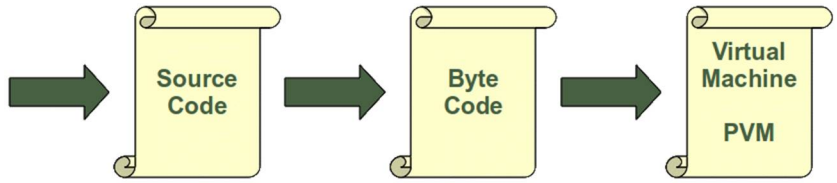
Pengantar Python

Python adalah bahasa berorientasi objek dan open source yang dikembangkan pada 1980-an oleh orang Belanda, Guido van Rossum. Raksasa teknologi seperti Cisco, IBM, Mozilla, Google, Quora, Hewlett-Packard, Dropbox, dan Qualcomm menggunakan bahasa ini karena kesederhanaan dan keanggunannya. Sebagian besar pengembang lebih suka Python daripada kebanyakan bahasa pemrograman di luar sana karena penekanannya pada keterbacaan dan efisiensi. Ada beberapa alasan mengapa Anda harus mempertimbangkan pelatihan Python.

Ketika script dijalankan, Python akan melakukan tahap kompilasi. Tahap ini tersembunyi dari user. Bila Python memiliki write-access, dia akan menyimpan byte code hasil kompilasi dengan akhiran .pyc. Bila tidak memiliki hak write-access, program akan tetap berjalan namun byte code yang dihasilkan akan diabaikan ketika program selesai. Ketika sebuah program Python dipanggil, Python akan memeriksa, apakah terdapat versi kompilasi dari program tersebut. Bila file .pyc dari file program tersebut harus lebih baru dibanding file .py. Bila ada, Python akan me-load byte code, yang akan mempercepat waktu start up. Bila tidak ada, Python akan membuat byte code sebelum meng-eksekusi program. Eksekusi sebuah program Python adalah berarti meng-eksekusi byte code dalam Python Virtual Machine (PVM).



Hello World

```
import time

try:
  while True:
    print(“Hello World”)
    time.sleep(2)
except:
  pass
```

Multi Thread

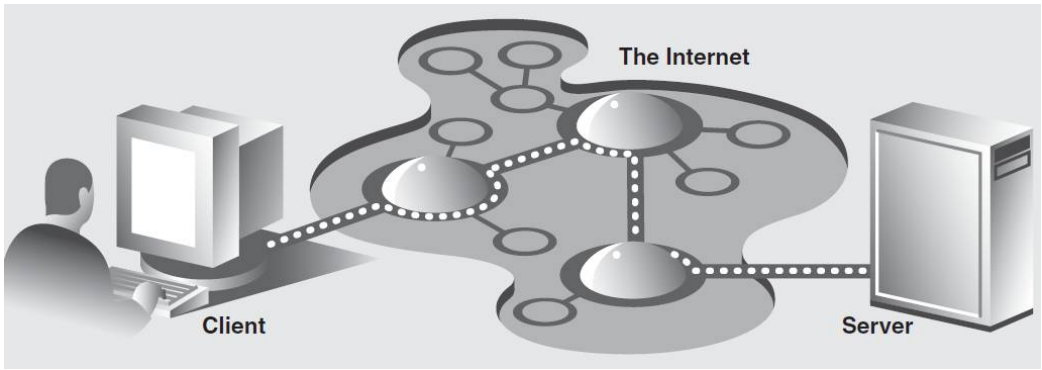
Threading dalam Python digunakan untuk menjalankan beberapa thread (task, pemanggilan fungsi) secara concurrent. Harap dicatat bahwa ini tidak berarti bahwa mereka dijalankan pada CPU yang berbeda. Thread tidak akan mempercepat jalannya program jika telah menggunakan 100% waktu CPU. Dalam kasus demikian, kita perlu melihat pemrograman secara paralel.

Thread pada Python digunakan dalam kasus dimana eksekusi dari sebuah task disertai kondisi menunggu. Sebagai contoh, interaksi dengan sebuah service yang berjalan di komputer lain, misalnya web server. Threading memungkinkan python untuk meng-eksekusi kode lain sementara menunggu, yang dapat di-simulasikan secara sederhana dengan fungsi sleep().

```
import time
import threading

def thread1():
  while True:
    print(“Hello World #1”)
    time.sleep(2)

def thread2():
  while True:
    print(“Hello World #2”)
    time.sleep(1)

th01 = threading.Thread(target=thread1)
th01.daemon = True
th01.start()
th02 = threading.Thread(target=thread2)
th02.daemon = True
th02.start()

try:
  while True:
    print(“Main Loop”)
    time.sleep(3)
except:
  pass
```

Python dan Socket

Python menyediakan dua tingkat akses ke layanan jaringan. Pada low level, kita dapat mengakses socket support pada ranah sistem operasi, yang mana memungkinkan kita untuk mengimplementasi client dan server untuk *connection-oriented* dan *connection-less* protocol.



Python juga menyediakan library untuk high level access ke protokol jaringan spesifik pada tingkat aplikasi, seperti FTP, HTTP dan yang lainnya.

Socket adalah titik akhir dari kanal komunikasi dua arah. Socket dapat berkomunikasi di dalam sebuah proses, antar proses pada mesin yang sama, atau antar proses yang berbeda wilayah.

Socket dapat di-implementaikan melalui sejumlah tipe kanal yang berbeda: UNIX domain socket, TCP, UDP, dan yang lainnya. Library socket menyediakan class spesifik untuk menangani transport, juga antarmuka yang umum untuk menangani setelahnya.

Socket UDP lebih sederhana dalam bekerja karena *connection-less*. Sebuah server UDP cukup memiliki sebuah socket dan menunggu data tiba, dan sebuah socket client mengirim data pada sebuah sebuah socket tanpa koneksi. Socket UDP digunakan pada komunikasi jaringan dimana kehandalan tidak terlalu penting dengan sejumlah alasan.

Untuk membuat sebuah socket dalam Python, kita dapat menggunakan method socket():

```
socket(socket_family, socket_type)
```

Parameter socket_family memiliki pilihan `AF_UNIX`, `AF_INET` atau `AF_INET6`. Ada beberapa socket_type, dua yang utama adalah `SOCK_STREAM` untuk socket TCP dan `SOCK_DGRAM` untuk socket UDP.

Untuk membuat sebuah socket TCP/IP:

```
import socket

tcpSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Untuk membuat sebuah socket UDP/IP:

```
import socket

udpSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

** TCP Server

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((“0.0.0.0”, 5500))
sock.listen(1)
conn, addr = sock.accept()

while True:
  data = conn.recv(1024)
  if not data:
    break
  conn.sendall(data)

conn.close()
```

** TCP Client

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((“192.168.1.80”, 5500))
sock.send(“Hello”)
data = sock.recv(1024)
print(data)
sock.close()
```

** UDP Server

```
import socket
from time import ctime

UDPSock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
UDPSock.bind(("0.0.0.0",5556))

while True:
  print(“Waiting for message ...”)
  data, addr = UDPSock.recvfrom(1024)
  UDPSock.sendto(data, addr)
  print(“Received from and returned to “, addr)

UDPSock.close()
```

** UDP Client

```
import socket

UDPSock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
while True:
  data = input(“> “)
  if not data:
    break
  UDPSock.sendto(data, ("192.168.0.10",5556))
```

```
  data, addr = UDPSock.recvfrom(1024)
  if not data:
    break
  print(data)

UDPSock.close()
```

Flask

Python adalah salah satu bahasa yang digunakan secara luas dalam pengembangan web, dan Django, secara de facto, adalah framework pengembangan web. Namun, masih ada sejumlah framework lain yang tidak terlalu populer, namun mendapat apresiasi dari penggunanya. Satu framework yang popularitasnya sedang naik adalah Flask. Flask juga digunakan secara luas untuk membuat **RESTful API** yang sederhana dan mudah.

Flask adalah sebuah web application framework yang ditulis dengan Python. Flask didasarkan pada Werkzeug WSGI toolkit dan Jinja2 template engine. Keduanya adalah proyek dari Pocco.

```
$ sudo pip install flask
```

Dalam contoh berikut, kita men-definisikan sebuah route / dan sebuah fungsi untuk memanggil hello:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
  return "Welcome to Python Flask App!"

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Host = 0.0.0.0 berarti, aplikasi web dapat di-akses oleh seluruh divais dalam jaringan. Bila, untuk tujuan tertentu, server hanya bisa di-akses secara lokal, gunakan opsi host = 127.0.0.1.

```
@app.route('/cakes')
def cakes():
  return 'Yummy cakes!'
```

Arahkan ke alamat - `http://127.0.0.1:5000/cakes`, dan di layar browser akan tampil pesan Yummy cakes!

Secara normal, Flask berjalan dalam mode single-threaded, dan hanya dapat menangani satu request dalam satu waktu, dan permintaan paralel harus menunggu sampai mereka dapat ditangani. Untuk menangani masalah ini cukup mudah, hanya perlu menempatkan threaded=True dalam script, sehingga Flask akan menangani setiap request dalam thread yang berbeda.

```
app.run(host='0.0.0.0', port=80, debug=True, threaded=True)
```

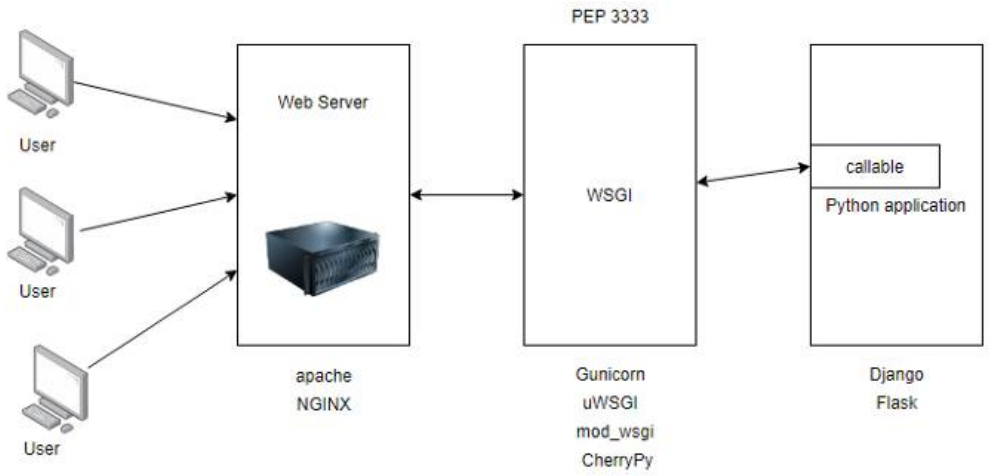
WSGI Server

Ketika aplikasi di run, muncul satu warning:

```
* Serving Flask app "myproject" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Waitress: WSGI Server untuk Python

WSGI Web Server Gateway Interface adalah sebuah spesifikasi yang menjelaskan komunikasi antara web server dan Python web application atau framework. Dia menjelaskan bagaimana sebuah web server berkomunikasi dengan python web application / framework dan bagaimana web application / framework dapat diikat untuk memroses sebuah request.



Instalasi:

```
$ pip install waitress
```

Berikut contoh penggunaannya:

```
from flask import Flask
from waitress import serve

app = Flask(__name__)

@app.route("/")
def hello():
  return "Hello World!"

if __name__ == "__main__":
  #app.run()
  serve(app, host='0.0.0.0', port=5000)
```

Waitress bekerja di lingkungan multithread. Default, menggunakan 4 threads. Jumlah thread dapat disesuaikan:

```
  serve(app, host='0.0.0.0', port=5000, threads=10)
```

HTTP Methods

Protokol HTTP adalah fondasi dari komunikasi data di world wide web.

Secara default, Flask route me-respons terhadap GET request. Namun, preferensi dapat disesuaikan dengan memberikan argumen method ke dekorator route().

```
from flask import Flask, request

app = Flask(__name__)

@app.route("/hello")
def hello():
   _nama = request.args.get('nama')
   if _nama == None:
     return "Hello"
   return _nama

if __name__=='__main__':
   app.run(host='0.0.0.0', port=5000, debug=True)
```

Buka browser, kemudian tuliskan URL berikut:

```
http://localhost:5000/hello?nama=HelloWorld
```

... maka di layar browser akan muncul tulisan HelloWorld.

Contoh lain:

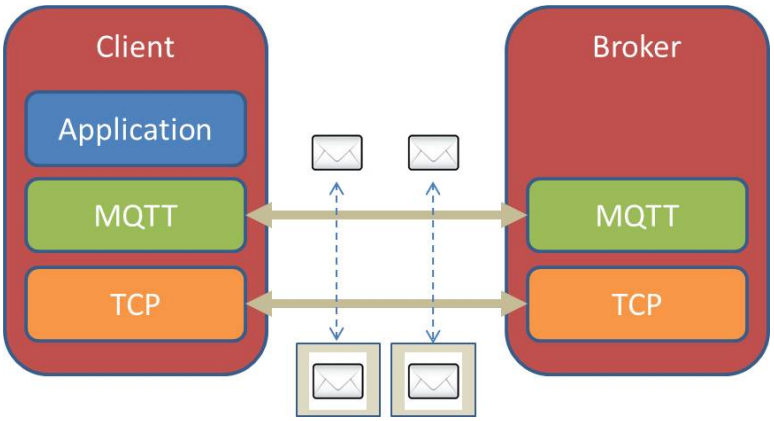
```
from flask import Flask, request

app = Flask(__name__)

@app.route("/hello")
def hello():
   _nama = request.args.get('username')
   _pword = request.args.get('password')
   return “OK”

if __name__=='__main__':
   app.run(host='0.0.0.0', port=5000, debug=True)
```

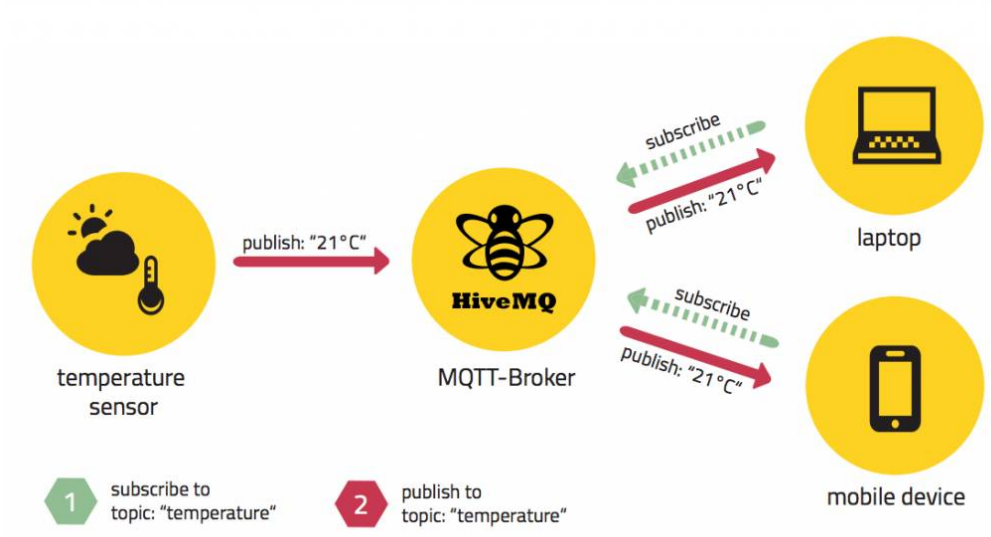
Buka browser, kemudian tuliskan URL berikut:

```
http://localhost:5000/hello?username=hello&password=world
```

Protokol MQTT

Protokol MQTT adalah protokol konektivitas *machine-to-machine*, menggunakan konsep *publish / subscribe messaging transport*.





The Server MUST allow ClientIds which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

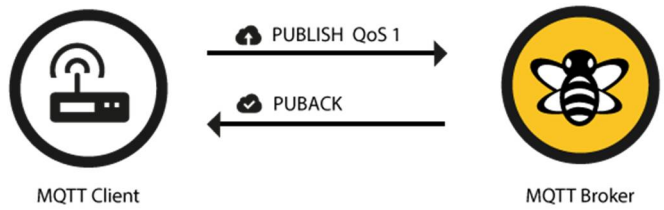
Quality of Service

Tingkat Quality of Service (QoS) adalah sebuah persetujuan antara pengirim dan penerima terkait dengan jaminan dalam pengiriman sebuah message.

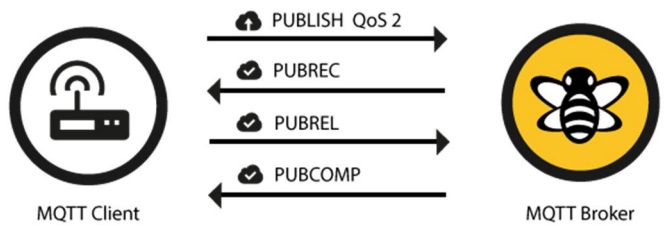
- 0 : Broker / Client akan mengirim message satu kali. Message tidak akan di-acknowledge oleh penerima atau disimpan dan dikirim ulang oleh pengirim. Level ini sering disebut “fire

and forget" dan menyediakan garansi yang sama sebagaimana yang ditetapkan oleh protokol TCP.

- 1 : Broker / Client akan mengirim message paling tidak satu kali. Pengirim akan menyimpan message sampai menerima *acknowledgement* dari penerima.



- 2 : Broker / Client akan mengirim message satu kali dengan menggunakan empat langkah handshake . Tingkat QoS ini adalah yang paling aman sekaligus paling lambat.



Pustaka MQTT Client

Instalasi

```
pip install paho-mqtt
```

Untuk upgrade:

```
pip install paho-mqtt --upgrade
```

Upgrade pip bisa dilakukan dengan cara:

```
python –m pip install --upgrade pip
```

Fungsi Callback

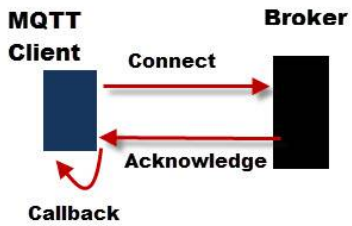
Callback adalah fungsi yang dipanggil dalam me-respons sebuah event. Berikut adalah sejumlah event dan callback yang didukung Paho MQTT client:

- Event Connection acknowledged Triggers the `on_connect` callback
- Event Disconnection acknowledged Triggers the `on_disconnect` callback
- Event Subscription acknowledged Triggers the `on_subscribe` callback
- Event Un-subscription acknowledged Triggers the `on_unsubscribe` callback
- Event Publish acknowledged Triggers the `on_publish` callback
- Event Message Received Triggers the `on_message` callback
- Event Log information available Triggers the `on_log` callback

Paho client dirancang untuk hanya menggunakan fungsi callback jika ada, dan tidak menyediakan fungsi callback default.

Connect

Terdapat sejumlah cara untuk terhubung dengan sebuah broker. Berikut adalah sebuah potongan contoh program:




```
import paho.mqtt.client as paho

def on_connect(client, userdata, flags, rc):
    print("CONNACK received with code %d." % (rc))

client = paho.Client("NSOneClient")
client.on_connect = on_connect
client.connect("broker.hivemq.com", 1883)
```

Konstruktor Client() memiliki beberapa parameter opsional:

```
paho.Client(client_id="", clean_session=True, userdata=None,
protocol=paho.MQTTv31, transport="tcp")

paho.Client(client_id="", clean_session=True, userdata=None,
protocol=MQTTv311, transport="tcp")
```

Dalam satu broker, ClientID harus unik.

Fungsi `connect()` akan menunggu hingga koneksi ke broker berhasil dibuat, dan setelahnya akan memanggil callback `on_connect()`.

MQTT menggunakan koneksi TCP/IP. Koneksi ini normalnya dibiarkan terbua oleh client, sehingga dapat digunakan untuk mengirim dan menerima data setiap saat. Jika tidak ada data yang mengalir pada periode waktu tertentu, maka client akan mengirimkan `PINGREQ` dan berharap untuk menerima `PINGRESP` dari broker. Hal ini untuk memastikan bahwa koneksi terbuka dan bekerja. Periode ini dikenal dengan keep alive period.

Default keep alive period pada Python MQTT client adalah 60 detik.

Fungsi connect dapat dipanggil dengan 4 parameter seperti berikut:

```
connect(host, port=1883, keepalive=60, bind_address="")
```

Ketika menggunakan Python MQTT client, kita tidak perlu membuat ping message, karena langkah ini telah ditangani oleh client ketika kita memanggil fungsi loop.

Kita dapat melihat ping request dan respons pada server dan client. Untuk melihatnya pada server, berikan opsi verbose ketika memulai mosquitto:

```
mosquitto -v
```

Untuk melihatnya di sisi client, gunakan `on_log` callback:

```
def on_log(client, userdata, level, buf):
  print("log: ",buf)

client.on_log=on_log # set client logging
```

Message PING akan dikirim setiap keep alive interval, walaupun sistem mengirim message secara reguler. Misal, message di-publish setiap 5 detik dan ping dikirim setiap 12 detik, karena keep alive di-set 12 detik.

Namun, bila kita menyesuaikan script, dengan menambahkan subscribing pada topik yang sama, sehingga kita menerima message yang sama dengan yang di-publish; kita akan melihat bahwa tidak ada PINGREQ dan PINGRESP yang dikirim.

Bila karena sejumlah sebab, PING request / response tidak berhasil, maka broker akan memaksa memutus hubungan dengan client. Spesifikasi menyebutkan, jika nilai keep alive tidak sama dengan nol dan server tidak menerima control packet dari client dalam satu setengah periode keep alive, maka hubungan dengan client tersebut harus diputus. Sebagai contoh, periode keep alive kita set 6 detik, maka bila PING request / response tidak berhasil dalam 9 detik, maka hubungan dengan client tersebut diputus.

```
def on_connect(client, userdata, flags, rc):
  if rc == 0:
    print("connected OK Returned code=",rc)
  else:
    print("Bad connection Returned code=",rc)
```

Result code:

0: Connection successful 1: Connection refused - incorrect protocol version 2: Connection refused - invalid client identifier 3: Connection refused - server unavailable 4: Connection refused - bad username or password 5: Connection refused - not authorised 6-255: Currently unused.

Untuk memiliki kontrol yang lebih baik, kita dapat memanfaatkan sebuah flag pada callback on_connect. Flag ini merupakan bagian dari objek client.

```
client.connected_flag = False
```

Pada awal script, kita membuat flag ini bernilai False dan dibuat True ketika koneksi berhasil, dan kembali dibuat Flase ketika disconnect.

```
def on_connect(client, userdata, flags, rc):
  if rc==0:
    client.connected_flag=True #set flag
    print("connected OK Returned code=",rc)
    #client.subscribe(topic)
  else:
    print("Bad connection Returned code= ",rc)
```

Sekarang, kita menggunakan flag ini untuk sebuah wait loop:

```
client.connect(broker_address)   #connect to broker
while not client.connected_flag: #wait in loop
  time.sleep(1)
```

Mempertahankan Koneksi

Sekali sebuah client terhubung, network traffic diantara client dan broker harus diproses. Hal ini dapat dilakukan dengan cara blocking atau dengan sebuah thread yang bekerja di background.

`client.loop_forever()` akan mem-blok, memroses network traffic dan melakukan koneksi ulang secara otomatis bila diperlukan. Fungsi ini sangat berguna bila kita hanya subscribe ke sebuah broker dan melakukan aksi yang sesuai dengan message yang kita terima.

`client.loop_start()` memulai sebuah thread di background untuk menangani network traffic dan akan kembali sesegera mungkin. Cara ini cocok untuk situasi dimana kita mengerjakan task lain dalam program yang kita buat. Untuk menghentikan background thread, kita menggunakan perintah `client.loop_stop()`.

Kita juga dapat memanggil loop() dalam program, pastikan memanggilnya secara terus menerus. Karena loop bersifat blocking, maka perlu memanggilnya dengan menyertakan timeout (default: 1 detik). Kita pun perlu membuat kode untuk menangani reconnect.

```
while ...
  some code
  client.loop(0.1) # blocks for 100 ms
  some code
```

Jika client script memiliki lebih dari satu koneksi client, maka perlu menuliskan client1.loop() dan client2.loop() dalam script.

```
import paho.mqtt.client as mqtt

clients=[]
nclients=20
mqtt.Client.connected_flag=False

#create clients
for i in range(nclients):
  cname = "Client"+str(i)
  client = mqtt.Client(cname)
  clients.append(client)

for client in clients:
  client.connect(broker)
  client.loop_start()
```

Berikut sebuah contoh lengkap:

```
import paho.mqtt.client as mqtt
import time

def on_connect(client, data, flags, rc):
  client.subscribe(“mychannel/myresource”, 1)

def on_message(client, data, msg):
  print(msg.topic, msg.payload)

client = mqtt.Client(“NSOneClient”)
client.on_connect = on_connect
client.on_message = on_message

client.connect("127.0.0.1", 1883, 60, True)
client.loop_start()

while True:
  client.publish("mychannel/myresource", "Hello World", 1)
  time.sleep(10)
```

Kita juga dapat menggunakan fungsi client.loop() untuk memroses network traffic secara manual. Bila kita menggunakannya, kita perlu menangani koneksi ulang sendiri. Disarankan untuk menggunakan satu dari kedua metoda di atas.

Koneksi dengan Username / Password

Untuk membangun koneksi dengan sebuah username dan password, panggil username_pw_set() sebelumnya:

```
client = paho.Client(“clientId”)
client.username_pw_set(“username”, “password”)
client.connect(“broker.mqttdashboard.com”)
```

Publish

Secara normal, jika sebuah publisher mem-publish sebuah message dengan topik tertentu, dan tidak ada satupun yang subscribe ke topik tersebut, maka broker akan mengabaikannya. Namun, publisher dapat memberitahukan kepada broker untuk menyimpan dan mempertahankan message terakhir pada topik tersebut dengan men-setting flag retained message.

Hal ini berguna, sebagai contoh, dalam kasus dimana kita memiliki sensor yang mem-publish statusnya saat ada perubahan, misal door sensor. Apa yang terjadi jika ada subscriber baru terhadap status ini? Tanpa retained message, subscriber harus menunggu perubahan status terlebih dahulu. Namun demikian, dengan retained message, subscriber dapat melihat status sensor saat itu.

Yang penting untuk dipahami adalah, hanya satu message disimpan dan dipertahankan per topik.

Berikut adalah sebuah contoh tanpa opsi retained message:

```
import paho.mqtt.client as mqtt
import time

def on_publish(client, userdata, mid):
  print("mid: "+str(mid))

client = mqtt.Client(“NSOneClient”)
client.on_publish = on_publish
client.connect(“broker.hivemq.com”, 1883)
client.loop_start()

while True:
  temperature = read_from_imaginary_thermometer()
  (rc, mid) = client.publish(“encyclopedia/temperature”, str(temperature),
qos=1)
  time.sleep(30)
```

Untuk publish dengan opsi retained message:

```
(rc, mid) = client.publish(“encyclopedia/temperature”, str(temperature),
qos=1, retain=True)
```

Publish sebuah message kosong dengan retain flag di-set True akan membersihkan retained message.

QoS yang dipilih tidak mempengaruhi retained message.

Subscribe

Subscribe method menerima 2 parameter: topik atau sejumlah topik, dan QOS (quality of service), seperti berikut, dengan nilai default-nya:

```
subscribe(topic, qos=0)
```

Dokumentasi menyebutkan tiga cara memanggil subscribe method:

```
client.subscribe(“house/bulb”, 1)
client.subscribe((“house/bulb”,2))
client.subscribe([(“house/bulb1”,2),(“house/bulb2”,1),(“house/bulb3”,0)])
```

Fungsi subscribe memberikan retur sebuah tuple yang meng-indikasikan berhasil, dan sebuah message id yang digunakan sebagai tracking code.

```
(resuld, mid)
```

MQTT broker akan meng-acknowledge subscription yang kemudian akan membangkitkan callback on_subscribe:

```
on_subscribe(client, userdata, mid, granted_qos)
```

Nilai mid dapat dibandingkan dengan nilai yang dikembalikan pada pemanggilan fungsi untuk memeriksa keberhasilan subscription request.

Jika kita ingin memastikan bahwa subscriber menerima message, maka kita perlu subscribe dengan quality of service 1 atau 2.

Namun demikian, walaupun langkah ini tidak menjamin bahwa mereka akan menerima message, kita juga perlu subscribe dengan opsi persistent connection (bukan default).

Jika kita subscribe ke sebuah topik yang memiliki message yang di-publish dengan retain flag di-set, maka kita akan menerima semua retained message. Client dapat men-deteksi retained message dengan memeriksa flag message.retain dalam callback on_message:

```
def on_message(client, userdata, message):
  print("message received  ",str(message.payload.decode("utf-8")),\
          "topic",message.topic,"retained ",message.retain)
  if message.retain==1:
    print("This is a retained message")
```

Berikut adalah sebuah contoh lengkap:

```
import paho.mqtt.client as paho

def on_subscribe(client, userdata, mid, granted_qos):
    print("Subscribed: "+str(mid)+" "+str(granted_qos))

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.qos)+" "+str(msg.payload))

client = paho.Client("NSOneClient")
client.on_subscribe = on_subscribe
client.on_message = on_message
client.connect("broker.mqttdashboard.com", 1883)
client.subscribe("encyclopedia/#", qos=1)

client.loop_forever()
```

Fungsi subscribe dapat dipanggil dengan cara sebagai berikut:

- `subscribe("my/topic", qos=2)`
- `subscribe(("my/topic", 1)`
- `subscribe([("my/topic",0), ("another/topic",2)])`

Untuk unsubscribe:

```
client.unsubscribe(topic)
```

Callback `on_unsubscribe` seperti berikut:

```
on_subscribe(client, userdata, mid)
```

Disconnect

```
client.disconnect()
```

Jika kita menggunakan fungsi loop_start() maka kita perlu menghentikannya ketika koneksi gagal. Cara termudah adalah menggunakan callback on_disconnect:

```
def on_disconnect(client, userdata, rc=0):
  logging.debug("Disconnected result code "+str(rc))
  client.loop_stop()
```

Contoh Implementasi

```
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc):

  if rc == 0:
    print("Connected to broker")
    global connected
    connected = True
  else:
    print("Connection failed")

connected = False

broker_address= "m11.cloudmqtt.com"
port = 12948
user = "yourUser"
password = "yourPassword"

client = mqtt.Client("Python")
client.username_pw_set(user, password=password)
client.on_connect = on_connect
client.connect(broker_address, port=port)

client.loop_start()

while connected != True:
  time.sleep(0.1)

try:
  while True:
    value = input('Enter the message:')
    client.publish("nsone/smartlamp", value)

except KeyboardInterrupt:
```

```
client.disconnect()
client.loop_stop()
```