EMQ 2.0 Documentation

Release 2.1.2

EMQ Enterprise, Inc. <contact@emqtt.io>

May 05, 2017

Contents

EMQ(Erlang MQTT Broker) is a distributed, massively scalable, highly extensible MQTT message broker written in Erlang/OTP.

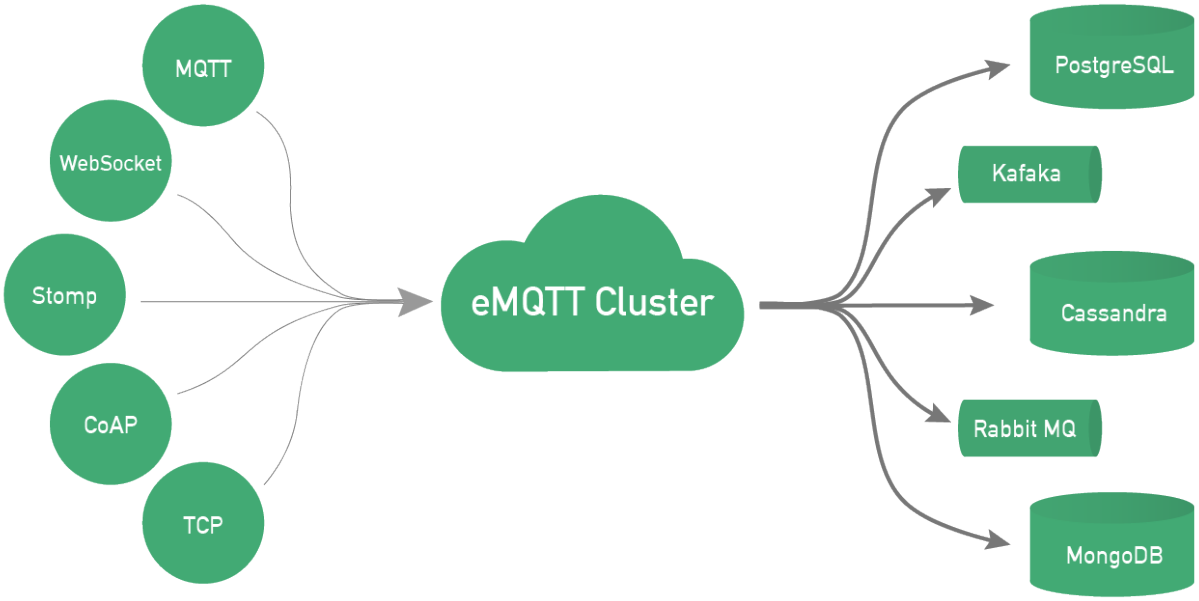
Note: Adopt a shortened project name since 2.0 release: EMQ

EMQ is fully open source and licensed under the Apache Version 2.0. *EMQ* implements both MQTT V3.1 and V3.1.1 protocol specifications, and supports MQTT-SN, CoAP, WebSocket, STOMP and SockJS at the same time.

EMQ provides a scalable, reliable, enterprise-grade MQTT message Hub for IoT, M2M, Smart Hardware and Mobile Messaging Applications. Sensors, Mobiles, Web Browsers and Application Servers could be connected by *EMQ* brokers with asynchronous PUB/SUB MQTT messages.

The 1.0 release of the *EMQ* broker has scaled to 1.3 million concurrent MQTT connections on a 12 Core, 32G CentOS server.

Please visit emqtt.io for more service. Follow us on Twitter: [@emqtt](https://twitter.com/emqtt)



Homepage:	http://emqtt.io
Downloads:	http://emqtt.io/downloads
GitHub:	https://github.com/emqtt
Twitter:	@emqtt
Forum:	https://groups.google.com/d/forum/emqtt
Mailing List:	emqtt@googlegroups.com
Author:	Feng Lee <feng@emqtt.io>

Contents:

CHAPTER 1

Get Started

Overview

EMQ (Erlang MQTT Broker) is an open source MQTT broker written in Erlang/OTP. Erlang/OTP is a concurrent, fault-tolerant, soft-realtime and distributed programming platform. MQTT is an extremely lightweight publish/subscribe messaging protocol powering IoT, M2M and Mobile applications.

The *EMQ* project is aimed to implement a scalable, distributed, extensible open-source MQTT broker for IoT, M2M and Mobile applications that hope to handle millions of concurrent MQTT clients.

Highlights of the *EMQ* broker:

- Full MQTT V3.1/3.1.1 Protocol Specifications Support
- Easy to Install - Quick Install on Linux, FreeBSD, Mac and Windows
- Massively scalable - Scaling to 1 million connections on a single server
- Cluster and Bridge Support
- Easy to extend - Hooks and plugins to customize or extend the broker
- Pluggable Authentication - LDAP, MySQL, PostgreSQL, Redis Authentication Plugins

Features

- Full MQTT V3.1/V3.1.1 protocol specification support
- QoS0, QoS1, QoS2 Publish and Subscribe
- Session Management and Offline Messages
- Retained Message
- Last Will Message
- TCP/SSL Connection

- MQTT Over WebSocket(SSL)
- HTTP Publish API
- STOMP protocol
- MQTT-SN Protocol
- CoAP Protocol
- STOMP over SockJS
- $SYS/# Topics
- ClientID Authentication
- IpAddress Authentication
- Username and Password Authentication
- Access control based on IpAddress, ClientID, Username
- Authentication with LDAP, Redis, MySQL, PostgreSQL and HTTP API
- Cluster brokers on several servers
- Bridge brokers locally or remotely
- mosquitto, RSMB bridge
- Extensible architecture with Hooks, Modules and Plugins
- Passed eclipse paho interoperability tests
- Local subscription
- Shared subscription

Quick Start

Download and Install

The *EMQ* broker is cross-platform, which could be deployed on Linux, FreeBSD, Mac, Windows and even Raspberry Pi.

Download binary package from: http://emqtt.io/downloads.

Installing on Mac, for example:

```
unzip emqttd-macosx-v2.0.zip && cd emqttd

# Start emqttd
./bin/emqttd start

# Check Status
./bin/emqttd_ctl status

# Stop emqttd
./bin/emqttd stop
```

Installing from Source

Note: The *EMQ* broker requires Erlang R18+ to build since 1.1 release.

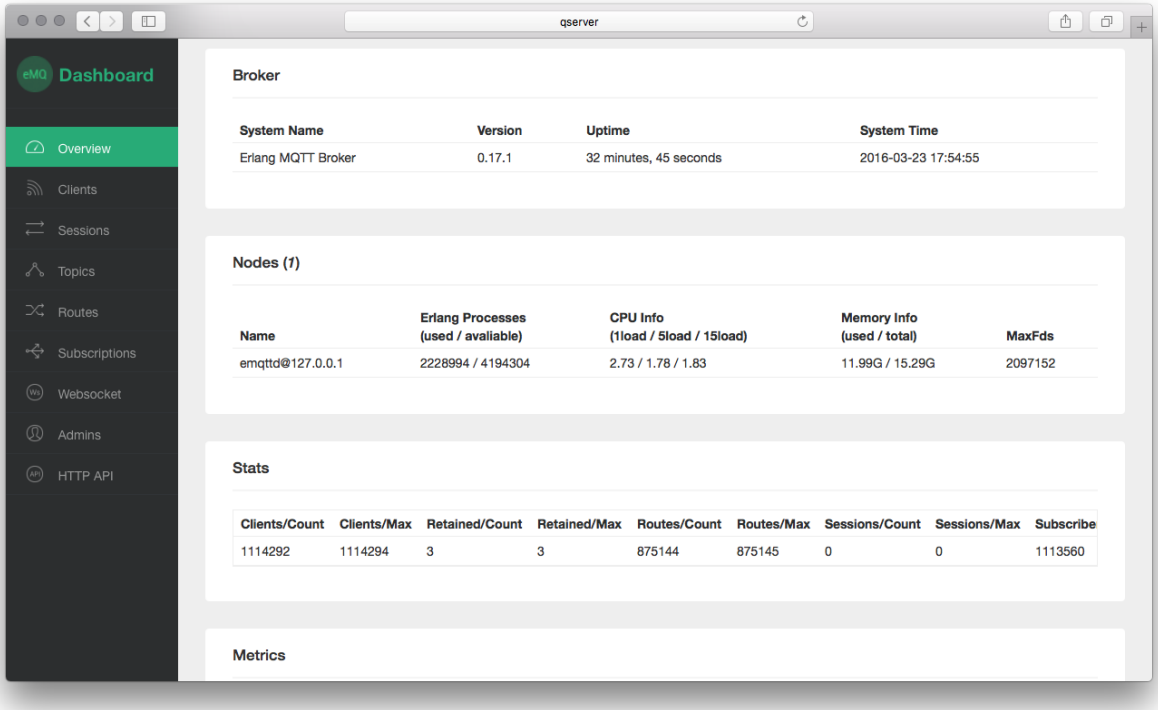
```
git clone https://github.com/emqtt/emqttd.git

cd emqttd && make rel

cd rel/emqttd && ./bin/emqttd console
```

Web Dashboard

A Web Dashboard will be loaded when the *EMQ* broker is started successfully.

The Dashboard helps check running status of the broker, monitor statistics and metrics of MQTT packets, query clients, sessions, topics and subscriptions.

Default Address	http://localhost:18083
Default User	admin
Default Password	public



Plugins

The *EMQ* broker could be extended by Plugins. A plugin is an Erlang application that adds extra feature to the *EMQ* broker:

emq_retainer	Store Retained Messages
emq_dashboard	Web Dashboard
emq_modules	Presence, Subscription, Rewrite Modules
emq_auth_clientid	Authentication with ClientId
emq_auth_username	Authentication with Username and Password
emq_plugin_template	Plugin template and demo
emq_auth_ldap	LDAP Auth Plugin
emq_auth_http	Authentication/ACL with HTTP API
emq_auth_mysql	Authentication with MySQL
emq_auth_pgsql	Authentication with PostgreSQL
emq_auth_redis	Authentication with Redis
'emq_mod_mongo'_	Authentication with MongoDB
emq_sn	MQTT-SN Protocol Plugin
emq_coap	CoAP Protocol Plugin
emq_stomp	STOMP Protocol Plugin
emq_sockjs	SockJS(Stomp) Plugin
emq_recon	Recon Plugin
emq_reloader	Reloader Plugin

A plugin could be enabled by 'bin/emqttd_ctl plugins load' command.

For example, enable 'emq_auth_pgsql' plugin:

```
./bin/emqttd_ctl plugins load emq_auth_pgsql
```

One Million Connections

Latest release of the *EMQ* broker is scaling to 1.3 million MQTT connections on a 12 Core, 32G CentOS server.

Note: The emqttd broker only allows 512 concurrent connections by default, for 'ulimit -n' limit is 1024 on most platform.

We need tune the OS Kernel, TCP Stack, Erlang VM and emqttd broker for one million connections benchmark.

Linux Kernel Parameters

```
# 2M:
sysctl -w fs.file-max=2097152
sysctl -w fs.nr_open=2097152
echo 2097152 > /proc/sys/fs/nr_open

# 1M:
ulimit -n 1048576
```

TCP Stack Parameters

```
# backlog
sysctl -w net.core.somaxconn=65536
```

Erlang VM

emqttd/etc/emq.conf:

```
## Erlang Process Limit
node.process_limit = 2097152

## Sets the maximum number of simultaneously existing ports for this system
node.max_ports = 1048576
```

Max Allowed Connections

emqttd/etc/emq.conf 'listeners':

```
## Size of acceptor pool
mqtt.listener.tcp.acceptors = 64

## Maximum number of concurrent clients
mqtt.listener.tcp.max_clients = 1000000
```

Test Client

```
sysctl -w net.ipv4.ip_local_port_range="500 65535"
echo 1000000 > /proc/sys/fs/nr_open
ulimit -n 100000
```

MQTT Client Libraries

GitHub: https://github.com/emqtt

emqttc	Erlang MQTT Client
emqtt_benchmark	MQTT benchmark Tool
CocoaMQTT	Swift MQTT Client
QMQTT	QT MQTT Client

Eclipse Paho: https://www.eclipse.org/paho/

MQTT.org: https://github.com/mqtt/mqtt.github.io/wiki/libraries

CHAPTER 2

Installation

The *EMQ* broker is cross-platform, which could be deployed on Linux, FreeBSD, Mac, Windows and even Raspberry Pi.

Note: Linux, FreeBSD Recommended.

Download Packages

Download binary packages from: http://emqtt.io/downloads

Debian	http://emqtt.com/downloads/latest/debian
Ubuntu12.04	http://emqtt.com/downloads/latest/ubuntu12_04
Ubuntu14.04	http://emqtt.com/downloads/latest/ubuntu14_04
Ubuntu16.04	http://emqtt.com/downloads/latest/ubuntu16_04
CentOS7	http://emqtt.com/downloads/latest/centos7
Debian7	http://emqtt.com/downloads/latest/debian7
Debian8	http://emqtt.com/downloads/latest/debian7
FreeBSD	http://emqtt.com/downloads/latest/freebsd
Windows7	http://emqtt.com/downloads/latest/windows7
Windows10	http://emqtt.com/downloads/latest/windows10
Mac OS X	http://emqtt.com/downloads/latest/macosx
Docker	http://emqtt.com/downloads/latest/docker

The package name consists of platform, version and release time.

For example: emqttd-centos64-v2.0.zip

Installing on Linux

Download CentOS Package from: http://emqtt.io/downloads/latest/centos7, and then unzip:

```
unzip emqttd-centos64-v2.0.zip
```

Start the broker in console mode:

```
cd emqttd && ./bin/emqttd console
```

If the broker is started successfully, console will print:

```
starting emqttd on node 'emqttd@127.0.0.1'
emqttd ctl is starting...[done]
emqttd trace is starting...[done]
emqttd pubsub is starting...[done]
emqttd stats is starting...[done]
emqttd metrics is starting...[done]
emqttd retainer is starting...[done]
emqttd pooler is starting...[done]
emqttd client manager is starting...[done]
emqttd session manager is starting...[done]
emqttd session supervisor is starting...[done]
emqttd broker is starting...[done]
emqttd alarm is starting...[done]
emqttd mod supervisor is starting...[done]
emqttd bridge supervisor is starting...[done]
emqttd access control is starting...[done]
emqttd system monitor is starting...[done]
http listen on 0.0.0.0:18083 with 4 acceptors.
mqtt listen on 0.0.0.0:1883 with 16 acceptors.
mqtts listen on 0.0.0.0:8883 with 4 acceptors.
http listen on 0.0.0.0:8083 with 4 acceptors.
Erlang MQTT Broker 2.0 is running now
Eshell V6.4  (abort with ^G)
(emqttd@127.0.0.1)1>
```

CTRL+C to close the console and stop the broker.

Start the broker in daemon mode:

```
./bin/emqttd start
```

Check the running status of the broker:

```
$ ./bin/emqttd_ctl status
Node 'emqttd@127.0.0.1' is started
emqttd 2.0 is running
```

Or check the status by URL:

```
http://localhost:8083/status
```

Stop the broker:

```
./bin/emqttd stop
```

Installing on FreeBSD

Download FreeBSD Package from: http://emqtt.io/downloads/latest/freebsd

The installing process is same to Linux.

Installing on Mac OS X

We could install the broker on Mac OS X to develop and debug MQTT applications.

Download Mac Package from: http://emqtt.io/downloads/latest/macosx

Configure log level in *etc/emq.conf*, all MQTT messages recevied/sent will be printed on console:

The install and boot process on Mac are same to Linux.

Installing on Windows

Download Package from: http://emqtt.io/downloads/latest/windows.

Unzip the package to install folder. Open the command line window and 'cd' to the folder.

Start the broker in console mode:

```
bin\emqttd console
```

If the broker started successfully, a Erlang console window will popup.

Close the console window and stop the emqttd broker. Prepare to register emqttd as window service.

Warning: Cannot register EMQ-2.0 as a windows service.

Install emqttd serivce:

```
bin\emqttd install
```

Start emqttd serivce:

```
bin\emqttd start
```

Stop emqttd serivce:

```
bin\emqttd stop
```

Uninstall emqttd service:

```
bin\emqttd uninstall
```

Install via Docker Image

Download *EMQ* 2.0 Docker Image:

http://emqtt.com/downloads/latest/docker

unzip emqttd-docker image:

```
unzip emqttd-docker-v2.0.zip
```

Load Docker Image:

```
docker load < emqttd-docker-v2.0
```

Run the Container:

```
docker run -tid --name emq20 -p 1883:1883 -p 8083:8083 -p 8883:8883 -p 8084:8084 -p␣
↪18083:18083 emqttd-docker-v2.0
```

Stop the broker:

```
docker stop emq20
```

Start the broker:

```
docker start emq20
```

Enter the running container:

```
docker exec -it emq20 /bin/sh
```

Installing From Source

The *EMQ* broker requires Erlang/OTP R18+ and git client to build:

Install Erlang: http://www.erlang.org/

Install Git Client: http://www.git-scm.com/

Could use apt-get on Ubuntu, yum on CentOS/RedHat and brew on Mac to install Erlang and Git.

When all dependencies are ready, clone the emqttd project from github.com and build:

```
git clone https://github.com/emqtt/emq-relx.git

cd emq-relx && make

cd _rel/emqttd && ./bin/emqttd console
```

The binary package output in folder:

```
_rel/emqttd
```

TCP Ports Used

1883	MQTT Port
8883	MQTT/SSL Port
8083	MQTT(WebSocket), HTTP API Port
8084	MQTT(WebSocket/SSL), HTTP API Port
18083	Web Dashboard Port

The TCP ports used can be configured in etc/emqttd.config:

```
## TCP Listener: 1883, 127.0.0.1:1883, ::1:1883
mqtt.listener.tcp = 1883

## SSL Listener: 8883, 127.0.0.1:8883, ::1:8883
mqtt.listener.ssl = 8883

## HTTP and WebSocket Listener
mqtt.listener.http = 8083
```

The 18083 port is used by Web Dashboard of the broker. Default login: admin, Password: public

Quick Setup

Two main configuration files of the *EMQ* broker:

etc/emq.conf	EMQ Broker Config
etc/plugins/*.conf	EMQ Plugins' Config

Two important parameters in etc/emq.conf:

node.process_limit	Max number of Erlang proccesses. A MQTT client consumes two proccesses. The value should be larger than max_clients * 2
node.max_ports	Max number of Erlang Ports. A MQTT client consumes one port. The value should be larger than max_clients.

Note: node.process_limit > maximum number of allowed concurrent clients * 2 node.max_ports > maximum number of allowed concurrent clients

The maximum number of allowed MQTT clients:

```
mqtt.listener.tcp = 1883

mqtt.listener.tcp.acceptors = 8

mqtt.listener.tcp.max_clients = 1024
```

/etc/init.d/emqttd

```
#!/bin/sh
#
# emqttd       Startup script for emqttd.
#
# chkconfig: 2345 90 10
# description: emqttd is mqtt broker.

# source function library
. /etc/rc.d/init.d/functions

# export HOME=/root

start() {
    echo "starting emqttd..."
    cd /opt/emqttd && ./bin/emqttd start
```

```
}

stop() {
    echo "stopping emqttd..."
    cd /opt/emqttd && ./bin/emqttd stop
}

restart() {
    stop
    start
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        restart
        ;;
    *)
        echo $"Usage: $0 {start|stop}"
        RETVAL=2
esac
```

chkconfig:

```
chmod +x /etc/init.d/emqttd
chkconfig --add emqttd
chkconfig --list
```

boot test:

```
service emqttd start
```

Note: ## erlexec: HOME must be set uncomment '# export HOME=/root' if "HOME must be set" error.

CHAPTER 3

Configuration

The main configuration files of the EMQ broker are under 'etc/' folder:

File	Description
etc/emq.conf	EMQ 2.0 Configuration File
etc/acl.conf	The default ACL File
etc/plugins/*.conf	Config Files of Plugins

EMQ 2.0 Config Syntax

The *EMQ* 2.0-rc.2 release integrated with *cuttlefish* library, and adopted a more user-friendly *k* = *v* syntax for configuration file:

```
## Node name
node.name = emqttd@127.0.0.1
...
## Max ClientId Length Allowed.
mqtt.max_clientid_len = 1024
...
```

The configuration files will be preprocessed and translated to Erlang *app.config* before the EMQ broker started:

```
----------------------                                                2.0/schema/*.schema
↪    --------------------
| etc/emq.conf       |                    ------------------              \|/
↪    | data/app.config |
|       +            | --> mergeconf --> | data/app.conf | -->  cuttlefish generate  -
↪-> |                  |
| etc/plugins/*.conf |                    ------------------
↪    | data/vm.args    |
----------------------
↪    --------------------
```

OS Environment Variables

EMQ_NODE_NAME	Erlang node name
EMQ_NODE_COOKIE	Cookie for distributed erlang node
EMQ_MAX_PORTS	Maximum number of opened sockets
EMQ_TCP_PORT	MQTT TCP Listener Port, Default: 1883
EMQ_SSL_PORT	MQTT SSL Listener Port, Default: 8883
EMQ_HTTP_PORT	HTTP/WebSocket Port, Default: 8083
EMQ_HTTPS_PORT	HTTPS/WebSocket Port, Default: 8084

EMQ Node and Cookie

The node name and cookie of *EMQ* should be configured when clustering:

```
## Node name
node.name = emqttd@127.0.0.1

## Cookie for distributed node
node.cookie = emq_dist_cookie
```

Erlang VM Arguments

Configure and Optimize Erlang VM:

```
## SMP support: enable, auto, disable
node.smp = auto

## Enable kernel poll
node.kernel_poll = on

## async thread pool
node.async_threads = 32

## Erlang Process Limit
node.process_limit = 256000

## Sets the maximum number of simultaneously existing ports for this system
node.max_ports = 65536

## Set the distribution buffer busy limit (dist_buf_busy_limit)
node.dist_buffer_size = 32MB

## Max ETS Tables.
## Note that mnesia and SSL will create temporary ets tables.
node.max_ets_tables = 256000

## Tweak GC to run more often
node.fullsweep_after = 1000

## Crash dump
node.crash_dump = log/crash.dump
```

```
## Distributed node ticktime
node.dist_net_ticktime = 60

## Distributed node port range
## node.dist_listen_min = 6000
## node.dist_listen_max = 6999
```

The two most important parameters for Erlang VM:

node.process_limit	Max number of Erlang proccesses. A MQTT client consumes two proccesses. The value should be larger than max_clients * 2
node.max_ports	Max number of Erlang Ports. A MQTT client consumes one port. The value should be larger than max_clients.

Log Level and File

Console Log

```
## Console log. Enum: off, file, console, both
log.console = console

## Console log level. Enum: debug, info, notice, warning, error, critical, alert,␣
↪emergency
log.console.level = error

## Console log file
## log.console.file = log/console.log
```

Error Log

```
## Error log file
log.error.file = log/error.log
```

Crash Log

```
## Enable the crash log. Enum: on, off
log.crash = on

log.crash.file = log/crash.log
```

Syslog

```
## Syslog. Enum: on, off
log.syslog = on

##  syslog level. Enum: debug, info, notice, warning, error, critical, alert,␣
↪emergency
log.syslog.level = error
```

MQTT Protocol Parameters

Maximum ClientId Length

```
## Max ClientId Length Allowed.
mqtt.max_clientid_len = 1024
```

Maximum Packet Size

```
## Max Packet Size Allowed, 64K by default.
mqtt.max_packet_size = 64KB
```

MQTT Client Idle Timeout

```
## Client Idle Timeout (Second)
mqtt.client.idle_timeout = 30
```

Enable Per Client Statistics

```
## Enable client Stats: on | off
mqtt.client.enable_stats = off
```

Force GC Count

```
## Force GC: integer. Value 0 disabled the Force GC.
mqtt.conn.force_gc_count = 100
```

Allow Anonymous and ACL File

Allow Anonymous

```
## Allow Anonymous authentication
mqtt.allow_anonymous = true
```

Default ACL File

Enable the default ACL module:

```
## Default ACL File
mqtt.acl_file = etc/acl.conf
```

Define ACL rules in etc/acl.conf. The rules by default:

```
%% Allow 'dashboard' to subscribe '$SYS/#'
{allow, {user, "dashboard"}, subscribe, ["$SYS/#"]}.

%% Allow clients from localhost to subscribe any topics
{allow, {ipaddr, "127.0.0.1"}, pubsub, ["$SYS/#", "#"]}.

%% Deny clients to subscribe '$SYS#' and '#'
{deny, all, subscribe, ["$SYS/#", {eq, "#"}]}.

%% Allow all by default
{allow, all}.
```

An ACL rule is an Erlang tuple. The Access control module of *EMQ* broker matches the rule one by one from top to bottom:

```
          ---------               ---------               ---------
Client -> | Rule1 | --nomatch--> | Rule2 | --nomatch--> | Rule3 | --> Default
          ---------               ---------               ---------
              |                       |                       |
            match                   match                   match
             \|/                     \|/                     \|/
       allow | deny            allow | deny            allow | deny
```

MQTT Session Parameters

```
## Upgrade QoS?
mqtt.session.upgrade_qos = off

## Max number of QoS 1 and 2 messages that can be "inflight" at one time.
## 0 means no limit
mqtt.session.max_inflight = 32

## Retry Interval for redelivering QoS1/2 messages.
mqtt.session.retry_interval = 20s

## Max Packets that Awaiting PUBREL, 0 means no limit
mqtt.session.max_awaiting_rel = 100

## Awaiting PUBREL Timeout
mqtt.session.await_rel_timeout = 20s

## Enable Statistics: on | off
mqtt.session.enable_stats = off

## Expired after 1 day:
## w - week
## d - day
## h - hour
## m - minute
## s - second
mqtt.session.expiry_interval = 2h
```

session.upgrade_qos	Upgrade QoS according to the subscription
session.max_inflight	Max number of QoS1/2 messages that can be delivered at the same time
session.retry_interval	Retry interval for unacked QoS1/2 messages.
session.await_rel_timeout	Awaiting PUBREL Timeout
session.max_awaiting_rel	Max number of Packets that Awaiting PUBREL
session.enable_stats	Interval of Statistics Collection
session.expiry_interval	Session expiry interval

MQTT Message Queue

The message queue of session stores:

1. Offline messages for persistent session.
2. Pending messages for inflight window is full

Queue parameters:

```
## Type: simple | priority
mqtt.queue.type = simple

## Topic Priority: 0~255, Default is 0
## mqtt.queue.priority = topic/1=10,topic/2=8

## Max queue length. Enqueued messages when persistent client disconnected,
## or inflight window is full.
mqtt.queue.max_length = infinity

## Low-water mark of queued messages
mqtt.queue.low_watermark = 20%

## High-water mark of queued messages
mqtt.queue.high_watermark = 60%

## Queue Qos0 messages?
mqtt.queue.qos0 = true
```

queue.type	Queue type: simple or priority
queue.priority	Topic priority
queue.max_length	Max Queue size, infinity means no limit
queue.low_watermark	Low watermark
queue.high_watermark	High watermark
queue.qos0	If Qos0 message queued?

Sys Interval of Broker

```
## System Interval of publishing broker $SYS Messages
mqtt.broker.sys_interval = 60
```

PubSub Parameters

```
## PubSub Pool Size. Default should be scheduler numbers.
mqtt.pubsub.pool_size = 8

mqtt.pubsub.by_clientid = true

##TODO: Subscribe Asynchronously
mqtt.pubsub.async = true
```

MQTT Bridge Parameters

```
## Bridge Queue Size
mqtt.bridge.max_queue_len = 10000

## Ping Interval of bridge node. Unit: Second
mqtt.bridge.ping_down_interval = 1
```

Plugins' Etc Folder

```
## Dir of plugins' config
mqtt.plugins.etc_dir = etc/plugins/

## File to store loaded plugin names.
mqtt.plugins.loaded_file = data/loaded_plugins
```

MQTT Listeners

Configure the TCP listeners for MQTT, MQTT(SSL), HTTP and HTTPS Protocols.

The most important parameter for MQTT listener is *max_clients*: max concurrent clients allowed.

The TCP Ports occupied by the *EMQ* broker by default:

1883	MQTT Port
8883	MQTT(SSL) Port
8083	MQTT(WebSocket), HTTP API Port

Listener Parameters:

mqtt.listener.*.acceptors	TCP Acceptor Pool
mqtt.listener.*.max_clients	Maximum number of concurrent TCP connections allowed
mqtt.listener.*.rate_limit	Maximum number of concurrent TCP connections allowed

TCP Listener - 1883

```
## TCP Listener: 1883, 127.0.0.1:1883, ::1:1883
mqtt.listener.tcp = 1883
```

```
## Size of acceptor pool
mqtt.listener.tcp.acceptors = 8

## Maximum number of concurrent clients
mqtt.listener.tcp.max_clients = 1024

## Rate Limit. Format is 'burst,rate', Unit is KB/Sec
## mqtt.listener.tcp.rate_limit = 100,10

## TCP Socket Options
mqtt.listener.tcp.backlog = 1024
## mqtt.listener.tcp.recbuf = 4096
## mqtt.listener.tcp.sndbuf = 4096
## mqtt.listener.tcp.buffer = 4096
## mqtt.listener.tcp.nodelay = true
```

SSL Listener - 8883

```
## SSL Listener: 8883, 127.0.0.1:8883, ::1:8883
mqtt.listener.ssl = 8883

## Size of acceptor pool
mqtt.listener.ssl.acceptors = 4

## Maximum number of concurrent clients
mqtt.listener.ssl.max_clients = 512

## Rate Limit. Format is 'burst,rate', Unit is KB/Sec
## mqtt.listener.ssl.rate_limit = 100,10

## Configuring SSL Options
mqtt.listener.ssl.handshake_timeout = 15
mqtt.listener.ssl.keyfile = etc/certs/key.pem
mqtt.listener.ssl.certfile = etc/certs/cert.pem
## mqtt.listener.ssl.cacertfile = etc/certs/cacert.pem
## mqtt.listener.ssl.verify = verify_peer
## mqtt.listener.ssl.fail_if_no_peer_cert = true
```

HTTP/WS Listener - 8083

```
## HTTP and WebSocket Listener
mqtt.listener.http = 8083
mqtt.listener.http.acceptors = 4
mqtt.listener.http.max_clients = 64
```

HTTPS/WSS Listener - 8084

```
## HTTP(SSL) Listener
mqtt.listener.https = 8084
mqtt.listener.https.acceptors = 4
mqtt.listener.https.max_clients = 64
mqtt.listener.https.handshake_timeout = 15
```

```
mqtt.listener.https.certfile = etc/certs/cert.pem
mqtt.listener.https.keyfile = etc/certs/key.pem
## mqtt.listener.https.cacertfile = etc/certs/cacert.pem
## mqtt.listener.https.verify = verify_peer
## mqtt.listener.https.fail_if_no_peer_cert = true
```

System Monitor

```
## Long GC, don't monitor in production mode for:
sysmon.long_gc = false

## Long Schedule(ms)
sysmon.long_schedule = 240

## 8M words. 32MB on 32-bit VM, 64MB on 64-bit VM.
sysmon.large_heap = 8MB

## Busy Port
sysmon.busy_port = false

## Busy Dist Port
sysmon.busy_dist_port = true
```

Plugin Configuration Files

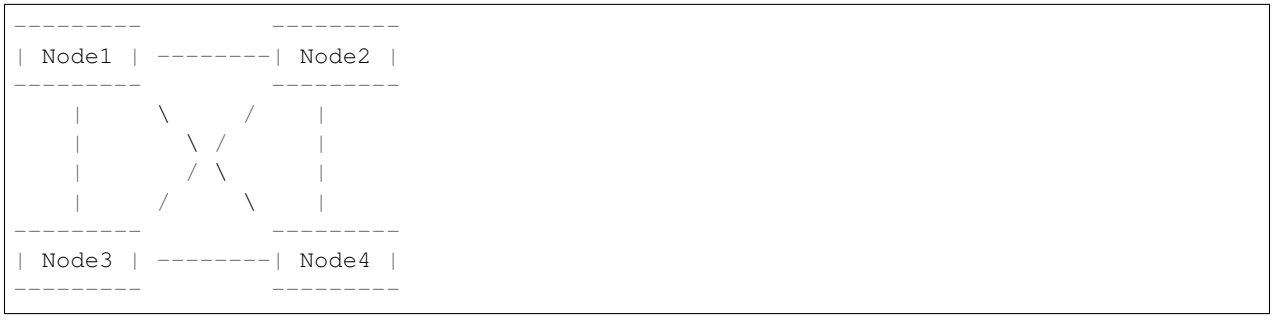
File	Description
etc/plugins/emq_auth_username.conf	Username/Password Auth Plugin
etc/plugins/emq_auth_clientid.conf	ClientId Auth Plugin
etc/plugins/emq_auth_http.conf	HTTP Auth/ACL Plugin Config
etc/plugins/emq_auth_mongo.conf	MongoDB Auth/ACL Plugin Config
etc/plugins/emq_auth_mysql.conf	MySQL Auth/ACL Plugin Config
etc/plugins/emq_auth_pgsql.conf	Postgre Auth/ACL Plugin Config
etc/plugins/emq_auth_redis.conf	Redis Auth/ACL Plugin Config
etc/plugins/emq_coap.conf	CoAP Protocol Plugin Config
etc/plugins/emq_mod_presence.conf	Presence Module Config
etc/plugins/emq_mod_retainer.conf	Retainer Module Config
etc/plugins/emq_mod_rewrite.config	Rewrite Module Config
etc/plugins/emq_mod_subscription.conf	Subscription Module Config
etc/plugins/emq_dashboard.conf	Dashboard Plugin Config
etc/plugins/emq_plugin_template.conf	Template Plugin Config
etc/plugins/emq_recon.conf	Recon Plugin Config
etc/plugins/emq_reloader.conf	Reloader Plugin Config
etc/plugins/emq_sn.conf	MQTT-SN Protocal Plugin Config
etc/plugins/emq_stomp.conf	Stomp Protocl Plugin Config

CHAPTER 4

Clustering

Distributed Erlang/OTP

Erlang/OTP is a concurrent, fault-tolerant, distributed programming platform. A distributed Erlang/OTP system consists of a number of Erlang runtime systems called 'node'. Nodes connect to each other with TCP/IP sockets and communicate by Message Passing.



Node

An erlang runtime system called 'node' is identified by a unique name like email addreass. Erlang nodes communicate with each other by the name.

Suppose we start four Erlang nodes on localhost:

```
erl -name node1@127.0.0.1
erl -name node2@127.0.0.1
erl -name node3@127.0.0.1
erl -name node4@127.0.0.1
```

connect all the nodes:

```
(node1@127.0.0.1)1> net_kernel:connect_node('node2@127.0.0.1').
true
(node1@127.0.0.1)2> net_kernel:connect_node('node3@127.0.0.1').
true
(node1@127.0.0.1)3> net_kernel:connect_node('node4@127.0.0.1').
true
(node1@127.0.0.1)4> nodes().
['node2@127.0.0.1','node3@127.0.0.1','node4@127.0.0.1']
```

epmd

epmd(Erlang Port Mapper Daemon) is a daemon service that is responsible for mapping node names to machine addresses(TCP sockets). The daemon is started automatically on every host where an Erlang node started.

```
(node1@127.0.0.1)6> net_adm:names().
{ok,[{"node1",62740},
     {"node2",62746},
     {"node3",62877},
     {"node4",62895}]}
```

Cookie

Erlang nodes authenticate each other by a magic cookie when communicating. The cookie could be configured by:

```
1. $HOME/.erlang.cookie

2. erl -setcookie <Cookie>
```

Note: Content of this chapter is from: http://erlang.org/doc/reference_manual/distributed.html

Cluster Design

The cluster architecture of emqttd broker is based on distrubuted Erlang/OTP and Mnesia database.

The cluster design could be summarized by the following two rules:

1. When a MQTT client SUBSCRIBE a Topic on a node, the node will tell all the other nodes in the cluster: I subscribed a Topic.
2. When a MQTT Client PUBLISH a message to a node, the node will lookup the Topic table and forward the message to nodes that subscribed the Topic.

Finally there will be a global route table(Topic -> Node) that replicated to all nodes in the cluster:

```
topic1 -> node1, node2
topic2 -> node3
topic3 -> node2, node4
```

Topic Trie and Route Table

Every node in the cluster will store a topic trie and route table in mnesia database.

Suppose that we create subscriptions:

Client	Node	Topics
client1	node1	t/+/x, t/+/y
client2	node2	t/#
client3	node3	t/+/x, t/a

Finally the topic trie and route table in the cluster:

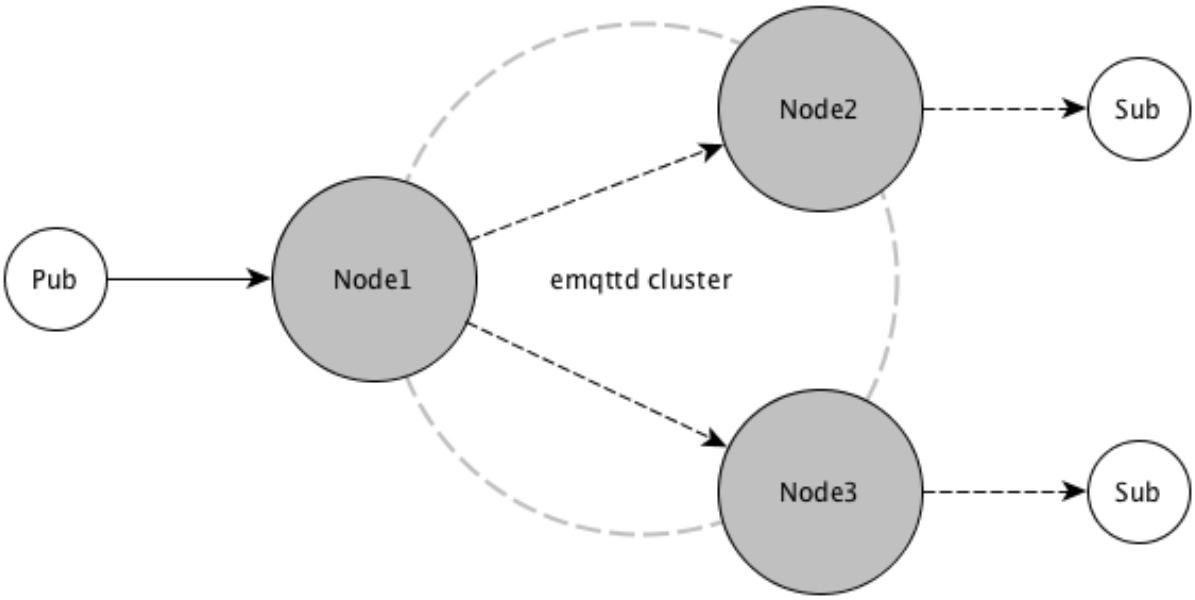
```
---------------------------
|           t             |
|          / \            |
|         +   #           |
|       /  \              |
|    x       y            |
---------------------------
| t/+/x -> node1, node3   |
| t/+/y -> node1          |
| t/#   -> node2          |
| t/a   -> node3          |
---------------------------
```

Message Route and Deliver

The brokers in the cluster route messages by topic trie and route table, deliver messages to MQTT clients by subscriptions. Subscriptions are mapping from topic to subscribers, are stored only in the local node, will not be replicated to other nodes.

Suppose client1 PUBLISH a message to the topic 't/a', the message Route and Deliver process:

```
title: Message Route and Deliver

client1->node1: Publish[t/a]
node1-->node2: Route[t/#]
node1-->node3: Route[t/a]
node2-->client2: Deliver[t/#]
node3-->client3: Deliver[t/a]
```



Cluster Setup

Suppose we deploy two nodes cluster on s1.emqtt.io, s2.emqtt.io:

Node	Host(FQDN)	IP and Port
emqttd@s1.emqtt.io or emqttd@192.168.0.10	s1.emqtt.io	192.168.0.10:1883
emqttd@s2.emqtt.io or emqttd@192.168.0.20	s2.emqtt.io	192.168.0.20:1883

Warning: The node name is Name@Host, where Host is IP address or the fully qualified host name.

emqttd@s1.emqtt.io config

etc/emq.conf:

```
node.name = emqttd@s1.emqtt.io

or

node.name = emqttd@192.168.0.10
```

Warning: The name cannot be changed after node joined the cluster.

emqttd@s2.emqtt.io config

etc/emq.conf:

```
node.name = emqttd@s2.emqtt.io

or

node.name = emqttd@192.168.0.20
```

Join the cluster

Start the two broker nodes, and 'cluster join ' on emqttd@s2.emqtt.io:

```
$ ./bin/emqttd_ctl cluster join emqttd@s1.emqtt.io

Join the cluster successfully.
Cluster status: [{running_nodes,['emqttd@s1.emqtt.io','emqttd@s2.emqtt.io']}]
```

Or 'cluster join' on emqttd@s1.emqtt.io:

```
$ ./bin/emqttd_ctl cluster join emqttd@s2.emqtt.io

Join the cluster successfully.
Cluster status: [{running_nodes,['emqttd@s1.emqtt.io','emqttd@s2.emqtt.io']}]
```

Query the cluster status:

```
$ ./bin/emqttd_ctl cluster status

Cluster status: [{running_nodes,['emqttd@s1.emqtt.io','emqttd@s2.emqtt.io']}]
```

Leave the cluster

Two ways to leave the cluster:

1. leave: this node leaves the cluster
2. remove: remove other nodes from the cluster

emqttd@s2.emqtt.io node tries to leave the cluster:

```
$ ./bin/emqttd_ctl cluster leave
```

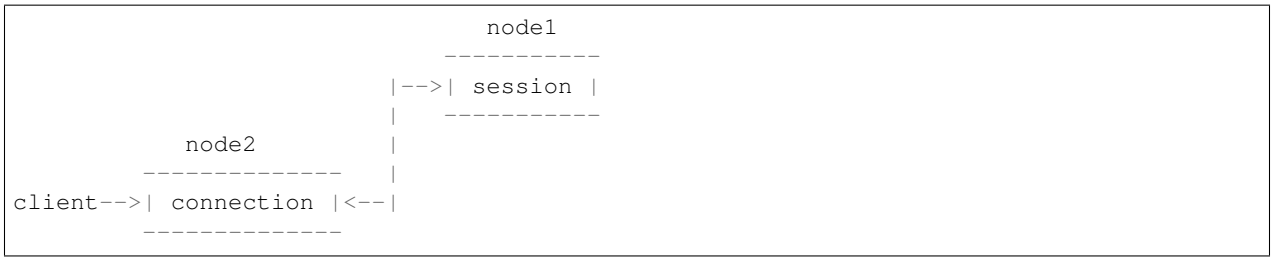
Or remove emqttd@s2.emqtt.io node from the cluster on emqttd@s1.emqtt.io:

```
$ ./bin/emqttd_ctl cluster remove emqttd@s2.emqtt.io
```

Session across Nodes

The persistent MQTT sessions (clean session = false) are across nodes in the cluster.

If a persistent MQTT client connected to node1 first, then disconnected and connects to node2, the MQTT connection and session will be located on different nodes:



The Firewall

If there is a firewall between clustered nodes, the cluster requires to open 4369 port used by epmd daemon, and a port segment for nodes' communication.

Configure the port segment in releases/2.0/sys.config, for example:

```
[{kernel, [
    ...
    {inet_dist_listen_min, 20000},
    {inet_dist_listen_max, 21000}
 ]},
 ...
```

Network Partitions

The emqttd 1.0 cluster requires reliable network to avoid network partitions. The cluster will not recover from a network partition automatically.

If a network partition occures, there will be critical logs in log/emqttd_error.log:

```
Mnesia inconsistent_database event: running_partitioned_network, emqttd@host
```

To recover from a network partition, you have to stop the nodes in a partition, clean the 'data/mneisa' of these nodes and reboot to join the cluster again.

Consistent Hash and DHT

Consistent Hash and DHT are popular in the design of NoSQL databases. Cluster of emqttd broker could support 10 million size of global routing table now. We could use the Consistent Hash or DHT to partition the routing table, and evolve the cluster to larger size.

CHAPTER 5

Bridge

EMQ Node Bridge

Two or more *EMQ* brokers could be bridged together. Bridges forward MQTT messages from one broker node to another:

```
                ---------                          ---------                          ---------
Publisher --> | Node1 | --Bridge Forward--> | Node2 | --Bridge Forward--> | Node3 | --
↪> Subscriber
                ---------                          ---------                          ---------
```

Configure Bridge

Suppose that we create two *EMQ* brokers on localhost:

Name	Node	MQTT Port
emqttd1	emqttd1@127.0.0.1	1883
emqttd2	emqttd2@127.0.0.1	2883

Create a bridge that forwards all the 'sensor/#' messages from emqttd1 to emqttd2.

1. Start Brokers

```
cd emqttd1/ && ./bin/emqttd start
cd emqttd2/ && ./bin/emqttd start
```

2. Create bridge: emqttd1–sensor/#–>emqttd2

```
$ cd emqttd1 && ./bin/emqttd_ctl bridges start emqttd2@127.0.0.1 sensor/#

bridge is started.

$ ./bin/emqttd_ctl bridges list

bridge: emqttd1@127.0.0.1--sensor/#-->emqttd2@127.0.0.1
```

3. Test the bridge

```
#emqttd2
mosquitto_sub -t sensor/# -p 2883 -d

#emqttd1
mosquitto_pub -t sensor/1/temperature -m "37.5" -d
```

4. Delete the bridge

```
./bin/emqttd_ctl bridges stop emqttd2@127.0.0.1 sensor/#
```

EMQ Bridge CLI

```
#query bridges
./bin/emqttd_ctl bridges list

#start bridge
./bin/emqttd_ctl bridges start <Node> <Topic>

#start bridge with options
./bin/emqttd_ctl bridges start <Node> <Topic> <Options>

#stop bridge
./bin/emqttd_ctl bridges stop <Node> <Topic>
```

mosquitto Bridge

Bridge mosquitto to emqttd broker:

mosquitto.conf

Suppose that we start an emqttd broker on localost:2883, and mosquitto on localhost:1883.

A bridge configured in mosquitto.conf:

```
connection emqttd
address 127.0.0.1:2883
topic sensor/# out 2

# Set the version of the MQTT protocol to use with for this bridge. Can be one
# of mqttv31 or mqttv311. Defaults to mqttv31.
bridge_protocol_version mqttv311
```

rsmb Bridge

Bridge RSMB to EMQ broker, same settings as mosquitto.

broker.cfg:

```
connection emqttd
addresses 127.0.0.1:2883
topic sensor/#
```

CHAPTER 6

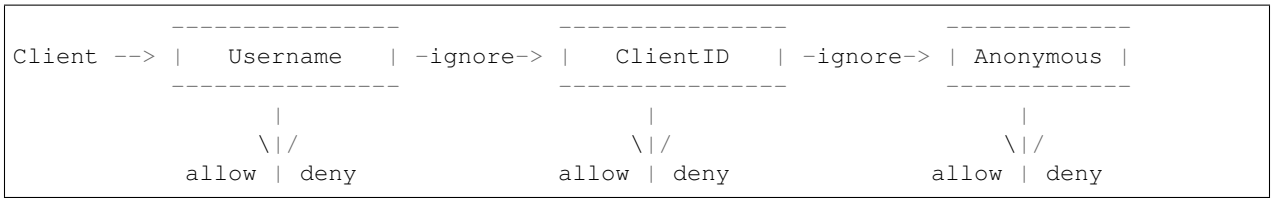
User Guide

Authentication

The *EMQ* broker supports to authenticate MQTT clients with ClientID, Username/Password, IpAddress and even HTTP Cookies.

The authentication is provided by a list of plugins such as MySQL, PostgreSQL and Redis...

If we enable several authentication plugins at the same time, the authentication process:



The authentication plugins implemented by default:

Plugin	Description
emq_auth_clientid	ClientId Auth Plugin
emq_auth_username	Username Auth Plugin
emq_auth_ldap	LDAP Auth Plugin
emq_auth_http	HTTP Auth/ACL Plugin
emq_auth_mysql	MySQL Auth/ACL Plugin
emq_auth_pgsql	Postgre Auth/ACL Plugin
emq_auth_redis	Redis Auth/ACL Plugin
emq_auth_mongo	MongoDB Auth/ACL Plugin

Allow Anonymous

Configure etc/emq.conf to allow anonymous authentication:

```
## Allow Anonymous authentication
mqtt.allow_anonymous = true
```

Username/Password

Authenticate MQTT client with Username/Password:

Configure default users in etc/plugins/emq_auth_username.conf:

```
auth.user.$N.username = admin
auth.user.$N.password = public
```

Enable emq_auth_username plugin:

```
./bin/emqttd_ctl plugins load emq_auth_username
```

Add user by './bin/emqttd_ctl users' command:

```
$ ./bin/emqttd_ctl users add <Username> <Password>
```

ClientId

Authentication with MQTT ClientId.

Configure Client Ids in etc/plugins/emq_auth_clientid.conf:

```
auth.client.$N.clientid = clientid
auth.client.$N.password = passwd
```

Enable emq_auth_clientid plugin:

```
./bin/emqttd_ctl plugins load emq_auth_clientid
```

LDAP

etc/plugins/emq_auth_ldap.conf:

```
auth.ldap.servers = 127.0.0.1

auth.ldap.port = 389

auth.ldap.timeout = 30

auth.ldap.user_dn = uid=%u,ou=People,dc=example,dc=com

auth.ldap.ssl = false
```

Enable LDAP plugin:

```
./bin/emqttd_ctl plugins load emq_auth_ldap
```

HTTP

etc/plugins/emq_auth_http.conf:

```
## Variables: %u = username, %c = clientid, %a = ipaddress, %P = password, %t = topic

auth.http.auth_req = http://127.0.0.1:8080/mqtt/auth
auth.http.auth_req.method = post
auth.http.auth_req.params = clientid=%c,username=%u,password=%P

auth.http.super_req = http://127.0.0.1:8080/mqtt/superuser
auth.http.super_req.method = post
auth.http.super_req.params = clientid=%c,username=%u
```

Enable HTTP Plugin:

```
./bin/emqttd_ctl plugins load emq_auth_http
```

MySQL

Authenticate with MySQL database. Suppose that we create a mqtt_user table:

```
CREATE TABLE `mqtt_user` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(100) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `salt` varchar(20) DEFAULT NULL,
  `created` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mqtt_username` (`username`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Configure the 'auth_query' and 'password_hash' in etc/plugins/emq_auth_mysql.conf:

```
## Mysql Server
auth.mysql.server = 127.0.0.1:3306

## Mysql Pool Size
auth.mysql.pool = 8

## Mysql Username
## auth.mysql.username =

## Mysql Password
## auth.mysql.password =

## Mysql Database
auth.mysql.database = mqtt

## Variables: %u = username, %c = clientid

## Authentication Query: select password only
auth.mysql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.mysql.password_hash = sha256
```

```
## %% Superuser Query
auth.mysql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1
```

Enable MySQL plugin:

```
./bin/emqttd_ctl plugins load emq_auth_mysql
```

PostgreSQL

Authenticate with PostgreSQL database. Create a mqtt_user table:

```
CREATE TABLE mqtt_user (
  id SERIAL primary key,
  username character varying(100),
  password character varying(100),
  salt character varying(40)
);
```

Configure the 'auth_query' and 'password_hash' in etc/plugins/emq_auth_pgsql.conf:

```
## Postgre Server
auth.pgsql.server = 127.0.0.1:5432

auth.pgsql.pool = 8

auth.pgsql.username = root

#auth.pgsql.password =

auth.pgsql.database = mqtt

auth.pgsql.encoding = utf8

auth.pgsql.ssl = false

## Variables: %u = username, %c = clientid, %a = ipaddress

## Authentication Query: select password only
auth.pgsql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.pgsql.password_hash = sha256

## sha256 with salt prefix
## auth.pgsql.password_hash = salt sha256

## sha256 with salt suffix
## auth.pgsql.password_hash = sha256 salt

## Superuser Query
auth.pgsql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1
```

Enable the plugin:

```
./bin/emqttd_ctl plugins load emq_auth_pgsql
```

Redis

Authenticate with Redis. MQTT users could be stored in redis HASH, the key is "mqtt_user:<Username>".

Configure 'auth_cmd' and 'password_hash' in etc/plugins/emq_auth_redis.conf:

```
## Redis Server
auth.redis.server = 127.0.0.1:6379

## Redis Pool Size
auth.redis.pool = 8

## Redis Database
auth.redis.database = 0

## Redis Password
## auth.redis.password =

## Variables: %u = username, %c = clientid

## Authentication Query Command
auth.redis.auth_cmd = HGET mqtt_user:%u password

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.redis.password_hash = sha256

## Superuser Query Command
auth.redis.super_cmd = HGET mqtt_user:%u is_superuser
```

Enable the plugin:

```
./bin/emqttd_ctl plugins load emq_auth_redis
```

MongoDB

Create a *mqtt_user* collection:

```
{
    username: "user",
    password: "password hash",
    is_superuser: boolean (true, false),
    created: "datetime"
}
```

Configure *super_query*, *auth_query* in etc/plugins/emq_auth_mongo.conf:

```
## Mongo Server
auth.mongo.server = 127.0.0.1:27017

## Mongo Pool Size
auth.mongo.pool = 8

## Mongo User
```

```
## auth.mongo.user =

## Mongo Password
## auth.mongo.password =

## Mongo Database
auth.mongo.database = mqtt

## auth_query
auth.mongo.auth_query.collection = mqtt_user

auth.mongo.auth_query.password_field = password

auth.mongo.auth_query.password_hash = sha256

auth.mongo.auth_query.selector = username=%u

## super_query
auth.mongo.super_query.collection = mqtt_user

auth.mongo.super_query.super_field = is_superuser

auth.mongo.super_query.selector = username=%u
```

Enable the plugin:

```
./bin/emqttd_ctl plugins load emq_auth_mongo
```

ACL

The ACL of *EMQ* broker is responsible for authorizing MQTT clients to publish/subscribe topics.

The ACL rules define:

```
Allow|Deny Who Publish|Subscribe Topics
```

Access Control Module of *EMQ* broker will match the rules one by one:

```
          ---------               ---------               ---------
Client -> | Rule1 | --nomatch--> | Rule2 | --nomatch--> | Rule3 | --> Default
          ---------               ---------               ---------
              |                       |                       |
            match                   match                   match
             \|/                     \|/                     \|/
        allow | deny            allow | deny            allow | deny
```

Internal

The default ACL of *EMQ* broker is implemented by an 'internal' module.

Enable the 'internal' ACL module in etc/emq.conf:

```
## Default ACL File
mqtt.acl_file = etc/acl.conf
```

The ACL rules of 'internal' module are defined in 'etc/acl.conf' file:

```
%% Allow 'dashboard' to subscribe '$SYS/#'
{allow, {user, "dashboard"}, subscribe, ["$SYS/#"]}.

%% Allow clients from localhost to subscribe any topics
{allow, {ipaddr, "127.0.0.1"}, pubsub, ["$SYS/#", "#"]}.

%% Deny clients to subscribe '$SYS#' and '#'
{deny, all, subscribe, ["$SYS/#", {eq, "#"}]}.

%% Allow all by default
{allow, all}.
```

HTTP API

ACL by HTTP API: https://github.com/emqtt/emq_auth_http

Configure etc/plugins/emq_auth_http.conf and enable the plugin:

```
## 'access' parameter: sub = 1, pub = 2
auth.http.acl_req = http://127.0.0.1:8080/mqtt/acl
auth.http.acl_req.method = get
auth.http.acl_req.params = access=%A,username=%u,clientid=%c,ipaddr=%a,topic=%t

auth.http.acl_nomatch = deny
```

MySQL

ACL with MySQL database. The *mqtt_acl* table and default data:

```
CREATE TABLE `mqtt_acl` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `allow` int(1) DEFAULT NULL COMMENT '0: deny, 1: allow',
  `ipaddr` varchar(60) DEFAULT NULL COMMENT 'IpAddress',
  `username` varchar(100) DEFAULT NULL COMMENT 'Username',
  `clientid` varchar(100) DEFAULT NULL COMMENT 'ClientId',
  `access` int(2) NOT NULL COMMENT '1: subscribe, 2: publish, 3: pubsub',
  `topic` varchar(100) NOT NULL DEFAULT '' COMMENT 'Topic Filter',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO mqtt_acl (id, allow, ipaddr, username, clientid, access, topic)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

Configure 'acl-query' and 'acl_nomatch' in etc/plugins/emq_auth_mysql.conf:

```
## ACL Query Command
auth.mysql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid = '
↪%c'
```

```
## ACL nomatch
auth.mysql.acl_nomatch = deny
```

PostgreSQL

ACL with PostgreSQL database. The mqtt_acl table and default data:

```
CREATE TABLE mqtt_acl (
  id SERIAL primary key,
  allow integer,
  ipaddr character varying(60),
  username character varying(100),
  clientid character varying(100),
  access  integer,
  topic character varying(100)
);

INSERT INTO mqtt_acl (id, allow, ipaddr, username, clientid, access, topic)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

Configure 'acl_query' and 'acl_nomatch' in etc/plugins/emq_auth_pgsql.conf:

```
## ACL Query. Comment this query, the acl will be disabled.
auth.pgsql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid = '
↪%c'

## If no rules matched, return...
auth.pgsql.acl_nomatch = deny
```

Redis

ACL with Redis. The ACL rules are stored in a Redis HashSet:

```
HSET mqtt_acl:<username> topic1 1
HSET mqtt_acl:<username> topic2 2
HSET mqtt_acl:<username> topic3 3
```

Configure *acl_cmd* and *acl_nomatch* in etc/plugins/emq_auth_redis.conf:

```
## ACL Query Command
auth.redis.acl_cmd = HGETALL mqtt_acl:%u

## ACL nomatch
auth.redis.acl_nomatch = deny
```

MongoDB

Store ACL Rules in a *mqtt_acl* collection:

```
{
    username: "username",
    clientid: "clientid",
    publish: ["topic1", "topic2", ...],
    subscribe: ["subtop1", "subtop2", ...],
    pubsub: ["topic/#", "topic1", ...]
}
```

For example, insert rules into *mqtt_acl* collection:

```
db.mqtt_acl.insert({username: "test", publish: ["t/1", "t/2"], subscribe: ["user/%u",
↪"client/%c"]})
db.mqtt_acl.insert({username: "admin", pubsub: ["#"]})
```

Configure *acl_query* and *acl_nomatch* in etc/plugins/emq_auth_mongo.conf:

```
## acl_query
auth.mongo.acl_query.collection = mqtt_user

auth.mongo.acl_query.selector = username=%u

## acl_nomatch
auth.mongo.acl_nomatch = deny
```

MQTT Publish/Subscribe

MQTT is a an extremely lightweight publish/subscribe messaging protocol desgined for IoT, M2M and Mobile applications.

Install and start the *EMQ* broker, and then any MQTT client could connect to the broker, subscribe topics and publish messages.

MQTT Client Libraries: https://github.com/mqtt/mqtt.github.io/wiki/libraries

For example, we use mosquitto_sub/pub commands:

```
mosquitto_sub -t topic -q 2
mosquitto_pub -t topic -q 1 -m "Hello, MQTT!"
```

MQTT V3.1.1 Protocol Specification: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

MQTT Listener of emqttd broker is configured in etc/emq.conf:

```

```

```
## TCP Listener: 1883, 127.0.0.1:1883, ::1:1883
mqtt.listener.tcp = 1883

## Size of acceptor pool
mqtt.listener.tcp.acceptors = 8

## Maximum number of concurrent clients
mqtt.listener.tcp.max_clients = 1024
```

MQTT(SSL) Listener, Default Port is 8883:

```
## SSL Listener: 8883, 127.0.0.1:8883, ::1:8883
mqtt.listener.ssl = 8883

## Size of acceptor pool
mqtt.listener.ssl.acceptors = 4

## Maximum number of concurrent clients
mqtt.listener.ssl.max_clients = 512
```

HTTP Publish API

The *EMQ* broker provides a HTTP API to help application servers publish messages to MQTT clients.

HTTP API: POST http://host:8083/mqtt/publish

Web servers such as PHP, Java, Python, NodeJS and Ruby on Rails could use HTTP POST to publish MQTT messages to the broker:

```
curl -v --basic -u user:passwd -d "qos=1&retain=0&topic=/a/b/c&message=hello from␣
↪http..." -k http://localhost:8083/mqtt/publish
```

Parameters of the HTTP API:

Name	Description
client	clientid
qos	QoS(0, 1, 2)
retain	Retain(0, 1)
topic	Topic
message	Payload

Note: The API uses HTTP Basic Authentication.

MQTT Over WebSocket

Web browsers could connect to the emqttd broker directly by MQTT Over WebSocket.

WebSocket URI:	ws(s)://host:8083/mqtt
Sec-WebSocket-Protocol:	'mqttv3.1' or 'mqttv3.1.1'

The Dashboard plugin provides a test page for WebSocket:

```
http://127.0.0.1:18083/websocket.html
```

Listener of WebSocket and HTTP Publish API is configured in etc/emqttd.config:

```
## HTTP and WebSocket Listener
mqtt.listener.http = 8083
mqtt.listener.http.acceptors = 4
mqtt.listener.http.max_clients = 64
```

$SYS Topics

The *EMQ* broker periodically publishes internal status, MQTT statistics, metrics and client online/offline status to $SYS/# topics.

For the *EMQ* broker could be clustered, the $SYS topic path is started with:

```
$SYS/brokers/${node}/
```

'${node}' is the erlang node name of emqttd broker. For example:

```
$SYS/brokers/emqttd@127.0.0.1/version

$SYS/brokers/emqttd@host2/uptime
```

Note: The broker only allows clients from localhost to subscribe $SYS topics by default.

Sys Interval of publishing $SYS messages, could be configured in etc/emqttd.config:

```
## System Interval of publishing broker $SYS Messages
mqtt.broker.sys_interval = 60
```

Broker Version, Uptime and Description

Topic	Description
$SYS/brokers	Broker nodes
$SYS/brokers/${node}/version	Broker Version
$SYS/brokers/${node}/uptime	Broker Uptime
$SYS/brokers/${node}/datetime	Broker DateTime
$SYS/brokers/${node}/sysdescr	Broker Description

Online/Offline Status of MQTT Client

The topic path started with: $SYS/brokers/${node}/clients/

Topic	Payload(JSON)	Description
${clientid}/connected	**{ipaddress: "127.0.0.1", username: "test",** session: false, version: 3, connack: 0, ts: 1432648482}	Publish when a client connected
${clientid}/disconnected	**{reason: "keepalive_timeout",** ts: 1432749431}	Publish when a client disconnected

Properties of 'connected' Payload:

```
ipaddress: "127.0.0.1",
username:  "test",
session:   false,
protocol:  3,
connack:   0,
ts:        1432648482
```

Properties of 'disconnected' Payload:

```
reason: normal,
ts:     1432648486
```

Broker Statistics

Topic path started with: $SYS/brokers/${node}/stats/

Clients

Topic	Description
clients/count	Count of current connected clients
clients/max	Max number of cocurrent connected clients

Sessions

Topic	Description
sessions/count	Count of current sessions
sessions/max	Max number of sessions

Subscriptions

Topic	Description
subscriptions/count	Count of current subscriptions
subscriptions/max	Max number of subscriptions

Topics

Topic	Description
topics/count	Count of current topics
topics/max	Max number of topics

Broker Metrics

Topic path started with: $SYS/brokers/${node}/metrics/

Bytes Sent/Received

Topic	Description
bytes/received	MQTT Bytes Received since broker started
bytes/sent	MQTT Bytes Sent since the broker started

Packets Sent/Received

Topic	Description
packets/received	MQTT Packets received
packets/sent	MQTT Packets sent
packets/connect	MQTT CONNECT Packet received
packets/connack	MQTT CONNACK Packet sent
packets/publish/received	MQTT PUBLISH packets received
packets/publish/sent	MQTT PUBLISH packets sent
packets/subscribe	MQTT SUBSCRIBE Packets received
packets/suback	MQTT SUBACK packets sent
packets/unsubscribe	MQTT UNSUBSCRIBE Packets received
packets/unsuback	MQTT UNSUBACK Packets sent
packets/pingreq	MQTT PINGREQ packets received
packets/pingresp	MQTT PINGRESP Packets sent
packets/disconnect	MQTT DISCONNECT Packets received

Messages Sent/Received

Topic	Description
messages/received	Messages Received
messages/sent	Messages Sent
messages/retained	Messages Retained
messages/stored	TODO: Messages Stored
messages/dropped	Messages Dropped

Broker Alarms

Topic path started with: $SYS/brokers/${node}/alarms/

Topic	Description
${alarmId}/alert	New Alarm
${alarmId}/clear	Clear Alarm

Broker Sysmon

Topic path started with: '$SYS/brokers/${node}/sysmon/'

Topic	Description
long_gc	Long GC Warning
long_schedule	Long Schedule
large_heap	Large Heap Warning
busy_port	Busy Port Warning
busy_dist_port	Busy Dist Port

Trace

The emqttd broker supports to trace MQTT packets received/sent from/to a client, or trace MQTT messages published to a topic.

Trace a client:

```
./bin/emqttd_ctl trace client "clientid" "trace_clientid.log"
```

Trace a topic:

```
./bin/emqttd_ctl trace topic "topic" "trace_topic.log"
```

Lookup Traces:

```
./bin/emqttd_ctl trace list
```

Stop a Trace:

```
./bin/emqttd_ctl trace client "clientid" off

./bin/emqttd_ctl trace topic "topic" off
```

CHAPTER 7

Advanced

EMQ 2.0 release supports *Local Subscription* and *Shared Subscription*.

Local Subscription

The *EMQ* broker will not create global routes for *Local Subscription*, and only dispatch MQTT messages on local node.

```
mosquitto_sub -t '$local/topic'

mosquitto_pub -t 'topic'
```

Usage: subscribe a topic with *$local/* prefix.

Shared Subscription

Shared Subscription supports Load balancing to distribute MQTT messages between multiple subscribers in the same group:

```
                              _________
                             |         | --Msg1--> Subscriber1
Publisher--Msg1,Msg2,Msg3--->|   EMQ   | --Msg2--> Subscriber2
                             |         | --Msg3--> Subscriber3
                              _________
```

Two ways to create a shared subscription:

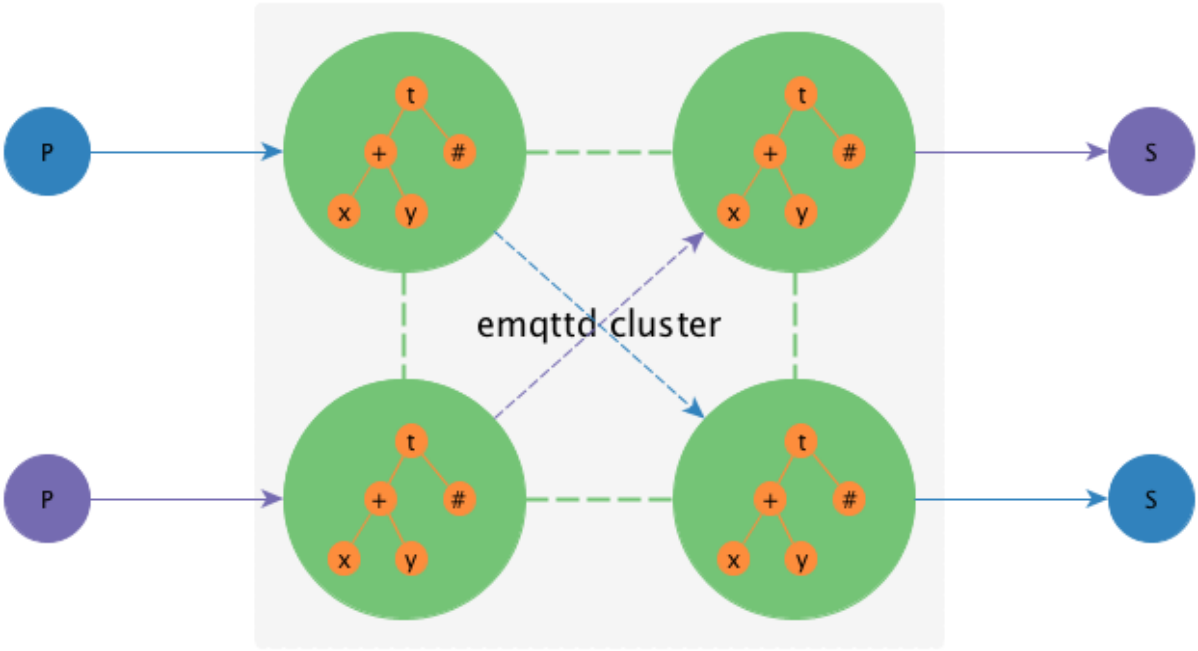
Prefix	Examples
$queue/	mosquitto_sub -t '$queue/topic'
$share/<group>/	mosquitto_sub -t '$share/group/topic'

CHAPTER 8

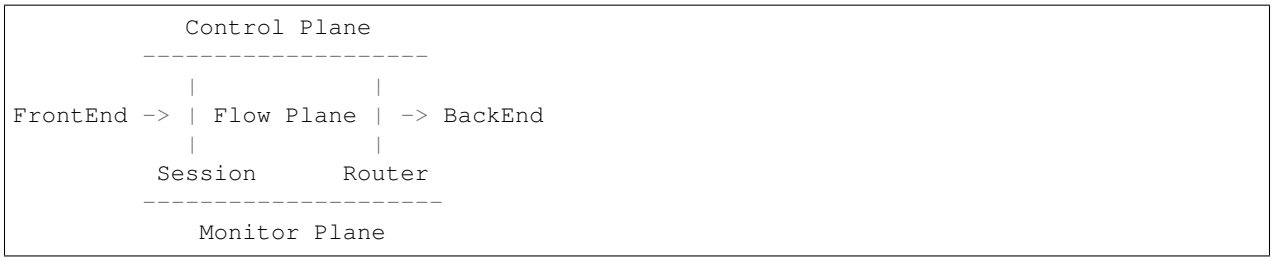
Design

Architecture

The *EMQ* broker 1.0 is more like a network Switch or Router, not a traditional enterprise message queue. Compared to a network router that routes packets based on IP or MPLS label, the *EMQ* broker routes MQTT messages based on topic trie.



The *EMQ* 2.0 seperated the Message Flow Plane and Monitor/Control Plane, the Architecture is something like:



Design Philosophy

1. Focus on handling millions of MQTT connections and routing MQTT messages between clustered nodes.
2. Embrace Erlang/OTP, The Soft-Realtime, Low-Latency, Concurrent and Fault-Tolerant Platform.
3. Layered Design: Connection, Session, PubSub and Router Layers.
4. Separate the Message Flow Plane and the Control/Management Plane.
5. Stream MQTT messages to various backends including MQ or databases.

System Layers

1. Connection Layer

   Handle TCP and WebSocket connections, encode/decode MQTT packets.

2. Session Layer

   Process MQTT PUBLISH/SUBSCRIBE Packets received from client, and deliver MQTT messages to client.

3. PubSub Layer

   Dispatch MQTT messages to subscribers in a node.

4. Routing(Distributed) Layer

   Route MQTT messages among clustered nodes.

Connection Layer

This layer is built on the eSockd library which is a general Non-blocking TCP/SSL Socket Server:

- Acceptor Pool and Asynchronous TCP Accept
- Parameterized Connection Module
- Max connections management
- Allow/Deny by peer address or CIDR
- Keepalive Support
- Rate Limit based on The Leaky Bucket Algorithm
- Fully Asynchronous TCP RECV/SEND

This layer is also responsible for encoding/decoding MQTT frames:

1. Parse MQTT frames received from client

2. Serialize MQTT frames sent to client
3. MQTT Connection Keepalive

Main erlang modules of this layer:

Module	Description
emqttd_client	TCP Client
emqttd_ws_client	WebSocket Client
emqttd_protocol	MQTT Protocol Handler
emqttd_parser	MQTT Frame Parser
emqttd_serializer	MQTT Frame Serializer

Session Layer

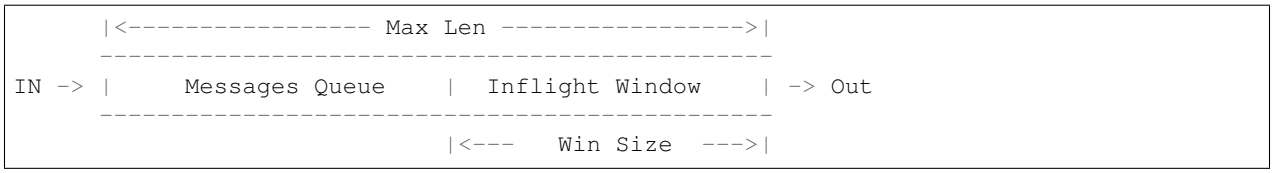
The session layer processes MQTT packets received from client and delivers PUBLISH packets to client.

A MQTT session will store the subscriptions and inflight messages in memory:

1. The Client's subscriptions.
2. Inflight qos1/2 messages sent to the client but unacked, QoS 2 messages which have been sent to the Client, but have not been completely acknowledged.
3. Inflight qos2 messages received from client and waiting for PUBREL. QoS 2 messages which have been received from the Client, but have not been completely acknowledged.
4. All qos1, qos2 messages published to when client is disconnected.

MQueue and Inflight Window

Concept of Message Queue and Inflight Window:



1. Inflight Window to store the messages delivered and await for PUBACK.
2. Enqueue messages when the inflight window is full.
3. If the queue is full, drop qos0 messages if store_qos0 is true, otherwise drop the oldest one.

The larger the inflight window size is, the higher the throughput is. The smaller the window size is, the more strict the message order is.

PacketId and MessageId

The 16-bit PacketId is defined by MQTT Protocol Specification, used by client/server to PUBLISH/PUBACK packets. A GUID(128-bit globally unique Id) will be generated by the broker and assigned to a MQTT message.

Format of the globally unique message id:

```
------------------------------------------------------
|        Timestamp       |  NodeID + PID  |  Sequence  |
|<------- 64bits ------->|<--- 48bits --->|<- 16bits ->|
------------------------------------------------------
```

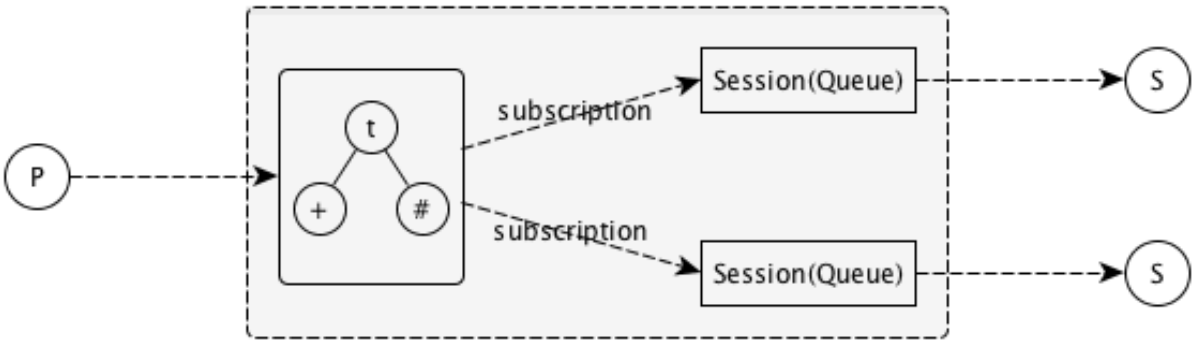
1. Timestamp: erlang:system_time if Erlang >= R18, otherwise os:timestamp
2. NodeId: encode node() to 2 bytes integer
3. Pid: encode pid to 4 bytes integer
4. Sequence: 2 bytes sequence in one process

The PacketId and MessageId in a End-to-End Message PubSub Sequence:

```
PktId <-- Session --> MsgId <-- Router --> MsgId <-- Session --> PktId
```

PubSub Layer

The PubSub layer maintains a subscription table and is responsible to dispatch MQTT messages to subscribers.



MQTT messages will be dispatched to the subscriber's session, which finally delivers the messages to client.

Routing Layer

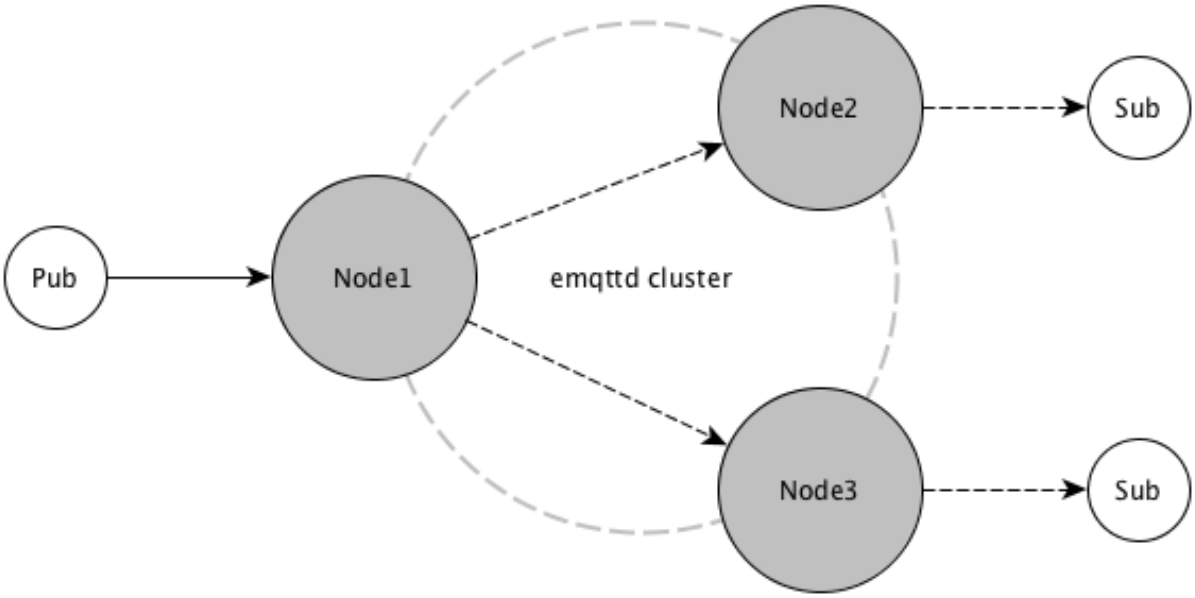
The routing(distributed) layer maintains and replicates the global Topic Trie and Routing Table. The topic tire is composed of wildcard topics created by subscribers. The Routing Table maps a topic to nodes in the cluster.

For example, if node1 subscribed 't/+/x' and 't/+/y', node2 subscribed 't/#' and node3 subscribed 't/a', there will be a topic trie and route table:

```
-------------------------
|             t           |
|            / \          |
|           +   #         |
|          /  \           |
|       x       y         |
-------------------------
| t/+/x -> node1, node3   |
| t/+/y -> node1          |
| t/#   -> node2          |
```

```
| t/a    -> node3          |
--------------------------
```

The routing layer would route MQTT messages among clustered nodes by topic trie match and routing table lookup:



The routing design follows two rules:

1. A message only gets forwarded to other cluster nodes if a cluster node is interested in it. This reduces the network traffic tremendously, because it prevents nodes from forwarding unnecessary messages.
2. As soon as a client on a node subscribes to a topic it becomes known within the cluster. If one of the clients somewhere in the cluster is publishing to this topic, the message will be delivered to its subscriber no matter to which cluster node it is connected.

Authentication and ACL

The *EMQ* broker supports an extensible authentication/ACL mechanism, which is implemented by emqttd_access_control, emqttd_auth_mod and emqttd_acl_mod modules.

emqttd_access_control module provides two APIs that help register/unregister auth or ACL module:

```
register_mod(auth | acl, atom(), list()) -> ok | {error, any()}.

register_mod(auth | acl, atom(), list(), non_neg_integer()) -> ok | {error, any()}.
```

Authentication Bahaviour

The emqttd_auth_mod defines an Erlang behaviour for authentication module:

```
-module(emqttd_auth_mod).

-ifdef(use_specs).
```

```
-callback init(AuthOpts :: list()) -> {ok, State :: any()}.

-callback check(Client, Password, State) -> ok | ignore | {error, string()} when
    Client    :: mqtt_client(),
    Password  :: binary(),
    State     :: any().

-callback description() -> string().

-else.

-export([behaviour_info/1]).

behaviour_info(callbacks) ->
    [{init, 1}, {check, 3}, {description, 0}];
behaviour_info(_Other) ->
    undefined.

-endif.
```

The authentication modules implemented by plugins:

Authorization(ACL)

The emqttd_acl_mod defines an Erlang behavihour for ACL module:

```
-module(emqttd_acl_mod).

-include("emqttd.hrl").

-ifdef(use_specs).

-callback init(AclOpts :: list()) -> {ok, State :: any()}.

-callback check_acl({Client, PubSub, Topic}, State :: any()) -> allow | deny | ignore␣
↪when
    Client    :: mqtt_client(),
    PubSub    :: pubsub(),
    Topic     :: binary().

-callback reload_acl(State :: any()) -> ok | {error, any()}.

-callback description() -> string().

-else.

-export([behaviour_info/1]).

behaviour_info(callbacks) ->
    [{init, 1}, {check_acl, 2}, {reload_acl, 1}, {description, 0}];
behaviour_info(_Other) ->
    undefined.

-endif.
```

emqttd_acl_internal implements the default ACL based on etc/acl.conf file:

```
%%%-----------------------------------------------------------------------
%%%
%%% -type who() :: all | binary() |
%%%                {ipaddr, esockd_access:cidr()} |
%%%                {client, binary()} |
%%%                {user, binary()}.
%%%
%%% -type access() :: subscribe | publish | pubsub.
%%%
%%% -type topic() :: binary().
%%%
%%% -type rule() :: {allow, all} |
%%%                 {allow, who(), access(), list(topic())} |
%%%                 {deny, all} |
%%%                 {deny, who(), access(), list(topic())}.
%%%
%%%-----------------------------------------------------------------------

{allow, {user, "dashboard"}, subscribe, ["$SYS/#"]}.

{allow, {ipaddr, "127.0.0.1"}, pubsub, ["$SYS/#", "#"]}.

{deny, all, subscribe, ["$SYS/#", {eq, "#"}]}.

{allow, all}.
```

Hooks Design

The *EMQ* broker implements a simple but powerful hooks mechanism to help users develop plugin. The broker would run the hooks when a client is connected/disconnected, a topic is subscribed/unsubscribed or a MQTT message is published/delivered/acked.

Hooks defined by the *EMQ* 2.0 broker:

Hook	Description
client.connected	Run when client connected to the broker successfully
client.subscribe	Run before client subscribes topics
client.unsubscribe	Run when client unsubscribes topics
session.subscribed	Run After client(session) subscribed a topic
session.unsubscribed	Run After client(session) unsubscribed a topic
message.publish	Run when a MQTT message is published
message.delivered	Run when a MQTT message is delivered
message.acked	Run when a MQTT message is acked
client.disconnected	Run when client disconnected from broker

The *EMQ* broker uses the Chain-of-responsibility_pattern to implement hook mechanism. The callback functions registered to hook will be executed one by one:

```
                 --------  ok | {ok, NewAcc}   --------  ok | {ok, NewAcc}   --------
(Args, Acc) --> | Fun1 | -------------------> | Fun2 | -------------------> | Fun3 | -
↪-> {ok, Acc} | {stop, Acc}
                 --------                      --------                      --------
                    |                             |                             |
               stop | {stop, NewAcc}         stop | {stop, NewAcc}         stop |
↪{stop, NewAcc}
```

The callback function for a hook should return:

Return	Description
ok	Continue
{ok, NewAcc}	Return Acc and Continue
stop	Break
{stop, NewAcc}	Return Acc and Break

The input arguments for a callback function are depending on the types of hook. Clone the emq_plugin_template project to check the argument in detail.

Hook Implementation

The hook APIs defined in emqttd module:

```
-module(emqttd).

%% Hooks API
-export([hook/4, hook/3, unhook/2, run_hooks/3]).
hook(Hook :: atom(), Callback :: function(), InitArgs :: list(any())) -> ok | {error,␣
↪any()}.

hook(Hook :: atom(), Callback :: function(), InitArgs :: list(any()), Priority ::␣
↪integer()) -> ok | {error, any()}.

unhook(Hook :: atom(), Callback :: function()) -> ok | {error, any()}.

run_hooks(Hook :: atom(), Args :: list(any()), Acc :: any()) -> {ok | stop, any()}.
```

And implemented in emqttd_hook module:

```
-module(emqttd_hook).

%% Hooks API
-export([add/3, add/4, delete/2, run/3, lookup/1]).

add(HookPoint :: atom(), Callback :: function(), InitArgs :: list(any())) -> ok.

add(HookPoint :: atom(), Callback :: function(), InitArgs :: list(any()), Priority ::␣
↪integer()) -> ok.

delete(HookPoint :: atom(), Callback :: function()) -> ok.

run(HookPoint :: atom(), Args :: list(any()), Acc :: any()) -> any().

lookup(HookPoint :: atom()) -> [#callback{}].
```

Hook Usage

The emq_plugin_template project provides the examples for hook usage:

```
-module(emq_plugin_template).

-export([load/1, unload/0]).

-export([on_message_publish/2, on_message_delivered/3, on_message_acked/3]).
```

```
load(Env) ->
    emqttd:hook('message.publish', fun ?MODULE:on_message_publish/2, [Env]),
    emqttd:hook('message.delivered', fun ?MODULE:on_message_delivered/3, [Env]),
    emqttd:hook('message.acked', fun ?MODULE:on_message_acked/3, [Env]).

on_message_publish(Message, _Env) ->
    io:format("publish ~s~n", [emqttd_message:format(Message)]),
    {ok, Message}.

on_message_delivered(ClientId, Message, _Env) ->
    io:format("delivered to client ~s: ~s~n", [ClientId, emqttd_
↪message:format(Message)]),
    {ok, Message}.

on_message_acked(ClientId, Message, _Env) ->
    io:format("client ~s acked: ~s~n", [ClientId, emqttd_message:format(Message)]),
    {ok, Message}.

unload() ->
    emqttd:unhook('message.publish', fun ?MODULE:on_message_publish/2),
    emqttd:unhook('message.acked', fun ?MODULE:on_message_acked/3),
    emqttd:unhook('message.delivered', fun ?MODULE:on_message_delivered/3).
```

Plugin Design

Plugin is a normal erlang application that can be started/stopped dynamically by a running *EMQ* broker.

emqttd_plugins Module

The plugin mechanism is implemented by emqttd_plugins module:

```
-module(emqttd_plugins).

-export([load/1, unload/1]).

%% @doc Load a Plugin
load(PluginName :: atom()) -> ok | {error, any()}.

%% @doc UnLoad a Plugin
unload(PluginName :: atom()) -> ok | {error, any()}.
```

Load a Plugin

Use './bin/emqttd_ctl' CLI to load/unload a plugin:

```
./bin/emqttd_ctl plugins load emq_auth_redis

./bin/emqttd_ctl plugins unload emq_auth_redis
```

Plugin Template

http://github.com/emqtt/emq_plugin_template

Mnesia/ETS Tables

Table	Type	Description
mqtt_trie	mnesia	Trie Table
mqtt_trie_node	mnesia	Trie Node Table
mqtt_route	mnesia	Global Route Table
mqtt_local_route	mnesia	Local Route Table
mqtt_pubsub	ets	PubSub Tab
mqtt_subscriber	ets	Subscriber Tab
mqtt_subscription	ets	Subscription Tab
mqtt_session	mnesia	Global Session Table
mqtt_local_session	ets	Local Session Table
mqtt_client	ets	Client Table
mqtt_retained	mnesia	Retained Message Table

CHAPTER 9

Commands

The './bin/emqttd_ctl' command line could be used to query and administrate the *EMQ* broker.

> **Warning:** Cannot work on Windows

status

Show running status of the broker:

```
$ ./bin/emqttd_ctl status

Node 'emqttd@127.0.0.1' is started
emqttd 2.0 is running
```

broker

Query basic information, statistics and metrics of the broker.

broker	Show version, description, uptime of the broker
broker pubsub	Show status of the core pubsub process
broker stats	Show statistics of client, session, topic, subscription and route of the broker
broker metrics	Show metrics of MQTT bytes, packets, messages sent/received.

Query version, description and uptime of the broker:

```
$ ./bin/emqttd_ctl broker

sysdescr  : Erlang MQTT Broker
version   : 0.15.0
```

```
uptime    : 1 hours, 25 minutes, 24 seconds
datetime  : 2016-01-16 13:17:32
```

broker stats

Query statistics of MQTT Client, Session, Topic, Subscription and Route:

```
$ ./bin/emqttd_ctl broker stats

clients/count       : 1
clients/max         : 1
queues/count        : 0
queues/max          : 0
retained/count      : 2
retained/max        : 2
routes/count        : 2
routes/reverse      : 2
sessions/count      : 0
sessions/max        : 0
subscriptions/count : 1
subscriptions/max   : 1
topics/count        : 54
topics/max          : 54
```

broker metrics

Query metrics of Bytes, MQTT Packets and Messages(sent/received):

```
$ ./bin/emqttd_ctl broker metrics

bytes/received          : 297
bytes/sent              : 40
messages/dropped        : 348
messages/qos0/received  : 0
messages/qos0/sent      : 0
messages/qos1/received  : 0
messages/qos1/sent      : 0
messages/qos2/received  : 0
messages/qos2/sent      : 0
messages/received       : 0
messages/retained       : 2
messages/sent           : 0
packets/connack         : 5
packets/connect         : 5
packets/disconnect      : 0
packets/pingreq         : 0
packets/pingresp        : 0
packets/puback/received : 0
packets/puback/sent     : 0
packets/pubcomp/received: 0
packets/pubcomp/sent    : 0
packets/publish/received: 0
packets/publish/sent    : 0
packets/pubrec/received : 0
packets/pubrec/sent     : 0
```

```
packets/pubrel/received : 0
packets/pubrel/sent     : 0
packets/received        : 9
packets/sent            : 9
packets/suback          : 4
packets/subscribe       : 4
packets/unsuback        : 0
packets/unsubscribe     : 0
```

cluster

Cluster two or more emqttd brokers.

cluster join <Node>	Join the cluster
cluster leave	Leave the cluster
cluster remove <Node>	Remove a node from the cluster
cluster status	Query cluster status and nodes

Suppose we create two emqttd nodes on localhost and cluster them:

Folder	Node	MQTT Port
emqttd1	emqttd1@127.0.0.1	1883
emqttd2	emqttd2@127.0.0.1	2883

Start emqttd1 node:

```
cd emqttd1 && ./bin/emqttd start
```

Start emqttd2 node:

```
cd emqttd2 && ./bin/emqttd start
```

Under emqttd2 folder:

```
$ ./bin/emqttd_ctl cluster join emqttd1@127.0.0.1

Join the cluster successfully.
Cluster status: [{running_nodes,['emqttd1@127.0.0.1','emqttd2@127.0.0.1']}]
```

Query cluster status:

```
$ ./bin/emqttd_ctl cluster status

Cluster status: [{running_nodes,['emqttd2@127.0.0.1','emqttd1@127.0.0.1']}]
```

Message Route between nodes:

```
# Subscribe topic 'x' on emqttd1 node
mosquitto_sub -t x -q 1 -p 1883

# Publish to topic 'x' on emqttd2 node
mosquitto_pub -t x -q 1 -p 2883 -m hello
```

emqttd2 leaves the cluster:

```
cd emqttd2 && ./bin/emqttd_ctl cluster leave
```

Or remove emqttd2 from the cluster on emqttd1 node:

```
cd emqttd1 && ./bin/emqttd_ctl cluster remove emqttd2@127.0.0.1
```

clients

Query MQTT clients connected to the broker:

clients list	List all MQTT clients
clients show <ClientId>	Show a MQTT Client
clients kick <ClientId>	Kick out a MQTT client

clients lists

Query All MQTT clients connected to the broker:

```
$ ./bin/emqttd_ctl clients list

Client(mosqsub/43832-airlee.lo, clean_sess=true, username=test, peername=127.0.0.
↪1:64896, connected_at=1452929113)
Client(mosqsub/44011-airlee.lo, clean_sess=true, username=test, peername=127.0.0.
↪1:64961, connected_at=1452929275)
...
```

Properties of the Client:

clean_sess	Clean Session Flag
username	Username of the client
peername	Peername of the TCP connection
connected_at	The timestamp when client connected to the broker

clients show <ClientId>

Show a specific MQTT Client:

```
./bin/emqttd_ctl clients show "mosqsub/43832-airlee.lo"

Client(mosqsub/43832-airlee.lo, clean_sess=true, username=test, peername=127.0.0.
↪1:64896, connected_at=1452929113)
```

clients kick <ClientId>

Kick out a MQTT Client:

```
./bin/emqttd_ctl clients kick "clientid"
```

sessions

Query all MQTT sessions. The broker will create a session for each MQTT client. Persistent Session if clean_session flag is true, transient session otherwise.

sessions list	List all Sessions
sessions list persistent	Query all persistent Sessions
sessions list transient	Query all transient Sessions
sessions show <ClientId>	Show a session

sessions list

Query all sessions:

```
$ ./bin/emqttd_ctl sessions list

Session(clientid, clean_sess=false, max_inflight=100, inflight_queue=0, message_
↪queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,␣
↪created_at=1452935508)
Session(mosqsub/44101-airlee.lo, clean_sess=true, max_inflight=100, inflight_queue=0,␣
↪message_queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,
↪ created_at=1452935401)
```

Properties of Session:

TODO:??

clean_sess	clean sess flag. false: persistent, true: transient
max_inflight	Inflight window (Max number of messages delivering)
inflight_queue	Inflight Queue Size
message_queue	Message Queue Size
message_dropped	Number of Messages Dropped for queue is full
awaiting_rel	The number of QoS2 messages received and waiting for PUBREL
awaiting_ack	The number of QoS1/2 messages delivered and waiting for PUBACK
awaiting_comp	The number of QoS2 messages delivered and waiting for PUBCOMP
created_at	Timestamp when the session is created

sessions list persistent

Query all persistent sessions:

```
$ ./bin/emqttd_ctl sessions list persistent

Session(clientid, clean_sess=false, max_inflight=100, inflight_queue=0, message_
↪queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,␣
↪created_at=1452935508)
```

sessions list transient

Query all transient sessions:

```
$ ./bin/emqttd_ctl sessions list transient

Session(mosqsub/44101-airlee.lo, clean_sess=true, max_inflight=100, inflight_queue=0,␣
↪message_queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,
↪ created_at=1452935401)
```

sessions show <ClientId>

Show a session:

```
$ ./bin/emqttd_ctl sessions show clientid

Session(clientid, clean_sess=false, max_inflight=100, inflight_queue=0, message_
↪queue=0, message_dropped=0, awaiting_rel=0, awaiting_ack=0, awaiting_comp=0,␣
↪created_at=1452935508)
```

routes

Show routing table of the broker.

routes list

List all routes:

```
$ ./bin/emqttd_ctl routes list

t2/# -> emqttd2@127.0.0.1
t/+/x -> emqttd2@127.0.0.1,emqttd@127.0.0.1
```

routes show <Topic>

Show a route:

```
$ ./bin/emqttd_ctl routes show t/+/x

t/+/x -> emqttd2@127.0.0.1,emqttd@127.0.0.1
```

topics

Query topic table of the broker.

topics list

Query all the topics:

```
$ ./bin/emqttd_ctl topics list

$SYS/brokers/emqttd@127.0.0.1/metrics/packets/subscribe: static
$SYS/brokers/emqttd@127.0.0.1/stats/subscriptions/max: static
$SYS/brokers/emqttd2@127.0.0.1/stats/subscriptions/count: static
...
```

topics show <Topic>

Show a topic:

```
$ ./bin/emqttd_ctl topics show '$SYS/brokers'

$SYS/brokers: static
```

subscriptions

Query the subscription table of the broker:

subscriptions list	List all subscriptions
subscriptions show <ClientId>	Show a subscription

subscriptions list

Query all subscriptions:

```
$ ./bin/emqttd_ctl subscriptions list

mosqsub/91042-airlee.lo -> t/y:1
mosqsub/90475-airlee.lo -> t/+/x:2
```

subscriptions list static

List all static subscriptions:

```
$ ./bin/emqttd_ctl subscriptions list static

clientid -> new_topic:1
```

subscriptions show <ClientId>

Show the subscriptions of a MQTT client:

```
$ ./bin/emqttd_ctl subscriptions show clientid

clientid: [{<<"x">>,1},{<<"topic2">>,1},{<<"topic3">>,1}]
```

plugins

List, load or unload plugins of emqttd broker.

plugins list	List all plugins
plugins load <Plugin>	Load Plugin
plugins unload <Plugin>	Unload (Plugin)

plugins list

List all plugins:

```
$ ./bin/emqttd_ctl plugins list

Plugin(emq_auth_clientid, version=2.0, description=Authentication with ClientId/
↪Password, active=false)
Plugin(emq_auth_http, version=2.0, description=Authentication/ACL with HTTP API,␣
↪active=false)
Plugin(emq_auth_ldap, version=2.0, description=Authentication/ACL with LDAP,␣
↪active=false)
Plugin(emq_auth_mongo, version=2.0, description=Authentication/ACL with MongoDB,␣
↪active=false)
Plugin(emq_auth_mysql, version=2.0, description=Authentication/ACL with MySQL,␣
↪active=false)
Plugin(emq_auth_pgsql, version=2.0, description=Authentication/ACL with PostgreSQL,␣
↪active=false)
Plugin(emq_auth_redis, version=2.0, description=Authentication/ACL with Redis,␣
↪active=false)
Plugin(emq_auth_username, version=2.0, description=Authentication with Username/
↪Password, active=false)
Plugin(emq_coap, version=0.2, description=CoAP Gateway, active=false)
Plugin(emq_dashboard, version=2.0, description=Dashboard, active=true)
Plugin(emq_mod_rewrite, version=2.0, description=EMQ Rewrite Module, active=false)
Plugin(emq_plugin_template, version=2.0, description=EMQ Plugin Template,␣
↪active=false)
Plugin(emq_recon, version=2.0, description=Recon Plugin, active=false)
Plugin(emq_reloader, version=3.0, description=Reloader Plugin, active=false)
Plugin(emq_sn, version=0.2, description=MQTT-SN Gateway, active=false)
Plugin(emq_stomp, version=2.0, description=Stomp Protocol Plugin, active=false)
```

Properties of a plugin:

version	Plugin Version
description	Plugin Description
active	If the plugin is Loaded

Load <Plugin>

Load a Plugin:

```
$ ./bin/emqttd_ctl plugins load emq_recon

Start apps: [recon,emq_recon]
Plugin emq_recon loaded successfully.
```

Unload <Plugin>

Unload a Plugin:

```
$ ./bin/emqttd_ctl plugins unload emq_recon

Plugin emq_recon unloaded successfully.
```

bridges

Bridge two or more *EMQ* brokers:

```
                ---------                          ---------
Publisher --> | node1 | --Bridge Forward--> | node2 | --> Subscriber
                ---------                          ---------
```

commands for bridge:

bridges list	List all bridges
bridges options	Show bridge options
bridges start <Node> <Topic>	Create a bridge
bridges start <Node> <Topic> <Options>	Create a bridge with options
bridges stop <Node> <Topic>	Delete a bridge

Suppose we create a bridge between emqttd1 and emqttd2 on localhost:

Name	Node	MQTT Port
emqttd1	emqttd1@127.0.0.1	1883
emqttd2	emqttd2@127.0.0.1	2883

The bridge will forward all the the 'sensor/#' messages from emqttd1 to emqttd2:

```
$ ./bin/emqttd_ctl bridges start emqttd2@127.0.0.1 sensor/#

bridge is started.

$ ./bin/emqttd_ctl bridges list

bridge: emqttd1@127.0.0.1--sensor/#-->emqttd2@127.0.0.1
```

The the 'emqttd1–sensor/#–>emqttd2' bridge:

```
#emqttd2 node

mosquitto_sub -t sensor/# -p 2883 -d

#emqttd1 node

mosquitto_pub -t sensor/1/temperature -m "37.5" -d
```

bridges options

Show bridge options:

```
$ ./bin/emqttd_ctl bridges options

Options:
  qos     = 0 | 1 | 2
  prefix  = string
  suffix  = string
  queue   = integer
Example:
  qos=2,prefix=abc/,suffix=/yxz,queue=1000
```

bridges stop <Node> <Topic>

Delete the emqttd1–sensor/#–>emqttd2 bridge:

```
$ ./bin/emqttd_ctl bridges stop emqttd2@127.0.0.1 sensor/#

bridge is stopped.
```

vm

Query the load, cpu, memory, processes and IO information of the Erlang VM.

vm all	Query all
vm load	Query VM Load
vm memory	Query Memory Usage
vm process	Query Number of Erlang Processes
vm io	Query Max Fds of VM

vm load

Query load:

```
$ ./bin/emqttd_ctl vm load

cpu/load1               : 2.21
cpu/load5               : 2.60
cpu/load15              : 2.36
```

vm memory

Query memory:

```
$ ./bin/emqttd_ctl vm memory

memory/total            : 23967736
memory/processes        : 3594216
memory/processes_used   : 3593112
memory/system           : 20373520
memory/atom             : 512601
memory/atom_used        : 491955
```

```
memory/binary           : 51432
memory/code             : 13401565
memory/ets              : 1082848
```

vm process

Query number of erlang processes:

```
$ ./bin/emqttd_ctl vm process

process/limit           : 8192
process/count           : 221
```

vm io

Query max, active file descriptors of IO:

```
$ ./bin/emqttd_ctl vm io

io/max_fds              : 2560
io/active_fds           : 1
```

trace

Trace MQTT packets, messages(sent/received) by ClientId or Topic.

trace list	List all the traces
trace client <ClientId> <LogFile>	Trace a client
trace client <ClientId> off	Stop tracing the client
trace topic <Topic> <LogFile>	Trace a topic
trace topic <Topic> off	Stop tracing the topic

trace client <ClientId> <LogFile>

Start to trace a client:

```
$ ./bin/emqttd_ctl trace client clientid log/clientid_trace.log

trace client clientid successfully.
```

trace client <ClientId> off

Stop tracing the client:

```
$ ./bin/emqttd_ctl trace client clientid off

stop tracing client clientid successfully.
```

trace topic <Topic> <LogFile>

Start to trace a topic:

```
$ ./bin/emqttd_ctl trace topic topic log/topic_trace.log

trace topic topic successfully.
```

trace topic <Topic> off

Stop tracing the topic:

```
$ ./bin/emqttd_ctl trace topic topic off

stop tracing topic topic successfully.
```

trace list

List all traces:

```
$ ./bin/emqttd_ctl trace list

trace client clientid -> log/clientid_trace.log
trace topic topic -> log/topic_trace.log
```

listeners

Show all the TCP listeners:

```
$ ./bin/emqttd_ctl listeners

listener on mqtt:ws:8083
  acceptors       : 4
  max_clients     : 64
  current_clients : 0
  shutdown_count  : []
listener on mqtt:ssl:8883
  acceptors       : 4
  max_clients     : 512
  current_clients : 0
  shutdown_count  : []
listener on mqtt:tcp:1883
  acceptors       : 8
  max_clients     : 1024
  current_clients : 0
  shutdown_count  : []
listener on dashboard:http:18083
  acceptors       : 2
  max_clients     : 512
  current_clients : 0
  shutdown_count  : []
```

listener parameters:

acceptors	TCP Acceptor Pool
max_clients	Max number of clients
current_clients	Count of current clients
shutdown_count	Statistics of client shutdown reason

mnesia

Show system_info of mnesia database.

admins

The 'admins' CLI is used to add/del admin account, which is registered by the dashboard plugin.

admins add <Username> <Password>	Add admin account
admins passwd <Username> <Password>	Reset admin password
admins del <Username>	Delete admin account

admins add

Add admin account:

```
$ ./bin/emqttd_ctl admins add root public
ok
```

admins passwd

Reset password:

```
$ ./bin/emqttd_ctl admins passwd root private
ok
```

admins del

Delete admin account:

```
$ ./bin/emqttd_ctl admins del root
ok
```

CHAPTER 10

Plugins

The *EMQ* broker could be extended by plugins. Users could develop plugins to customize authentication, ACL and functions of the broker, or integrate the broker with other systems.

The plugins that *EMQ* 2.0-rc.2 released:

Plugin	Description
emq_dashboard	Web Dashboard
emq_retainer	Store Retained Messages
emq_modules	Extend Modules Plugin
emq_auth_clientid	ClientId Auth Plugin
emq_auth_username	Username/Password Auth Plugin
emq_auth_ldap	LDAP Auth
emq_auth_http	HTTP Auth/ACL Plugin
emq_auth_mysql	MySQL Auth/ACL Plugin
emq_auth_pgsql	PostgreSQL Auth/ACL Plugin
emq_auth_redis	Redis Auth/ACL Plugin
emq_auth_mongo	MongoDB Auth/ACL Plugin
emq_coap	CoAP Protocol Plugin
emq_sn	MQTT-SN Protocol Plugin
emq_stomp	STOMP Protocol Plugin
emq_sockjs	STOMP over SockJS Plugin
emq_recon	Recon Plugin
emq_reloader	Reloader Plugin
emq_plugin_template	Template Plugin

emq_plugin_template - Template Plugin

A plugin is just a normal Erlang application which has its own configuration file: 'etc/<PluginName>.conf|config'.

emq_plugin_template is a plugin template.

Load, unload Plugin

Use 'bin/emqttd_ctl plugins' CLI to load, unload a plugin:

```
./bin/emqttd_ctl plugins load <PluginName>

./bin/emqttd_ctl plugins unload <PluginName>

./bin/emqttd_ctl plugins list
```

emq_retainer - Retainer Plugin

Renamed the **'emq_mod_retainer'_** to emq_retainer project in 2.1-beta release.

Configure Retainer Plugin

etc/plugins/emq_retainer.conf:

```
## disc: disc_copies, ram: ram_copies
## Notice: retainer's storage_type on each node in a cluster must be the same!
retainer.storage_type = disc

## Max number of retained messages
retainer.max_message_num = 1000000

## Max Payload Size of retained message
retainer.max_payload_size = 64KB

## Expiry interval. Never expired if 0
## h - hour
## m - minute
## s - second
retainer.expiry_interval = 0
```

emq_auth_clientid - ClientID Auth Plugin

Released in 2.0-rc.2: https://github.com/emqtt/emq_auth_clientid

Configure ClientID Auth Plugin

etc/plugins/emq_auth_clientid.conf:

```
##auth.client.$N.clientid = clientid
##auth.client.$N.password = passwd

## Examples
##auth.client.1.clientid = id
##auth.client.1.password = passwd
##auth.client.2.clientid = dev:devid
##auth.client.2.password = passwd2
```

```
##auth.client.3.clientid = app:appid
##auth.client.3.password = passwd3
```

Load ClientId Auth Plugin

```
./bin/emqttd_ctl plugins load emq_auth_clientid
```

emq_auth_username - Username Auth Plugin

Released in 2.0-rc.2: https://github.com/emqtt/emq_auth_username

Configure Username Auth Plugin

etc/plugins/emq_auth_username.conf:

```
##auth.user.$N.username = admin
##auth.user.$N.password = public

## Examples:
##auth.user.1.username = admin
##auth.user.1.password = public
##auth.user.2.username = feng@emqtt.io
##auth.user.2.password = public
```

Add username/password by *./bin/emqttd_ctl users* CLI:

```
$ ./bin/emqttd_ctl users add <Username> <Password>
```

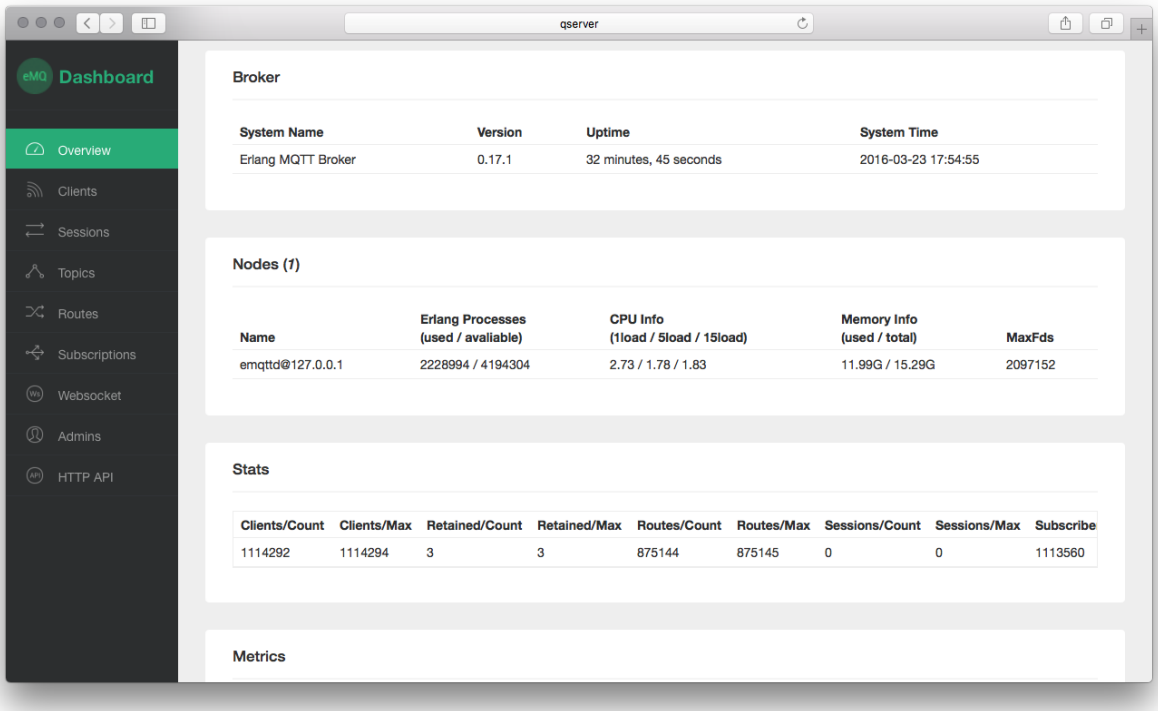
Load Username Auth Plugin

```
./bin/emqttd_ctl plugins load emq_auth_username
```

emq_dashboard - Dashboard Plugin

The Web Dashboard for *EMQ* broker. The plugin will be loaded automatically when the broker started successfully.

Address	http://localhost:18083
Default User	admin
Default Password	public



Configure Dashboard Plugin

etc/plugins/emq_dashboard.conf:

```
## HTTP Listener
dashboard.listener.http = 18083
dashboard.listener.http.acceptors = 2
dashboard.listener.http.max_clients = 512

## HTTPS Listener
## dashboard.listener.https = 18084
## dashboard.listener.https.acceptors = 2
## dashboard.listener.https.max_clients = 512
## dashboard.listener.https.handshake_timeout = 15
## dashboard.listener.https.certfile = etc/certs/cert.pem
## dashboard.listener.https.keyfile = etc/certs/key.pem
## dashboard.listener.https.cacertfile = etc/certs/cacert.pem
## dashboard.listener.https.verify = verify_peer
## dashboard.listener.https.fail_if_no_peer_cert = true
```

emq_auth_ldap: LDAP Auth Plugin

LDAP Auth Plugin: https://github.com/emqtt/emq_auth_ldap

Note: Released in 2.0-beta.1

Configure LDAP Plugin

etc/plugins/emq_auth_ldap.conf:

```
auth.ldap.servers = 127.0.0.1

auth.ldap.port = 389

auth.ldap.timeout = 30

auth.ldap.user_dn = uid=%u,ou=People,dc=example,dc=com

auth.ldap.ssl = false
```

Load LDAP Plugin

./bin/emqttd_ctl plugins load emq_auth_ldap

emq_auth_http - HTTP Auth/ACL Plugin

MQTT Authentication/ACL with HTTP API: https://github.com/emqtt/emq_auth_http

Note: Supported in 1.1 release

Configure HTTP Auth/ACL Plugin

etc/plugins/emq_auth_http.conf:

```
## Variables: %u = username, %c = clientid, %a = ipaddress, %P = password, %t = topic

auth.http.auth_req = http://127.0.0.1:8080/mqtt/auth
auth.http.auth_req.method = post
auth.http.auth_req.params = clientid=%c,username=%u,password=%P

auth.http.super_req = http://127.0.0.1:8080/mqtt/superuser
auth.http.super_req.method = post
auth.http.super_req.params = clientid=%c,username=%u

## 'access' parameter: sub = 1, pub = 2
auth.http.acl_req = http://127.0.0.1:8080/mqtt/acl
auth.http.acl_req.method = get
auth.http.acl_req.params = access=%A,username=%u,clientid=%c,ipaddr=%a,topic=%t

auth.http.acl_nomatch = deny
```

HTTP Auth/ACL API

Return 200 if ok

Return 4xx if unauthorized

Load HTTP Auth/ACL Plugin

```
./bin/emqttd_ctl plugins load emq_auth_http
```

emq_auth_mysql - MySQL Auth/ACL Plugin

MQTT Authentication, ACL with MySQL database.

MQTT User Table

```
CREATE TABLE `mqtt_user` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(100) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `salt` varchar(20) DEFAULT NULL,
  `is_superuser` tinyint(1) DEFAULT 0,
  `created` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mqtt_username` (`username`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

MQTT ACL Table

```
CREATE TABLE `mqtt_acl` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `allow` int(1) DEFAULT NULL COMMENT '0: deny, 1: allow',
  `ipaddr` varchar(60) DEFAULT NULL COMMENT 'IpAddress',
  `username` varchar(100) DEFAULT NULL COMMENT 'Username',
  `clientid` varchar(100) DEFAULT NULL COMMENT 'ClientId',
  `access` int(2) NOT NULL COMMENT '1: subscribe, 2: publish, 3: pubsub',
  `topic` varchar(100) NOT NULL DEFAULT '' COMMENT 'Topic Filter',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `mqtt_acl` (`id`, `allow`, `ipaddr`, `username`, `clientid`, `access`,␣
↪`topic`)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

Configure MySQL Auth/ACL Plugin

etc/plugins/emq_auth_mysql.conf:

```
## Mysql Server
auth.mysql.server = 127.0.0.1:3306

## Mysql Pool Size
auth.mysql.pool = 8

## Mysql Username
## auth.mysql.username =

## Mysql Password
## auth.mysql.password =

## Mysql Database
auth.mysql.database = mqtt

## Variables: %u = username, %c = clientid

## Authentication Query: select password only
auth.mysql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.mysql.password_hash = sha256

## %% Superuser Query
auth.mysql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1

## ACL Query Command
auth.mysql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid = '
↪%c'

## ACL nomatch
auth.mysql.acl_nomatch = deny
```

Load MySQL Auth/ACL plugin

```
./bin/emqttd_ctl plugins load emq_auth_mysql
```

emq_auth_pgsql - PostgreSQL Auth/ACL Plugin

MQTT Authentication/ACL with PostgreSQL Database.

Postgre MQTT User Table

```
CREATE TABLE mqtt_user (
  id SERIAL primary key,
  is_superuser boolean,
  username character varying(100),
  password character varying(100),
  salt character varying(40)
);
```

Postgre MQTT ACL Table

```
CREATE TABLE mqtt_acl (
  id SERIAL primary key,
  allow integer,
  ipaddr character varying(60),
  username character varying(100),
  clientid character varying(100),
  access  integer,
  topic character varying(100)
);

INSERT INTO mqtt_acl (id, allow, ipaddr, username, clientid, access, topic)
VALUES
    (1,1,NULL,'$all',NULL,2,'#'),
    (2,0,NULL,'$all',NULL,1,'$SYS/#'),
    (3,0,NULL,'$all',NULL,1,'eq #'),
    (5,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (6,1,'127.0.0.1',NULL,NULL,2,'#'),
    (7,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

Configure Postgre Auth/ACL Plugin

Plugin Config: etc/plugins/emq_auth_pgsql.conf.

Configure host, username, password and database of PostgreSQL:

```
## Postgre Server
auth.pgsql.server = 127.0.0.1:5432

auth.pgsql.pool = 8

auth.pgsql.username = root

#auth.pgsql.password =

auth.pgsql.database = mqtt

auth.pgsql.encoding = utf8

auth.pgsql.ssl = false

## Variables: %u = username, %c = clientid, %a = ipaddress

## Authentication Query: select password only
auth.pgsql.auth_query = select password from mqtt_user where username = '%u' limit 1

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.pgsql.password_hash = sha256

## sha256 with salt prefix
## auth.pgsql.password_hash = salt sha256

## sha256 with salt suffix
## auth.pgsql.password_hash = sha256 salt

## Superuser Query
```

```
auth.pgsql.super_query = select is_superuser from mqtt_user where username = '%u'␣
↪limit 1

## ACL Query. Comment this query, the acl will be disabled.
auth.pgsql.acl_query = select allow, ipaddr, username, clientid, access, topic from␣
↪mqtt_acl where ipaddr = '%a' or username = '%u' or username = '$all' or clientid = '
↪%c'

## If no rules matched, return...
auth.pgsql.acl_nomatch = deny
```

Load Postgre Auth/ACL Plugin

```
./bin/emqttd_ctl plugins load emq_auth_pgsql
```

emq_auth_redis - Redis Auth/ACL Plugin

MQTT Authentication, ACL with Redis: https://github.com/emqtt/emq_auth_redis

Configure Redis Auth/ACL Plugin

etc/plugins/emq_auth_redis.conf:

```
## Redis Server
auth.redis.server = 127.0.0.1:6379

## Redis Pool Size
auth.redis.pool = 8

## Redis Database
auth.redis.database = 0

## Redis Password
## auth.redis.password =

## Variables: %u = username, %c = clientid

## Authentication Query Command
auth.redis.auth_cmd = HGET mqtt_user:%u password

## Password hash: plain, md5, sha, sha256, pbkdf2
auth.redis.password_hash = sha256

## Superuser Query Command
auth.redis.super_cmd = HGET mqtt_user:%u is_superuser

## ACL Query Command
auth.redis.acl_cmd = HGETALL mqtt_acl:%u

## ACL nomatch
auth.redis.acl_nomatch = deny
```

Redis User Hash

Set a 'user' hash with 'password' field, for example:

```
HSET mqtt_user:<username> is_superuser 1
HSET mqtt_user:<username> password "passwd"
```

Redis ACL Rule Hash

The plugin uses a redis Hash to store ACL rules:

```
HSET mqtt_acl:<username> topic1 1
HSET mqtt_acl:<username> topic2 2
HSET mqtt_acl:<username> topic3 3
```

Note: 1: subscribe, 2: publish, 3: pubsub

Redis Subscription Hash

The plugin can store static subscriptions in a redis Hash:

```
HSET mqtt_subs:<username> topic1 0
HSET mqtt_subs:<username> topic2 1
HSET mqtt_subs:<username> topic3 2
```

Load Redis Auth/ACL Plugin

```
./bin/emqttd_ctl plugins load emq_auth_redis
```

emq_auth_mongo - MongoDB Auth/ACL Plugin

MQTT Authentication/ACL with MongoDB: https://github.com/emqtt/emq_auth_mongo

Configure MongoDB Auth/ACL Plugin

etc/plugins/emq_auth_mongo.conf:

```
## Mongo Server
auth.mongo.server = 127.0.0.1:27017

## Mongo Pool Size
auth.mongo.pool = 8

## Mongo User
## auth.mongo.user =

## Mongo Password
```

```
## auth.mongo.password =

## Mongo Database
auth.mongo.database = mqtt

## auth_query
auth.mongo.auth_query.collection = mqtt_user

auth.mongo.auth_query.password_field = password

auth.mongo.auth_query.password_hash = sha256

auth.mongo.auth_query.selector = username=%u

## super_query
auth.mongo.super_query.collection = mqtt_user

auth.mongo.super_query.super_field = is_superuser

auth.mongo.super_query.selector = username=%u

## acl_query
auth.mongo.acl_query.collection = mqtt_user

auth.mongo.acl_query.selector = username=%u

## acl_nomatch
auth.mongo.acl_nomatch = deny
```

MongoDB Database

MongoDB User Collection

```
{
    username: "user",
    password: "password hash",
    is_superuser: boolean (true, false),
    created: "datetime"
}
```

For example:

```
db.mqtt_user.insert({username: "test", password: "password hash", is_superuser: false}
↪)
db.mqtt_user:insert({username: "root", is_superuser: true})
```

MongoDB ACL Collection

```
{
    username: "username",
    clientid: "clientid",
    publish: ["topic1", "topic2", ...],
    subscribe: ["subtop1", "subtop2", ...],
```

```
    pubsub: ["topic/#", "topic1", ...]
}
```

For example:

```
db.mqtt_acl.insert({username: "test", publish: ["t/1", "t/2"], subscribe: ["user/%u",
↪"client/%c"]})
db.mqtt_acl.insert({username: "admin", pubsub: ["#"]})
```

Load MongoDB Auth/ACL Plugin

```
./bin/emqttd_ctl plugins load emq_auth_mongo
```

emq_modules - Modules Plugin

Merged the emq_mod_presence, emq_mod_subscription, emq_mod_rewrite into one emq_modules project.

Configure Modules Plugin

```
##--------------------------------------------------------------------
## Presence Module
##--------------------------------------------------------------------

## Enable Presence, Values: on | off
module.presence = on

module.presence.qos = 1

##--------------------------------------------------------------------
## Subscription Module
##--------------------------------------------------------------------

## Enable Subscription, Values: on | off
module.subscription = on

## Subscribe the Topics automatically when client connected
module.subscription.1.topic = $client/%c
## Qos of the subscription: 0 | 1 | 2
module.subscription.1.qos = 1

## module.subscription.2.topic = $user/%u
## module.subscription.2.qos = 1

##--------------------------------------------------------------------
## Rewrite Module
##--------------------------------------------------------------------

## Enable Rewrite, Values: on | off
module.rewrite = off

## {rewrite, Topic, Re, Dest}
```

```
## module.rewrite.rule.1 = "x/# ^x/y/(.+)$ z/y/$1"
## module.rewrite.rule.2 = "y/+/z/# ^y/(.+)/z/(.+)$ y/z/$2"
```

emq_mod_presence - Presence Module

Presence module will publish presence message to $SYS topic when a client connected or disconnected:

Note: This project has been deprecated in 2.1-beta release.

Configure Presence Module

etc/plugins/emq_mod_presence.conf:

```
## Enable presence module
## Values: on | off
module.presence = on

module.presence.qos = 0
```

Load Presence Module

Note: This module will be loaded by default.

```
./bin/emqttd_ctl plugins load emq_mod_presence
```

emq_mod_retainer - Retainer Module

Retainer module is responsible for storing MQTT retained messages.

Note: This project has been deprecated in 2.1-beta release.

Configure Retainer Module

etc/plugins/emq_mod_retainer.conf:

```
## disc: disc_copies, ram: ram_copies
module.retainer.storage_type = ram

## Max number of retained messages
module.retainer.max_message_num = 100000

## Max Payload Size of retained message
module.retainer.max_payload_size = 64KB
```

```
## Expired after seconds, never expired if 0
module.retainer.expired_after = 0
```

Load Retainer Module

Note: This module will be loaded by default.

```
./bin/emqttd_ctl plugins load emq_mod_retainer
```

emq_mod_subscription - Subscription Module

Subscription module forces the client to subscribe some topics when connected to the broker:

Note: This project has been deprecated in 2.1-beta release.

Configure Subscription Module

etc/plugins/emq_mod_subscription.conf:

```
## Subscribe the Topics automatically when client connected
module.subscription.1.topic = $client/%c
## Qos of the subscription: 0 | 1 | 2
module.subscription.1.qos = 1

##module.subscription.2.topic = $user/%u
##module.subscription.2.qos = 1
```

Load Subscription Module

Note: This module will be loaded by default.

```
./bin/emqttd_ctl plugins load emq_mod_subscription
```

emq_mod_rewrite - Topic Rewrite Module

Released in 2.0-rc.2: https://github.com/emqtt/emq_mod_rewrite

Note: This project has been deprecated in 2.1-beta release.

Configure Rewrite Module

etc/plugins/emq_mod_rewrite.config:

```
[
  {emq_mod_rewrite, [
    {rules, [
      %% {rewrite, Topic, Re, Dest}

      %% Example: x/y/ -> z/y/
      %% {rewrite, "x/#", "^x/y/(.+)$", "z/y/$1"},

      %% {rewrite, "y/+/z/#", "^y/(.+)/z/(.+)$", "y/z/$2"}
    ]}
  ]}
].
```

Load Rewrite Module

```
./bin/emqttd_ctl plugins load emq_mod_rewrite
```

emq_coap: CoAP Protocol Plugin

CoAP Protocol Plugin: https://github.com/emqtt/emqttd_coap

Configure CoAP Plugin

```
coap.server = 5683

coap.prefix.mqtt = mqtt

coap.handler.mqtt = emq_coap_gateway
```

Load CoAP Protocol Plugin

```
./bin/emqttd_ctl plugins load emq_coap
```

libcoap Client

```
yum install libcoap

% coap client publish message
coap-client -m post -e "qos=0&retain=0&message=payload&topic=hello" coap://localhost/
↪mqtt
```

emq_sn: MQTT-SN Protocol

MQTT-SN Protocol/Gateway Plugin.

Configure MQTT-SN Plugin

Note: UDP Port for MQTT-SN: 1884

etc/plugins/emq_sn.conf:

```
mqtt.sn.port = 1884
```

Load MQTT-SN Plugin

```
./bin/emqttd_ctl plugins load emq_sn
```

emq_stomp - STOMP Protocol

Support STOMP 1.0/1.1/1.2 clients to connect to emqttd broker and communicate with MQTT Clients.

Configure Stomp Plugin

etc/plugins/emq_stomp.conf:

Note: Default Port for STOMP Protocol: 61613

```
stomp.default_user.login = guest

stomp.default_user.passcode = guest

stomp.allow_anonymous = true

stomp.frame.max_headers = 10

stomp.frame.max_header_length = 1024

stomp.frame.max_body_length = 8192

stomp.listener = 61613

stomp.listener.acceptors = 4

stomp.listener.max_clients = 512
```

Load Stomp Plugin

```
./bin/emqttd_ctl plugins load emq_stomp
```

emq_sockjs - STOMP/SockJS Plugin

emq_sockjs plugin enables web browser to connect to emqttd broker and communicate with MQTT clients.

Warning: The plugin is deprecated in 2.0

Configure SockJS Plugin

Note: Default TCP Port: 61616

```
[
  {emq_sockjs, [

    {sockjs, []},

    {cowboy_listener, {stomp_sockjs, 61616, 4}},

    %% TODO: unused...
    {stomp, [
      {frame, [
        {max_headers,       10},
        {max_header_length, 1024},
        {max_body_length,   8192}
      ]}
    ]}
  ]}
].
```

Load SockJS Plugin

```
./bin/emqttd_ctl plugins load emqttd_sockjs
```

SockJS Demo Page

http://localhost:61616/index.html

emq_recon - Recon Plugin

The plugin loads recon library on a running *EMQ* broker. Recon libray helps debug and optimize an Erlang application.

Load Recon Plugin

```
./bin/emqttd_ctl plugins load emq_recon
```

Recon CLI

```
./bin/emqttd_ctl recon

recon memory                 #recon_alloc:memory/2
recon allocated              #recon_alloc:memory(allocated_types, current|max)
recon bin_leak               #recon:bin_leak(100)
recon node_stats             #recon:node_stats(10, 1000)
recon remote_load Mod        #recon:remote_load(Mod)
```

emq_reloader - Reloader Plugin

Erlang Module Reloader for Development

Note: Don't load the plugin in production!

Load *Reloader* Plugin

```
./bin/emqttd_ctl plugins load emq_reloader
```

reload CLI

```
./bin/emqttd_ctl reload

reload <Module>              # Reload a Module
```

Plugin Development Guide

Create a Plugin Project

Clone emq_plugin_template source from github.com:

```
git clone https://github.com/emqtt/emq_plugin_template.git
```

Create a plugin project with erlang.mk and depends on 'emqttd' application, the 'Makefile':

```
PROJECT = emq_plugin_abc
PROJECT_DESCRIPTION = emqttd abc plugin
PROJECT_VERSION = 1.0

BUILD_DEPS = emqttd
```

```
dep_emqttd = git https://github.com/emqtt/emqttd master

COVER = true

include erlang.mk
```

Template Plugin: https://github.com/emqtt/emq_plugin_template

Register Auth/ACL Modules

emq_auth_demo.erl - demo authentication module:

```
-module(emq_auth_demo).

-behaviour(emqttd_auth_mod).

-include_lib("emqttd/include/emqttd.hrl").

-export([init/1, check/3, description/0]).

init(Opts) -> {ok, Opts}.

check(#mqtt_client{client_id = ClientId, username = Username}, Password, _Opts) ->
    io:format("Auth Demo: clientId=~p, username=~p, password=~p~n",
              [ClientId, Username, Password]),
    ok.

description() -> "Demo Auth Module".
```

emq_acl_demo.erl - demo ACL module:

```
-module(emq_acl_demo).

-include_lib("emqttd/include/emqttd.hrl").

%% ACL callbacks
-export([init/1, check_acl/2, reload_acl/1, description/0]).

init(Opts) ->
    {ok, Opts}.

check_acl({Client, PubSub, Topic}, Opts) ->
    io:format("ACL Demo: ~p ~p ~p~n", [Client, PubSub, Topic]),
    allow.

reload_acl(_Opts) ->
    ok.

description() -> "ACL Module Demo".
```

emq_plugin_template_app.erl - Register the auth/ACL modules:

```
ok = emqttd_access_control:register_mod(auth, emq_auth_demo, []),
ok = emqttd_access_control:register_mod(acl, emq_acl_demo, []),
```

Register Callbacks for Hooks

The plugin could register callbacks for hooks. The hooks will be run by the broker when a client connected/disconnected, a topic subscribed/unsubscribed or a message published/delivered:

Name	Description
client.connected	Run when a client connected to the broker successfully
client.subscribe	Run before a client subscribes topics
client.unsubscribe	Run when a client unsubscribes topics
session.subscribed	Run after a client subscribed a topic
session.unsubscribed	Run after a client unsubscribed a topic
message.publish	Run when a message is published
message.delivered	Run when a message is delivered
message.acked	Run when a message(qos1/2) is acked
client.disconnected	Run when a client is disconnnected

emq_plugin_template.erl for example:

```
%% Called when the plugin application start
load(Env) ->
    emqttd:hook('client.connected', fun ?MODULE:on_client_connected/3, [Env]),
    emqttd:hook('client.disconnected', fun ?MODULE:on_client_disconnected/3, [Env]),
    emqttd:hook('client.subscribe', fun ?MODULE:on_client_subscribe/4, [Env]),
    emqttd:hook('session.subscribed', fun ?MODULE:on_session_subscribed/4, [Env]),
    emqttd:hook('client.unsubscribe', fun ?MODULE:on_client_unsubscribe/4, [Env]),
    emqttd:hook('session.unsubscribed', fun ?MODULE:on_session_unsubscribed/4, [Env]),
    emqttd:hook('message.publish', fun ?MODULE:on_message_publish/2, [Env]),
    emqttd:hook('message.delivered', fun ?MODULE:on_message_delivered/4, [Env]),
    emqttd:hook('message.acked', fun ?MODULE:on_message_acked/4, [Env]).
```

Register CLI Modules

emq_cli_demo.erl:

```
-module(emqttd_cli_demo).

-include_lib("emqttd/include/emqttd_cli.hrl").

-export([cmd/1]).

cmd(["arg1", "arg2"]) ->
    ?PRINT_MSG("ok");

cmd(_) ->
    ?USAGE([{"cmd arg1 arg2", "cmd demo"}]).
```

emq_plugin_template_app.erl - register the CLI module to *EMQ* broker:

```
emqttd_ctl:register_cmd(cmd, {emq_cli_demo, cmd}, []).
```

There will be a new CLI after the plugin loaded:

```
./bin/emqttd_ctl cmd arg1 arg2
```

Create Configuration File

Create *etc/${plugin_name}.conf\config* file for the plugin. The *EMQ* broker supports two type of config syntax:

1. ${plugin_name}.config with erlang syntax:

```
[
  {plugin_name, [
    {key, value}
  ]}
].
```

2. ${plugin_name}.conf with a general *k* = *v* syntax:

```
plugin_name.key = value
```

Build and Release the Plugin

1. clone emq-relx project:

```
git clone https://github.com/emqtt/emq-relx.git
```

2. Add *DEPS* in Makefile:

```
DEPS += plugin_name
dep_plugin_name = git url_of_plugin
```

3. Add the plugin in relx.config:

```
{plugin_name, load},
```

CHAPTER 11

Tuning Guide

Tuning the Linux Kernel, Networking, Erlang VM and the *EMQ* broker for one million concurrent MQTT connections.

Linux Kernel Tuning

The system-wide limit on max opened file handles:

```
# 2 million system-wide
sysctl -w fs.file-max=2097152
sysctl -w fs.nr_open=2097152
echo 2097152 > /proc/sys/fs/nr_open
```

The limit on opened file handles for current session:

```
ulimit -n 1048576
```

/etc/sysctl.conf

Add the 'fs.file-max' to /etc/sysctl.conf, make the changes permanent:

```
fs.file-max = 1048576
```

/etc/security/limits.conf

Persist the limits on opened file handles for users in /etc/security/limits.conf:

```
*      soft   nofile      1048576
*      hard   nofile      1048576
```

Network Tuning

Increase number of incoming connections backlog:

```
sysctl -w net.core.somaxconn=32768
sysctl -w net.ipv4.tcp_max_syn_backlog=16384
sysctl -w net.core.netdev_max_backlog=16384
```

Local Port Range:

```
sysctl -w net.ipv4.ip_local_port_range="1000 65535"
```

Read/Write Buffer for TCP connections:

```
sysctl -w net.core.rmem_default=262144
sysctl -w net.core.wmem_default=262144
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.core.optmem_max=16777216

#sysctl -w net.ipv4.tcp_mem='16777216 16777216 16777216'
sysctl -w net.ipv4.tcp_rmem='1024 4096 16777216'
sysctl -w net.ipv4.tcp_wmem='1024 4096 16777216'
```

Connection Tracking:

```
sysctl -w net.nf_conntrack_max=1000000
sysctl -w net.netfilter.nf_conntrack_max=1000000
sysctl -w net.netfilter.nf_conntrack_tcp_timeout_time_wait=30
```

The TIME-WAIT Buckets Pool, Recycling and Reuse:

```
net.ipv4.tcp_max_tw_buckets=1048576

# Enable fast recycling of TIME_WAIT sockets.  Enabling this
# option is not recommended for devices communicating with the
# general Internet or using NAT (Network Address Translation).
# Since some NAT gateways pass through IP timestamp values, one
# IP can appear to have non-increasing timestamps.
# net.ipv4.tcp_tw_recycle = 1
# net.ipv4.tcp_tw_reuse = 1
```

Timeout for FIN-WAIT-2 sockets:

```
net.ipv4.tcp_fin_timeout = 15
```

Erlang VM Tuning

Tuning and optimize the Erlang VM in etc/emq.conf file:

```
## Erlang Process Limit
node.process_limit = 2097152

## Sets the maximum number of simultaneously existing ports for this system
node.max_ports = 1048576
```

The EMQ Broker

Tune the acceptor pool, max_clients limit and sockopts for TCP listener in etc/emqttd.config:

```
## TCP Listener
mqtt.listener.tcp = 1883
mqtt.listener.tcp.acceptors = 64
mqtt.listener.tcp.max_clients = 1000000
```

Client Machine

Tune the client machine to benchmark emqttd broker:

```
sysctl -w net.ipv4.ip_local_port_range="500 65535"
sysctl -w fs.file-max=1000000
echo 1000000 > /proc/sys/fs/nr_open
ulimit -n 100000
```

emqtt_benchmark

Test tool for concurrent connections: http://github.com/emqtt/emqtt_benchmark

CHAPTER 12

Changes

Version 2.1.2

Release Data: 2017-04-21

Fix *emqttd_ctl sessions list* CLI

Newline character in emq.conf causing error;(emqttd#1000)

Fix crash caused by duplicated PUBREC packet (emqttd#1004)

Unload the 'session.created' and 'session.teminated' hooks (emq-plugin-template)

Version 2.1.1

Release Data: 2017-04-14

Localhost:8083/status returns 404 when AWS LB check the health of EMQ (emqttd#984)

Https listener not working in 2.1.0 as in 2.0.7 (emq-dashboard#105)

Fix mqtt-sn Gateway not working (emq-sn#12)

Upgrade emq-sn Plugin (emq-sn#11)

Upgrade emq-coap Plugin (emq-coap#21)

Version 2.1.0

Release Data: 2017-04-07

The stable release of 2.1 version.

Trouble with auth.mysql.acl_query (emq-auth-mysql#38)

Filter the empty fields in ACL table (emq-auth-mysql#39)

Version 2.1.0-rc.2

Release Data: 2017-03-31

Support pbkdf2 hash (emq-auth-mongo#46)

Kickout the conflict WebSocket connection (emqttd#963)

Correct licence in app.src (emqttd#958)

SSL options to connect to pgsql (emq-auth-pgsql#41)

Version 2.1.0-rc.1

Release Data: 2017-03-24

EMQ fails to start if run under a different linux user than that which first ran it (emqttd#842)

Depend on emqtt/pbkdf2 to fix the building errors of Travis CI (emqttd#957)

Depend on goldrush and emqtt/pbkdf2 to resolve the building errors (emqttd#956)

Fix 'rebar command not found' (emq-relx#33)

Compile error in v2.1.0-beta.2 (emq-relx#32)

Support salt with passwords (emq-auth-mongo#11)

Change the default storage_type to 'ram' (emq-retainer#13)

Version 2.1.0-beta.2

Release Data: 2017-03-13

Cannot find AwaitingAck (emqttd#597)

EMQ V2.1 crash when public with QoS = 2 (emqttd#919)

Support pbkdf2 hash (emqttd#940)

Add src/emqttd.app.src to be compatible with rebar3 (emqttd#920)

Add more test cases (emqttd#944)

CRASH REPORT Process <0.1498.0> with 0 neighbours crashed with reason: {ssl_error,{tls_alert,"certificate unknown"}} in esockd_connection:upgrade (emqttd#915)

auth.redis.password_hash = plain by default (emq-auth-redis#20)

Version 2.1.0-beta.1

Release Date: 2017-02-24

EMQ v2.1.0-beta.1 is now available.

Warning: EMQ 2.1+ Requires Erlang/OTP R19+ to build.

Since 2.1.0 release, we will tag EMQ versions accoding to the Semantic Versioning 2.0.0 principles. And we will release EMQ versions monthly, odd number releases for bugfix and optimization, and even number releases for bugfix and new features.

Tuning GC

1. All the WebSocket, Client, Session processes will hiberante and GC after a period of idle time.
2. Add 'mqtt.conn.force_gc_count' configuration to force the Client, Session processes to GC when high message throughput.
3. Tune the 'fullsweep_after' option of WebSocket, Client, Session processes.

Hooks API

Hooks module now support to register the same function with different tags.

Bugfix

emqttd#916: Add 'mqtt_msg_from()' type

emq-auth-http#15: ACL endpoint isnt called

Version 2.1-beta

Version 2.1-beta

Release Date: 2017-02-18

EMQ v2.1-beta is now available. We improved the design of Session/Inflight and use one timer to redeliver the inflight QoS1/2 messages, and improved the GC mechanism of MQTT connection process to reduce CPU usage at the high rate of messages.

Per Client, Session Statistics

Support Per Client, Session Statistics. Enable by configuration in etc/emq.conf:

```
mqtt.client.enable_stats = 60s

mqtt.session.enable_stats = 60s
```

Add 'missed' Metrics

The 'missed' metrics will be increased when EMQ broker received PUBACK, PUBREC, PUBREL, PUBCOMP packets from clients, but missing in inflight window:

```
packets/puback/missed

packets/pubrec/missed

packets/pubrel/missed

packets/pubcomp/missed
```

Integrate Syslog

Output EMQ log to syslog:

```
## Syslog. Enum: on, off
log.syslog = on

##  syslog level. Enum: debug, info, notice, warning, error, critical, alert,␣
↪emergency
log.syslog.level = error
```

Upgrade QoS

Support to upgrade QoS accoding to the subscription:

```
mqtt.session.upgrade_qos = on
```

Add 'acl reload' CLI

Reload acl.conf without restarting emqttd service (#885)

etc/emq.conf Changes

1. Rename mqtt.client_idle_timeout to mqtt.client.idle_timeout
2. Add mqtt.client.enable_stats
3. Add mqtt.session.upgrade_qos
4. Delete mqtt.session.collect_interval
5. Add mqtt.session.enable_stats
6. Rename mqtt.session.expired_after to mqtt.session.expiry_interval

Merge modules to emq_modules

Merge the emq_mod_presence, emq_mod_subscription, emq_mod_rewrite into emq_modules

Rename emq_mod_retainer to emq_retainer project

Dashboard Plugin

Overview page: Add 'missed' metrics Client page: Add 'SendMsg', 'RecvMsg' Fields Session page: DeliverMsgEnqueueMsg Fields

recon Plugin

Change the datatype of 'recon.gc_interval' to duration

reloader Plugin

Change the datatype of 'reloader.interval' to duration

Version 2.0.7

Release Date: 2017-01-20

The Last Maintenance Release for EMQ 2.0, and support to build RPM/DEB Packages.

Create the emq-package project: https://github.com/emqtt/emq-package

emq-auth-http#9: Update the priv/emq_auth_http.schema, *cuttlefish:unset()* if no super_req/acl_req config exists

emq-auth-mongo#31: *cuttlefish:unset()* if no ACL/super config exists

emq-dashboard#91: Fix the exception caused by binary payload

emq-relx#21: Improve the *binemqttd.cmd* batch script for windows platform

emqttd#873: Documentation: installing-from-source

emqttd#870: Documentation: The word in Documents is wrong

emqttd#864: Hook 'client.unsubscribe' need to handle 'stop'

emqttd#856: Support variables in etc/emq.conf: {{ runner_etc_dir }}, {{ runner_etc_dir }}, {{ runner_data_dir }}

Version 2.0.6

Release Date: 2017-01-08

Upgrade the esockd library to v4.1.1

esockd#41: Fast close the TCP socket if ssl:ssl_accept failed

emq-relx#15: The EMQ 2.0 broker cannot run on Windows.

emq-auth-mongo#31: Mongodb ACL Cannot work?

Version 2.0.5

Release Date: 2016-12-24

emq-auth-http#9: Disable ACL support

emq-auth-mongo#29: Disable ACL support

emq-auth-mongo#30: {datatype, flag}

Version 2.0.4

Release Date: 2016-12-16

emqttd#822: Test cases for SSL connections

emqttd#818: trap_exit to link WebSocket process

emqttd#799: Can't publish via HTTPS

Version 2.0.3

Release Date: 2016-12-12

emqttd#796: Unable to forbidden tcp lisener

emqttd#814: Cannot remove a 'DOWN' node from the cluster

emqttd#813: Change parameters order

emqttd#795: Fix metrics of websocket connections

emq-dashboard#88: Rename the default topic from "/World" to "world"

emq-dashboard#86: Lookup all online clients

emq-dashboard#85: Comment the default listener port

emq-mod-retainer#3: Retained messages get lost after EMQTT broker restart.

Version 2.0.2

Release Date: 2016-12-05

emqttd#787: Stop plugins before the broker stopped, clean routes when a node down

emqttd#790: Unable to start emqttd service if username/password contains special characters

emq-auth-clientid#4: Improve the configuration of emq_auth_clientid.conf to resolve emqttd#790

emq-auth-username#4: Improve the configuration of emq_auth_username.conf to resolve emqttd#790

Version 2.0.1

Release Date: 2016-11-30

emqttd#781: Update README for EMQ 2.0

emq_dashboard#84: Show the Cluster Status of Node

emq_dashboard#79: disc_copies to store mqtt_admin table

emq_auth_clientid: disc_copies to store mqtt_auth_clientid table

emq_auth_username: disc_copies to store mqtt_auth_username table

emq_mod_subscription#3: Remove emq_mod_subscription table and *module.subscription.backend* config

emq_plugin_template#5: Unregister Auth/ACL modules when the plugin unloaded

Version 2.0 "West of West Lake"

Release Date: 2016-11-24

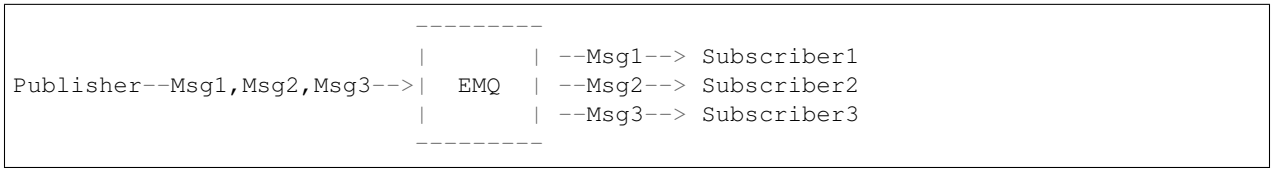
Release Name: West of West Lake

The *EMQ* Version 2.0, named "West of West Lake", has been released with a lot of improvements and enhancements, and is ready to deploy in production now.

1. First of all, the *EMQ* broker now supports *Shared Subscription* and *Local Subscription*.
2. Supports CoAP(RFC 7252) and MQTT-SN protocol/gateway.
3. Adopt a more user-friendly *k = v* syntax for the new configuration file.
4. Add more hooks and new plugins, integrate with HTTP, LDAP, Redis, MySQL, PostgreSQL and MongoDB.
5. Cross-platform Builds and Deployment. Run the broker on Linux, Unix, Windows, Raspberry Pi and ARM platform.

Shared Subscription

Shared Subscription supports Load balancing to distribute MQTT messages between multiple subscribers in the same group:



Create a shared subscription with *$queue/* or *$share/<group>/* prefix:

Prefix	Examples
$queue/	mosquitto_sub -t '$queue/topic
$share/<group>/	mosquitto_sub -t '$share/group/topic

Local Subscription

The *Local Subscription* will not create global routes on clustered nodes, and only dispatch MQTT messages on local node.

Usage: subscribe a topic with *$local/* prefix.

erlang.mk and relx

The *EMQ* 2.0 adopts erlang.mk and relx tools to build the whole projects on Linux, Unix and Windows.

CoAP Support

The *EMQ* 2.0 supports CoAP(RFC7252) protocol/gateway now, and supports communication between CoAP, MQTT-SN and MQTT clients.

CoAP Protocol Plugin: https://github.com/emqtt/emqttd_coap

MQTT-SN Support

The *EMQ* 2.0 now supports MQTT-SN protocol/gateway.

MQTT-SN Plugin: https://github.com/emqtt/emq_sn

New Configuration File

The release integrated with *cuttlefish* library, and adopted a more user-friendly *k* = *v* syntax for the new configuration file:

```
## Node name
node.name = emqttd@127.0.0.1
...
## Max ClientId Length Allowed.
mqtt.max_clientid_len = 1024
...
```

The new configuration files will be preprocessed and translated to an Erlang *app.config* before the EMQ broker started:

```
----------------------                                              2.0/schema/*.schema
↪    --------------------
| etc/emq.conf        |                   ------------------              \|/
↪    | data/app.config |
|        +            | --> mergeconf --> | data/app.conf | -->  cuttlefish generate  -
↪-> |                   |
| etc/plugins/*.conf |                    ------------------
↪    | data/vm.args    |
----------------------
↪    --------------------
```

OS Environment Variables

EMQ_NODE_NAME	Erlang node name
EMQ_NODE_COOKIE	Cookie for distributed erlang node
EMQ_MAX_PORTS	Maximum number of opened sockets
EMQ_TCP_PORT	MQTT TCP Listener Port, Default: 1883
EMQ_SSL_PORT	MQTT SSL Listener Port, Default: 8883
EMQ_HTTP_PORT	HTTP/WebSocket Port, Default: 8083
EMQ_HTTPS_PORT	HTTPS/WebSocket Port, Default: 8084

Docker Image

We released an official Docker Image for *EMQ* 2.0. The open source project for Dockerfile: https://github.com/emqtt/emq_docker.

Full Support for Windows

The *EMQ* 2.0 fully supports Windows platform. You can run 'emqttd_ctl' command and cluster two nodes on Windows now.

Bugfix and Enhancements

#764: add mqtt.cache_acl option

#667: Configuring emqttd from environment variables

#722: *mqtt/superuser* calls two times *emqtt_auth_http*

#754: "-heart" option for EMQ 2.0

#741: emq_auth_redis cannot use hostname as server address

Plugins

Plugin	Description
emq_dashboard	Web Dashboard
emq_auth_clientid	ClientId Auth Plugin
emq_auth_username	Username/Password Auth Plugin
emq_auth_ldap	LDAP Auth
emq_auth_http	HTTP Auth/ACL Plugin
emq_auth_mysql	MySQL Auth/ACL Plugin
emq_auth_pgsql	PostgreSQL Auth/ACL Plugin
emq_auth_redis	Redis Auth/ACL Plugin
emq_auth_mongo	MongoDB Auth/ACL Plugin
emq_mod_presence	Presence Module
emq_mod_retainer	Retainer Module
emq_mod_rewrite	Topic Rewrite Module
emq_mod_subscription	Subscription Module
emq_coap	CoAP Protocol Plugin
emq_sn	MQTT-SN Protocol Plugin
emq_stomp	STOMP Protocol Plugin
emq_sockjs	STOMP over SockJS Plugin
emq_recon	Recon Plugin
emq_reloader	Reloader Plugin
emq_plugin_template	Template Plugin

Version 2.0-rc.3

Release Date: 2016-11-01

1. Change the three modules(Presence, Retainer, Subscription) to standalone plugins:

emq_mod_retainer	Retained Message Storage
emq_mod_presence	Publish presence message to $SYS topics when client connected or disconnected
emq_mod_subscription	Subscribe topics automatically when client connected

2. Update the SSL certificates under the etc/certs/ folder.

3. Bugfix: Fixed a typo (#716)

4. Bugfix: emqttd_http can not use emq_auth_http? #739
5. Bugfix: emq_auth_redis cannot use hostname as server address (#741)
6. Upgrade Redis, MySQL, Postgre and MongoDB plugins to support hostname.

Version 2.0-rc.2

Release Date: 2016-10-19

1. A more user-friendly configuration for the EMQ broker. Integrate with *cuttlefish* library and adopt *K = V* syntax:

```
node.name = emqttd@127.0.0.1

...

mqtt.listener.tcp = 1883

...
```

2. Support OS Environments:

```
EMQ_NODE_NAME
EMQ_NODE_COOKIE
EMQ_MAX_PORTS
EMQ_TCP_PORT
EMQ_SSL_PORT
EMQ_HTTP_PORT
EMQ_HTTPS_PORT
```

3. Refactor all the modules and plugins, and adopt new configuration syntax.

TODO: issues closed.

Version 2.0-rc.1

Release Date: 2016-10-03

1. *mqtt/superuser* POST called two times in *emqtt_auth_http* (#696)
2. Close MQTT TCP connection if authentication failed (#707)
3. Improve the plugin management. Developer don't need to add plugin's config to rel/sys.config
4. Add *BUILD_DEPS* in the plugin's Makefile:

```
BUILD_DEPS = emqttd
dep_emqttd = git https://github.com/emqtt/emqttd emq20
```

5. Improve the design of Redis ACL.

Version 2.0-beta.3

Release Date: 2016-09-18

New Features

Shared Suscriptions (#639, #416):

```
mosquitto_sub -t '$queue/topic'
mosquitto_sub -t '$share/group/topic'
```

Local Subscriptions that will not create global routes:

```
mosquitto_sub -t '$local/topic'
```

Bugfix

Error on Loading *emqttd_auth_http* (#691)

Remove 'emqttd' application from dependencies (emqttd_coap PR#3)

Version 2.0-beta.2

Release Date: 2016-09-10

CoAP Support

Release an experimental CoAP Gateway: https://github.com/emqtt/emqttd_coap

API Breaking Changes

'$u', '$c' variables in emqttd.conf and modules/acl.conf changed to '%u', '%c'

Improve the design of mqtt retained message, replace emqttd_retainer with emqttd_mod_retainer.

Add 'session.subscribed', 'session.unsubscribed' hooks, remove 'client.subscribe.after' hook

Tab 'retained_message' -> 'mqtt_retained'

Bugfix

[2.0 beta1] FORMAT ERROR: "~s PUBLISH to ~s: ~p" (PR #671)

Fixing issues in cluster mode. (PR #681)

Fixing issues with unsubscribe hook (PR #673)

Version 2.0-beta.1

Release Date: 2016-08-30

Release Name: West of West Lake

EMQ - Shortened Project Name

Adopt a shortened project name: EMQ(Erlang/Enterprise/Elastic MQTT Broker)E means Erlang/OTP, Enterprise and Elastic.

Improve the Release Management

In order to iterate the project fast, we will adopt a new release management strategy since 2.0. There will be two or three 'Preview Release' named beta1, beta2 or beta3, and then one or two 'Release Candidate' named rc1, rc2 before a Major version is production ready.

Seperate Rel from Application

We split the emqttd 1.x project into two projects since 2.0-beta1 release to resolve the plugins' dependency issue.

A new project named **'emqttd-relx'_** is created and responsible for buiding the emqttd application and the plugins:

```
git clone https://github.com/emqtt/emqttd-relx.git

cd emqttd-relx && make

cd _rel/emqttd && ./bin/emqttd console
```

erlang.mk and relx

The rebar which is used in 1.x release is replaced by erlang.mk and relx tools since 2.0-beta1 release.

You can check the 'Makefile' and 'relx.config' in the release project of the borker: **'emqttd-relx'_** .

Improve Git Branch Management

stable	1.x Stable Branch
master	2.x Master Branch
emq10	1.x Developement Branch
emq20	2.x Development Branch
emq30	3.x Development Branch
issue#{id}	BugFix Branch

New Config Syntax

Since 2.0-beta1 release the configuration file of the broker and plugins adopt a new syntax like rebar.config and relx.config:

etc/emqttd.conf for example:

```
%% Max ClientId Length Allowed.
{mqtt_max_clientid_len, 512}.

%% Max Packet Size Allowed, 64K by default.
{mqtt_max_packet_size, 65536}.
```

```
%% Client Idle Timeout.
{mqtt_client_idle_timeout, 30}. % Second
```

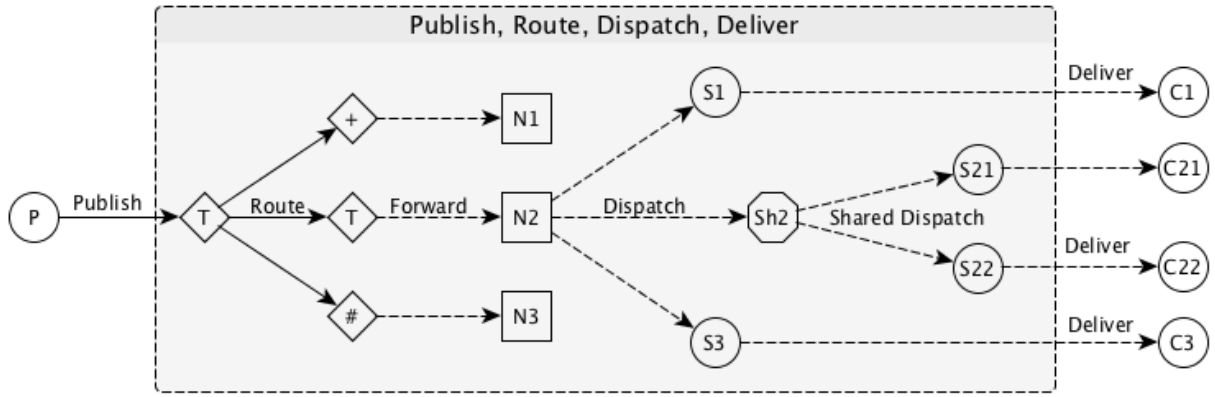
MQTT-SN Protocol Plugin

The MQTT-SN Protocol Plugin emqttd_sn has been ready in 2.0-beta1 release. The default UDP port of MQTT-SN is 1884.

Load the plugin:

```
./bin/emqttd_ctl plugins load emqttd_sn
```

Improve the PubSub Design



Improve the Plugin Management

The plugin of EMQ 2.0 broker is a normal erlang application which depends on and extends 'emqttd'. You can create a standalone plugin application project, and add it to **'emqttd-relx'_** Makefile as a DEP.

All the plugins' config files will be copied to emqttd/etc/plugins/ folder when making emqttd brinary packages in **'emqttd-relx'_** project:

```
emqttd/
  etc/
    modules/
    plugins/
       emqtt_coap.conf
       emqttd.conf
       emqttd_auth_http.conf
       emqttd_auth_mongo.conf
       emqttd_auth_mysql.conf
       emqttd_auth_pgsql.conf
       emqttd_auth_redis.conf
       emqttd_coap.conf
       emqttd_dashboard.conf
       emqttd_plugin_template.conf
       emqttd_recon.conf
```

```
        emqttd_reloader.conf
        emqttd_sn.conf
        emqttd_stomp.conf
```

EMQ 2.0 Documentation

http://emqtt.io/docs/v2/index.html

Version 1.1.3

Release Date: 2016-08-19

Support './bin/emqttd_ctl users list' CLI (#621)

Cannot publish payloads with a size of the order 64K using WebSockets (#643)

Optimize the procedures that retrieve the Broker version and Borker description in the tick timer (PR#627)

Fix SSL certfile, keyfile config (#651)

Version 1.1.2

Version 1.1.2

Release Date: 2016-06-30

Upgrade mysql-otp driver to 1.2.0 (#564, #523, #586, #596)

Fix WebSocket Client Leak (PR #612)

java.io.EOFException using paho java client (#551)

Send message from paho java client to javascript client (#552)

Compatible with the Qos0 PUBREL packet (#575)

Empty clientId with non-clean session accepted (#599)

Update docs to fix typos (#601, #607)

Version 1.1.1

Release Date: 2016-06-04

Compatible with the Qos0 PUBREL packet (#575)

phpMqtt Client Compatibility (#572)

java.io.EOFException using paho java client (#551)

Version 1.1

Release Date: 2016-06-01

Highlights

Upgrade eSockd library to 4.0 and Support IPv6

Support to listen on specific IP Address:

```
{mqtt, {"192.168.1.20", 1883}, [
    ...
]},
```

Add MongoDB, HTTP Authentication/ACL Plugins

Upgrade MySQL, PostgreSQL, Redis Plugins to support superuser authentication and avoid SQL Injection

Enhancements

Allow human-friendly IP addresses (PR#395)

File operation error: emfile (#445)

emqttd_plugin_mongo not found in emqttd (#489)

emqttd_plugin_mongo Error While Loading in emqttd (#505)

Feature request: HTTP Authentication (#541)

Compatible with the Qos0 PUBREL packet (#575)

Bugfix

Bugfix: function_clause exception occurs when registering a duplicated authentication module (#542)

Bugfix: ./emqttd_top msg_q result: {"init terminating in do_boot",{undef,[{etop,start,[],[]},{init,start_it,1,[]},{init,start_em,1,[]}]}} (#557)

Tests

111 common test cases.

Dashboard Plugin

WebSocket Page: Support 'Clean Session', Qos, Retained parameters (emqttd_dashboard#52)

Upgrade eSockd library to 4.0, Show OTP Release on Overview Page (emqttd_dashboard#61)

Changing dashboard credentials for username authentication (emqttd_dashboard#56)

Add './bin/emqttd_ctl admins' CLIsupport to add/delete admins

HTTP Auth Plugin

Authentication/ACL by HTTP API: https://github.com/emqtt/emqttd_auth_http

MongoDB Plugin

Upgrade Erlang MongoDB driver to v1.0.0

Support superuser authentication

Support ACL (emqttd_plugin_mongo#3)

MySQL Plugin

Support superuser authentication

Use parameterized query to avoid SQL Injection

Postgre Plugin

Support superuser authentication

Use parameterized query to avoid SQL Injection

Redis Plugin

Support superuser authentication

Support ClientId authentication by '%c' variable

Reloader Plugin

Reload modified modules during development automatically.

Version 1.0.3

Release Date: 2016-05-23

eSockd 3.2

MochiWeb 4.0.1

Version 1.0.2

Release Date: 2016-05-04

Issue#534 - './bin/emqttd_ctl vm' - add 'port/count', 'port/limit' statistics

Issue#535 - emqttd_client should be terminated properly even if exception happened when sending data

PR#519 - The erlang '-name' requires the fully qualified host name

emqttd_reloader plugin - help reload modified modules during development.

Version 1.0.1

Release Date: 2016-04-16

PR#515 - Fix '$queue' pubsub, add 'pubsub_queue' test and update docs

Version 1.0 (The Seven Mile Journey)

Release Date: 2016-04-13

Release Name: The Seven Mile Journey

We finally released Version 1.0 (The Seven Mile Journey) with full documentation after two years' development and more than fifty iterations.

The emqttd 1.0 implements a fully-featured, scalable, distributed and extensible open-source MQTT broker for IoT, M2M and Mobile applications:

1. Full MQTT V3.1/3.1.1 Protocol Specifications Support
2. Massively scalable - Scaling to 1 million connections on a single server
3. Distributed - Route MQTT Messages among clustered or bridged broker nodes
4. Extensible - LDAP, MySQL, PostgreSQL, Redis Authentication/ACL Plugins

Bugfix and Enhancements

Possible race condition using emqttd_cm (#486)

Improve the design of retained message expiration (#503)

Do not expire the retained messages from $SYS/# topics (#500)

Documentation

http://emqtt.io/docs

http://docs.emqtt.com/

Thanks

Thank Ericsson for the Great Erlang/OTP Platform (http://erlang.org/)!

Contributors on GitHub: @callbay @lsxredrain @hejin1026 @desoulter @turtleDeng @Hades32 @huangdan @phanimahesh @dvliman @Prots @joaohf

Partners: EACG (http://eacg.de/)

Favorite Band: The Seven Mile Journey (http://www.thesevenmilejourney.dk/)

Version 0.17.1-beta

Release Date: 2016-03-22

Enhancements

Time unit of session 'expired_after' changed to minute. (#479)

Dashboard

Code Review and improve the design of Dashboard.

Version 0.17.0-beta

Release Date: 2016-03-15

Highlights

Installation and Configuration Guide released on http://docs.emqtt.com

Improve and Consolidate the design of Hook, Server, PubSub and Router

Upgrade the [Web Dashboard](https://github.com/emqtt/emqttd_dashboard) to support pagination

Bridge emqttd broker to another emqttd broker & emqttd to mosquitto bridge (#438)

Enhancements

emqttd_ctl: better error message (#450)

./bin/emqttd_ctl: add 'routes' command:

```
routes list             # List all routes
routes show <Topic>     # Show a route
```

Add 'backend_subscription' table and support static subscriptions (emqttd_backend)

Add 'retained_message' table and refactor emqttd_retainer module (emqttd_backend)

A New Hook and Callback Design (emqttd_hook)

Add PubSub, Hooks APIs to emqttd module (emqttd)

Move start_listeners/0, stop_listeners/0 APIs to emqttd_app module (emqttd_app)

Tests

Add 100+ common test cases.

Plugins

Upgrade Dashboard, Redis, Stomp and Template Plugins

Version 0.16.0-beta

Release Date: 2016-02-16

Highlights

Licensed under the Apache License, Version 2.0 Now.

Improve the design of cluster, support to join or leave the cluster (#449):

```
$ ./bin/emqttd_ctl cluster
cluster join <Node>                     #Join the cluster
cluster leave                           #Leave the cluster
cluster remove <Node>                   #Remove the node from cluster
cluster status                          #Cluster status
```

Improve the design of Trie and Route, only the wildcard topics stored in Trie.

Common Test to replace EUnit.

Enhancements

mqtt_message record: add 'sender' field (#440)

refactor the emqttd, emqttd_time, emqttd_opts, emqttd_node modules.

Bugfix

noproc error when call to gen_server2:call(false, {add_route,Topic,<0.685.0>}, infinity) (#446)

Plugins

Changed the license of all plugins.

Version 0.15.0-beta

Release Date: 2016-01-31

Highlights

Optimize for Push Application, 500K+ Subscribers to a Topic.

Optimization for Route ETS insertion (#427)

Priority Message Queue for Persistent Session (#432)

Add Redis, MongoDB Plugins (#417)

Enhancements

Username/Password Authentication: Support to configure default users (#428)

Improve CLI Commands: pubsub, bridges, trace (#429)

emqttd_mod_subscription: fix client_connected/3

emqttd_auth_mod: add passwd_hash/2 function

priority_queue: add plen/2, out/2 functions

Bugfix

Fix dequeue/1 of emqttd_bridge...

Add emqttd:seed_now/0 function

Plugins

emqttd_plubin_mysql: Changed mysql driver to mysql-otp

emqttd_plugin_pgsql: Integrate with ecpool

emqttd_plugin_redis: First release

emqttd_plugin_mongo: First release

Version 0.14.1-beta

Release Date: 2015-12-28

Bugfix: emqttd_ws_client.erl: Unexpected Info: {'EXIT',<0.27792.18>,{shutdown,destroy}} (#413)

Improve: fix spec errors found by dialyzer

Version 0.14.0-beta

Release Date: 2015-12-18

Highlights

Scaling to 1.3 Million Concurrent MQTT Connections on a 12 Core, 32G CentOS server.

New PubSub, Router Design (#402). Prepare for scaling to 10 millions on one cluster.

Enhancements

Improve the gproc_pool usage with a general emqttd_pool_sup

Improve the design of emqttd_pubsub, add a new emqttd_router module

Improve the design of the whole supervisor tree

Route aging mechanism to remove the topics that have no subscriptions

Improve the dashboard, mysql, pgsql, stomp, sockjs plugins

Add 'topics', 'subscriptions' admin commands

Avoid using mnesia table index and mnesia:index_read API to lower CPU usage

Subscribe timeout exception (#366)

Long Delay on Multiple Topic Subscription (#365)

Subscriptions persistence (#344)

emqttd_ctl: 'subscriptions' command to force clients to subscribe some topics (#361)

Bugfix

emqttd_sm: spec of lookup_session/1 is not right BUG (#411)

Observer application should be removed from reltool.config for 'wx' app is not available (#410)

Benchmark

1.3 million concurrent MQTT connections on a 12 Core, 32G CentOS Server, consume about 15G Memory and 200% CPU.

Version 0.13.1-beta

Release Date: 2015-11-28

Bugfix: Plugin pathes error under windows (#387)

Improve: Too many error logs "[error] Session ..... Unexpected EXIT: client_pid=<0.14137.35>, exit_pid=<0.30829.22>, reason=nop..." (#383)

Improve: Define QOS0/1/2, Pooler Error (PR#382)

Improve: High CPU load when 400K unstable mobile connections (#377)

BugFix: emqttd_plugin_pgsql - error using same query with latest update plugin (pgsql#5)

Version 0.13.0-beta

Release Date: 2015-11-08

Highlights

Rate Limiting based on [Token Bucket](https://en.wikipedia.org/wiki/Token_bucket) and [Leaky Bucket](https://en.wikipedia.org/wiki/Leaky_bucket#The_Leaky_Bucket_Algorithm_as_a_Meter) Algorithm

Upgrade eSockd and MochiWeb libraries to support Parameterized Connection Module

Improve emqttd_client to support fully asynchronous socket networking

Enhancements

Protocol Compliant - Session Present Flag (#163)

Compilation fails if repo is cloned with a different name (#348)

emqttd_client: replace gen_tcp:send with port_command (#358)

TCP sndbuf, recbuf, buffer tuning (#359)

emqttd_client.erl to handle 'inet_async', 'inet_reply' properly (#360)

Refator the [client/session management design](https://github.com/emqtt/emqttd/blob/master/doc/design/ClientSession.md)

Bugfix

Cannot kick transient client out when clientId collision (#357)

Fix the order of emqttd_app:start_server/1 (#367)

emqttd_session:subscribe/2 will crash (#374)

Benchmark

[benchmark for 0.13.0 release](https://github.com/emqtt/emqttd/wiki/benchmark-for-0.13.0-release)

3.1G memory and 50+ CPU/core:

```
Connections: 250K
Subscribers: 250K
Topics:      50K
Qos1 Messages/Sec In:  4K
Qos1 Messages/Sec Out: 20K
Traffic In(bps):  12M+
Traffic Out(bps): 56M+
```

Version 0.12.3-beta

Release Date: 2015-10-22

Bugfix: emqttd_sysmon crasher for 'undefined' process_info (#350)

Bugfix: emqttd_client: catch parser exception (#353)

Version 0.12.2-beta

Release Date: 2015-10-16

Bugfix: Retained messages should not be expired if 'broker.retained.expired_after = 0' (#346)

Version 0.12.1-beta

Release Date: 2015-10-15

Highlight: Release for Bugfix and Code Refactor.

Feature: Retained message expiration (#182)

Improve: '$SYS/#' publish will not match '#' or '+/#' (#68)

Improve: Add more metrics and ignore '$SYS/#' publish (#266)

Improve: emqttd_sm should be optimized for clustered nodes may be crashed (#282)

Improve: Refactor emqttd_sysmon and suppress 'monitor' messages (#328)

Task: benchmark for 0.12.0 release (#225)

Benchmark: About 900K concurrent connections established on a 20Core, 32G CentOS server.

Version 0.12.0-beta

Release Date: 2015-10-08

Highlights

Enhance the **emqttd_ctl** module to allow plugins to register new commands (#256)

Add [emqttd_recon plugin](https://github.com/emqtt/emqttd_recon) to debug/optimize the broker (#235)

Add **'./bin/emqttd_ctl broker pubsub'** command to check the status of core pubsub processes

Add **'./bin/emqttd_top'** command(like etop) to show the top 'msg_q', 'reductions', 'memory' or 'runtime' processes

'rel/files/emqttd.config.production' for production deployment(default)

'rel/files/emqttd.config.development' for development deployment

Enhancements

Qos1/2 messages will not be dropped under unstable mobile network (#264)

emqttd_session:subscribe/2, emqttd_session:unsubscribe/2 APIs should be asynchronous (#292)

etc/emqttd.config: 'idle_timeout' option to close the idle client(socket connected but no 'CONNECT' frame received)

etc/emqttd.config: 'unack_retry_interval' option for redelivering Qos1/2 messages

How to monitor large 'message_queue_len' (#283)

Bugfix

Behaviour emqttd_auth_mod is missing init callback (#318)

Benchmark

Write a new [benchmark tool](https://github.com/emqtt/emqtt_benchmark) to benchmark this release

Hw requirements - 5K users, 25-50 msgs/sec, QoS=1 (#209)

Supported Number of Connections Greatly Reduced When Clients are Subscribing (#324)

Version 0.11.0-beta

Release Date: 2015-09-25

Highlight: Rebar to manage plugin dependencies.

Highlight: [Stomp](https://github.com/emqtt/emqttd_stomp) and [SockJS](https://github.com/emqtt/emqttd_sockjs) Plugins!

Improve: add rel/files/emqttd.config.development|production.

Improve: rel/reltool.config.script to release deps of plugin.

Improve: persist mnesia schema on slave nodes.

Improve: use timer:seconds/1 api.

Improve: The binary release will be compiled with R18.1 now.

Bugfix: issue#306 - emqttd_cm should unregister the duplicated client

Bugfix: issue#310 - usage of emqttd_ctl error: 'session list' should be 'sessions list'

Bugfix: issue#311 - './bin/emqttd_ctl sessions list' error

Bugfix: issue#312 - unsubcribe will lead to crash if emqttd_plugin_template plugin loaded

Version 0.10.4-beta

Release Date: 2015-09-18

Optimize session management and upgrade eSockd library to 2.7.1

[Benchmark for 0.10.4 release](https://github.com/emqtt/emqttd/wiki/benchmark-for-0.10.4-release)

Improve: issue#294 - [error] failed to start connection on 0.0.0.0:1883 - enotconn

Improve: issue#297 - How do I allow user with some pattern to access topic with some pattern?

Bugfix: issue#291 - "./bin/emqttd attach ..." cannot work

Bugfix: issue#284 - Should not use erlang:list_to_atom/1 in emqttd_vm.erl

Version 0.10.3-beta

Release Date: 2015-08-30

Bugfix: issue#271 - add emqttd_ws_client:subscribe/2 function

Bugfix: issue#269 - bin/emqttd Syntax error on ubuntu

Improve: issue#265 - client under unstable mobile network generate a lot of logs

Version 0.10.2-beta

Release Date: 2015-08-26

Improve: issue#257 - After the node name changed, the broker cannot restart for mnesia schema error.

Version 0.10.1-beta

Release Date: 2015-08-25

Bugfix: issue#259 - when clustered the emqttd_dashboard port is close, and the 'emqttd' application cannot stop normally.

Feature: issue#262 - Add 'http://host:8083/mqtt/status' Page for health check

Version 0.10.0-beta

Release Date: 2015-08-20

[Web Dashboard](https://github.com/emqtt/emqttd_dashboard) and [MySQL](https://github.com/emqtt/emqttd_plugin_mysql), [PostgreSQL](https://github.com/emqtt/emqttd_plugin_pgsql) Authentication/ACL Plugins!

Highlight: Web Dashboard to monitor Statistics, Metrics, Clients, Sessions and Topics of the broker.

Highlight: JSON/HTTP API to query all clients connected to broker.

Highlight: A new [Plugin Design](https://github.com/emqtt/emqttd/wiki/Plugin%20Design) and a [Template project](https://github.com/emqtt/emqttd_plugin_template) for plugin development.

Highlight: Authentication/ACL with MySQL, PostreSQl databases (#194, #172)

Feature: Session Statistics including inflight_queue, message_queue, message_dropped, awaiting_rel, awaiting_ack, awaiting_comp (#213)

Feature: Cookie based authentication for MQTT over websocket connections (#231)

Feature: Get all clients connected to the broker (#228, #230, #148, #129)

Feature: "./bin/emqttd_ctl clients show ClientId" to query client status (#226)

Feature: "./bin/emqttd_ctl clients kick ClientId" to kick out a client

Feature: "./bin/emqttd_ctl sessions list" to show all sessions

Feature: "./bin/emqttd_ctl sessions show ClientId" to show a session

Feature: Erlang VM metrics monitor with Web Dashboard (#59)

Improve: Too many "inflight queue is full!" log when session is overloaded (#247)

Improve: There are two many "MQueue(~s) drop ~s" logs if the message queue of session is small (#244)

Improve: gen_server2(from RabbitMQ) to improve emqttd_session, emqttd_pubsub

Improve: Makefile to build plugins

Bugfix: emqttd_broker:unhook/2 cannot work (#238)

Bugfix: emqttd plugin cannot include_lib("emqttd/include/emqttd.hrl") (#233)

Bugfix: Too many 'Session ~s cannot find PUBACK' logs (#212)

Bugfix: emqttd_pooler cannot work

Version 0.9.3-alpha

Release Date: 2015-07-25

Wiki: [Bridge](https://github.com/emqtt/emqttd/wiki/Bridge)

Improve: emqttd_protocol.hrl to define 'QOS_I'

Improve: emqttd_pubsub to add subscribe/2 API

Improve: ./bin/emqttd_ctl to support new bridges command

Bugfix: issue #206 - Cannot bridge two nodes

Version 0.9.2-alpha

Release Date: 2015-07-18

Improve: issue #196 - Add New Hook 'client.subscribe.after'

Version 0.9.1-alpha

Release Date: 2015-07-10

Bugfix: issue #189 - MQTT over WebSocket(SSL) cannot work?

Bugfix: issue #193 - 'client.ack' hook should be renamed to 'message.acked', and called by emqttd_broker:foreach_hooks

Version 0.9.0-alpha

Release Date: 2015-07-09

[Session, Queue, Inflight Window, Hooks, Global MessageId and More Protocol Compliant](https://github.com/emqtt/emqttd/releases/tag/0.9.0-alpha) Now!

Feature: Session/Queue/Inflight Window Design (#145).

Feature: Support to resume a persistent session on other clustered node.

Feature: Support alarm management.

Feature: emqttd_guid to generate global unique message id.

Feature: Hooks for message pub/ack.

Feature: Protocol compliant - message ordering, timeout and retry.

Improve: Every client will start_link a session process, whether or not the client is persistent.

Improve: etc/emqttd.config to support more session, queue configuration.

Improve: issue #179 - Max offline message queue {max_queue, 100} meaning.

Improve: issue #180 - Should change project structure for other projects maybe depend on 'emqttd'. Merge emqtt, emqttd apps.

Improve: issue #185 - PacketId and MessageId: the broker should generate global unique message id.

Improve: issue #187 - etc/emqttd.config to support https listener

Improve: issue #186 - emqttd_cm to store client details

Improve: issue #174 - add 'from' field to mqtt_message record.

Improve: issue #170 - $SYS Topics should support alarms.

Improve: issue #169 - Add More [Hooks](https://github.com/emqtt/emqttd/wiki/Hooks-Design)

Improve: issue #167 - Inflight window to assure message ordering.

Improve: issue #166 - Message delivery timeout and retry.

Improve: issue #143 - Qos1, Qos2 PubSub message timeout.

Improve: issue #122 - Labeling message with unique id. emqttd_guid module to generate global unique msgid.

Improve: emqttd_bridge to support pending message queue, and fix the wrong Qos design.

Improve: mqtt_message record to add 'msgid', 'from' and 'sys' fields.

Change: Add emqttd_mqueue, emqttd_guid, emqttd_alarm modules.

Bugfix: issue #184 - emqttd_stats:setstats is not right.

Bugfix: Closed issues #181, #119.

Tests: fix the parser, acl test cases.

Version 0.8.6-beta

Release Date: 2015-06-17

Bugfix: issue #175 - publish Will message when websocket is closed without 'DISCONNECT' packet

Version 0.8.5-beta

Release Date: 2015-06-10

Bugfix: issue #53 - client will receive duplicate messages when overlapping subscription

Version 0.8.4-beta

Release Date: 2015-06-08

Bugfix: issue #165 - duplicated message when publish 'retained' message to persistent client

Version 0.8.3-beta

Release Date: 2015-06-05

Bugfix: issue #158 - should queue:in new message after old one dropped

Bugfix: issue #155 - emqtt_parser.erl: parse_topics/3 should reverse topics

Bugfix: issue #149 - Forget to merge plugins/emqttd_auth_mysql from 'dev' branch to 'master' in 0.8.x release

Version 0.8.2-alpha

Release Date: 2015-06-01

Bugfix: issue #147 - WebSocket client cannot subscribe queue '$Q/queue/${clientId}'

Bugfix: issue #146 - emqttd_auth_ldap: fill(Username, UserDn) is not right

Version 0.8.1-alpha

Release Date: 2015-05-28

Client [Presence](https://github.com/emqtt/emqttd/wiki/Presence) Support and [$SYS Topics](https://github.com/emqtt/emqttd/wiki/\protect\T1\textdollarSYS-Topics) Redesigned!

Bugfix: issue #138 - when client disconnected normally, broker will not publish disconnected $SYS message

Bugfix: fix websocket url in emqttd/priv/www/websocket.html

Improve: etc/emqttd.config to allow websocket connections from any hosts

Improve: rel/reltool.config to exclude unnecessary apps.

Version 0.8.0-alpha

Release Date: 2015-05-25

[Hooks](https://github.com/emqtt/emqttd/wiki/Hooks%20Design), Modules and [Plugins](https://github.com/emqtt/emqttd/wiki/Plugin%20Design) to extend the broker Now!

Plugin: emqttd_auth_mysql - MySQL authentication plugin (issues #116, #120)

Plugin: emqttd_auth_ldap - LDAP authentication plugin

Feature: emqttd_broker to support Hooks API

Feature: issue #111 - Support 'Forced Subscriptions' by emqttd_mod_autosub module

Feature: issue #126 - Support 'Rewrite rules' by emqttd_mod_rewrite module

Improve: Support hooks, modules to extend the broker

Improve: issue #76 - dialyzer check

Improve: 'Get Started', 'User Guide', 'Developer Guide' Wiki

Improve: emqtt_topic to add join/1, feed_var/3, is_queue/1

Improve: emqttd_pooler to execute common tasks

Improve: add emqttd_sm_sup module, and use 'hash' gproc_pool to manage sessions

Tests: add more test cases for 'emqttd' app

Version 0.7.1-alpha

Release Date: 2015-05-04

Add doc/design/* and merge doc/* to github Wiki

Bugfix: issue #121 - emqttd cluster issuse

Bugfix: issue #123 - emqttd:unload_all_plugins/0 cannot unload any plugin

Bugfix: fix errors found by dialyzer

Version 0.7.0-alpha

Release Date: 2015-05-02

[MQTT over WebSocket(SSL)](https://github.com/emqtt/emqttd/wiki/MQTT-Over-WebSocket) Now!

[Plugin Achitecture](https://github.com/emqtt/emqttd/wiki/Plugin%20Design) based on OTP application

[Trace MQTT Packets or Messages](https://github.com/emqtt/emqttd/wiki/Trace%20Design) to log files

Feature: issue #40, #115 - WebSocket/SSL Support

Feature: issue #49, #105 - Plugin Architecture Support

Feature: issue #93 - Trace API Design

Improve: issue #109 - emqttd_broker should add subscribe, notify API

Improve: update README.md to add 'Goals', 'Contributors' chapters

Change: rename etc/app.config to etc/emqttd.config

Change: etc/emqttd.config changed

Bugfix: critical issue #54 - error when resume session!

Bugfix: issue #118 - error report when UNSUBSCRIBE with no topics

Bugfix: issue #117 - sys_interval = 0 config cannot work

Bugfix: issue #112 - Makefile to support build plugins

Bugfix: issue #96 - "make clean" cannot work

Version 0.6.2-alpha

Release Date: 2015-04-24

Bugfix: critical issue #54, #104, #106 - error when resume session

Improve: add emqttd_cm_sup module, and use 'hash' gproc_pool to register/unregister client ids

Improve: kick old client out when session is duplicated.

Improve: move mnesia dir config from etc/app.config to etc/vm.args

Version 0.6.1-alpha

Release Date: 2015-04-20

Integrate with [gproc library](https://github.com/uwiger/gproc) to support pool

Feature: issues#91 - should use worker_pool to handle some async work?

Feature: issues#95 - Topic filters in ACL rule should support 'eq' tag

Improve: issues#84 - emqttd_pubsub is redesigned again to protect mnesia transaction

Improve: issues#74 - ACL Support and update [ACL Design Wiki](https://github.com/emqtt/emqttd/wiki/ACL-Design)

Version 0.6.0-alpha

Release Date: 2015-04-17

ACL Support Now: [ACL-Design Wiki](https://github.com/emqtt/emqttd/wiki/ACL-Design)

Authentication with username, clientid Now: [Authentication Wiki](https://github.com/emqtt/emqttd/wiki/Authentication)

Seperate common MQTT library to 'emqtt' application

Redesign message pubsub, route and retain modules

Redesign mnesia database cluster

Feature: issues#47 - authentication, authorization support

Feature: issues#92 - merge emqttd_acl and emqttd_auth to emqttd_access_control

Feature: emqttd_acl_mod, emqttd_auth_mod behaviour to extend ACL, authentication

Feature: issues#85 - lager:info to log subscribe, unsubscribe actions

Feature: issues#77 - authentication with clientid, ipaddress

Improve: issues#90 - fix lager_file_backend log format, and rotate 10 log files

Improve: issues#88 - use '-mneisa_create', '-mnesia_replicate' attributes to init mneisa

Improve: issues#87 - record mqtt_user and mqtt_client is duplicated

Improve: issues#81 - redesign nodes cluster to support disc_copies mnesia tables

Improve: issues#80 - redesign emqttd_cm to handle more concurrent connections

Improve: issues#70 - how to handle connection flood? Now could support 2K+ CONNECT/sec

Change: redesign mnesia tables: message, topic, subscriber, trie, trie_node

Bugfix: issues#83 - emqttd_broker stats cannot work

Bugfix: issues#75 - careless about function name when emqttd_pubsub handle getstats message

Version 0.5.5-beta

Release Date: 2015-04-09

Bugfix: issue #75 - careless about function name when emqttd_pubsub handle getstats message.

Bugfix: issue #79 - cannot find topic_subscriber table after cluster with other nodes.

Version 0.5.4-alpha

Release Date: 2015-03-22

Benchmark this release on a ubuntu/14.04 server with 8 cores, 32G memory from QingCloud.com:

```
200K Connections,
30K Messages/Sec,
20Mbps In/Out Traffic,
200K Topics,
200K Subscribers,

Consumed 7G memory, 40% CPU/core
```

Benchmark code: https://github.com/emqtt/emqttd_benchmark

Change: rewrite emqttd_pubsub to handle more concurrent subscribe requests.

Change: ./bin/emqttd_ctl add 'stats', 'metrics' commands.

Bugfix: issue #71, #72

Version 0.5.3-alpha

Release Date: 2015-03-19

Bugfix: issues#72 - emqttd_cm, emqtt_sm ets:match_delete/2 with wrong pattern

Version 0.5.2-alpha

Release Date: 2015-03-18

Change: upgrade esockd to 2.1.0-alpha, do not tune socket buffer for mqtt connection.

Version 0.5.1-alpha

Release Date: 2015-03-13

Change: upgrade esockd to v1.2.0-beta, rename 'acceptor_pool' to 'acceptors'

Version 0.5.0-alpha

Release Date: 2015-03-12

RENAME 'emqtt' to 'emqttd'!

Support [Broker Bridge](https://github.com/emqtt/emqttd/wiki/Bridge-Design) Now!

Change: rename project from 'emqtt' to 'emqttd'

Change: lager:debug to dump RECV/SENT packets

Feature: emqttd_bridge, emqttd_bridge_sup to support broker bridge

Feature: emqtt_event to publish client connected/disconnected message to $SYS topics

Feature: ./bin/emqttd_ctl add more commands: listeners, broker, bridges, start_bridge, stop_bridge...

Feature: issue#57 - support to configure max packet size

Feature: issue#68 - if sys_interval = 0, emqttd_broker will not publish messages to $SYS/brokers/#

Bugfix: issue#67 - subscribe '#' to receive all messages

Bugfix: issue#64 - emqtt_app start/2: should wait_for_databases

Test: emqttd_topic_tests add more '_match_test'

Version 0.4.0-alpha

Release Date: 2015-03-10

Support [$SYS Topics of Broker](https://github.com/emqtt/emqttd/wiki/\protect\T1\textdollarSYS-Topics-of-Broker) Now!

Feature: emqtt_broker to publish version, uptime, datetime to $SYS/brokers/# topics

Feature: emqtt_broker to publish count of clients, sessions, suscribers to $SYS/brokers/# topics

Feature: emqtt_metrics to publish bytes, packets, messages metrics to $SYS/brokers/# topics

Feature: add include/emqtt_systop.hrl

Change: emqtt_cm to count current clients

Change: emqtt_sm to count current sessions

Change: emqtt_pubsub to count current topics and suscribers

Change: emqtt_pubsub to add create/1 API

Change: emqtt_pubsub dispatch/2 to return number of subscribers

Change: emqtt_pubsub to count 'dropped' messages

Change: emqtt_opts to add merge/2 function

Test: add emqtt_serialiser_tests.erl

Version 0.3.4-beta

Release Date: 2015-03-08

Bugfix: emqtt_serialiser.erl cannot serialise UNSUBACK packets

Version 0.3.3-beta

Release Date: 2015-03-07

Bugfix: emqtt_serialiser.erl cannot serialise PINGRESP issue#60

Version 0.3.2-beta

Release Date: 2015-03-05

Improve: merge emqttc serialiser, parser, packet

Add: emqtt_opts to merge socket options

Version 0.3.1-beta

Release Date: 2015-03-02

Feature: SSL Socket Support

Feature: issue#44 HTTP API should add Qos parameter

Bugfix: issue#52 emqtt_session crash

Bugfix: issue#53 sslsocket keepalive error

Upgrade: esockd to v0.2.0

Upgrade: mochiweb to v3.0.0

Version 0.3.0-beta

Release Date: 2015-01-19

Feature: HTTP POST API to support 'qos', 'retain' parameters

Feature: $SYS system topics support

Change: Rewrite emqtt_topic.erl, use '', '#', '+' to replace <<"">>, <<"#">>, <<"+">>

Change: fix emqtt_pubsub.erl to match '#', '+'

Tests: emqtt_topic_tests.erl add more test cases

Version 0.3.0-alpha

Release Date: 2015-01-08

NOTICE: Full MQTT 3.1.1 support now!

Feature: Passed org.eclipse.paho.mqtt.testing/interoperability tests

Feature: Qos0, Qos1 and Qos2 publish and suscribe

Feature: session(clean_sess=false) management and offline messages

Feature: redeliver awaiting puback/pubrec messages(doc: Chapter 4.4)

Feature: retain messages, add emqtt_server module

Feature: MQTT 3.1.1 null client_id support

Bugfix: keepalive timeout to send will message

Improve: overlapping subscription support

Improve: add emqtt_packet:dump to dump packets

Test: passed org.eclipse.paho.mqtt.testing/interoperability

Test: simple cluster test

Closed Issues: #22, #24, #27, #28, #29, #30, #31, #32, #33, #34, #36, #37, #38, #39, #41, #42, #43

Version 0.2.1-beta

Release Date: 2015-01-08

pull request 26: Use binaries for topic paths and fix wildcard topics

emqtt_pubsub.erl: fix wildcard topic match bug caused by binary topic in 0.2.0

Makefile: deps -> get-deps

rebar.config: fix mochiweb git url

tag emqtt release accoding to [Semantic Versioning](http://semver.org/)

max clientId length is 1024 now.

Version 0.2.0

Release Date: 2014-12-07

rewrite the project, integrate with esockd, mochiweb

support MQTT 3.1.1

support HTTP to publish message

Version 0.1.5

Release Date: 2013-01-05

Bugfix: remove QOS_1 match when handle PUBREL request

Bugfix: reverse word in emqtt_topic:words/1 function

Version 0.1.4

Release Date: 2013-01-04

Bugfix: fix "mosquitto_sub -q 2 ....." bug

Bugfix: fix keep alive bug

Version 0.1.3

Release Date: 2013-01-04

Feature: Support QOS2 PUBREC, PUBREL, PUBCOMP messages

Bugfix: fix emqtt_frame to encode/decoe PUBREC/PUBREL messages

Version 0.1.2

Release Date: 2012-12-27

Feature: release support like riak

Bugfix: use ?INFO/?ERROR to print log in tcp_listener.erl

Version 0.1.1

Release Date: 2012-09-24

Feature: use rebar to generate release

Feature: support retained messages

Bugfix: send will msg when network error

Version 0.1.0

Release Date: 2012-09-21

The first public release.

CHAPTER 13

Upgrade

Upgrade to 2.0

Note: Cannot upgrade 1.x releases to 2.0 smoothly.

Upgrade steps:

1. Download and install emqttd-2.0 to the new directory, for example:

```
Old installation: /opt/emqttd-1.1.3/

New installation: /opt/emqttd-2.0/
```

2. Configure the etc/emq.conf for the 2.0 installation.
3. Configure etc/plugins/{plugin}.conf for the 2.0 new installation if you loaded plugins.
4. Edit the data/loaded_plugins, and add the plugins loaded in old installation.
5. Stop the old emqttd, and start the 2.0 installation.

Upgrade to 1.1.2

Note: 1.0+ releases can be upgraded to 1.1.2 smoothly

Steps:

1. Download and install emqttd-1.1.2 to the new directory, for example:

```
Old installation: /opt/emqttd_1_0_0/

New installation: /opt/emqttd_1_1_2/
```

2. Copy the 'etc/' and 'data/' from the old installation:

```
cp -R /opt/emqttd_1_0_0/etc/* /opt/emqttd_1_1_2/etc/

cp -R /opt/emqttd_1_0_0/data/* /opt/emqttd_1_1_2/data/
```

3. Copy the plugins/{plugin}/etc/* from the old installation if you loaded plugins.
4. Stop the old emqttd, and start the new one.

CHAPTER 14

License

Apache License Version 2.0