

---

## CONTENTS

Prerequisites

Step 1 — Installing Mosquitto

Step 2 — Configuring MQTT Passwords

Step 3 — Configuring MQTT SSL

Step 4 — Configuring MQTT Over Websockets (Optional)

Conclusion

// Tutorial //

# How to Install and Secure the Mosquitto MQTT Messaging Broker on Debian 10

[MQTT](#) is a machine-to-machine messaging protocol, designed to provide lightweight publish/subscribe communication to “Internet of Things” devices. It is commonly used for geo-tracking fleets of vehicles, home automation, environmental sensor networks, and utility-scale data collection.

[Mosquitto](#) is a popular MQTT server (or *broker*, in MQTT parlance) that has great community support and is easy to install and configure.

In this tutorial, we'll install Mosquitto and set up our broker to use SSL to secure our password-protected MQTT communications.

## Prerequisites

Before starting this tutorial, you will need:

- A Debian 10 server with a non-root, sudo-enabled user and basic firewall set up, as detailed in [this Debian 10 server setup tutorial](#).
- A domain name pointed at your server, as documented in our [DigitalOcean DNS product documentation](#). This tutorial will use `mqtt.example.com` throughout.
- An auto-renewable Let's Encrypt SSL certificate for use with your domain and

Mosquitto, generated using the Certbot tool. You can learn how to set this up in [How To Use Certbot Standalone Mode to Retrieve Let's Encrypt SSL Certificates on Debian 10](#). You can add `systemctl restart mosquitto` as a `renew_hook` in Step 4. Be sure to use the same domain configured in the previous prerequisite step.

## Step 1 – Installing Mosquitto

Debian 10 has a fairly recent version of Mosquitto in its default software repository, so we can install it from there.

First, log in using your non-root user and update the package lists using `apt update`:

```
$ sudo apt update
```

Copy

Now, install Mosquitto using `apt install`:

```
$ sudo apt install mosquitto mosquitto-clients
```

Copy

By default, Debian will start the Mosquitto service after install. Let's test the default configuration. We'll use one of the Mosquitto clients we just installed to subscribe to a topic on our broker.

*Topics* are labels that you publish messages to and subscribe to. They are arranged as a hierarchy, so you could have `sensors/outside/temp` and `sensors/outside/humidity`, for example. How you arrange topics is up to you and your needs. Throughout this tutorial we will use a simple test topic to test our configuration changes.

Log in to your server a second time, so you have two terminals side-by-side. In the new terminal, use `mosquitto_sub` to subscribe to the test topic:

```
$ mosquitto_sub -h localhost -t test
```

Copy

`-h` is used to specify the hostname of the MQTT server, and `-t` is the topic name. You'll see no output after hitting `ENTER` because `mosquitto_sub` is waiting for messages to arrive. Switch back to your other terminal and publish a message:

```
$ mosquitto_pub -h localhost -t test -m "hello world"
```

Copy

The options for `mosquitto_pub` are the same as `mosquitto_sub`, though this time we

use the additional `-m` option to specify our message. Hit `ENTER`, and you should see **hello world** pop up in the other terminal. You've sent your first MQTT message!

Enter `CTRL+C` in the second terminal to exit out of `mosquitto_sub`, but keep the connection to the server open. We'll use it again for another test in Step 5.

Next, we'll secure our installation using password-based authentication.

## Step 2 – Configuring MQTT Passwords

Let's configure Mosquitto to use passwords. Mosquitto includes a utility to generate a special password file called `mosquitto_passwd`. This command will prompt you to enter a password for the specified username, and place the results in `/etc/mosquitto/passwd`.

```
$ sudo mosquitto_passwd -c /etc/mosquitto/passwd sammy
```

 Copy

Now we'll open up a new configuration file for Mosquitto and tell it to use this password file to require logins for all connections:

```
$ sudo nano /etc/mosquitto/conf.d/default.conf
```

 Copy

This should open an empty file. Paste in the following:

```
/etc/mosquitto/conf.d/default.conf
```

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Be sure to leave a trailing newline at the end of the file.

`allow_anonymous false` will disable all non-authenticated connections, and the `password_file` line tells Mosquitto where to look for user and password information. Save and exit the file.

Now we need to restart Mosquitto and test our changes.

```
$ sudo systemctl restart mosquitto
```

 Copy

Try to publish a message without a password:

```
$ mosquitto_pub -h localhost -t "test" -m "hello world"
```

Copy

The message should be rejected:

#### Output

```
Connection Refused: not authorised.  
Error: The connection was refused.
```

Before we try again with the password, switch to your second terminal window again, and subscribe to the 'test' topic, using the username and password this time:

```
$ mosquitto_sub -h localhost -t test -u "sammy" -P "password"
```

Copy

It should connect and sit, waiting for messages. You can leave this terminal open and connected for the rest of the tutorial, as we'll periodically send it test messages.

Now publish a message with your other terminal, again using the username and password:

```
$ mosquitto_pub -h localhost -t "test" -m "hello world" -u "sammy" Copy is
```

The message should go through as in Step 1. We've successfully added password protection to Mosquitto. Unfortunately, we're sending passwords unencrypted over the internet. We'll fix that next by adding SSL encryption to Mosquitto.

## Step 3 – Configuring MQTT SSL

To enable SSL encryption, we need to tell Mosquitto where our Let's Encrypt certificates are stored. Open up the configuration file we previously started:

```
$ sudo nano /etc/mosquitto/conf.d/default.conf
```

Copy

Paste in the following at the end of the file, leaving the two lines we already added:

```
/etc/mosquitto/conf.d/default.conf
```

```
. . .  
listener 1883 localhost  
  
listener 8883  
certfile /etc/letsencrypt/live/mqtt.example.com/cert.pem
```

```
cafile /etc/letsencrypt/live/mqtt.example.com/chain.pem
keyfile /etc/letsencrypt/live/mqtt.example.com/privkey.pem
```

Again, be sure to leave a trailing newline at the end of the file.

We're adding two separate `listener` blocks to the config. The first, `listener 1883 localhost`, updates the default MQTT listener on port 1883, which is what we've been connecting to so far. 1883 is the standard unencrypted MQTT port. The `localhost` portion of the line instructs Mosquitto to only bind this port to the localhost interface, so it's not accessible externally. External requests would have been blocked by our firewall anyway, but it's good to be explicit.

`listener 8883` sets up an encrypted listener on port 8883. This is the standard port for MQTT + SSL, often referred to as MQTTS. The next three lines, `certfile`, `cafile`, and `keyfile`, all point Mosquitto to the appropriate Let's Encrypt files to set up the encrypted connections.

Save and exit the file, then restart Mosquitto to update the settings:

```
$ sudo systemctl restart mosquitto
```

 Copy

Update the firewall to allow connections to port 8883.

```
$ sudo ufw allow 8883
```

 Copy

### Output

```
Rule added
Rule added (v6)
```

Now we test again using `mosquitto_pub`, with a few different options for SSL:

```
$ mosquitto_pub -h mqtt.example.com -t test -m "hello again" -p 8888
```

 Copy p

Note that we're using the full hostname instead of `localhost`. Because our SSL certificate is issued for `mqtt.example.com`, if we attempt a secure connection to `localhost` we'll get an error saying the hostname does not match the certificate hostname (even though they both point to the same Mosquitto server).

`--capath /etc/ssl/certs/` enables SSL for `mosquitto_pub`, and tells it where to look for root certificates. These are typically installed by your operating system, so the path is different for Mac OS, Windows, etc. `mosquitto_pub` uses the root certificate to

verify that the Mosquitto server's certificate was properly signed by the Let's Encrypt certificate authority. It's important to note that `mosquitto_pub` and `mosquitto_sub` will not attempt an SSL connection without this option (or the similar `--cafile` option), even if you're connecting to the standard secure port of `8883`.

If all goes well with the test, we'll see **hello again** show up in the other `mosquitto_sub` terminal. This means your server is fully set up! If you'd like to extend the MQTT protocol to work with websockets, you can follow the final step.

## Step 4 – Configuring MQTT Over Websockets (Optional)

In order to speak MQTT using JavaScript from within web browsers, the protocol was adapted to work over standard websockets. If you don't need this functionality, you may skip this step.

We need to add one more `listener` block to our Mosquitto config:

```
$ sudo nano /etc/mosquitto/conf.d/default.conf
```

Copy

At the end of the file, add the following:

```
/etc/mosquitto/conf.d/default.conf
```

```
. . .
listener 8083
protocol websockets
certfile /etc/letsencrypt/live/mqtt.example.com/cert.pem
cafile /etc/letsencrypt/live/mqtt.example.com/chain.pem
keyfile /etc/letsencrypt/live/mqtt.example.com/privkey.pem
```

Again, be sure to leave a trailing newline at the end of the file.

This is mostly the same as the previous block, except for the port number and the `protocol websockets` line. There is no official standardized port for MQTT over websockets, but `8083` is the most common.

Save and exit the file, then restart Mosquitto.

```
$ sudo systemctl restart mosquitto
```

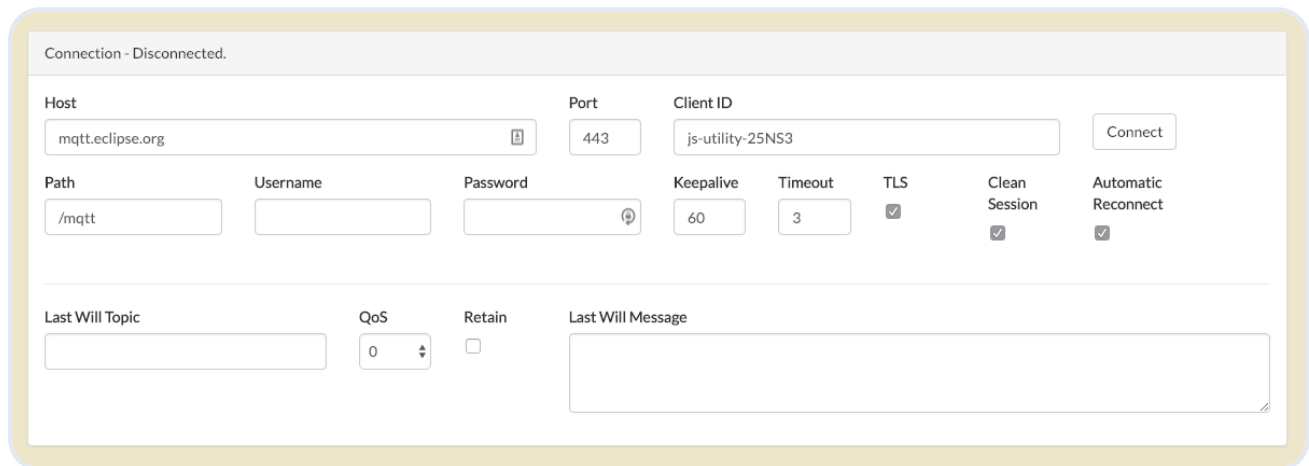
Copy

Now, open up port `8083` in the firewall.

```
$ sudo ufw allow 8083
```

Copy

To test this functionality, we'll use a public, browser-based MQTT client. There are a few out there, but the [Eclipse Paho JavaScript Client](#) is simple and straightforward to use. [Open the Paho client in your browser](#). You'll see the following:



The screenshot shows the Eclipse Paho JavaScript Client connection interface. At the top, it says "Connection - Disconnected." Below this, there are several input fields and checkboxes. The "Host" field contains "mqtt.eclipse.org". The "Port" field contains "443". The "Client ID" field contains "js-utility-25NS3". There is a "Connect" button to the right of the "Client ID" field. Below these, there are fields for "Path" (containing "/mqtt"), "Username" (empty), and "Password" (empty with a password icon). To the right of these are "Keepalive" (60), "Timeout" (3), "TLS" (checked), "Clean Session" (checked), and "Automatic Reconnect" (checked). At the bottom, there are fields for "Last Will Topic" (empty), "QoS" (0), "Retain" (unchecked), and "Last Will Message" (empty text area).

Fill out the connection information as follows:

- **Host** should be the domain for your Mosquitto server, `mqtt.example.com`.
- **Port** should be `8083`.
- **ClientId** can be left to the default value, `js-utility-DI1m6`.
- **Path** can be left to the default value, `/mqtt`.
- **Username** should be your Mosquitto username; here, we used **sammy**.
- **Password** should be the password you chose.

The remaining fields can be left to their default values.

After pressing **Connect**, the Paho browser-based client will connect to your Mosquitto server.

To publish a message, navigate to the **Publish Message** pane, fill out **Topic** as **test**, and enter any message in the **Message** section. Next, press **Publish**. The message will show up in your `mosquitto_sub` terminal.

## Conclusion

We've now set up a secure, password-protected and SSL-secured MQTT server. This can serve as a robust and secure messaging platform for whatever projects you dream up. Some popular software and hardware that work well with the MQTT protocol include:

- [OwnTracks](#), an open-source geo-tracking app you can install on your phone.

OwnTracks will periodically report position information to your MQTT server, which you could then store and display on a map, or create alerts and activate IoT hardware based on your location.

- [Node-RED](#) is a browser-based graphical interface for ‘wiring’ together the Internet of Things. You drag the output of one node to the input of another, and can route information through filters, between various protocols, into databases, and so on. MQTT is very well supported by Node-RED.
- The [ESP32](#) is an inexpensive wifi microcontroller with MQTT capabilities. You could wire one up to publish temperature data to a topic, or perhaps subscribe to a barometric pressure topic and sound a buzzer when a storm is coming!

These are just a few popular examples from the MQTT ecosystem. There is much more hardware and software out there that speaks the protocol. If you already have a favorite hardware platform, or software language, it probably has MQTT capabilities.

## About the authors



[Brian Boucheron](#) Author

Developer and author at DigitalOcean.



[Hanif Jetha](#) Author

Developer and author at DigitalOcean.



This work is licensed under a Creative Commons Attribution-NonCommercial- ShareAlike 4.0 International License.