Cookie settings

mosquitto.conf man page

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.