# ¿What is MQTT?

MQTT (Message Queuing Telemetry Transport) is a M2M (machine-to-machine) messages light protocol on the top of TCP/IP protocol. Although its origins are with IBM, since 2014 is managed by the [global nonprofit consortium OASIS](https://en.wikipedia.org/wiki/OASIS_(organization)). Its 3.1.1 version has been accepted under the international standard [**ISO/IEC 20922:2016**](http://www.iso.org/iso/catalogue_detail.htm?csnumber=69466)**.**

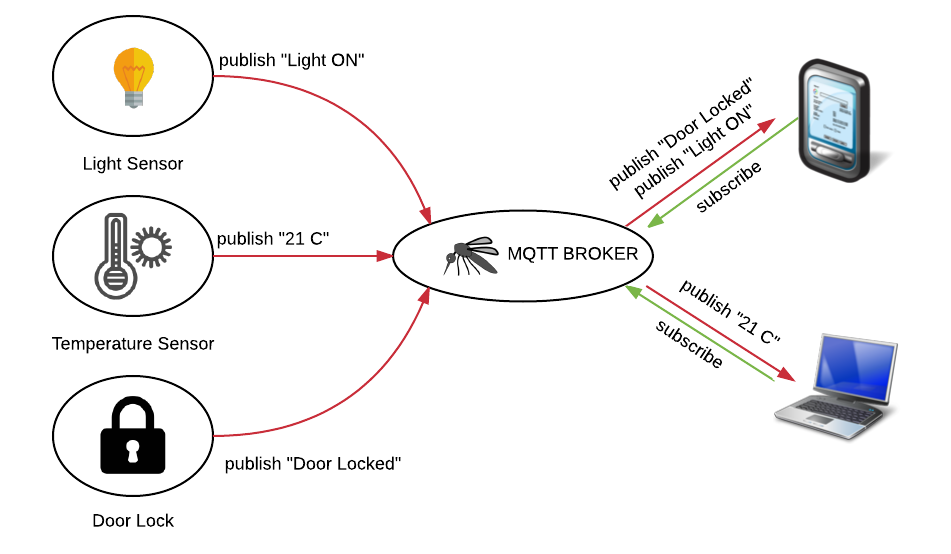
MQTT has publish-subscribe messaging pattern. This operation requires the use of a messages broker, who is in charge of message delivery to different clients, based on its topic. Currently, MQTT is the connectivity protocol for IoT devices with the widest diffusion.

Brokers show a TCP/IP interface connection, usually in tcp port 1883, which has been [registered by IANA for MQTT protocol](http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=1883).

Subscriber clients connect to this port indicating the *topic* they subscribe to, and remain waiting for the broker to send messages.

Publish clients, publish a message with a specific *topic*, connecting to the broker and sending the message to it.

Both publish and subscribe client are known as agents (publish agent/subscribe agent); hence an agent that publishes could be a light sensor, and an agent that subscribes could be an App from our mobile phone. The broker reviews the *topic* of the message and resend it to the agents subscribe to that *topic.*



There are several MQTT broker implementations, being “[**Mosquitto**](https://mosquitto.org/)” the most widespread, thanks to its [Open Source (EPL/EDL) license](https://wikipedia.org/wiki/Eclipse_Public_License).

## 1.1 Publish/Subscribe hierarchy

Publish/subscribe technique used by MQTT uses hierarchy for topics. Hierarchies are established through the use of "/"character. In this way we could have the *topic: home/floor1/room1.* This *topic* will generate three branches: *home, home/floor1, home/floor1/room1*.

Those agents, who want to receive a published message to that *topic*, could subscribe to any of these three branches. In order to receive the message while it has subscribed to any of the two first branches, it is needed to add the reserved character *"#"*, which act as a wildcard. Hence a message could be published:

**publish --topic "home/floor1/room2" --message "light on"**

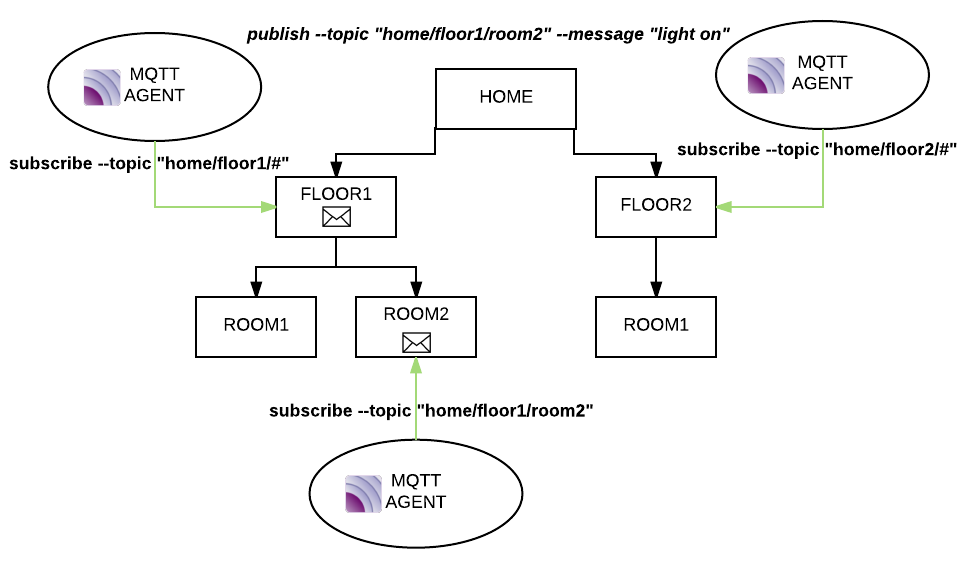
and it will be received by agents:

**subscribe --topic "home/floor1/room2"**

**subscribe --topic "home/floor1/#"**

but it would not be received by agent:

**subscribe --topic "home/floor2/#"**



# 2. Security in MQTT broker

MQTT protocol has very basic security mechanism because its basis is to be light and be able to be implemented in any IoT device. That is the reason why security lays over the protections that could exist, or can be implemented, at network level (for example, VPN), as well as the ones provided by the broker.

Mosquitto broker has different security features to control publish/subscribe messages. All of them are carried out by an authorization/authentication plugin. Each plugin defines its own security controls.

Mosquitto authentication plugin (auth-plugin) has the following security features implemented:

## 2.1 Authentication User/Password

This is the simplest and easiest control to configure. Each agent publishing a message can use the user/password combination and the broker will verify the credentials.

For a ***"publish agent"***, this action is done for each published message. The same configuration can be executed in a **"*subscribe agent***", although for this situation, the broker will check only the user/password when the agent makes the connection and not in each delivered message to the agent.

## 2.2 ACLs - Access Control Lists

With this mechanism, a control over the publish/subscribe functionality can be obtained, through *topics* filtering. The main idea of an ACL is to establish a control over a specific *topic* or any branches inherit from it, if wildcards are used.

ACLs can be seen as a set of rules; when a *topic* matches a filter, the element to be controlled is checked (for example, if the *topic* was published by an agent with a specific user) and, when the control is evaluated, the broker decides if the message is discarded or not.

ACL entry format is:

*<topic> <control element>*

As explained earlier, this configuration is specific for each broker.

ACLs examples using auth-plugin for Mosquito:

***home/floor1/room1 john***

***home/floor2/# john, paul***

In this example, an agent using *paul* user can not publish or subscribe to *home/floor1/room1 topic,* but it could do it to any *topic* inherited from *home/floor2* (including this *topic* itself).

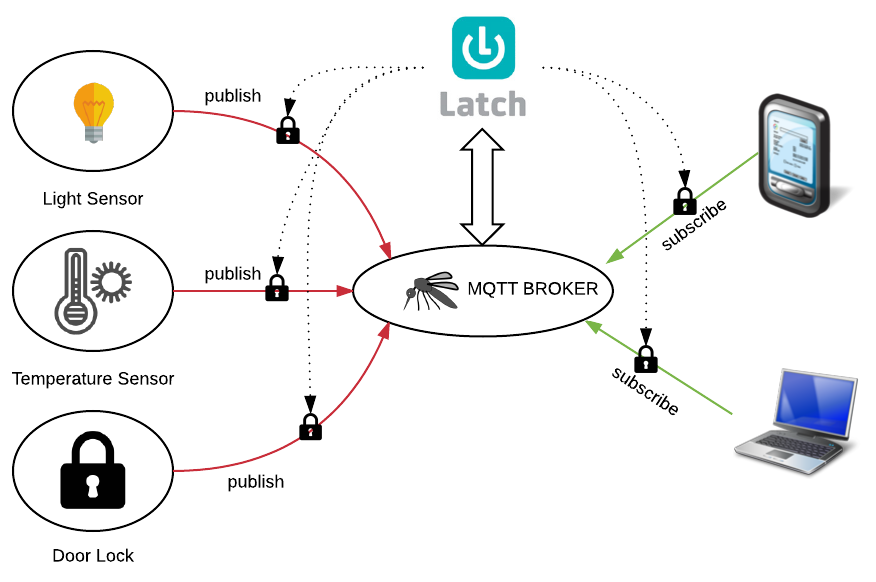
# 3. Latch Plugin for Mosquitto

Latch Plugin for Mosquitto allows to add, in an easy way, a second factor authentication inside IoT ecosystem, doing the integration in the platform (MQTT Broker), instead of devices, allowing to free resources from them, and maximizing compatibility and scalability.

It is developed in Python over Mosquitto auth-plugin basis, therefore it is supported the use of standard ACLs, as well as user/password authentication with Latch as 2FA.

Latch plugin operates in different ways:

* Over ***publish/subscribe*** operations, allowing 2FA when a message is published or when a subscriber is connected.
* Over ***topics***, allowing adding a 2FA dynamically, through **Latch Instances** use.



## 3.1 Pairing

Users of the plugin that are paired will be those with capacity to latch publish/subscribe operation or a *topic,* depend on the configuration made.

For each pairing done, we will get an *Account ID* that it will be the user identifier in Latch. This value is stored in the file *latch.accounts*. In order for this process to be carried out, previously the user has to be created, using the script *tools/users\_op.py*.

The pairing process can be done in two different ways:

* Through the use of a *Python* script: *tools/pair\_op.py*. This script asks for the username, password and token generated by Latch App.
* Enable the option *dynamic\_pairing* in the configuration file *latch.conf* and publish, from any agent, to the *topic:* ***LATCH/pairing=<token provided>***

In a similar way, the unpairing process can be done:

* Through the use of a *Python* script: *tools/pair\_op.py*.
* Enable the option *dynamic\_pairing* in the configuration file *latch.conf* and publish, from any agent, to the *topic:* ***LATCH/unpairing***

***LATCH/# topic*** is reserved by Latch plugin for Latch management. Doing publish to this *topic* has to go with a user and password, because the plugin will do automatically the pairing/unpairing with the user used in the publish. The payload of the published message to *LATCH/#* is ignored.

## 3.2 Latch Plugin for publish/subscribe operations

The easiest way to latch MQTT broker is through publish and subscribe operations. Latch plugin allows to control both operations at broker level, forcing to verify the authorization before publish or subscribe.

This process is independent from any security control configured before in the broker. In this way, for example, the MQTT broker can verify user/password first, and later, use Latch as second factor authorization for the requested operation (publish or subscribe), as long as the agent is paired.

When a paired agent requests to publish a message, each publish is authorized by Latch.

For paired agents subscribers, it will be verified with Latch each subscription, but when they are authorized, it is not checked again Latch status.



## 3.3 Latch Plugin for topics

This is the most powerful feature of Latch plugin, allowing to perform a **latch** over any topic dynamically. In this way, when a paired agent publishes a message in a *topic* or a paired agent subscribes to a *topic*, after all security controls already defined in the plugin, a second verification towards Latch can be done. In Latch, a *topic* is represented by an instance.

When a paired agent publishes a message, each of them, will be authorized by Latch. For paired agent that subscribes, it will be authorized by Latch each time they subscribe, but once they are authorized, they are not checked again.



Latch plugin allows creating these "***latchables" topic*** or instances, in several ways:

* Using the *python* script: *tools/instances\_op.py,* which can create or delete *topics*/instances.
* Publishing by any agent to the *topic*:   
    
   ***LATCH/create\_instance=<instance\_name>***    
    
  The plugin will create automatically the new instance with the name indicated and it will be associated with the user specified in the publish.
* Dynamically, using the option ***instance\_mode=dynamic*** in the configuration file *latch.conf.* After this option is enabled, each published message will create a new instance, if it does not exist, with the name of the *topic* used and linked to the user used in the publish.

# 4. Latch Plugin installation

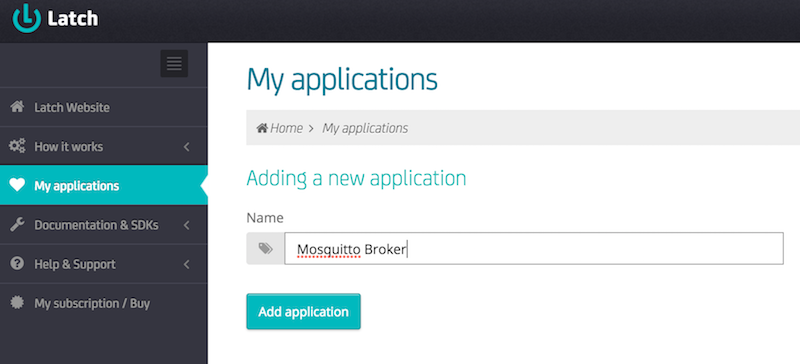
## 4.1 Plugin files

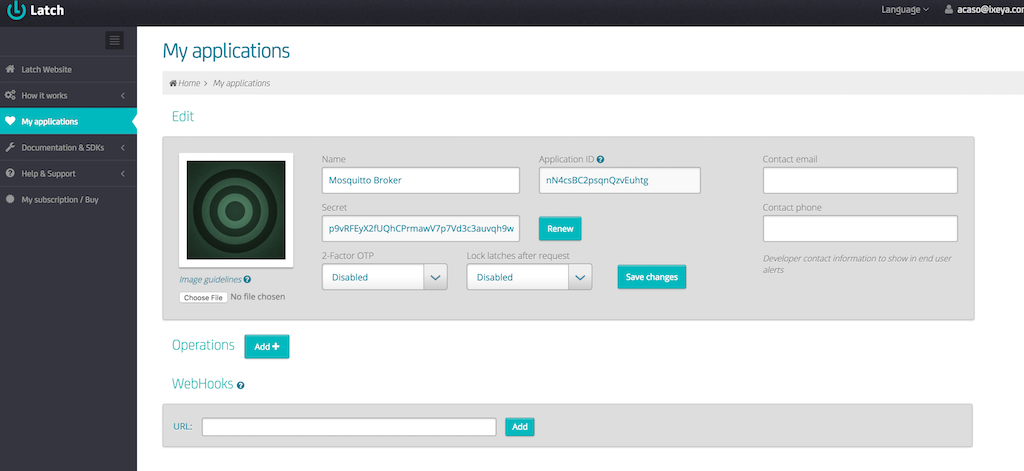
Latch plugin consist of the following files:

* *mosquitto\_latch.py:* Latch plugin for Mosquitto.
* *mosquitto\_latch\_bag.py:* Latch plugin library for Mosquitto.
* *latch.conf: Latch plugin configuration file.*
* *latch.instances*: Latch instances file.
* *latch.accounts*: Latch accountID file.
* *mosquitto.acl*: ACLs file.
* *mosquitto.users*: users file.
* *tools/install.py*: Plugin installation script.
* *tools/instances\_op.py*: instances static management script.
* *tools/pair\_op.py*: static pairing management script.
* *tools/users\_op.py*: user management script.

## 4.2 Pre-installation

Before starting the installation of the Plugin, first you need to create the application inside Latch portal (<https://latch.elevenpaths.com>). The application can be created with any name, and has to have *"2-Factor OTP"* and "*Lock latches after request* " disabled. Once the application is created, you have to write down the values of *Application ID* and *Secret*, they will be used lately in the plugin configuration.





After the previous process is ended, you need to verify that the system where our broker MQTT is installed, has the required software for Latch plugin. To do this, the following libraries are mandatories:

**sudo apt-get install python-pip python-dev mosquitto-dev libmosquitto-dev**

Also it will be needed a special library for Python: paho-mqtt. It can be installed with:

**pip install paho-mqtt**

Likewise Latch plugin needs a Mosquitto plugin for the plugin security development in Python. It can be installed from <https://github.com/oribit/mosquitto_pyauth> following the instructions described in the webpage.

## 4.3 Installation

With all prerequisite satisfied, Latch plugin can be downloaded from <https://github.com/oribit/latch-plugin>. Please, download this repository to the path you want to use for the plugin.

Once the plugin is downloaded, we will execute from the directory where it was downloaded *python ./tools/install.py* ,which will proceed with the verification and installation of the plugin in the system. This script will request the values of *Application ID* and *Secret* previously mentioned, and they will be stored in the file *latch.conf*.

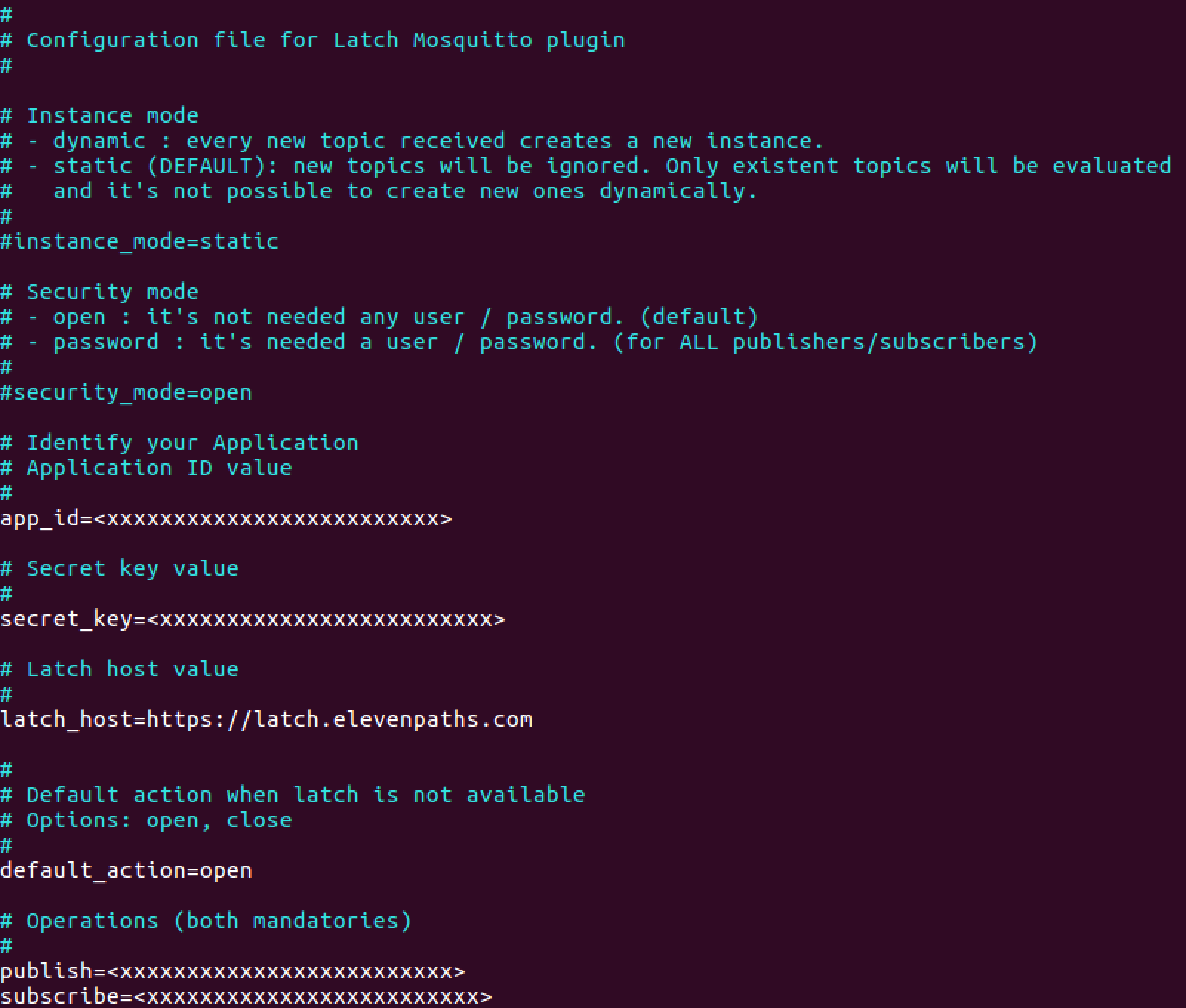
The installation script will perform a validation of the prerequisites, and then it will perform the copy of the Plugin files in the specified directory.

Configuration files and tools scripts will be copied in the directory: */etc/mosquitto/plugin/latch*. The installation probably will need **root permissions (sudo)** when it is launched, because Mosquitto configuration file, usually has root owner.

When the installation is finished, it will be necessary to create a user to start using **Latch**. This can be done with the script: *python ./tools/users\_op.py* After the user is created, the "pairing" can be done following section 3.1.

## 4.4 Configuration

After the installation is completed, the Plugin configuration file will be located in */etc/mosquitto/plugin/latch/latch.conf:*



*app\_id and secret\_key* fields will contain the information introduced during the execution of the installation script. *Publish* and *subscribe* fields will contain the IDs created automatically during the installation process.

## 4.5 Execution

The execution of the Plugin is done automatically together iwth the broker MQTT execution. The installation script will add the following line to the Mosquitto configuration file:

***auth\_plugin /usr/lib/mosquitto/auth\_plugin\_pyauth.so***

***auth\_opt\_pyauth\_module mosquitto\_latch***

***auth\_opt\_latch\_conf latch.conf***

All messages generated by the Plugin are redirected to the configured standard output in the broker, therefore if it is configured the output to a syslog or a file, Plugin message will be written in the same place.

In addition to the standard redirection, the Plugin publishes to the reserved *topic* ***LATCH/status*** any lock done by Latch, both global level and operation (*publish/subscribe*) or *topic* level. Any agent could subscribe to this topic.