mosquitto.conf man page

Name

mosquitto.conf — the configuration file for mosquitto

Synopsis

```
mosquitto.conf
```

Description

mosquitto.conf is the configuration file for mosquitto. This file can reside anywhere as long as mosquitto can read it. By default, mosquitto does not need a configuration file and will use the default values listed below. See mosquitto(8) for information on how to load a configuration file.

Mosquitto can be instructed to reload the configuration file by sending a SIGHUP signal as described in the Signals section of mosquitto(8). Not all configuration options can be reloaded, as detailed in the options below.

File Format

All lines with a # as the very first character are treated as a comment.

Configuration lines start with a variable name. The variable value is separated from the name by a single space.

Authentication

The authentication options described below allow a wide range of possibilities in conjunction with the listener options. This section aims to clarify the possibilities. An overview is also available at https://mosquitto.org/documentation/authentication-methods/

The simplest option is to have no authentication at all. This is the default if no other options are given. Unauthenticated encrypted support is provided by using the certificate based SSL/TLS based options certfile and keyfile.

MQTT provides username/password authentication as part of the protocol. Use the password_file option to define the valid usernames and passwords. Be sure to use network encryption if you are

using this option otherwise the username and password will be vulnerable to interception. Use the `per_listener_settings` to control whether passwords are required globally or on a per-listener basis.

Mosquitto provides the Dynamic Security plugin which handles username/password authentication and access control in a much more flexible way than a password file. See https://mosquitto.org/documentation/dynamic-security/

When using certificate based encryption there are three options that affect authentication. The first is require_certificate, which may be set to true or false. If false, the SSL/TLS component of the client will verify the server but there is no requirement for the client to provide anything for the server: authentication is limited to the MQTT built in username/password. If require_certificate is true, the client must provide a valid certificate in order to connect successfully. In this case, the second and third options, use_identity_as_username and use_subject_as_username, become relevant. If set to true, use_identity_as_username causes the Common Name (CN) from the client certificate to be used instead of the MQTT username for access control purposes. The password is not used because it is assumed that only authenticated clients have valid certificates. This means that any CA certificates you include in cafile or capath will be able to issue client certificates that are valid for connecting to your broker. If use_identity_as_username is false, the client must authenticate as normal (if required by password_file) through the MQTT options. The same principle applies for the use_subject_as_username option, but the entire certificate subject is used as the username instead of just the CN.

When using pre-shared-key based encryption through the psk_hint and psk_file options, the client must provide a valid identity and key in order to connect to the broker before any MQTT communication takes place. If use_identity_as_username is true, the PSK identity is used instead of the MQTT username for access control purposes. If use_identity_as_username is false, the client may still authenticate using the MQTT username/password if using the password_file option.

Both certificate and PSK based encryption are configured on a per-listener basis.

Authentication plugins can be created to augment the password_file, acl_file and psk_file options with e.g. SQL based lookups.

It is possible to support multiple authentication schemes at once. A config could be created that had a listener for all of the different encryption options described above and hence a large number of ways of authenticating.

General Options

`acl_file` *file path*

Set the path to an access control list file. If defined, the contents of the file are used to control client access to topics on the broker.

If this parameter is defined then only the topics listed will have access. Topic access is added with lines of the format:

```
topic [read|write|readwrite|deny] <topic>
```

The access type is controlled using "read", "write", "readwrite" or "deny". This parameter is optional (unless <topic> includes a space character) - if not given then the access is read/write. <topic> can contain the + or # wildcards as in subscriptions. The "deny" option can used to explicitly deny access to a topic that would otherwise be granted by a broader read/write/readwrite statement. Any "deny" topics are handled before topics that grant read/write access.

The first set of topics are applied to anonymous clients, assuming `allow_anonymous` is true. User specific topic ACLs are added after a user line as follows:

```
user <username>
```

The username referred to here is the same as in `password_file` . It is not the clientid.

It is also possible to define ACLs based on pattern substitution within the topic. The form is the same as for the topic keyword, but using pattern as the keyword.

```
pattern [read|write|readwrite|deny] <topic>
```

The patterns available for substition are:

- `%c` to match the client id of the client
- `%u` to match the username of the client

The substitution pattern must be the only text for that level of hierarchy. Pattern ACLs apply to all users even if the "user" keyword has previously been given.

Example:

```
pattern write sensor/%u/data
```

Allow access for bridge connection messages:

```
pattern write $SYS/broker/connection/%c/state
```

If the first character of a line of the ACL file is a # it is treated as a comment.

If `per_listener_settings` is *`true`* , this option applies to the current listener being configured only. If `per_listener_settings` is *`false`* , this option applies to all listeners.

Reloaded on reload signal. The currently loaded ACLs will be freed and reloaded. Existing subscriptions will be affected after the reload.

See also https://mosquitto.org/documentation/dynamic-security/

`allow_anonymous` [true | false]

Boolean value that determines whether clients that connect without providing a username are allowed to connect. If set to *`false`* then another means of connection should be created to control authenticated client access.

Defaults to *`false`* , unless no listeners are defined in the configuration file, in which case it set to *`true`* , but connections are only allowed from the local machine.

If `per_listener_settings` is *`true`* , this option applies to the current listener being configured only. If `per_listener_settings` is *`false`* , this option applies to all listeners.

Important

In version 1.6.x and earlier, this option defaulted to *`true`* unless there was another security option set.

Reloaded on reload signal.

`allow_duplicate_messages` [true | false]

This option is deprecated and will be removed in a future version. The behaviour will default to true.

If a client is subscribed to multiple subscriptions that overlap, e.g. foo/# and foo/+/baz , then MQTT expects that when the broker receives a message on a topic that matches both subscriptions, such as foo/bar/baz, then the client should only receive the message once.

Mosquitto keeps track of which clients a message has been sent to in order to meet this requirement. This option allows this behaviour to be disabled, which may be useful if you have a large number of clients subscribed to the same set of topics and want to minimise memory usage.

It can be safely set to *`true`* if you know in advance that your clients will never have overlapping subscriptions, otherwise your clients must be able to correctly deal with duplicate messages even when then have QoS=2.

Defaults to *true* .

This option applies globally.

Reloaded on reload signal.

`allow_zero_length_clientid` [true | false]

MQTT 3.1.1 and MQTT 5 allow clients to connect with a zero length client id and have the broker generate a client id for them. Use this option to allow/disallow this behaviour. Defaults to true.

See also the `auto_id_prefix` option.

If `per_listener_settings` is *true* , this option applies to the current listener being configured only. If `per_listener_settings` is *false* , this option applies to all listeners.

Reloaded on reload signal.

`auth_plugin_deny_special_chars` [true | false]

If *true* then before an ACL check is made, the username/client id of the client needing the check is searched for the presence of either a '+' or '#' character. If either of these characters is found in either the username or client id, then the ACL check is denied before it is sent to the plugin.

This check prevents the case where a malicious user could circumvent an ACL check by using one of these characters as their username or client id. This is the same issue as was reported with mosquitto itself as CVE-2017-7650.

If you are entirely sure that the plugin you are using is not vulnerable to this attack (i.e. if you never use usernames or client ids in topics) then you can disable this extra check and hence have all ACL checks delivered to your plugin by setting this option to *false* .

Defaults to *true* .

Applies to the current authentication plugin being configured.

Not currently reloaded on reload signal.

`auto_id_prefix` *prefix*

If `allow_zero_length_clientid` is *true* , this option allows you to set a string that will be prefixed to the automatically generated client ids to aid visibility in logs. Defaults to `auto-` .

If `per_listener_settings` is *true* , this option applies to the current listener being configured only. If `per_listener_settings` is *false* , this option applies to all listeners.

Reloaded on reload signal.

`autosave_interval` *seconds*

The number of seconds that mosquitto will wait between each time it saves the in-memory database to disk. If set to 0, the in-memory database will only be saved when mosquitto exits or when receiving the SIGUSR1 signal. Note that this setting only has an effect if persistence is enabled. Defaults to 1800 seconds (30 minutes).

This option applies globally.

Reloaded on reload signal.

`autosave_on_changes` [true | false]

If *true*, mosquitto will count the number of subscription changes, retained messages received and queued messages and if the total exceeds `autosave_interval` then the in-memory database will be saved to disk. If *false*, mosquitto will save the in-memory database to disk by treating `autosave_interval` as a time in seconds.

This option applies globally.

Reloaded on reload signal.

`check_retain_source` [true | false]

This option affects the scenario when a client subscribes to a topic that has retained messages. It is possible that the client that published the retained message to the topic had access at the time they published, but that access has been subsequently removed. If `check_retain_source` is set to true, the default, the source of a retained message will be checked for access rights before it is republished. When set to false, no check will be made and the retained message will always be published.

This option applies globally, regardless of the `per_listener_settings` option.

`clientid_prefixes` *prefix*

This option is deprecated and will be removed in a future version.

If defined, only clients that have a clientid with a prefix that matches clientid_prefixes will be allowed to connect to the broker. For example, setting "secure-" here would mean a client "secure-client" could connect but another with clientid "mqtt" couldn't. By default, all client ids are valid.

This option applies globally.

Reloaded on reload signal. Note that currently connected clients will be unaffected by any changes.

`connection_messages` [true | false]

If set to *true*, the log will include entries when clients connect and disconnect. If set to *false*,

these entries will not appear.

This option applies globally.

Reloaded on reload signal.

`include_dir` *dir*

External configuration files may be included by using the include_dir option. This defines a directory that will be searched for config files. All files that end in '.conf' will be loaded as a configuration file. It is best to have this as the last option in the main file. This option will only be processed from the main configuration file. The directory specified must not contain the main configuration file.

The configuration files in `include_dir` are loaded in case sensitive alphabetical order, with the upper case of each letter ordered before the lower case of the same letter.

Example Load Order for include_dir.
Given the files *b.conf*, *A.conf*, *01.conf*, *a.conf*, *B.conf*, and *00.conf* inside `include_dir`, the config files would be loaded in this order:

```
00.conf
01.conf
A.conf
a.conf
B.conf
b.conf
```

If this option is used multiple times, then each `include_dir` option is processed completely in the order that they are written in the main configuration file.

Example Load Order for Multiple include_dir.
Assuming a directory *one.d* containing files *B.conf* and *C.conf*, and a second directory *two.d* containing files *A.conf* and *D.conf*, and a config:

```
include_dir one.d
include_dir two.d
```

Then the config files would be loaded in this order:

```
# files from one.d
B.conf
C.conf
```

```
# files from two.d
A.conf
D.conf
```

`log_dest` *destinations*

Send log messages to a particular destination. Possible destinations are: `stdout` `stderr` `syslog` `topic` `file` `dlt` .

`stdout` and `stderr` log to the console on the named output.

`syslog` uses the userspace syslog facility which usually ends up in /var/log/messages or similar.

`topic` logs to the broker topic '$SYS/broker/log/<severity>', where severity is one of E, W, N, I, M which are error, warning, notice, information and message. Message type severity is used by the subscribe and unsubscribe log_type options and publishes log messages at $SYS/broker/log/M /subscribe and $SYS/broker/log/M/unsubscribe. Debug messages are never logged on topics.

The `file` destination requires an additional parameter which is the file to be logged to, e.g. "log_dest file /var/log/mosquitto.log". The file will be closed and reopened when the broker receives a HUP signal. Only a single file destination may be configured.

The `dlt` destination is for the automotive `Diagnostic Log and Trace` tool. This requires that Mosquitto has been compiled with DLT support.

Use "log_dest none" if you wish to disable logging. Defaults to stderr. This option may be specified multiple times.

Note that if the broker is running as a Windows service it will default to "log_dest none" and neither stdout nor stderr logging is available.

Reloaded on reload signal.

`log_facility` *local facility*

If using syslog logging (not on Windows), messages will be logged to the "daemon" facility by default. Use the `log_facility` option to choose which of local0 to local7 to log to instead. The option value should be an integer value, e.g. "log_facility 5" to use local5.

`log_timestamp` [true | false]

Boolean value, if set to *true* a timestamp value will be added to each log entry. The default is *true* .

Reloaded on reload signal.

`log_timestamp_format` *format*

Set the format of the log timestamp. If left unset, this is the number of seconds since the Unix epoch. This option is a free text string which will be passed to the strftime function as the format specifier. To get an ISO 8601 datetime, for example:

```
log_timestamp_format %Y-%m-%dT%H:%M:%S
```

Reloaded on reload signal.

`log_type` *types*

Choose types of messages to log. Possible types are: *debug* , *error* , *warning* , *notice* , *information* , *subscribe* , *unsubscribe* , *websockets* , *none* , *all* .

Defaults to *error* , *warning* , *notice* and *information* . This option may be specified multiple times. Note that the *debug* type (used for decoding incoming/outgoing network packets) is never logged in topics.

Reloaded on reload signal.

`max_inflight_bytes` *count*

Outgoing QoS 1 and 2 messages will be allowed in flight until this byte limit is reached. This allows control of outgoing message rate based on message size rather than message count. If the limit is set to 100, messages of over 100 bytes are still allowed, but only a single message can be in flight at once. Defaults to 0. (No limit).

See also the `max_inflight_messages` option.

This option applies globally.

Reloaded on reload signal.

`max_inflight_messages` *count*

The maximum number of outgoing QoS 1 or 2 messages that can be in the process of being transmitted simultaneously. This includes messages currently going through handshakes and messages that are being retried. Defaults to 20. Set to 0 for no maximum. If set to 1, this will guarantee in-order delivery of messages.

This option applies globally.

Reloaded on reload signal.

`max_keepalive` *value*

For MQTT v5 clients, it is possible to have the server send a "server keepalive" value that will override the keepalive value set by the client. This is intended to be used as a mechanism to say that the server will disconnect the client earlier than it anticipated, and that the client should use the new keepalive value. The max_keepalive option allows you to specify that clients may only connect with keepalive less than or equal to this value, otherwise they will be sent a server keepalive telling them to use max_keepalive. This only applies to MQTT v5 clients. The maximum value allowable, and default value, is 65535. Do not set below 10 seconds.

For MQTT v3.1.1 and v3.1 clients, there is no mechanism to tell the client what keepalive value they should use. If an MQTT v3.1.1 or v3.1 client specifies a keepalive time greater than max_keepalive they will be sent a CONNACK message with the "identifier rejected" reason code, and disconnected.

This option applies globally.

Reloaded on reload signal.

`max_packet_size` *value*

For MQTT v5 clients, it is possible to have the server send a "maximum packet size" value that will instruct the client it will not accept MQTT packets with size greater than `value` bytes. This applies to the full MQTT packet, not just the payload. Setting this option to a positive value will set the maximum packet size to that number of bytes. If a client sends a packet which is larger than this value, it will be disconnected. This applies to all clients regardless of the protocol version they are using, but v3.1.1 and earlier clients will of course not have received the maximum packet size information. Defaults to no limit.

This option applies to all clients, not just those using MQTT v5, but it is not possible to notify clients using MQTT v3.1.1 or MQTT v3.1 of the limit.

Setting below 20 bytes is forbidden because it is likely to interfere with normal client operation even with small payloads.

This option applies globally.

Reloaded on reload signal.

`max_queued_bytes` *count*

The number of outgoing QoS 1 and 2 messages above those currently in-flight will be queued (per client) by the broker. Once this limit has been reached, subsequent messages will be silently dropped. This is an important option if you are sending messages at a high rate and/or have clients who are slow to respond or may be offline for extended periods of time. Defaults to 0. (No maximum).

See also the `max_queued_messages` option. If both max_queued_messages and max_queued_bytes are specified, packets will be queued until the first limit is reached.

This option applies globally.

Reloaded on reload signal.

`max_queued_messages` *count*

The maximum number of QoS 1 or 2 messages to hold in the queue (per client) above those messages that are currently in flight. Defaults to 1000. Set to 0 for no maximum (not recommended). See also the `queue_qos0_messages` and `max_queued_bytes` options.

This option applies globally.

Reloaded on reload signal.

`memory_limit` *limit*

This option sets the maximum number of heap memory bytes that the broker will allocate, and hence sets a hard limit on memory use by the broker. Memory requests that exceed this value will be denied. The effect will vary depending on what has been denied. If an incoming message is being processed, then the message will be dropped and the publishing client will be disconnected. If an outgoing message is being sent, then the individual message will be dropped and the receiving client will be disconnected. Defaults to no limit.

This option is only available if memory tracking support is compiled in.

Reloaded on reload signal. Setting to a lower value and reloading will not result in memory being freed.

`message_size_limit` *limit*

This option sets the maximum publish payload size that the broker will allow. Received messages that exceed this size will not be accepted by the broker. This means that the message will not be forwarded on to subscribing clients, but the QoS flow will be completed for QoS 1 or QoS 2 messages. MQTT v5 clients using QoS 1 or QoS 2 will receive a PUBACK or PUBREC with the "implementation specific error" reason code.

The default value is 0, which means that all valid MQTT messages are accepted. MQTT imposes a maximum payload size of 268435455 bytes.

This option applies globally.

Reloaded on reload signal.

`password_file` *file path*

Set the path to a password file. If defined, the contents of the file are used to control client access to the broker. The file can be created using the mosquitto_passwd(1) utility. If mosquitto is compiled without TLS support (it is recommended that TLS support is included), then the password file should be a text file with each line in the format "username:password", where the colon and password are optional but recommended. If `allow_anonymous` is set to *false*, only users defined in this file will be able to connect. Setting `allow_anonymous` to *true* when *password_file* is defined is valid and could be used with acl_file to have e.g. read only guest/anonymous accounts and defined users that can publish.

If `per_listener_settings` is *true*, this option applies to the current listener being configured only. If `per_listener_settings` is *false*, this option applies to all listeners.

Reloaded on reload signal. The currently loaded username and password data will be freed and reloaded. Clients that are already connected will not be affected.

See also mosquitto_passwd(1) and https://mosquitto.org/documentation/dynamic-security/

`per_listener_settings` [true | false]

If *true*, then authentication and access control settings will be controlled on a per-listener basis. The following options are affected:

`password_file`, `acl_file`, `psk_file`, `allow_anonymous`, `allow_zero_length_clientid`, `auto_id_prefix`.

`plugin`, `plugin_opt_*`,

Note that if set to true, then a durable client (i.e. with clean session set to false) that has disconnected will use the ACL settings defined for the listener that it was most recently connected to.

The default behaviour is for this to be set to *false*, which maintains the settings behaviour from previous versions of mosquitto.

Reloaded on reload signal.

`persistence` [true | false]

If *true*, connection, subscription and message data will be written to the disk in mosquitto.db at the location dictated by persistence_location. When mosquitto is restarted, it will reload the information stored in mosquitto.db. The data will be written to disk when mosquitto closes and also at periodic intervals as defined by autosave_interval. Writing of the persistence database may also be forced by sending mosquitto the SIGUSR1 signal. If *false*, the data will be stored in memory

only. Defaults to *false* .

The persistence file may change its format in a new version. The broker can currently read all old formats, but will only save in the latest format. It should always be safe to upgrade, but cautious users may wish to take a copy of the persistence file before installing a new version so that they can roll back to an earlier version if necessary.

This option applies globally.

Reloaded on reload signal.

`persistence_file` *file name*

The filename to use for the persistent database. Defaults to mosquitto.db.

This option applies globally.

Reloaded on reload signal.

`persistence_location` *path*

The path where the persistence database should be stored. If not given, then the current directory is used.

This option applies globally.

Reloaded on reload signal.

`persistent_client_expiration` *duration*

This option allows persistent clients (those with clean session set to false) to be removed if they do not reconnect within a certain time frame. This is a non-standard option. As far as the MQTT spec is concerned, persistent clients persist forever.

Badly designed clients may set clean session to false whilst using a randomly generated client id. This leads to persistent clients that will never reconnect. This option allows these clients to be removed.

The expiration period should be an integer followed by one of h d w m y for hour, day, week, month and year respectively. For example:

- `persistent_client_expiration 2m`
- `persistent_client_expiration 14d`
- `persistent_client_expiration 1y`

As this is a non-standard option, the default if not set is to never expire persistent clients.

This option applies globally.

Reloaded on reload signal.

`pid_file` *file path*

Write a pid file to the file specified. If not given (the default), no pid file will be written. If the pid file cannot be written, mosquitto will exit.

If mosquitto is being automatically started by an init script it will usually be required to write a pid file. This should then be configured as e.g. /var/run/mosquitto/mosquitto.pid

Not reloaded on reload signal.

`plugin_opt_*` *value*

Options to be passed to the most recent `plugin` defined in the configuration file. See the specific plugin instructions for details of what options are available.

Applies to the current plugin being configured.

This is also available as the `auth_opt_*` option, but this use is deprecated and will be removed in a future version.

`plugin` *file path*

Specify an external module to use for authentication and access control. This allows custom username/password and access control functions to be created.

Can be specified multiple times to load multiple plugins. The plugins will be processed in the order that they are specified.

If `password_file` , or `acl_file` are used in the config file alongsize `plugin` , the plugin checks will run after the built in checks.

Not currently reloaded on reload signal.

See also https://mosquitto.org/documentation/dynamic-security/

This is also available as the `auth_plugin` option, but this use is deprecated and will be removed in a future version.

`psk_file` *file path*

Set the path to a pre-shared-key file. This option requires a listener to be have PSK support enabled. If defined, the contents of the file are used to control client access to the broker. Each line

should be in the format "identity:key", where the key is a hexadecimal string with no leading "0x". A client connecting to a listener that has PSK support enabled must provide a matching identity and PSK to allow the encrypted connection to proceed.

If `per_listener_settings` is *`true`*, this option applies to the current listener being configured only. If `per_listener_settings` is *`false`*, this option applies to all listeners.

Reloaded on reload signal. The currently loaded identity and key data will be freed and reloaded. Clients that are already connected will not be affected.

`queue_qos0_messages` [true | false]

Set to *`true`* to queue messages with QoS 0 when a persistent client is disconnected. These messages are included in the limit imposed by max_queued_messages. Defaults to *`false`*.

Note that the MQTT v3.1.1 spec states that only QoS 1 and 2 messages should be saved in this situation so this is a non-standard option.

This option applies globally.

Reloaded on reload signal.

`retain_available` [true | false]

If set to false, then retained messages are not supported. Clients that send a message with the retain bit will be disconnected if this option is set to false. Defaults to true.

This option applies globally.

Reloaded on reload signal.

`set_tcp_nodelay` [true | false]

If set to true, the TCP_NODELAY option will be set on client sockets to disable Nagle's algorithm. This has the effect of reducing latency of some messages at potentially increasing the number of TCP packets being sent. Defaults to false.

This option applies globally.

Reloaded on reload signal.

`sys_interval` *`seconds`*

The integer number of seconds between updates of the $SYS subscription hierarchy, which provides status information about the broker. If unset, defaults to 10 seconds.

Set to 0 to disable publishing the $SYS hierarchy completely.

This option applies globally.

Reloaded on reload signal.

`upgrade_outgoing_qos` [true | false]

The MQTT specification requires that the QoS of a message delivered to a subscriber is never upgraded to match the QoS of the subscription. Enabling this option changes this behaviour. If `upgrade_outgoing_qos` is set *`true`*, messages sent to a subscriber will always match the QoS of its subscription. This is a non-standard option not provided for by the spec. Defaults to *`false`*.

This option applies globally.

Reloaded on reload signal.

`user` *`username`*

When run as root, change to this user and its primary group on startup. If set to "mosquitto" or left unset, and if the "mosquitto" user does not exist, then mosquitto will change to the "nobody" user instead. If this is set to another value and mosquitto is unable to change to this user and group, it will exit with an error. The user specified must have read/write access to the persistence database if it is to be written. If run as a non-root user, this setting has no effect. Defaults to mosquitto.

This setting has no effect on Windows and so you should run mosquitto as the user you wish it to run as.

Not reloaded on reload signal.

Listeners

The network ports that mosquitto listens on can be controlled using listeners. The default listener options can be overridden and further listeners can be created.

General Options

`bind_address` *`address`*

This option is deprecated and will be removed in a future version. Use the `listener` instead.

Listen for incoming network connections on the specified IP address/hostname only. This is useful to restrict access to certain network interfaces. To restrict access to mosquitto to the local host only, use "bind_address localhost". This only applies to the default listener. Use the `listener` option to control other listeners.

It is recommended to use an explicit `listener` rather than rely on the implicit default listener

options like this.

Not reloaded on reload signal.

`bind_interface` *device*

Listen for incoming network connections only on the specified interface. This is similar to the `bind_address` option but is useful when an interface has multiple addresses or the address may change.

If used at the same time as the `bind_address` for the default listener, or the *bind address/host* part of the `listener`, then `bind_interface` will take priority.

This option is not available on Windows.

Not reloaded on reload signal.

`http_dir` *directory*

When a listener is using the websockets protocol, it is possible to serve http data as well. Set `http_dir` to a directory which contains the files you wish to serve. If this option is not specified, then no normal http connections will be possible.

Not reloaded on reload signal.

`listener` *port [bind address/host/unix socket path]*

Listen for incoming network connection on the specified port. A second optional argument allows the listener to be bound to a specific ip address/hostname. If this variable is used and neither the global `bind_address` nor `port` options are used then the default listener will not be started.

The `bind address/host` option allows this listener to be bound to a specific IP address by passing an IP address or hostname. For websockets listeners, it is only possible to pass an IP address here.

On systems that support Unix Domain Sockets, this option can also be used to create a Unix socket rather than opening a TCP socket. In this case, the port must be set to 0, and the unix socket path must be given.

This option may be specified multiple times. See also the `mount_point` option.

Not reloaded on reload signal.

`max_connections` *count*

Limit the total number of clients connected for the current listener. Set to `-1` to have "unlimited" connections. Note that other limits may be imposed that are outside the control of mosquitto. See

e.g. limits.conf.

Not reloaded on reload signal.

`max_qos` *value*

Limit the QoS value allowed for clients connecting to this listener. Defaults to 2, which means any QoS can be used. Set to 0 or 1 to limit to those QoS values. This makes use of an MQTT v5 feature to notify clients of the limitation. MQTT v3.1.1 clients will not be aware of the limitation. Clients publishing to this listener with a too-high QoS will be disconnected.

Not reloaded on reload signal.

`max_topic_alias` *number*

This option sets the maximum number topic aliases that an MQTT v5 client is allowed to create. This option applies per listener. Defaults to 10. Set to 0 to disallow topic aliases. The maximum value possible is 65535.

Not reloaded on reload signal.

`mount_point` *topic prefix*

This option is used with the listener option to isolate groups of clients. When a client connects to a listener which uses this option, the string argument is attached to the start of all topics for this client. This prefix is removed when any messages are sent to the client. This means a client connected to a listener with mount point *example* can only see messages that are published in the topic hierarchy *example* and below.

Not reloaded on reload signal.

`port` *port number*

This option is deprecated and will be removed in a future version. Use the `listener` instead.

Set the network port for the default listener to listen on. Defaults to 1883.

Not reloaded on reload signal.

It is recommended to use an explicit `listener` rather than rely on the implicit default listener options like this.

`protocol` *value*

Set the protocol to accept for the current listener. Can be `mqtt` , the default, or `websockets` if available.

Websockets support is currently disabled by default at compile time. Certificate based TLS may be

used with websockets, except that only the `cafile`, `certfile`, `keyfile`, `ciphers`, and `ciphers_tls1.3` options are supported.

Not reloaded on reload signal.

`socket_domain` [ipv4 | ipv6]

By default, a listener will attempt to listen on all supported IP protocol versions. If you do not have an IPv4 or IPv6 interface you may wish to disable support for either of those protocol versions. In particular, note that due to the limitations of the websockets library, it will only ever attempt to open IPv6 sockets if IPv6 support is compiled in, and so will fail if IPv6 is not available.

Set to `ipv4` to force the listener to only use IPv4, or set to `ipv6` to force the listener to only use IPv6. If you want support for both IPv4 and IPv6, then do not use the `socket_domain` option.

Not reloaded on reload signal.

`use_username_as_clientid` [true | false]

Set `use_username_as_clientid` to true to replace the clientid that a client connected with its username. This allows authentication to be tied to the clientid, which means that it is possible to prevent one client disconnecting another by using the same clientid. Defaults to false.

If a client connects with no username it will be disconnected as not authorised when this option is set to true. Do not use in conjunction with `clientid_prefixes`.

See also `use_identity_as_username`.

Not reloaded on reload signal.

`websockets_log_level` *level*

Change the websockets logging level. This is a global option, it is not possible to set per listener. This is an integer that is interpreted by libwebsockets as a bit mask for its lws_log_levels enum. See the libwebsockets documentation for more details.

To use this option, `log_type websockets` must also be enabled. Defaults to 0.

`websockets_headers_size` *size*

Change the websockets headers size. This is a global option, it is not possible to set per listener. This option sets the size of the buffer used in the libwebsockets library when reading HTTP headers. If you are passing large header data such as cookies then you may need to increase this value. If left unset, or set to 0, then the default of 1024 bytes will be used.

Certificate based SSL/TLS Support

The following options are available for all listeners to configure certificate based SSL support. See also "Pre-shared-key based SSL/TLS support".

`cafile` *file path*

`cafile` is used to define the path to a file containing the PEM encoded CA certificates that are trusted when checking incoming client certificates.

`capath` *directory path*

`capath` is used to define a directory that contains PEM encoded CA certificates that are trusted when checking incoming client certificates. For `capath` to work correctly, the certificates files must have ".pem" as the file ending and you must run "openssl rehash <path to capath>" each time you add/remove a certificate.

`certfile` *file path*

Path to the PEM encoded server certificate. This option and `keyfile` must be present to enable certificate based TLS encryption.

The certificate pointed to by this option will be reloaded when Mosquitto receives a SIGHUP signal. This can be used to load new certificates prior to the existing ones expiring.

`ciphers` *cipher:list*

The list of allowed ciphers for this listener, for TLS v1.2 and earlier only, each separated with a colon. Available ciphers can be obtained using the "openssl ciphers" command.

`ciphers_tls1.3` *cipher:list*

The list of allowed ciphersuites for this listener, for TLS v1.3, each separated with a colon.

`crlfile` *file path*

If you have `require_certificate` set to *true* , you can create a certificate revocation list file to revoke access to particular client certificates. If you have done this, use crlfile to point to the PEM encoded revocation file.

`dhparamfile` *file path*

To allow the use of ephemeral DH key exchange, which provides forward security, the listener must load DH parameters. This can be specified with the dhparamfile option. The dhparamfile can be generated with the command e.g.

```
openssl dhparam -out dhparam.pem 2048
```

`keyfile` *file path*

Path to the PEM encoded server key. This option and `certfile` must be present to enable

certificate based TLS encryption.

The private key pointed to by this option will be reloaded when Mosquitto receives a SIGHUP signal. This can be used to load new keys prior to the existing ones expiring.

`require_certificate` [true | false]

By default an SSL/TLS enabled listener will operate in a similar fashion to a https enabled web server, in that the server has a certificate signed by a CA and the client will verify that it is a trusted certificate. The overall aim is encryption of the network traffic. By setting `require_certificate` to *`true`*, a client connecting to this listener must provide a valid certificate in order for the network connection to proceed. This allows access to the broker to be controlled outside of the mechanisms provided by MQTT.

`tls_engine` *`engine`*

A valid openssl engine id. These can be listed with openssl engine command.

`tls_engine_kpass_sha1` *`engine_kpass_sha1`*

SHA1 of the private key password when using an TLS engine. Some TLS engines such as the TPM engine may require the use of a password in order to be accessed. This option allows a hex encoded SHA1 hash of the password to the engine directly, instead of the user being prompted for the password.

`tls_keyform` [pem | engine]

Specifies the type of private key in use when making TLS connections.. This can be "pem" or "engine". This parameter is useful when a TPM module is being used and the private key has been created with it. Defaults to "pem", which means normal private key files are used.

`tls_version` *`version`*

Configure the minimum version of the TLS protocol to be used for this listener. Possible values are *`tlsv1.3`*, *`tlsv1.2`* and *`tlsv1.1`*. If left unset, the default of allowing TLS v1.3 and v1.2.

In Mosquitto version 1.6.x and earlier, this option set the only TLS protocol version that was allowed, rather than the minimum.

`use_identity_as_username` [true | false]

If `require_certificate` is *`true`*, you may set `use_identity_as_username` to *`true`* to use the CN value from the client certificate as a username. If this is *`true`*, the `password_file` option will not be used for this listener.

This takes priority over `use_subject_as_username` if both are set to *`true`*.

See also `use_subject_as_username`

`use_subject_as_username` [true | false]

If `require_certificate` is *`true`* , you may set `use_subject_as_username` to *`true`* to use the complete subject value from the client certificate as a username. If this is *`true`* , the `password_file` option will not be used for this listener.

The subject will be generated in a form similar to `CN=test client,OU=Production,O=Server,L=Nottingham,ST=Nottinghamshire,C=GB` .

See also `use_identity_as_username`

Pre-shared-key based SSL/TLS Support

The following options are available for all listeners to configure pre-shared-key based SSL support. See also "Certificate based SSL/TLS support".

`ciphers` *`cipher:list`*

When using PSK, the encryption ciphers used will be chosen from the list of available PSK ciphers. If you want to control which ciphers are available, use this option. The list of available ciphers can be optained using the "openssl ciphers" command and should be provided in the same format as the output of that command.

`psk_hint` *`hint`*

The `psk_hint` option enables pre-shared-key support for this listener and also acts as an identifier for this listener. The hint is sent to clients and may be used locally to aid authentication. The hint is a free form string that doesn't have much meaning in itself, so feel free to be creative.

If this option is provided, see `psk_file` to define the pre-shared keys to be used or create a security plugin to handle them.

`tls_version` *`version`*

Configure the minimum version of the TLS protocol to be used for this listener. Possible values are *`tlsv1.3`* , *`tlsv1.2`* and *`tlsv1.1`* . If left unset, the default of allowing TLS v1.3 and v1.2.

In Mosquitto version 1.6.x and earlier, this option set the only TLS protocol version that was allowed, rather than the minimum.

`use_identity_as_username` [true | false]

Set `use_identity_as_username` to have the psk identity sent by the client used as its username. The username will be checked as normal, so `password_file` or another means of authentication checking must be used. No password will be used.

Configuring Bridges

Multiple bridges (connections to other brokers) can be configured using the following variables.

Bridges cannot currently be reloaded on reload signal.

`address` *address[:port] [address[:port]]* ,

`addresses` *address[:port] [address[:port]]*

Specify the address and optionally the port of the bridge to connect to. This must be given for each bridge connection. If the port is not specified, the default of 1883 is used.

If you use an IPv6 address, then the port is not optional.

Multiple host addresses can be specified on the address config. See the `round_robin` option for more details on the behaviour of bridges with multiple addresses.

`bridge_attempt_unsubscribe` [true | false]

If a bridge has topics that have "out" direction, the default behaviour is to send an unsubscribe request to the remote broker on that topic. This means that changing a topic direction from "in" to "out" will not keep receiving incoming messages. Sending these unsubscribe requests is not always desirable, setting `bridge_attempt_unsubscribe` to *false* will disable sending the unsubscribe request. Defaults to *true* .

`bridge_bind_address` *ip address*

If you need to have the bridge connect over a particular network interface, use bridge_bind_address to tell the bridge which local IP address the socket should bind to, e.g. `bridge_bind_address 192.168.1.10` .

`bridge_max_packet_size` *value*

If you wish to restrict the size of messages sent to a remote bridge, use this option. This sets the maximum number of bytes for the total message, including headers and payload. Note that MQTT v5 brokers may provide their own maximum-packet-size property. In this case, the smaller of the two limits will be used. Set to 0 for "unlimited".

`bridge_outgoing_retain` [true | false]

Some MQTT brokers do not allow retained messages. MQTT v5 gives a mechanism for brokers to tell clients that they do not support retained messages, but this is not possible for MQTT v3.1.1 or v3.1. If you need to bridge to a v3.1.1 or v3.1 broker that does not support retained messages, set the `bridge_outgoing_retain` option to *false* . This will remove the retain bit on all outgoing messages to that bridge, regardless of any other setting. Defaults to *true* .

`bridge_protocol_version` *version*

Set the version of the MQTT protocol to use with for this bridge. Can be one of *mqttv50*, *mqttv311* or *mqttv31*. Defaults to *mqttv311*.

`cleansession` [true | false]

Set the clean session option for this bridge. Setting to *false* (the default), means that all subscriptions on the remote broker are kept in case of the network connection dropping. If set to *true*, all subscriptions and messages on the remote broker will be cleaned up if the connection drops. Note that setting to *true* may cause a large amount of retained messages to be sent each time the bridge reconnects.

If you are using bridges with `cleansession` set to *false* (the default), then you may get unexpected behaviour from incoming topics if you change what topics you are subscribing to. This is because the remote broker keeps the subscription for the old topic. If you have this problem, connect your bridge with `cleansession` set to *true*, then reconnect with cleansession set to *false* as normal.

`local_cleansession` [true | false]

The regular `cleansession` covers both the local subscriptions and the remote subscriptions. local_cleansession allows splitting this. Setting *false* will mean that the local connection will preserve subscription, independent of the remote connection.

Defaults to the value of bridge.cleansession unless explicitly specified.

`connection` *name*

This variable marks the start of a new bridge connection. It is also used to give the bridge a name which is used as the client id on the remote broker.

`keepalive_interval` *seconds*

Set the number of seconds after which the bridge should send a ping if no other traffic has occurred. Defaults to 60. A minimum value of 5 seconds is allowed.

`idle_timeout` *seconds*

Set the amount of time a bridge using the lazy start type must be idle before it will be stopped. Defaults to 60 seconds.

`local_clientid` *id*

Set the clientid to use on the local broker. If not defined, this defaults to `local.<remote_clientid>`. If you are bridging a broker to itself, it is important that local_clientid and remote_clientid do not match.

`local_password` *password*

Configure the password to be used when connecting this bridge to the local broker. This may be important when authentication and ACLs are being used.

`local_username` *username*

Configure the username to be used when connecting this bridge to the local broker. This may be important when authentication and ACLs are being used.

`notifications` [true | false]

If set to *true*, publish notification messages to the local and remote brokers giving information about the state of the bridge connection. Retained messages are published to the topic $SYS/broker/connection/<remote_clientid>/state unless otherwise set with `notification_topic`s. If the message is 1 then the connection is active, or 0 if the connection has failed. Defaults to *true*.

This uses the Last Will and Testament (LWT) feature.

`notifications_local_only` [true | false]

If set to *true*, only publish notification messages to the local broker giving information about the state of the bridge connection. Defaults to *false*.

`notification_topic` *topic*

Choose the topic on which notifications will be published for this bridge. If not set the messages will be sent on the topic $SYS/broker/connection/<remote_clientid>/state.

`remote_clientid` *id*

Set the client id for this bridge connection. If not defined, this defaults to 'name.hostname', where name is the connection name and hostname is the hostname of this computer.

This replaces the old "clientid" option to avoid confusion with local/remote sides of the bridge. "clientid" remains valid for the time being.

`remote_password` *value*

Configure a password for the bridge. This is used for authentication purposes when connecting to a broker that supports MQTT v3.1 and up and requires a username and/or password to connect. This option is only valid if a remote_username is also supplied.

This replaces the old "password" option to avoid confusion with local/remote sides of the bridge. "password" remains valid for the time being.

`remote_username` *name*

Configure a username for the bridge. This is used for authentication purposes when connecting to a broker that supports MQTT v3.1 and up and requires a username and/or password to connect. See

also the `remote_password` option.

This replaces the old "username" option to avoid confusion with local/remote sides of the bridge. "username" remains valid for the time being.

`restart_timeout` *base cap* ,
`restart_timeout` *constant*

Set the amount of time a bridge using the automatic start type will wait until attempting to reconnect.

This option can be configured to use a constant delay time in seconds, or to use a backoff mechanism based on "Decorrelated Jitter", which adds a degree of randomness to when the restart occurs, starting at the base and increasing up to the cap. Set a constant timeout of 20 seconds:

```
restart_timeout 20
```

Set backoff with a base (start value) of 10 seconds and a cap (upper limit) of 60 seconds:

```
restart_timeout 10 30
```

Defaults to jitter with a base of 5 seconds and cap of 30 seconds.

`round_robin` [true | false]

If the bridge has more than one address given in the address/addresses configuration, the round_robin option defines the behaviour of the bridge on a failure of the bridge connection. If round_robin is *false* , the default value, then the first address is treated as the main bridge connection. If the connection fails, the other secondary addresses will be attempted in turn. Whilst connected to a secondary bridge, the bridge will periodically attempt to reconnect to the main bridge until successful.

If round_robin is *true* , then all addresses are treated as equals. If a connection fails, the next address will be tried and if successful will remain connected until it fails.

`start_type` [automatic | lazy | once]

Set the start type of the bridge. This controls how the bridge starts and can be one of three types: *automatic* , *lazy* and *once* . Note that RSMB provides a fourth start type "manual" which isn't currently supported by mosquitto.

automatic is the default start type and means that the bridge connection will be started automatically when the broker starts and also restarted after a short delay (30 seconds) if the

connection fails.

Bridges using the *lazy* start type will be started automatically when the number of queued messages exceeds the number set with the `threshold` option. It will be stopped automatically after the time set by the `idle_timeout` parameter. Use this start type if you wish the connection to only be active when it is needed.

A bridge using the *once* start type will be started automatically when the broker starts but will not be restarted if the connection fails.

`threshold` *count*

Set the number of messages that need to be queued for a bridge with lazy start type to be restarted. Defaults to 10 messages.

`topic` *pattern* [[[out | in | both] qos-level] local-prefix remote-prefix]

Define a topic pattern to be shared between the two brokers. Any topics matching the pattern (which may include wildcards) are shared. The second parameter defines the direction that the messages will be shared in, so it is possible to import messages from a remote broker using *in*, export messages to a remote broker using *out* or share messages in both directions. If this parameter is not defined, the default of *out* is used. The QoS level defines the publish/subscribe QoS level used for this topic and defaults to 0.

The *local-prefix* and *remote-prefix* options allow topics to be remapped when publishing to and receiving from remote brokers. This allows a topic tree from the local broker to be inserted into the topic tree of the remote broker at an appropriate place.

For incoming topics, the bridge will prepend the pattern with the remote prefix and subscribe to the resulting topic on the remote broker. When a matching incoming message is received, the remote prefix will be removed from the topic and then the local prefix added.

For outgoing topics, the bridge will prepend the pattern with the local prefix and subscribe to the resulting topic on the local broker. When an outgoing message is processed, the local prefix will be removed from the topic then the remote prefix added.

When using topic mapping, an empty prefix can be defined using the place marker `""`. Using the empty marker for the topic itself is also valid. The table below defines what combination of empty or value is valid. The `Full Local Topic` and `Full Remote Topic` show the resulting topics that would be used on the local and remote ends of the bridge. For example, for the first table row if you publish to `L/topic` on the local broker, then the remote broker will receive a message on the topic `R/topic`.

Pattern	***Local Prefix***	***Remote Prefix***	***Validity***	***Full Local Topic***	***Full Remote Topic***
pattern	L/	R/	valid	L/pattern	R/pattern
pattern	L/	""	valid	L/pattern	pattern
pattern	""	R/	valid	pattern	R/pattern
pattern	""	""	valid (no remapping)	pattern	pattern
""	local	remote	valid (remap single local topic to remote)	local	remote
""	local	""	invalid		
""	""	remote	invalid		
""	""	""	invalid		

To remap an entire topic tree, use e.g.:

```
topic # both 2 local/topic/ remote/topic/
```

This option can be specified multiple times per bridge.

Care must be taken to ensure that loops are not created with this option. If you are experiencing high CPU load from a broker, it is possible that you have a loop where each broker is forever forwarding each other the same messages.

See also the `cleansession` option if you have messages arriving on unexpected topics when using incoming topics.

Example Bridge Topic Remapping.
The configuration below connects a bridge to the broker at `test.mosquitto.org` . It subscribes to the remote topic `$SYS/broker/clients/total` and republishes the messages received to the local topic `test/mosquitto/org/clients/total`

```
connection test-mosquitto-org
address test.mosquitto.org
cleansession true
topic clients/total in 0 test/mosquitto/org/ $SYS/broker/
```

`try_private` [true | false]

If try_private is set to *`true`* , the bridge will attempt to indicate to the remote broker that it is a bridge not an ordinary client. If successful, this means that loop detection will be more effective and that retained messages will be propagated correctly. Not all brokers support this feature so it may be necessary to set `try_private` to *`false`* if your bridge does not connect properly.

Defaults to *`true`* .

SSL/TLS Support

The following options are available for all bridges to configure SSL/TLS support.

`bridge_alpn` *`alpn`*

Configure the application layer protocol negotiation option for the TLS session. Useful for brokers that support both websockets and MQTT on the same port.

`bridge_cafile` *`file path`*

One of `bridge_cafile` or `bridge_capath` must be provided to allow SSL/TLS support.

bridge_cafile is used to define the path to a file containing the PEM encoded CA certificates that have signed the certificate for the remote broker.

`bridge_capath` *`file path`*

One of `bridge_capath` or `bridge_cafile` must be provided to allow SSL/TLS support.

bridge_capath is used to define the path to a directory containing the PEM encoded CA certificates that have signed the certificate for the remote broker. For bridge_capath to work correctly, the certificate files must have ".crt" as the file ending and you must run "openssl rehash <path to bridge_capath>" each time you add/remove a certificate.

`bridge_certfile` *`file path`*

Path to the PEM encoded client certificate for this bridge, if required by the remote broker.

`bridge_identity` *`identity`*

Pre-shared-key encryption provides an alternative to certificate based encryption. A bridge can be configured to use PSK with the `bridge_identity` and `bridge_psk` options. This is the client

identity used with PSK encryption. Only one of certificate and PSK based encryption can be used on one bridge at once.

`bridge_insecure` [true | false]

When using certificate based TLS, the bridge will attempt to verify the hostname provided in the remote certificate matches the host/address being connected to. This may cause problems in testing scenarios, so `bridge_insecure` may be set to *`true`* to disable the hostname verification.

Setting this option to *`true`* means that a malicious third party could potentially impersonate your server, so it should always be set to *`false`* in production environments.

`bridge_keyfile` *`file path`*

Path to the PEM encoded private key for this bridge, if required by the remote broker.

`bridge_psk` *`key`*

Pre-shared-key encryption provides an alternative to certificate based encryption. A bridge can be configured to use PSK with the `bridge_identity` and `bridge_psk` options. This is the pre-shared-key in hexadecimal format with no "0x". Only one of certificate and PSK based encryption can be used on one bridge at once.

`bridge_require_ocsp` [true | false]

When set to true, the bridge requires OCSP on the TLS connection it opens as client.

`bridge_tls_version` *`version`*

Configure the version of the TLS protocol to be used for this bridge. Possible values are *`tlsv1.3`*, *`tlsv1.2`* and *`tlsv1.1`*. Defaults to *`tlsv1.2`*. The remote broker must support the same version of TLS for the connection to succeed.

Files

mosquitto.conf

Bugs

mosquitto bug information can be found at https://github.com/eclipse/mosquitto/issues

See Also

mosquitto(8) , mosquitto_passwd(1) , mosquitto-tls(7) , mqtt(7) , limits.conf(5)