aMQTT Documentation

Release 0.10.0-alpha.1

Nicolas Jouanin

Apr 29, 2022

CONTENTS

aMQTT is an open source MQTT client and broker implementation.

Built on top of asyncio, Python's standard asynchronous I/O framework, aMQTT provides a straightforward API based on coroutines, making it easy to write highly concurrent applications.

CHAPTER

ONE

FEATURES

aMQTT implements the full set of MQTT 3.1.1 protocol specifications and provides the following features:

- Support QoS 0, QoS 1 and QoS 2 messages flow
- Client auto-reconnection on network lost
- Authentication through password file (more methods can be added through a plugin system)
- Basic `$SYS` topics
- TCP and websocket support
- SSL support over TCP and websocket
- Plugin system

CHAPTER

TWO

REQUIREMENTS

aMQTT is built on Python's *asyncio*. It requires Python 3.6 or newer.

CHAPTER

THREE

INSTALLATION

It is not recommended to install third-party library in Python system packages directory. The preferred way for installing aMQTT is to create a virtual environment and then install all the dependencies you need. Refer to PEP 405 to learn more.

Once you have a environment setup and ready, aMQTT can be installed with the following command

```
(venv) $ pip install amqtt
```

`pip` will download and install aMQTT and all its dependencies.

CHAPTER

FOUR

USER GUIDE

If you need aMQTT for running a MQTT client or deploying a MQTT broker, the *Quickstart* describes how to use console scripts provided by aMQTT.

If you want to develop an application which needs to connect to a MQTT broker, the *MQTTClient API* documentation explains how to use aMQTT API for connecting, publishing and subscribing with a MQTT broker.

If you want to run you own MQTT broker, th *Broker API reference* reference documentation explains how to embed a MQTT broker inside a Python application.

News and updates are listed in the *Changelog*.

4.1 Quickstart

A quick way for getting started with `aMQTT` is to use console scripts provided for :

- publishing a message on some topic on an external MQTT broker.
- subscribing some topics and getting published messages.
- running a autonomous MQTT broker

These scripts are installed automatically when installing `aMQTT` with the following command

```
(venv) $ pip install amqtt
```

4.1.1 Publishing messages

`hbmqtt_pub` is a command-line tool which can be used for publishing some messages on a topic. It requires a few arguments like broker URL, topic name, QoS and data to send. Additional options allow more complex use case.

Publishing `` `some_data `` to as `/test` topic on is as simple as :

```
$ hbmqtt_pub --url mqtt://test.mosquitto.org -t /test -m some_data
[2015-11-06 22:21:55,108] :: INFO - hbmqtt_pub/5135-MacBook-Pro.local Connecting to␣
↪broker
[2015-11-06 22:21:55,333] :: INFO - hbmqtt_pub/5135-MacBook-Pro.local Publishing to '/
↪test'
[2015-11-06 22:21:55,336] :: INFO - hbmqtt_pub/5135-MacBook-Pro.local Disconnected from␣
↪broker
```

This will use insecure TCP connection to connect to test.mosquitto.org. `hbmqtt_pub` also allows websockets and secure connection:

```
$ hbmqtt_pub --url ws://test.mosquitto.org:8080 -t /test -m some_data
[2015-11-06 22:22:42,542] :: INFO - hbmqtt_pub/5157-MacBook-Pro.local Connecting to␣
↪broker
[2015-11-06 22:22:42,924] :: INFO - hbmqtt_pub/5157-MacBook-Pro.local Publishing to '/
↪test'
[2015-11-06 22:22:52,926] :: INFO - hbmqtt_pub/5157-MacBook-Pro.local Disconnected from␣
↪broker
```

hbmqtt_pub can read from file or stdin and use data read as message payload:

```
$ some_command | hbmqtt_pub --url mqtt://localhost -t /test -l
```

See *hbmqtt_pub* reference documentation for details about available options and settings.

4.1.2 Subscribing a topic

hbmqtt_sub is a command-line tool which can be used to subscribe for some pattern(s) on a broker and get date from messages published on topics matching these patterns by other MQTT clients.

Subscribing a /test/# topic pattern is done with :

```
$ hbmqtt_sub --url mqtt://localhost -t /test/#
```

This command will run forever and print on the standard output every messages received from the broker. The -n option allows to set a maximum number of messages to receive before stopping.

See *hbmqtt_sub* reference documentation for details about available options and settings.

4.1.3 URL Scheme

aMQTT command line tools use the --url to establish a network connection with the broker. The --url parameter value must conform to the MQTT URL scheme. The general accepted form is :

```
[mqtt|ws][s]://[username][:password]@host.domain[:port]
```

Here are some examples of URL:

```
mqtt://localhost
mqtt://localhost:1884
mqtt://user:password@localhost
ws://test.mosquitto.org
wss://user:password@localhost
```

4.1.4 Running a broker

`hbmqtt` is a command-line tool for running a MQTT broker:

```
$ hbmqtt
[2015-11-06 22:45:16,470] :: INFO - Listener 'default' bind to 0.0.0.0:1883 (max_
↪connections=-1)
```

See *hbmqtt* reference documentation for details about available options and settings.

4.2 Changelog

4.3 0.10.1 - 2022-04-29

- Fixed a major bug in plugin api, see https://github.com/Yakifo/amqtt/pull/92

4.4 0.10.0 - 2021-08-04

- first release under new package name: amqtt
- reworked unit tests
- dropped support for python3.5 and earlier
- added support for python3.8 and 3.9
- Pass in loop to PluginManager, from https://github.com/beerfactory/hbmqtt/pull/126
- Fixes taboo topic checking without session username, from https://github.com/beerfactory/hbmqtt/pull/151
- Move scripts module into hbmqtt module, from https://github.com/beerfactory/hbmqtt/pull/167
- Download mosquitto certificate on the fly
- importing *hbmqtt* is deprecated, use *amqtt*
- Security fix: If an attacker could produce a KeyError inside an authentication plugin, the authentication was accepted instead of rejected

4.4.1 0.9.5

- fix more issues
- fix a few issues

4.4.2 0.9.2

- fix a few issues

4.4.3 0.9.1

- See commit log

4.4.4 0.9.0

- fix a serie of issues
- improve plugin performance
- support Python 3.6
- upgrade to `websockets` 3.3.0

4.4.5 0.8.0

- fix a serie of issues

4.4.6 0.7.3

- fix deliver message client method to raise TimeoutError (#40)
- fix topic filter matching in broker (#41)

Version 0.7.2 has been jumped due to troubles with pypi…

4.4.7 0.7.1

- Fix duplicated $SYS topic name .

4.4.8 0.7.0

- Fix a serie of issues reported by Christoph Krey

4.4.9 0.6.3

- Fix issue #22.

4.4.10 0.6.2

- Fix issue #20 (`mqtt` subprotocol was missing).
- Upgrade to `websockets` 3.0.

4.4.11 0.6.1

- Fix issue #19

4.4.12 0.6

- Added compatibility with Python 3.5.
- Rewritten documentation.
- Add command-line tools *hbmqtt*, *hbmqtt_pub* and *hbmqtt_sub*.

4.5 Transitioning from HBMQTT to aMQTT

This document is for those porting from HBMQTT to aMQTT. Basically you can search and replace `hbmqtt` with `amqtt` and all should work out. Details below.

4.5.1 Imports

The module changed from `hbmqtt` to `amqtt`. For the 0.10.x releases it will still be possible to `import hbmqtt`. In 0.11.x only `amqtt` will work.

Since the `amqtt` package does provide a `hbmqtt` module, installing the `hbmqtt` package in the same python installation is not possible.

4.5.2 Random Client ID

When not providing a client_id, a random id is automatically generated. These names were also changed from `hbmqtt/<random>` to `amqtt/<random>`.

4.5.3 plugins / entrypoints

If you make use of python's entrypoint system to build aMQTT plugins, make sure to use the `amqtt.*.plugins` names instead of `hbmqtt.*.plugins` names. During the transition plugins with `hbmqtt` entrypoint should keep working for 0.10.x releases.

4.5.4 CLI tools

Will also be renamed.

4.6 References

Reference documentation for HBMQTT console scripts and programming API.

4.6.1 Console scripts

- *hbmqtt_pub* : MQTT client for publishing messages to a broker
- *hbmqtt_sub* : MQTT client for subscribing to a topics and retrieved published messages
- *hbmqtt* : Autonomous MQTT broker

4.6.2 Programming API

- *MQTTClient API* : MQTT client API reference
- *Broker API reference* : MQTT broker API reference
- *Common API* : Common API

TBD

hbmqtt_pub

`hbmqtt_pub` is a MQTT client that publishes simple messages on a topic from the command line.

Usage

`hbmqtt_pub` usage :

```
hbmqtt_pub --version
hbmqtt_pub (-h | --help)
hbmqtt_pub --url BROKER_URL -t TOPIC (-f FILE | -l | -m MESSAGE | -n | -s) [-c CONFIG_
↪FILE] [-i CLIENT_ID] [-d]
           [-q | --qos QOS] [-d] [-k KEEP_ALIVE] [--clean-session]
           [--ca-file CAFILE] [--ca-path CAPATH] [--ca-data CADATA]
           [ --will-topic WILL_TOPIC [--will-message WILL_MESSAGE] [--will-qos WILL_QOS]␣
↪[--will-retain] ]
           [--extra-headers HEADER]
```

Note that for simplicity, `hbmqtt_pub` uses mostly the same argument syntax as mosquitto_pub.

Options

--version	HBMQTT version information
-h, --help	Display `hbmqtt_pub` usage help
-c	Set the YAML configuration file to read and pass to the client runtime.
-d	Enable debugging informations.
--ca-file	Define the path to a file containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-path	Define the path to a directory containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-data	Set the PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--clean-session	If given, set the CONNECT clean session flag to True.
-f	Send the contents of a file as the message. The file is read line by line, and `hbmqtt_pub` will publish a message for each line read.
-i	The id to use for this client. If not given, defaults to `hbmqtt_pub/` appended with the process id and the hostname of the client.
-l	Send messages read from stdin. `hbmqtt_pub` will publish a message for each line read. Blank lines won't be sent.
-k	Set the CONNECT keep alive timeout.
-m	Send a single message from the command line.
-n	Send a null (zero length) message.
-q, --qos	Specify the quality of service to use for the message, from 0, 1 and 2. Defaults to 0.
-s	Send a message read from stdin, sending the entire content as a single message.
-t	The MQTT topic on which to publish the message.
--url	Broker connection URL, conforming to MQTT URL scheme.
--will-topic	The topic on which to send a Will, in the event that the client disconnects unexpectedly.
--will-message	Specify a message that will be stored by the broker and sent out if this client disconnects unexpectedly. This must be used in conjunction with `--will-topic`.
--will-qos	The QoS to use for the Will. Defaults to 0. This must be used in conjunction with `--will-topic`.
--will-retain	If given, if the client disconnects unexpectedly the message sent out will be treated as a retained message. This must be used in conjunction with `--will-topic`.
--extra-headers	Specify a JSON object string with key-value pairs representing additional headers that are transmitted on the initial connection, but only when using a websocket connection

Configuration

If `-c` argument is given, `hbmqtt_pub` will read specific MQTT settings for the given configuration file. This file must be a valid YAML file which may contains the following configuration elements :

- `keep_alive` : Keep-alive timeout sent to the broker. Defaults to `10` seconds.
- `ping_delay` : Auto-ping delay before keep-alive timeout. Defaults to 1. Setting to `0` will disable to 0 and may lead to broker disconnection.
- `default_qos` : Default QoS for messages published. Defaults to 0.
- `default_retain` : Default retain value to messages published. Defaults to `false`.
- `auto_reconnect` : Enable or disable auto-reconnect if connectection with the broker is interrupted. Defaults to `false`.
- `reconnect_retries` : Maximum reconnection retries. Defaults to `2`. Negative value will cause client to reconnect infinietly.
- `reconnect_max_interval` : Maximum interval between 2 connection retry. Defaults to `10`.

Examples

Examples below are adapted from mosquitto_pub documentation.

Publish temperature information to localhost with QoS 1:

```
hbmqtt_pub --url mqtt://localhost -t sensors/temperature -m 32 -q 1
```

Publish timestamp and temperature information to a remote host on a non-standard port and QoS 0:

```
hbmqtt_pub --url mqtt://192.168.1.1:1885 -t sensors/temperature -m "1266193804 32"
```

Publish light switch status. Message is set to retained because there may be a long period of time between light switch events:

```
hbmqtt_pub --url mqtt://localhost -r -t switches/kitchen_lights/status -m "on"
```

Send the contents of a file in two ways:

```
hbmqtt_pub --url mqtt://localhost -t my/topic -f ./data

hbmqtt_pub --url mqtt://localhost -t my/topic -s < ./data
```

Publish temperature information to localhost with QoS 1 over mqtt encapsulated in a websocket connection and additional headers:

```
hbmqtt_pub --url wss://localhost -t sensors/temperature -m 32 -q 1 --extra-headers '{
↪"Authorization": "Bearer <token>"}'
```

hbmqtt_sub

hbmqtt_sub is a command line MQTT client that subscribes to some topics and output data received from messages published.

Usage

hbmqtt_sub usage :

```
hbmqtt_sub --version
hbmqtt_sub (-h | --help)
hbmqtt_sub --url BROKER_URL -t TOPIC... [-n COUNT] [-c CONFIG_FILE] [-i CLIENT_ID] [-q |␣
↪--qos QOS] [-d]
          [-k KEEP_ALIVE] [--clean-session] [--ca-file CAFILE] [--ca-path CAPATH] [--ca-
↪data CADATA]
          [ --will-topic WILL_TOPIC [--will-message WILL_MESSAGE] [--will-qos WILL_QOS]␣
↪[--will-retain] ]
          [--extra-headers HEADER]
```

Note that for simplicity, hbmqtt_sub uses mostly the same argument syntax as mosquitto_sub.

Options

--version	HBMQTT version information
-h, --help	Display hbmqtt_sub usage help
-c	Set the YAML configuration file to read and pass to the client runtime.
-d	Enable debugging informations.
--ca-file	Define the path to a file containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-path	Define the path to a directory containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-data	Set the PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--clean-session	If given, set the CONNECT clean session flag to True.
-i	The id to use for this client. If not given, defaults to hbmqtt_sub/ appended with the process id and the hostname of the client.
-k	Set the CONNECT keep alive timeout.
-n	Number of messages to read before ending. Read forever if not given.
-q, --qos	Specify the quality of service to use for receiving messages. This QoS is sent in the subscribe request.
-t	Topic filters to subcribe.
--url	Broker connection URL, conforming to MQTT URL scheme.
--will-topic	The topic on which to send a Will, in the event that the client disconnects unexpectedly.

--will-message	Specify a message that will be stored by the broker and sent out if this client disconnects unexpectedly. This must be used in conjunction with `--will-topic`.
--will-qos	The QoS to use for the Will. Defaults to 0. This must be used in conjunction with `--will-topic`.
--will-retain	If given, if the client disconnects unexpectedly the message sent out will be treated as a retained message. This must be used in conjunction with `--will-topic`.
--extra-headers	Specify a JSON object string with key-value pairs representing additional headers that are transmitted on the initial connection, but only when using a websocket connection

Configuration

If `-c` argument is given, `hbmqtt_sub` will read specific MQTT settings for the given configuration file. This file must be a valid YAML file which may contains the following configuration elements :

- `keep_alive` : Keep-alive timeout sent to the broker. Defaults to `10` seconds.
- `ping_delay` : Auto-ping delay before keep-alive timeout. Defaults to 1. Setting to `0` will disable to 0 and may lead to broker disconnection.
- `default_qos` : Default QoS for messages published. Defaults to 0.
- `default_retain` : Default retain value to messages published. Defaults to `false`.
- `auto_reconnect` : Enable or disable auto-reconnect if connectection with the broker is interrupted. Defaults to `false`.
- `reconnect_retries` : Maximum reconnection retries. Defaults to `2`. Negative value will cause client to reconnect infinietly.
- `reconnect_max_interval` : Maximum interval between 2 connection retry. Defaults to `10`.

Examples

Examples below are adapted from mosquitto_sub documentation.

Subscribe with QoS 0 to all messages published under $SYS/:

```
hbmqtt_sub --url mqtt://localhost -t '$SYS/#' -q 0
```

Subscribe to 10 messages with QoS 2 from /#:

```
hbmqtt_sub --url mqtt://localhost -t /# -q 2 -n 10
```

Subscribe with QoS 0 to all messages published under $SYS/: over mqtt encapsulated in a websocket connection and additional headers:

```
hbmqtt_sub --url wss://localhost -t '$SYS/#' -q 0 --extra-headers '{"Authorization":
↪"Bearer <token>"}'
```

hbmqtt

`hbmqtt` is a command-line script for running a MQTT 3.1.1 broker.

Usage

`hbmqtt` usage :

```
hbmqtt --version
hbmqtt (-h | --help)
hbmqtt [-c <config_file> ] [-d]
```

Options

--version	HBMQTT version information
-h, --help	Display `hbmqtt_sub` usage help
-c	Set the YAML configuration file to read and pass to the client runtime.

Configuration

If `-c` argument is given, `hbmqtt` will read specific MQTT settings for the given configuration file. This file must be a valid YAML file which may contains the following configuration elements :

- `listeners` : network bindings configuration list
- `timeout-disconnect-delay` : client disconnect timeout after keep-alive timeout
- `auth` : authentication configuration

Without the `-c` argument, the broker will run with the following default configuration:

```
listeners:
    default:
        type: tcp
        bind: 0.0.0.0:1883
sys_interval: 20
auth:
    allow-anonymous: true
plugins:
    - auth_file
    - auth_anonymous
```

Using this configuration, `hbmqtt` will start a broker :

- listening on TCP port 1883 on all network interfaces.
- Publishing `$SYS``_ update messages every ``20` seconds.
- Allowing anonymous login, and password file bases authentication.

Configuration example

```
listeners:
    default:
        max-connections: 50000
        type: tcp
    my-tcp-1:
        bind: 127.0.0.1:1883
        my-tcp-2:
        bind: 1.2.3.4:1883
        max-connections: 1000
    my-tcp-ssl-1:
        bind: 127.0.0.1:8883
        ssl: on
        cafile: /some/cafile
        capath: /some/folder
        capath: certificate data
        certfile: /some/certfile
        keyfile: /some/key
    my-ws-1:
        bind: 0.0.0.0:8080
        type: ws
timeout-disconnect-delay: 2
auth:
    plugins: ['auth.anonymous']
    allow-anonymous: true
    password-file: /some/passwd_file
```

This configuration example shows use case of every parameter.

The `listeners` section define 3 bindings :

- `my-tcp-1` : a unsecured TCP listener on port 1883 allowing `1000` clients connections simultaneously
- `my-tcp-ssl-1` : a secured TCP listener on port 8883 allowing `50000` clients connections simultaneously
- `my-ws-1` : a unsecured websocket listener on port 8080 allowing `50000` clients connections simultaneously

Authentication allows anonymous logins and password file based authentication. Password files are required to be text files containing user name and password in the form of :

```
username:password
```

where `password` should be the encrypted password. Use the `mkpasswd -m sha-512` command to build encoded passphrase. Password file example:

```
# Test user with 'test' password encrypted with sha-512
test:$6$l4zQEHEcowc1Pnv4
↪$HHrh8xnsZoLItQ8BmpFHM4r6q5UqK3DnXp2GaTm5zp5buQ7NheY3Xt9f6godVKbEtA.hOC7IEDwnok3pbAOip.
```

MQTTClient API

The `MQTTClient` class implements the client part of MQTT protocol. It can be used to publish and/or subscribe MQTT message on a broker accessible on the network through TCP or websocket protocol, both secured or unsecured.

Usage examples

Subscriber

The example below shows how to write a simple MQTT client which subscribes a topic and prints every messages received from the broker :

```
import logging
import asyncio

from hbmqtt.client import MQTTClient, ClientException
from hbmqtt.mqtt.constants import QOS_1, QOS_2

logger = logging.getLogger(__name__)

async def uptime_coro():
    C = MQTTClient()
    await C.connect('mqtt://test.mosquitto.org/')
    # Subscribe to '$SYS/broker/uptime' with QOS=1
    # Subscribe to '$SYS/broker/load/#' with QOS=2
    await C.subscribe([
            ('$SYS/broker/uptime', QOS_1),
            ('$SYS/broker/load/#', QOS_2),
         ])
    try:
        for i in range(1, 100):
            message = await C.deliver_message()
            packet = message.publish_packet
            print("%d:  %s => %s" % (i, packet.variable_header.topic_name, str(packet.
↪payload.data)))
        await C.unsubscribe(['$SYS/broker/uptime', '$SYS/broker/load/#'])
        await C.disconnect()
    except ClientException as ce:
        logger.error("Client exception: %s" % ce)

if __name__ == '__main__':
    formatter = "[%(asctime)s] %(name)s {%(filename)s:%(lineno)d} %(levelname)s -
↪%(message)s"
    logging.basicConfig(level=logging.DEBUG, format=formatter)
    asyncio.get_event_loop().run_until_complete(uptime_coro())
```

When executed, this script gets the default event loop and asks it to run the `uptime_coro` until it completes. `uptime_coro` starts by initializing a `MQTTClient` instance. The coroutine then call `connect()` to connect to the broker, here `test.mosquitto.org`. Once connected, the coroutine subscribes to some topics, and then wait for 100 messages. Each message received is simply written to output. Finally, the coroutine unsubscribes from topics and disconnects from the broker.

Publisher

The example below uses the `MQTTClient` class to implement a publisher. This test publish 3 messages asynchronously to the broker on a test topic. For the purposes of the test, each message is published with a different Quality Of Service. This example also shows to method for publishing message asynchronously.

```
import logging
import asyncio

from hbmqtt.client import MQTTClient
from hbmqtt.mqtt.constants import QOS_0, QOS_1, QOS_2

logger = logging.getLogger(__name__)

async def test_coro():
    C = MQTTClient()
    await C.connect('mqtt://test.mosquitto.org/')
    tasks = [
        asyncio.ensure_future(C.publish('a/b', b'TEST MESSAGE WITH QOS_0')),
        asyncio.ensure_future(C.publish('a/b', b'TEST MESSAGE WITH QOS_1', qos=QOS_1)),
        asyncio.ensure_future(C.publish('a/b', b'TEST MESSAGE WITH QOS_2', qos=QOS_2)),
    ]
    await asyncio.wait(tasks)
    logger.info("messages published")
    await C.disconnect()


async def test_coro2():
    try:
        C = MQTTClient()
        ret = await C.connect('mqtt://test.mosquitto.org:1883/')
        message = await C.publish('a/b', b'TEST MESSAGE WITH QOS_0', qos=QOS_0)
        message = await C.publish('a/b', b'TEST MESSAGE WITH QOS_1', qos=QOS_1)
        message = await C.publish('a/b', b'TEST MESSAGE WITH QOS_2', qos=QOS_2)
        #print(message)
        logger.info("messages published")
        await C.disconnect()
    except ConnectException as ce:
        logger.error("Connection failed: %s" % ce)
        asyncio.get_event_loop().stop()


if __name__ == '__main__':
    formatter = "[%(asctime)s] %(name)s {%(filename)s:%(lineno)d} %(levelname)s -
↪%(message)s"
    logging.basicConfig(level=logging.DEBUG, format=formatter)
    asyncio.get_event_loop().run_until_complete(test_coro())
    asyncio.get_event_loop().run_until_complete(test_coro2())
```

As usual, the script runs the publish code through the async loop. `test_coro()` and `test_coro()` are ran in sequence. Both do the same job. `test_coro()` publish 3 messages in sequence. `test_coro2()` publishes the same message asynchronously. The difference appears the looking at the sequence of MQTT messages sent.

`test_coro()` achieves:

```
hbmqtt/YDYY;NNRpYQSy3?o -out-> PublishPacket(ts=2015-11-11 21:54:48.843901,␣
↪fixed=MQTTFixedHeader(length=28, flags=0x0), variable=PublishVariableHeader(topic=a/b,␣
↪packet_id=None), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_0'"))
hbmqtt/YDYY;NNRpYQSy3?o -out-> PublishPacket(ts=2015-11-11 21:54:48.844152,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x2), variable=PublishVariableHeader(topic=a/b,␣
↪packet_id=1), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_1'"))
hbmqtt/YDYY;NNRpYQSy3?o <-in-- PubackPacket(ts=2015-11-11 21:54:48.979665,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=1), payload=None)
hbmqtt/YDYY;NNRpYQSy3?o -out-> PublishPacket(ts=2015-11-11 21:54:48.980886,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x4), variable=PublishVariableHeader(topic=a/b,␣
↪packet_id=2), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_2'"))
hbmqtt/YDYY;NNRpYQSy3?o <-in-- PubrecPacket(ts=2015-11-11 21:54:49.029691,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/YDYY;NNRpYQSy3?o -out-> PubrelPacket(ts=2015-11-11 21:54:49.030823,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x2), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/YDYY;NNRpYQSy3?o <-in-- PubcompPacket(ts=2015-11-11 21:54:49.092514,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)fixed=MQTTFixedHeader(length=2, flags=0x0),␣
↪variable=PacketIdVariableHeader(packet_id=2), payload=None)
```

while `test_coro2()` runs:

```
hbmqtt/LYRf52W[56SOjW04 -out-> PublishPacket(ts=2015-11-11 21:54:48.466123,␣
↪fixed=MQTTFixedHeader(length=28, flags=0x0), variable=PublishVariableHeader(topic=a/b,␣
↪packet_id=None), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_0'"))
hbmqtt/LYRf52W[56SOjW04 -out-> PublishPacket(ts=2015-11-11 21:54:48.466432,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x2), variable=PublishVariableHeader(topic=a/b,␣
↪packet_id=1), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_1'"))
hbmqtt/LYRf52W[56SOjW04 -out-> PublishPacket(ts=2015-11-11 21:54:48.466695,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x4), variable=PublishVariableHeader(topic=a/b,␣
↪packet_id=2), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_2'"))
hbmqtt/LYRf52W[56SOjW04 <-in-- PubackPacket(ts=2015-11-11 21:54:48.613062,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=1), payload=None)
hbmqtt/LYRf52W[56SOjW04 <-in-- PubrecPacket(ts=2015-11-11 21:54:48.661073,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/LYRf52W[56SOjW04 -out-> PubrelPacket(ts=2015-11-11 21:54:48.661925,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x2), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/LYRf52W[56SOjW04 <-in-- PubcompPacket(ts=2015-11-11 21:54:48.713107,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
```

Both coroutines have the same results except that `test_coro2()` manages messages flow in parallel which may be more efficient.

Reference

MQTTClient API

MQTTClient configuration

The `MQTTClient __init__` method accepts a `config` parameter which allow to setup some behaviour and defaults settings. This argument must be a Python dict object which may contain the following entries:

- `keep_alive`: keep alive (in seconds) to send when connecting to the broker (defaults to `10` seconds). `MQTTClient` will *auto-ping* the broker if not message is sent within the keep-alive interval. This avoids disconnection from the broker.
- `ping_delay`: *auto-ping* delay before keep-alive times out (defaults to `1` seconds).
- `default_qos`: Default QoS (`0`) used by `publish()` if `qos` argument is not given.
- `default_retain`: Default retain (`False`) used by `publish()` if `qos` argument is not given.,
- `auto_reconnect`: enable or disable auto-reconnect feature (defaults to `True`).
- `reconnect_max_interval`: maximum interval (in seconds) to wait before two connection retries (defaults to `10`).
- `reconnect_retries`: maximum number of connect retries (defaults to `2`). Negative value will cause client to reconnect infinietly.

Default QoS and default retain can also be overriden by adding a `topics` with may contain QoS and retain values for specific topics. See the following example:

```
config = {
    'keep_alive': 10,
    'ping_delay': 1,
    'default_qos': 0,
    'default_retain': False,
    'auto_reconnect': True,
    'reconnect_max_interval': 5,
    'reconnect_retries': 10,
    'topics': {
        '/test': { 'qos': 1 },
        '/some_topic': { 'qos': 2, 'retain': True }
    }
}
```

With this setting any message published will set with QOS_0 and retain flag unset except for :

- messages sent to `/test` topic : they will be sent with QOS_1
- messages sent to `/some_topic` topic : they will be sent with QOS_2 and retain flag set

In any case, the `qos` and `retain` argument values passed to method `publish()` will override these settings.

Broker API reference

The `Broker` class provides a complete MQTT 3.1.1 broker implementation. This class allows Python developers to embed a MQTT broker in their own applications.

Usage example

The following example shows how to start a broker using the default configuration:

```
import logging
import asyncio
import os
from hbmqtt.broker import Broker


async def broker_coro():
    broker = Broker()
    await broker.start()


if __name__ == '__main__':
    formatter = "[%(asctime)s] :: %(levelname)s :: %(name)s :: %(message)s"
    logging.basicConfig(level=logging.INFO, format=formatter)
    asyncio.get_event_loop().run_until_complete(broker_coro())
    asyncio.get_event_loop().run_forever()
```

When executed, this script gets the default event loop and asks it to run the `broker_coro` until it completes. `broker_coro` creates `Broker` instance and then `start()` the broker for serving. Once completed, the loop is ran forever, making this script never stop …

Reference

Broker API

Broker configuration

The `Broker` `__init__` method accepts a `config` parameter which allow to setup some behaviour and defaults settings. This argument must be a Python dict object. For convinience, it is presented below as a YAML file[1].

```
listeners:
    default:
        max-connections: 50000
        type: tcp
    my-tcp-1:
        bind: 127.0.0.1:1883
    my-tcp-2:
        bind: 1.2.3.4:1884
        max-connections: 1000
    my-tcp-ssl-1:
```

(continues on next page)

[1] See PyYAML for loading YAML files as Python dict.

(continued from previous page)

```
        bind: 127.0.0.1:8885
        ssl: on
        cafile: /some/cafile
        capath: /some/folder
        capath: certificate data
        certfile: /some/certfile
        keyfile: /some/key
    my-ws-1:
        bind: 0.0.0.0:8080
        type: ws
timeout-disconnect-delay: 2
auth:
    plugins: ['auth.anonymous'] #List of plugins to activate for authentication among␣
↪all registered plugins
    allow-anonymous: true / false
    password-file: /some/passwd_file
topic-check:
    enabled: true / false  # Set to False if topic filtering is not needed
    plugins: ['topic_acl'] #List of plugins to activate for topic filtering among all␣
↪registered plugins
    acl:
        # username: [list of allowed topics]
        username1: ['repositories/+/master', 'calendar/#', 'data/memes']  # List of␣
↪topics on which client1 can publish and subscribe
        username2: ...
        anonymous: []  # List of topics on which an anonymous client can publish and␣
↪subscribe
```

The `listeners` section allows to define network listeners which must be started by the `Broker`. Several listeners can be setup. `default` subsection defines common attributes for all listeners. Each listener can have the following settings:

- `bind`: IP address and port binding.
- `max-connections`: Set maximum number of active connection for the listener. `0` means no limit.
- `type`: transport protocol type; can be `tcp` for classic TCP listener or `ws` for MQTT over websocket.
- `ssl` enables (`on`) or disable secured connection over the transport protocol.
- `cafile`, `cadata`, `certfile` and `keyfile` : mandatory parameters for SSL secured connections.

The `auth` section setup authentication behaviour:

- `plugins`: defines the list of activated plugins. Note the plugins must be defined in the `amqtt.broker.plugins` entry point.
- `allow-anonymous` : used by the internal `hbmqtt.plugins.authentication.AnonymousAuthPlugin` plugin. This parameter enables (`on`) or disable anonymous connection, ie. connection without username.
- `password-file` : used by the internal `hbmqtt.plugins.authentication.FileAuthPlugin` plugin. This parameter gives to path of the password file to load for authenticating users.

The `topic-check` section setup access control policies for publishing and subscribing to topics:

- `enabled`: set to true if you want to impose an access control policy. Otherwise, set it to false.
- `plugins`: defines the list of activated plugins. Note the plugins must be defined in the `amqtt.broker.plugins` entry point.

- **additional parameters: depending on the plugin used for access control, additional parameters should be added.**
  - **In case of `topic_acl` plugin, the Access Control List (ACL) must be defined in the parameter `acl`.**
    - For each username, a list with the allowed topics must be defined.
    - If the client logs in anonymously, the `anonymous` entry within the ACL is used in order to grant/deny subscriptions.

Common API

This document describes HBMQTT common API both used by *MQTTClient API* and *Broker API reference*.

Reference

ApplicationMessage

4.7 License

```
The MIT License (MIT)

Copyright (c) 2015 Nicolas JOUANIN
Copyright (c) 2021 aMQTT Contributers (https://github.com/Yakifo/amqtt/graphs/
↪contributors)

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```