



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Implementación de una aplicación basada en MQTT con Python.

**Prácticas de laboratorio
de Diseño de Servicios Telemáticos**

Introducción y Objetivo

El objetivo de la práctica consiste en comprender como funciona el protocolo MQTT utilizando las librerías de las que dispone el lenguaje de programación de alto nivel Python.

Para la realización de la práctica se requiere un conocimiento básico del protocolo MQTT (utilidad, componentes del sistema, tipos de mensajes, topics, etc.) Por tanto, antes de realizarla es necesario que el alumno haya estudiado el punto correspondiente del Tema IoT.

Un par de enlaces con información básica:

- <https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/>
- <https://www.luisllamas.es/que-son-y-como-usar-los-topics-en-mqtt-correctamente>

1 Instalación de la librería MQTT

Como sabemos MQTT (*Message Queuing Telemetry Transport*) es un protocolo de tipo PUB/SUB donde existen distintos elementos:

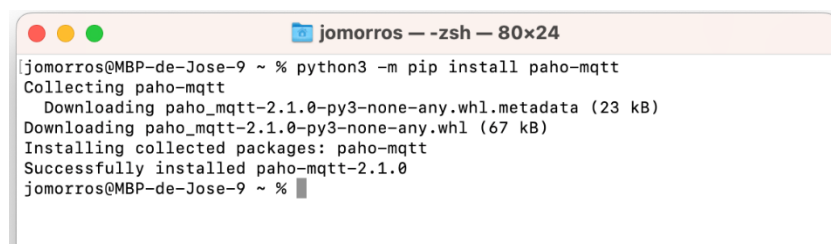
- Clientes ‘Subscriber’, que informan que quieren recibir un tipo de mensaje.
- Clientes ‘Publisher’, que pueden publicar dichos mensajes.
- Un BROKER que recibe los mensajes de los ‘Publisher’ y los distribuye a los ‘Subscriber’.

Para estudiar su funcionamiento vamos a instalar una librería Python que implementa la parte de cliente del protocolo MQTT. La librería se denomina Eclipse Paho y puedes encontrar toda la información sobre ella aquí:

<https://pypi.org/project/paho-mqtt/>

Realice la instalación ejecutando el siguiente comando:

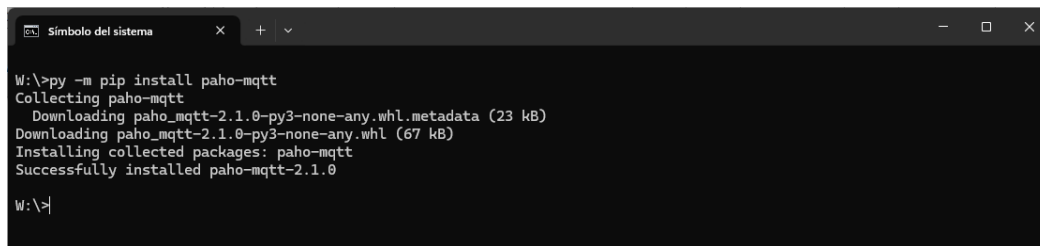
- Linux/macOS (ver Fig. 1): Abra una ventana de Terminal y escriba el comando **python3 -m pip install paho-mqtt**



```
jomorros — zsh — 80x24
jomorros@MBP-de-Jose-9 ~ % python3 -m pip install paho-mqtt
Collecting paho-mqtt
  Downloading paho_mqtt-2.1.0-py3-none-any.whl.metadata (23 kB)
Downloading paho_mqtt-2.1.0-py3-none-any.whl (67 kB)
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-2.1.0
jomorros@MBP-de-Jose-9 ~ %
```

Fig. 1. Instalación de una librería utilizando pip en Linux/macOS.

- Windows (ver Fig. 2): Abra la línea de comandos ejecutando el programa cmd.exe y escriba el comando **pip install paho-mqtt**



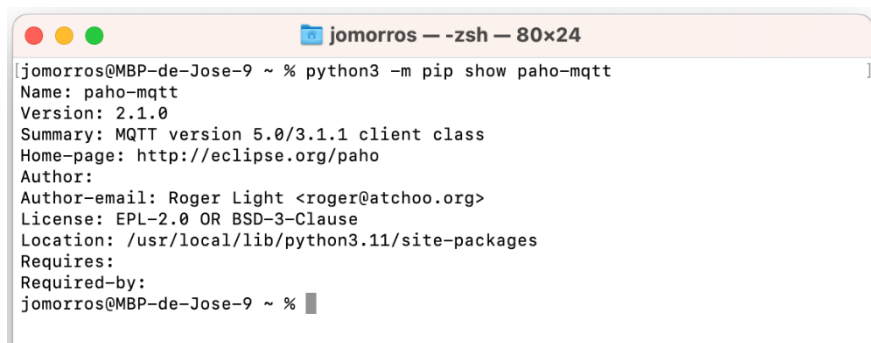
```
W:\>py -m pip install paho-mqtt
Collecting paho-mqtt
  Downloading paho_mqtt-2.1.0-py3-none-any.whl.metadata (23 kB)
  Downloading paho_mqtt-2.1.0-py3-none-any.whl (67 kB)
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-2.1.0
W:\>
```

Fig. 2. Instalación de una librería Python utilizando pip en Windows.

La versión actual de la librería Eclipse Paho es la versión 2.1 que incluye cambios significativos para mejorar el soporte con MQTT v5 respecto a la versión 1.0. Lo que provoca que aquellos programas que funcionaban con la versión de la librería 1.0 dejen de funcionar. Por lo tanto, aquellos sistemas que tengan instaladas versiones de la librería inferiores a la 2.0 es necesario proceder a su actualización.

Para comprobar que versión de la librería esta instalada en su equipo puede ejecutar el siguiente comando:

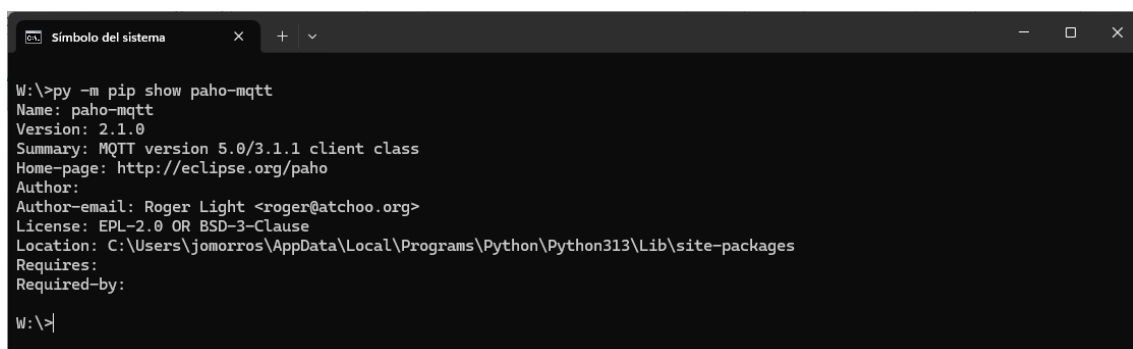
- Linux/macOS (ver Fig. 3): Abra una ventana de Terminal y escriba el comando `python3 -m pip show paho-mqtt`



```
jomorros@MBP-de-Jose-9 ~ % python3 -m pip show paho-mqtt
Name: paho-mqtt
Version: 2.1.0
Summary: MQTT version 5.0/3.1.1 client class
Home-page: http://eclipse.org/paho
Author:
Author-email: Roger Light <roger@atchoo.org>
License: EPL-2.0 OR BSD-3-Clause
Location: /usr/local/lib/python3.11/site-packages
Requires:
Required-by:
jomorros@MBP-de-Jose-9 ~ %
```

Fig. 3. Versión de la librería Eclipse Paho instalada en el sistema (Linux/macOS).

- Windows (ver Fig. 4): Abra la línea de comandos ejecutando el programa cmd.exe y escriba el comando `py -m pip show paho-mqtt`

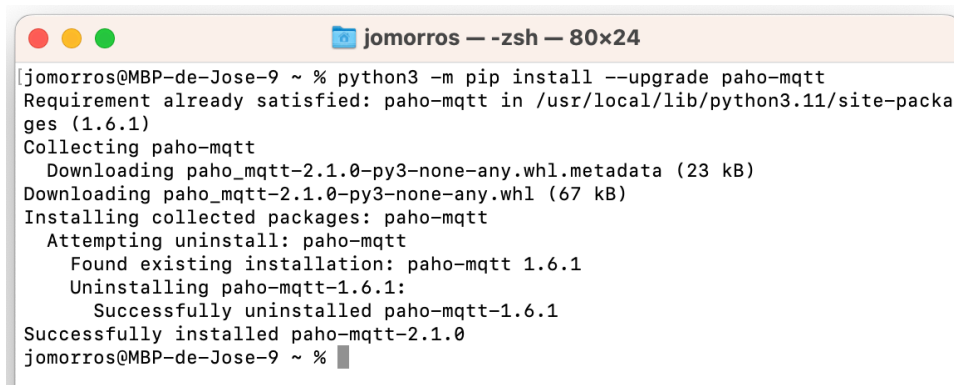


```
W:\>py -m pip show paho-mqtt
Name: paho-mqtt
Version: 2.1.0
Summary: MQTT version 5.0/3.1.1 client class
Home-page: http://eclipse.org/paho
Author:
Author-email: Roger Light <roger@atchoo.org>
License: EPL-2.0 OR BSD-3-Clause
Location: C:\Users\jomorros\AppData\Local\Programs\Python\Python313\Lib\site-packages
Requires:
Required-by:
W:\>
```

Fig. 4. Versión de la librería Eclipse Paho instalada en el sistema (Windows).

Para proceder a la actualización de la librería, en el caso que la versión instalada de la librería sea inferior a la 2.0, deberá ejecutar el siguiente comando:

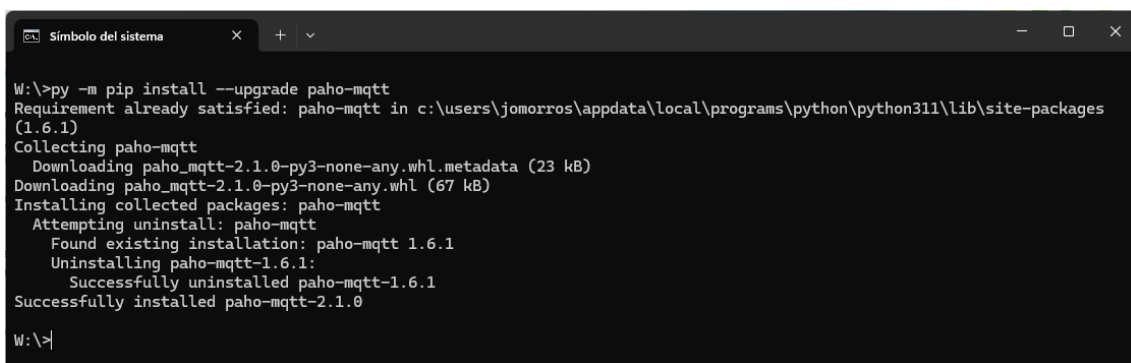
- Linux/macOS (ver Fig. 5): Abra una ventana de Terminal y escriba el comando `python3 -m pip install --upgrade paho-mqtt`



```
jomorros — -zsh — 80x24
[jomorros@MBP-de-Jose-9 ~ % python3 -m pip install --upgrade paho-mqtt
Requirement already satisfied: paho-mqtt in /usr/local/lib/python3.11/site-packa
ges (1.6.1)
Collecting paho-mqtt
  Downloading paho_mqtt-2.1.0-py3-none-any.whl.metadata (23 kB)
  Downloading paho_mqtt-2.1.0-py3-none-any.whl (67 kB)
Installing collected packages: paho-mqtt
  Attempting uninstall: paho-mqtt
    Found existing installation: paho-mqtt 1.6.1
    Uninstalling paho-mqtt-1.6.1:
      Successfully uninstalled paho-mqtt-1.6.1
Successfully installed paho-mqtt-2.1.0
jomorros@MBP-de-Jose-9 ~ %
```

Fig. 5. Actualización de la librería Eclipse Paho en Linux/macOS.

- Windows (ver Fig. 6): Abra la línea de comandos ejecutando el programa cmd.exe y escriba el comando `py -m pip install --upgrade paho-mqtt`



```
W:\>py -m pip install --upgrade paho-mqtt
Requirement already satisfied: paho-mqtt in c:\users\jomorros\appdata\local\programs\python\python311\lib\site-packages
(1.6.1)
Collecting paho-mqtt
  Downloading paho_mqtt-2.1.0-py3-none-any.whl.metadata (23 kB)
  Downloading paho_mqtt-2.1.0-py3-none-any.whl (67 kB)
Installing collected packages: paho-mqtt
  Attempting uninstall: paho-mqtt
    Found existing installation: paho-mqtt 1.6.1
    Uninstalling paho-mqtt-1.6.1:
      Successfully uninstalled paho-mqtt-1.6.1
Successfully installed paho-mqtt-2.1.0
W:\>
```

Fig. 6. Actualización de la librería Paho MQTT en Windows.

2 Implementación de un PUBLISHER y un SUBSCRIBER.

A continuación, se va a implementar un publicador básico. Cree un archivo en Python con el nombre `pub.py` y con el siguiente código:

```
1. import paho.mqtt.client as mqtt
2.
3. #MQTT_BROKER = "test.mosquitto.org"
4. MQTT_BROKER = "158.42.32.220"
5. MQTT_PORT = 1883
6.
7. MQTT_TOPIC = "salon/luz"
8. MQTT_MSG = input("Introduzca valor ON/OFF:")
9.
10. cliente = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
11.
12. cliente.connect(MQTT_BROKER, MQTT_PORT)
13. print("conectado al Broker: ", MQTT_BROKER)
14.
15. cliente.publish(MQTT_TOPIC, MQTT_MSG)
16. print("publicado el mensaje: ",MQTT_MSG," en el topic: ",MQTT_TOPIC)
17. cliente.disconnect()
18.
```

La explicación del código es sencilla. En primer lugar, vamos a importar la clase `paho.mqtt.client` (línea 1).

En las líneas 4 y 5 indicamos la dirección del broker y el puerto. Existen brokers públicos para la realización de pruebas como el indicado en la línea comentada 3. En

nuestro caso vamos a usar como bróker la dirección 158.42.32.220 que se corresponde con una máquina donde se ha instalado un bróker MQTT Mosquitto. Mosquitto es una de las implementaciones más populares existiendo versiones para todos los sistemas operativos. Opcionalmente puede instalarse un bróker en su máquina siguiendo las indicaciones de su web oficial: <https://mosquitto.org/>

El siguiente paso es definir el Topic donde queremos publicar, y preguntamos el mensaje a enviar. Para finalizar, en la línea 10 creamos el objeto cliente de tipo `mqttn.Client`, en la línea 12 realizamos la conexión con el bróker, en la 14 publicamos el mensaje en el Topic que habíamos definido previamente y terminamos en la línea 17 donde nos desconectamos.

Podemos ejecutar el código para comprobar que no existe ningún error de sintaxis y que nos podemos conectar correctamente al Broker.

A continuación, vamos a implementar el subscriptor. Cree un archivo en Python con el nombre `sub.py` y con el siguiente código:

```
1. import paho.mqtt.client as mqtt
2. import time
3.
4. def on_message(client, userdata, message):
5.     print("mensaje recibido: " + str(message.payload.decode("utf-8"))) + " en el topic: " + message.topic)
6.
7. #MQTT_BROKER = "test.mosquitto.org"
8. MQTT_BROKER = "158.42.32.220"
9. MQTT_PORT = 1883
10.
11. MQTT_TOPIC = "salon/#"
12.
13. cliente = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
14. cliente.on_message = on_message
15.
16. cliente.connect(MQTT_BROKER, MQTT_PORT)
17. print("conectado al broker: ", MQTT_BROKER)
18.
19. cliente.subscribe(MQTT_TOPIC)
20. print("subscrito al topic: ",MQTT_TOPIC)
21.
22. cliente.loop_start()
23. time.sleep(15) #15 segundos
24. cliente.loop_stop()
25.
26. cliente.disconnect()
27.
```

La principal diferencia es que ahora vamos a usar lo que se denomina "*callbacks*". *Callback* es una función que es llamada cuando se produce un evento. El cliente Paho MQTT tiene implementadas diferentes funciones *callbacks* como son `on_connect`, `on_subscribe`, `on_publish`, `on_message`, etc. Para poder utilizarlas primero hay que crear una función *callback* y además hay asignar la función creada a la *callback* del cliente.

Así, en la línea 4 definimos una función `on_message`. Podríamos asignarle cualquier nombre, pero suele usarse el mismo nombre que la función *callback* a la que la asignaremos. Esa asignación se realiza en la 14. El resultado es que cuando se recibe un mensaje (porque estamos suscritos a un Topic) se ejecuta la función `on_message` que según la línea 5 se encarga únicamente de mostrar un mensaje por pantalla.

Tenemos que dar un tiempo para que el cliente reciba los mensajes a los que está suscrito. Para ello utilizamos la función `loop_start()`. Eso hace que se ejecute un hilo separado que será el encargado de recibir los eventos que generan las *callbacks*. Así las *callbacks* pueden ocurrir en cualquier momento durante la ejecución del código y son asíncronas. En este caso el hilo se cierra con `loop_stop()` desde el hilo principal después de esperar 15 segundos (puede cambiar el valor por el que quiera).

A continuación, se muestran un par de enlaces para afianzar estos conceptos:

- <http://www.steves-internet-guide.com/into-mqtt-python-client/>
- <http://www.steves-internet-guide.com/mqtt-python-callbacks/>

Podemos crear un script con el siguiente código que nos ayudará a entender la idea de las funciones *callback* y la función *loop*.

Básicamente aquí creamos un cliente MQTT que se conecta con el Broker y entra en un bucle `while` donde espera 0.1 segundos e imprime un contador. Se ha llamado a la función *loop* con `loop_start()` que se encarga de iniciar un hilo paralelo para poder capturar los eventos que invocan a las *callbacks*. Hemos definido la función *callback* `on_connect` que se activa al conectarse con el Broker. Esta función `on_connect` únicamente muestra un mensaje en pantalla y modifica la variable `Flag` que nos permite que salgamos del bucle `while`.

```
1. import paho.mqtt.client as mqtt
2. import time
3.
4. def on_connect(client, userdata, flags, reason_code, properties):
5.     global Flag
6.     print(" \n Estoy en on_connect callback ")
7.     Flag=0
8.
9. broker_address="158.42.32.220"
10.
11. client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
12. client.on_connect=on_connect
13. client.loop_start()
14.
15. client.connect(broker_address)
16.
17. Flag=1
18. counter=0
19.
20. while Flag==1:
21.     print("Esperando la callback on_connect ",counter)
22.     time.sleep(.1) #espero 0.1 seg.
23.     counter+=1
24.     if counter >50:
25.         print("algo va mal")
26.         break
27.
28. client.disconnect()
29. client.loop_stop()
30.
```

Para crear un nivel de control de qué dispositivos operan con el servidor MQTT, el protocolo permite una validación por usuario y contraseña. Para realizar esto en el archivo de configuración del Broker Mosquitto se le han añadido varias líneas de configuración para indicarle el archivo donde están los usuarios y sus passwords, para decirle que vamos a permitir conectarse sin usuario (como hemos hecho hasta ahora),

y para indicarle el archivo con los permisos que tiene cada usuario incluido los permisos cuando no indicamos usuario.

A continuación, vamos a estudiar cómo nos conectamos al Broker por medio de un usuario y contraseña. Primero vamos a volver a ejecutar el subscritor y publicador anterior cambiando el topic por DST/PRUEBA. ¿El subscritor recibe el valor enviado al topic?. El problema es que al conectarse sin usuario el Broker Mosquitto tiene configurado únicamente permisos para publicar/subscribirse al topic salon/#. Añada tanto al publicador como al subscritor la línea siguiente, justo antes de hacer el método connect:

```
cliente.username_pw_set(username="dst",password="dst")
```

3 Trabajo a realizar

Los profesores hemos implementado una aplicación basada en un cliente MQTT que explicamos a continuación. La aplicación genera una serie de códigos aleatorios de 5 cifras, de un solo uso y que incluyen un mecanismo de integridad. Estos códigos son publicados en el Topic **DST/CODIGO**.

Para no inundar la red con mensajes de publicación con los códigos, la aplicación de los profesores está suscrita al Topic **DST/PETICION** y solamente cuando recibe un mensaje en ese topic con el **dni válido** de un alumno matriculado en la asignatura (tal como lo utiliza la UPV: sin la letra final, ni espacios) genera y publica un código. El dni es guardado en un archivo log como herramienta de control.

Finalmente, la aplicación está suscrita también al Topic **DST/SOLUCION**. En ese Topic se reciben mensajes del tipo **xxxxx;nombre_del_alumno;alumno@teleco.upv.es** donde xxxxx se corresponde con el valor de un código válido. Por ejemplo, un mensaje recibido podría ser: 12345;Peter Lim;pelim@teleco.upv.es

Es importante destacar la importancia del formato del mensaje. Por ejemplo, la aplicación usa los caracteres “;” para diferenciar los distintos campos. Espacios antes del código, por ejemplo, harán que el código se interprete como no válido.

Al recibir un mensaje en el topic **DST/SOLUCION** la aplicación del profesor realiza una serie de comprobaciones que indicamos a continuación:

1. Comprueba que el código es válido.
2. Comprueba que ese código ha sido generado y enviado por ella.
3. Comprueba que el código no ha sido utilizado previamente.

Si todas las comprobaciones son correctas la aplicación incorpora el nombre del alumno a un fichero que sirve para certificar que el alumno en cuestión ha realizado la práctica. Por último, y como confirmación para el alumno, la aplicación envía un email a la dirección del alumno suministrada.

Se muestra a continuación el código de la aplicación hecha por los profesores y que acabamos de explicar. **No es el que tenéis que implementar**. Para facilitar la comprensión del mismo, y centrarnos en el protocolo MQTT, hemos ocultado las definiciones de las funciones que generan un código y que comprueban que el código recibido cumple los criterios ya mencionados.

```

MQTT_BROKER = "158.42.32.220"
MQTT_PORT = 1883

MQTT_TOPIC = "DST/PETICION"
MQTT_TOPIC2 = "DST/CODIGO"
MQTT_TOPIC3 = "DST/SOLUCION"

def codigo():
    # devuelve un código de 5 cifras que incluye un sistema de Integridad

def comprobacion(mensaje):
    # devuelve 1 si el código incluido en el mensaje es correcto
    # devuelve 0 si el código ya ha sido utilizado
    # devuelve 0 si el código no había sido generado por él.

def lista(dni):
    # devuelve 1 si el dni es de un alumno matriculado

def envio_email(email_to):
    # intenta enviar un email a la dirección suministrada

def on_message(client, userdata, message):
    if message.topic == MQTT_TOPIC:
        N_DNI=str(message.payload.decode("utf-8")).strip()
        if lista(N_DNI) == 1:
            cliente.publish(MQTT_TOPIC2, codigo())

    if message.topic == MQTT_TOPIC3:
        m_recibido = str(message.payload.decode("utf-8"))
        m_lista=m_recibido.split(";")
        if len(m_lista)==3:
            if comprobacion(m_lista[0]) == 1:
                f = open("ok.txt", "a")
                f.write (m_recibido + "\n")
                f.close()
                email_alumno=m_lista[2].strip()
                ack_correo=envio_email(email_alumno)

cliente = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
cliente.on_message = on_message
cliente.username_pw_set(username=ADMIN_USER,password=ADMIN_PASS)

cliente.connect(MQTT_BROKER, MQTT_PORT)
print("conectado al broker: ", MQTT_BROKER)

cliente.subscribe(MQTT_TOPIC)
cliente.subscribe(MQTT_TOPIC3)

cliente.loop_forever()

```

Por tanto, **el alumno tiene que realizar una aplicación basada en un cliente MQTT que realice lo siguiente:**

- El cliente publica en el Topic correspondiente solicitando un código con su número de dni sin letra.
- El cliente se subscribe al Topic correspondiente para recibir el código.
- Su aplicación crea un mensaje con el formato:
codigo_recibido;nombre_alumno;alumno@teleco.upv.es. Repetimos que es

importante que el formato sea correcto puesto que la aplicación que hemos creado descartará el mensaje si no puede validar el código.

- El cliente publica el mensaje creado en el Topic correspondiente.
- El cliente tendrá que conectarse con el Broker usando el usuario y password “dst”. Este usuario tiene los permisos necesarios para realizar la práctica, pero por ejemplo no puede publicar códigos, o subscribirse al topic de peticiones para ver el dni de un compañero.

3.1 Evaluación

La idea es que todas las tareas que se tienen que realizar se implementen en un único código. Este código se puede realizar únicamente con comandos Python mostrados en este enunciado.

Sin embargo, es evidente que la tarea es mucho más sencilla si se realiza con 2 o 3 códigos independientes. Por ejemplo, introduciendo a mano el mensaje a enviar con el código recibido anteriormente en otro script.

Por tanto, se valorará la práctica con 0.3 puntos cualquier alumno cuyo nombre esté en el fichero de mensajes recibidos. Si el alumno ha realizado la tarea utilizando un único código la puntuación será 0.4 puntos. En este caso hay que enviar el código realizado a la cuenta de correo practica.dst@gmail.com. Los códigos recibidos se pasarán por una aplicación antiplagio para comprobar su originalidad. Simplemente recordaros el funcionamiento de un XOR: Dos valores iguales dan resultado 0.