myOnMessageReceived (self, paho_mqtt , userdata, msg):
            # A new message is received
            print ("Topic:'" + msg.topic+"', QoS: '"+str(msg.qos)+"' Message:␣
↪'"+str(msg.payload) + "'")
```

1.1 General purpose MQTT implementation

Let's look at the two pieces of code written below

[1]:
```
import paho.mqtt.client as PahoMQTT

class MyMQTT:
    def __init__(self, clientID, broker, port, notifier):
        self.broker = broker
        self.port = port
        self.notifier = notifier
        self.clientID = clientID

        self._topic = ""
        self._isSubscriber = False

        # create an instance of paho.mqtt.client
        self._paho_mqtt = PahoMQTT.Client(clientID, False)

        # register the callback
        self._paho_mqtt.on_connect = self.myOnConnect
        self._paho_mqtt.on_message = self.myOnMessageReceived


    def myOnConnect (self, paho_mqtt, userdata, flags, rc):
        print ("Connected to %s with result code: %d" % (self.broker, rc))

    def myOnMessageReceived (self, paho_mqtt , userdata, msg):
        # A new message is received
        self.notifier.notify (msg.topic, msg.payload)


    def myPublish (self, topic, msg):
        # if needed, you can do some computation or error-check before␣
↪publishing
        print ("publishing '%s' with topic '%s'" % (msg, topic))
        # publish a message with a certain topic
        self._paho_mqtt.publish(topic, msg, 2)
```

```
    def mySubscribe (self, topic):
        # if needed, you can do some computation or error-check before␣
↪subscribing
        print ("subscribing to %s" % (topic))
        # subscribe for a topic
        self._paho_mqtt.subscribe(topic, 2)

        # just to remember that it works also as a subscriber
        self._isSubscriber = True
        self._topic = topic

    def start(self):
        #manage connection to broker
        self._paho_mqtt.connect(self.broker , self.port)
        self._paho_mqtt.loop_start()

    def stop (self):
        if (self._isSubscriber):
            # remember to unsuscribe if it is working also as subscriber
            self._paho_mqtt.unsubscribe(self._topic)

        self._paho_mqtt.loop_stop()
        self._paho_mqtt.disconnect()
```

[3]:
```
from MyMQTT import MyMQTT

class DoSomething():
    def __init__(self, clientID):
        # create an instance of MyMQTT class
        self.clientID = clientID
        self.myMqttClient = MyMQTT(self.clientID, "iot.eclipse.org", 1883, self)



    def run(self):
        # if needed, perform some other actions befor starting the mqtt␣
↪communication
        print ("running %s" % (self.clientID))
        self.myMqttClient.start()

    def end(self):
        # if needed, perform some other actions befor ending the software
        print ("ending %s" % (self.clientID))
        self.myMqttClient.stop ()

    def notify(self, topic, msg):
```

```
        # manage here your received message. You can perform some error-check␣
↪here
        print ("received '%s' under topic '%s'" % (msg, topic))
```

1.2 Exercise 1

Try to create a script that mimics a light that has a status that can be on/off and has to to the topic *led*. Then create a client that uses MQTT to set the status of the light

1.3 Exercise 2

Try to improve the previous exercise by creating a REST client to set the status of the light. You can use the file *'index.html'* as page for the GET request, when you will click on the button the page will execute a PUT request where the uri indicates the status we want to set.

1.4 Exercise 3

Try to create a simple chat client that uses MQTT. We would like two have at leat two client that are subscribed to the same topic (i.e. *"chat"*) but can also publish to this topic. We want to have a client that allows to write a new message only if the last message has been written from another user, that means:

YES

John: Hi

Yoko: Hi, how are you?

John: Good

NO

John: Let

John: it

John: be