EMQ X Enterprise Documentation

3.4.5

<contact@emqx.io>

2020 03 06

Contents

CHAPTER 1

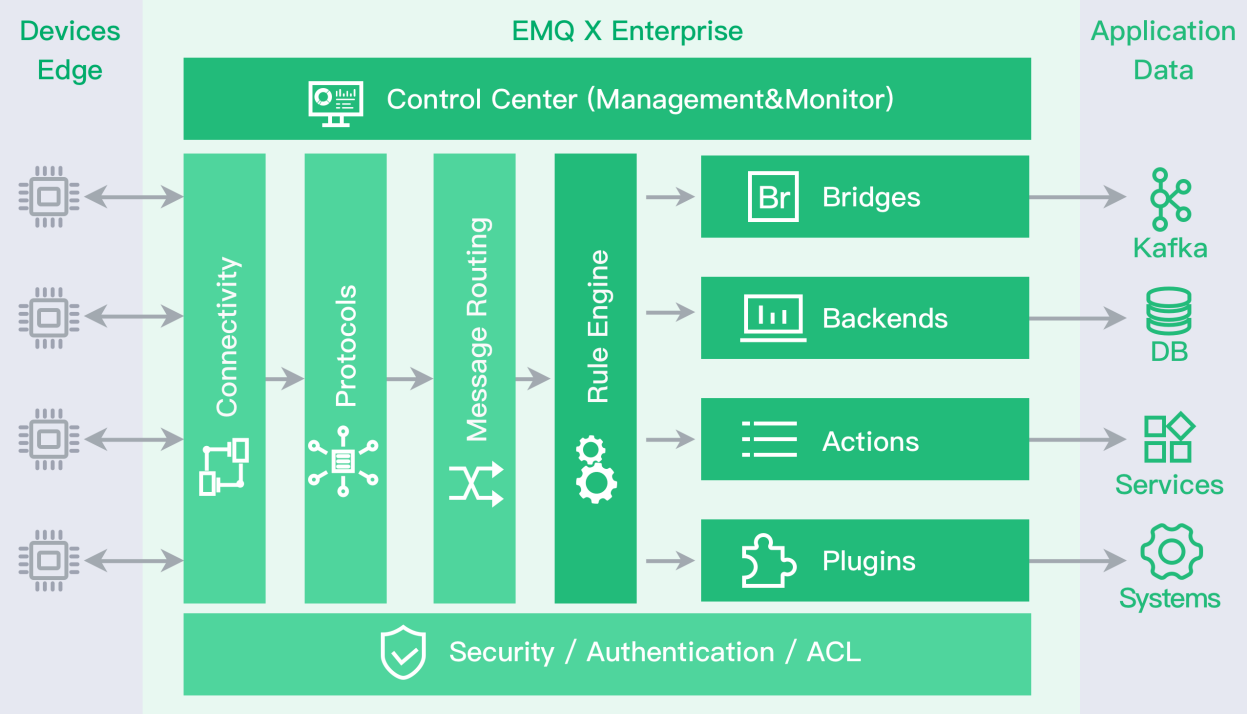
Overview

EMQ is a distributed, massively scalable, highly extensible MQTT message broker which can sustain million level connections. When the document is being writen, EMQ hits 200,000 downloads on Github, it is chosen by more than 3000 users worldwide. More than 10,000 nodes were already deployed and serve 30 million mobile and IoT connections.

EMQ X is the enterprise edition of the EMQ broker which extends the function and enhance the performance of EMQ. It improves the system architecture of EMQ, adopts Scalable RPC mechanism, provides more reliable clustering and higher performance of message routing.

EMQ X supports persistence MQTT messages to Redis, MySQL, PostgreSQL, MongoDB, Cassandra and other databases. It also supports bridging and forwarding MQTT messages to enterprise messaging middleware like Kafka and RabbitMQ.

EMQ X can be used as a scalable, reliable, enterprise-grade access platform for IoT, M2M, smart hardware, smart home and mobile messaging applications that serve millions of device terminals.



1.1 Design Objective

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 design objectives of EMQ X focus on enterprise-level requirements, such as high reliability, massive connections and extremely low latency of message delivery:

1. Steadily sustains massive MQTT client connections. A single node is able to handles about 1 million connections.
2. Distributed clustering, low-latency message routing. Single cluster handles 10 million level subscriptions.
3. Extensible broker design. Allow customizing various Auth/ACL extensions and data persistence extensions.
4. Supports comprehensive IoT protocols: MQTT, MQTT-SN, CoAP, WebSocket and other proprietary protocols.

1.2 Features

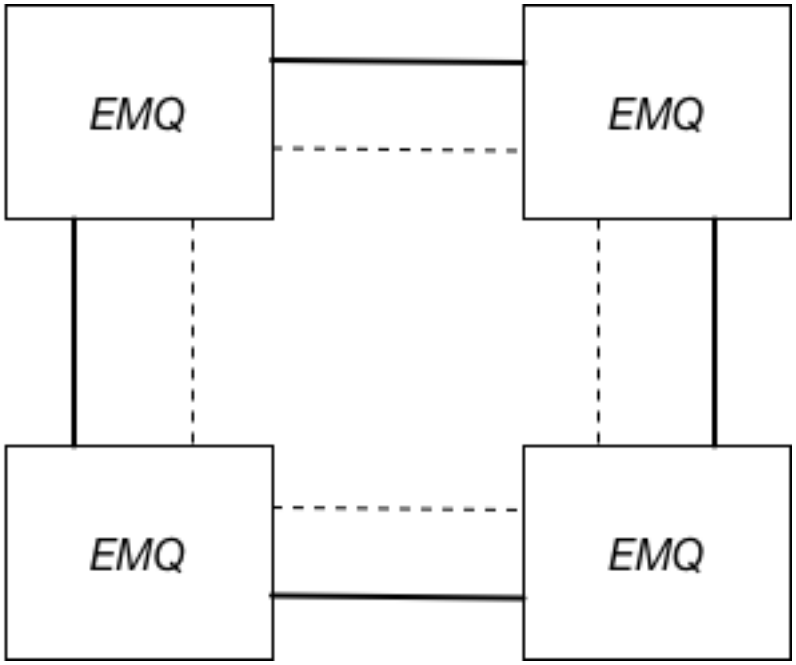
1. Scalable RPC Architecture: segregated cluster management channel and data channel between nodes.
2. Persistence to Redis: subscriptions, client connection status, MQTT messages, retained messages, SUB/UNSUB events.

3. Persistence to MySQL: subscriptions, client connection status, MQTT messages, retained messages.
4. Persistence to PostgreSQL: subscriptions, client connection status, MQTT messages, retained messages.
5. Persistence to MongoDB: subscriptions, client connection status, MQTT messages, retained messages.
6. Persistence to Cassandra: subscriptions, client connection status, MQTT messages, retained messages.
7. Persistence to DynamoDB: subscriptions, client connection status, MQTT messages, retained messages.
8. Persistence to InfluxDB: MQTT messages.
9. Persistence to OpenTDSB: MQTT messages.
10. Persistence to TimescaleDB: MQTT messages.
11. Bridge to Kafka: EMQ X forwards MQTT messages, client connected/disconnected event to Kafka.
12. Bridge to RabbitMQ: EMQ X forwards MQTT messages, client connected/disconnected event to RabbitMQ.
13. Bridge to Pulsar: EMQ X forwards MQTT messages, client connected/disconnected event to Pulsar.
14. Rule EngineConvert EMQ X events and messages to a specified format, then save them to a database table, or send them to a message queue.

1.3 Scalable RPC Architecture

EMQ X improves the communication mechanism between distributed nodes, segregates the cluster management channel and the data channel, and greatly improved the message throughput and the cluster reliability.

: the dash line indicates the cluster management and the solid line indicates the data exchange.



Scalable RPC configuration:

```
## TCP server port.
rpc.tcp_server_port = 5369

## Default TCP port for outgoing connections
rpc.tcp_client_port = 5369

## Client connect timeout
rpc.connect_timeout = 5000

## Client and Server send timeout
rpc.send_timeout = 5000

## Authentication timeout
rpc.authentication_timeout = 5000

## Default receive timeout for call() functions
rpc.call_receive_timeout = 15000

## Socket keepalive configuration
rpc.socket_keepalive_idle = 7200

## Seconds between probes
rpc.socket_keepalive_interval = 75

## Probes lost to close the connection
rpc.socket_keepalive_count = 9
```

: If firewalls are deployed between nodes, the 5369 port on each node must be opened.

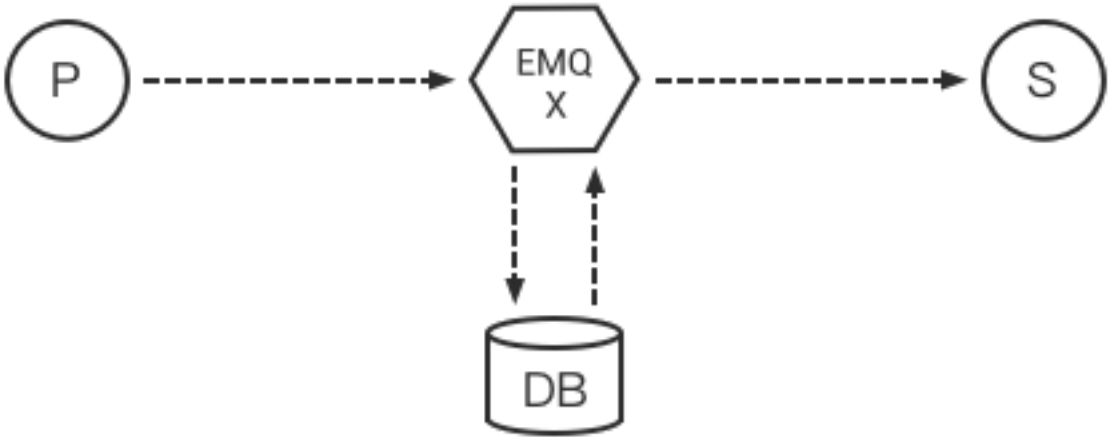
1.4 Subscription by Broker

EMQ X supports subscription by broker. A client doesn't need to expressly subscribe to some particular topics. The EMQ X broker will subscribe to this specified topics on behalf of the client. The topics are loaded from Redis or databases.

EMQ X subscription by broker is suitable for devices requiring low power consumption and narrow network bandwidth. This feature brings convenience to massive device management too.

1.5 MQTT Data Persistence

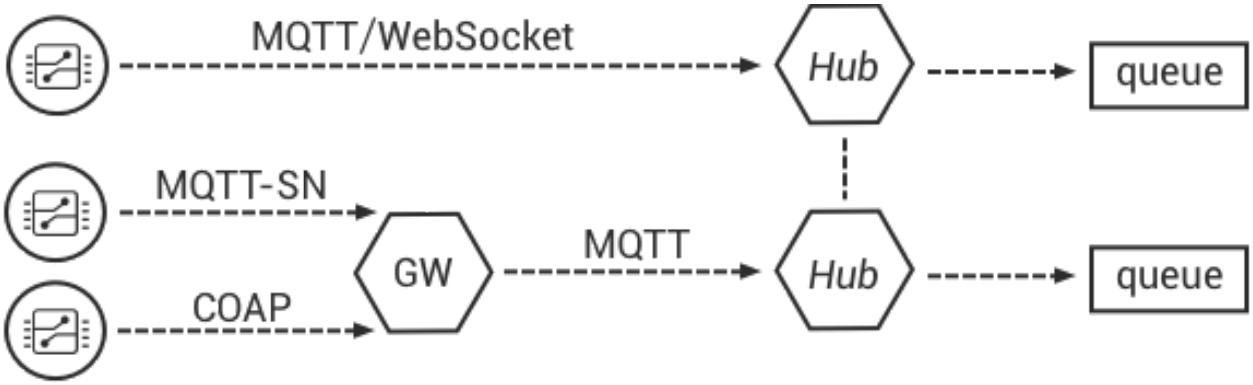
EMQ X supports MQTT data (subscription, messages, client online/offline status) persistence to Redis, MySQL, PostgreSQL, MongoDB and Cassandra databases:



For details please refer to the "Backends" chapter.

1.6 Message Bridge & Forward

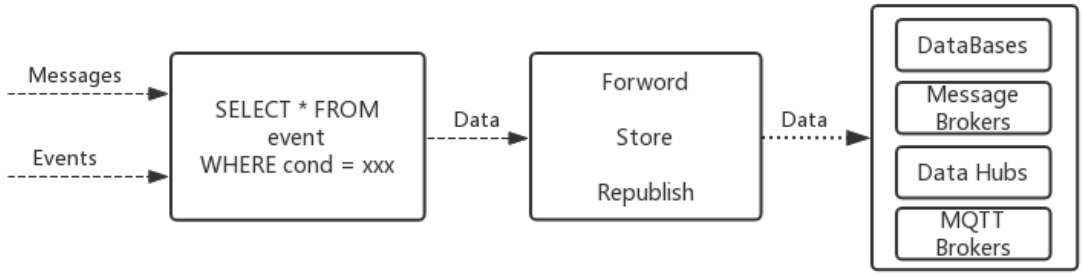
EMQ X allows bridging and forwarding MQTT messages to message-oriented middleware such as RabbitMQ and Kafka. It can be deployed as an IoT Hub:



1.7 Rule Engine

The EMQ X rules engine has the flexibility to handle messages and events.

1. Message Republish.
2. Bridges data to Kafka, Pulsar, RabbitMQ, MQTT Broker.
3. Persistence data to MySQL, PostgreSQL, Redis, MongoDB, DynamoDB, Cassandra, InfluxDB, OpenTSDB, TimescaleDB.
4. Sends data to WebServer.



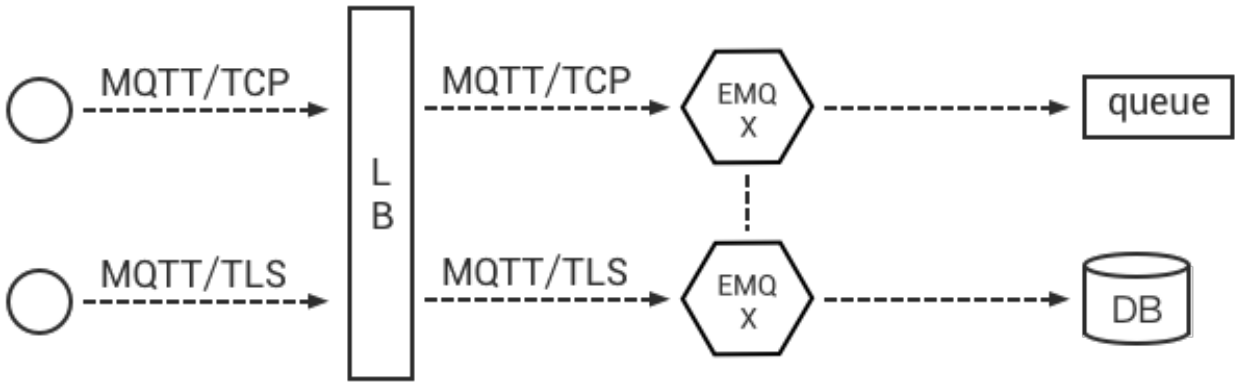
For details please refer to the "Rule Engine" chapter.

CHAPTER 2

Deployment

EMQ X Cluster can be deployed as an IoT Hub on an enterprise's private cloud, or on public clouds such as AWS, Azure and QingCloud.

Typical deployment architecture:



2.1 Load Balancer (LB)

The Load Balancer (LB) distributes MQTT connections and traffic from devices across the *EMQ X* clusters. LB enhances the HA of the clusters, balances the loads among the cluster nodes and makes the dynamic expansion possible.

It is recommended that SSL connections are terminated by a LB. The links between devices and the LB are secured by SSL, while the links between the LB and *EMQ X* cluster nodes are plain TCP connections. By this setup, a single *EMQ X* cluster can serve a million devices.

LB products of public cloud providers:

Cloud provider	SSL Termination	LB Product DOC/URL
QingCloud	Y	https://docs.qingcloud.com/guide/loadbalancer.html
AWS	Y	https://aws.amazon.com/cn/elasticloadbalancing/
aliyun	N	https://www.aliyun.com/product/slb

LBs for Private Cloud:

Open-Source LB	SSL Termination	DOC/URL
HAProxy	Y	https://www.haproxy.com/solutions/load-balancing.html
NGINX	PLUS Edition	https://www.nginx.com/solutions/load-balancing/

Recommend AWS with ELB for a public cloud deployment, and HAProxy for a private cloud deployment.

2.2 EMQ X Cluster

EMQ X cluster nodes are deployed behind LB. It is suggested that the nodes are deployed on VPCs or on a private network. Cloud provider, such as AWS, Azure or QingCloud, usually provides VPC network.

EMQ X Provides the MQTT service on following TCP ports by default:

1883	MQTT
8883	MQTT/SSL
8083	MQTT/WebSocket
8084	MQTT/WebSocket/SSL
8080	Management API
18083	Dashboard

Firewall should make the relevant ports accessible for public.

Following ports are opened for cluster internal communication:

4369	Node discovery port
5369	Cluster PRC
6369	Cluster channel

If deployed between nodes, firewalls should be configured that the above ports are inter-accessible between the nodes.

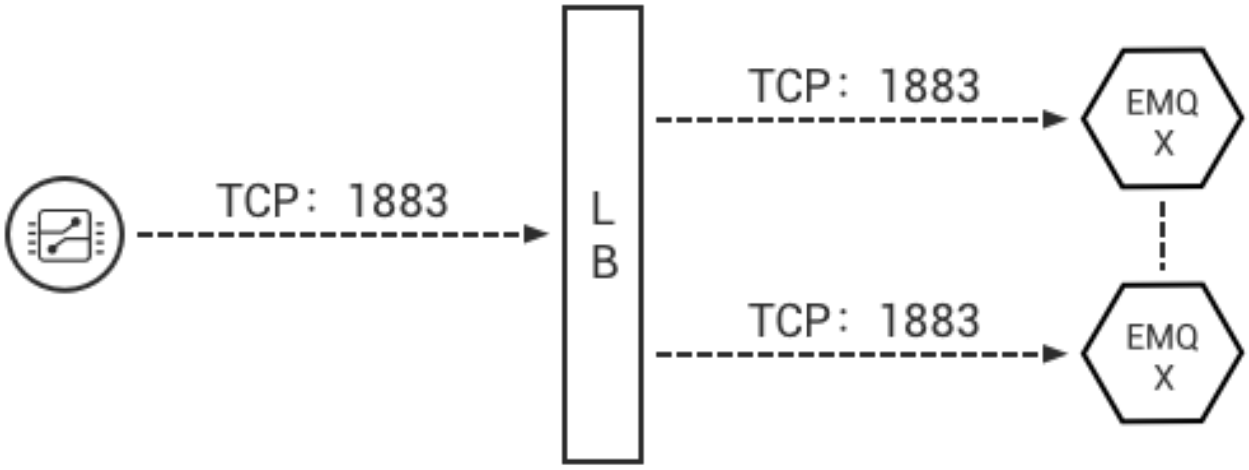
2.3 Deploying on QingCloud

1. Create VPC network.
2. Create a 'private network' for *EMQ X* cluster inside the VPC network, e.g. 192.168.0.0/24

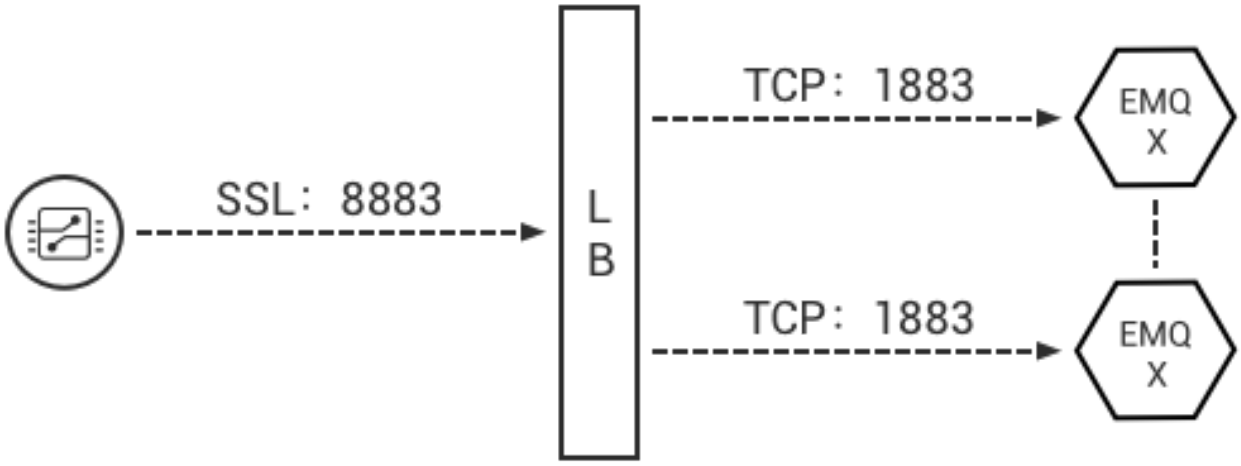
3. Create 2 *EMQ X* hosts inside the private network, like:

emqx1	192.168.0.2
emqx2	192.168.0.3

4. Install and cluster *EMQ X* on these two hosts. Please refer to the sections of cluster installation for details.
5. Create LB and assign the public IP address.
6. Create MQTT TCP listener:



Or create SSL listener and terminate the SSL connections on LB:



7. Connect the MQTT clients to the LB using the public IP address and test the deployment.

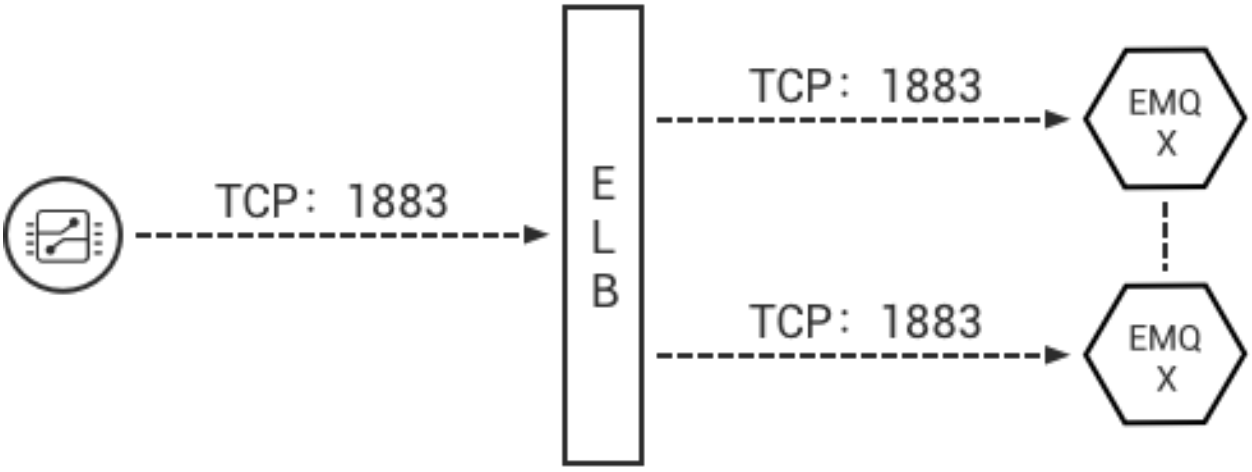
2.4 Deploying on AWS

1. Create VPC network.
2. Create a 'private network' for *EMQ X* cluster inside the VPC network, e.g. 192.168.0.0/24

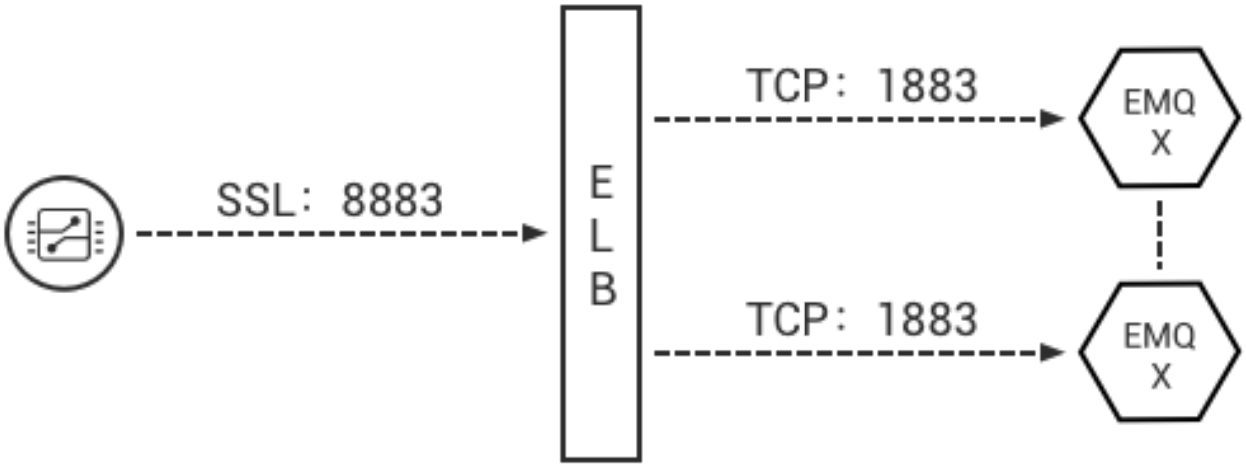
3. Create 2 hosts inside the private network, like:

emqx1	192.168.0.2
emqx2	192.168.0.3

4. Open the TCP ports for MQTT services (e.g. 1883,8883) on the security group.
5. Install and cluster *EMQ X* on these two hosts. Please refer to the sections of cluster installation for details.
6. Create ELB (Classic Load Balancer), assign the VPC network, and assign the public IP address.
7. Create MQTT TCP listener on the ELB:



Or create SSL listener and terminate the SSL connections on the ELB:



8. Connect the MQTT clients to the ELB using the public IP address and test the deployment.

2.5 Deploying on private network

2.5.1 Direct connection of EMQ X cluster

EMQ X cluster should be DNS-resolvable and the clients access the cluster via domain name or IP list:

1. Deploy *EMQ X* cluster. Please refer to the sections of 'Installation' and '*EMQ X* nodes clustering' for details.
2. Enable the access to the MQTT ports on the firewall (e.g. 1883, 8883).
3. Client devices access the *EMQ X* cluster via domain name or IP list.

: This kind of deployment is NOT recommended.

2.5.2 HAProxy -> EMQ X Cluster

HAProxy serves as a LB for *EMQ X* cluster and terminates the SSL connections:

1. Create *EMQ X* Cluster nodes like following:

node	IP
emqx1	192.168.0.2
emqx2	192.168.0.3

2. Modify the /etc/haproxy/haproxy.cfg accordingly. An example:

```
listen mqtt-ssl
    bind *:8883 ssl crt /etc/ssl/emqx/emqx.pem no-sslv3
    mode tcp
    maxconn 50000
    timeout client 600s
    default_backend emqx_cluster

backend emqx_cluster
    mode tcp
    balance source
    timeout server 50s
    timeout check 5000
    server emqx1 192.168.0.2:1883 check inter 10000 fall 2 rise 5 weight␣
↪1
    server emqx2 192.168.0.3:1883 check inter 10000 fall 2 rise 5 weight␣
↪1
    source 0.0.0.0 usesrc clientip
```

2.5.3 NGINX Plus -> EMQ X Cluster

NGINX Plus serves as a LB for *EMQ X* cluster and terminates the SSL connections:

1. Install the NGINX Plus. An instruction for Ubuntu: https://cs.nginx.com/repo_setup
2. Create *EMQ X* cluster nodes like following:

node	IP
emqx1	192.168.0.2
emqx2	192.168.0.3

3. Modify the /etc/nginx/nginx.conf. An example:

```
stream {
    # Example configuration for TCP load balancing

    upstream stream_backend {
        zone tcp_servers 64k;
        hash $remote_addr;
        server 192.168.0.2:1883 max_fails=2 fail_timeout=30s;
        server 192.168.0.3:1883 max_fails=2 fail_timeout=30s;
    }

    server {
        listen 8883 ssl;
        status_zone tcp_server;
        proxy_pass stream_backend;
        proxy_buffer_size 4k;
        ssl_handshake_timeout 15s;
        ssl_certificate     /etc/emqx/certs/cert.pem;
        ssl_certificate_key /etc/emqx/certs/key.pem;
    }
}
```

CHAPTER 3

Installation

The *EMQ X* broker is cross-platform, it can be deployed on Linux, FreeBSD and Windows.

: It is recommended to deploy EMQ X on Linux for production environment.

3.1 Get free trial License files

Log in to https://emqx.io to get free trial License files

3.2 *EMQ X* Package Download

Each version of the EMQ X broker will release packages of CentOS, Ubuntu, Debian, openSUSE, FreeBSD, macOS, Windows platform and Docker images.

Download address: https://www.emqx.io/downloads#enterprise

3.3 CentOS

- CentOS6.X
- CentOS7.X

3.3.1 Install via Repository

1. Delete the old EMQ X

```
$ sudo yum remove emqx emqx-edge emqx-ee
```

2. Install the required dependencies

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

3. Set up a stable repository, taking the CentOS7 as an example.

```
$ sudo yum-config-manager --add-repo https://repos.emqx.io/emqx-ee/
↪redhat/centos/7/emqx-ee.repo
```

4. Install the latest version of EMQ X

```
$ sudo yum install emqx-ee
```

5. Install a specific version of EMQ X

   1. Query available version

```
$ yum list emqx --showduplicates | sort -r

emqx-ee.x86_64                       3.2.0-1.el7                                ␣
↪emqx-ee-stable
```

   2. Install a specific version based on the version string in the second column, such as 3.1.0

```
$ sudo yum install emqx-ee-3.2.0
```

6. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

7. Start EMQ X

   - Directly start

```
$ emqx start
emqx v3.2.0 is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

   - systemctl start

```
$ sudo systemctl start emqx
```

   - service start

```
$ sudo service emqx start
```

8. Configuration file path
    - Configuration file path: `/etc/emqx`
    - Log file path: `/var/log/emqx`
    - Data file path: ‘‘/var/lib/emqx‘

3.3.2 Install via rpm

1. Select the CentOS version via emqx.io and download the rpm package for the EMQ X version to be installed.
2. Install EMQ X

```
$ sudo rpm -ivh emqx-ee-centos7-v3.2.0.x86_64.rpm
```

3. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

4. Start EMQ X
    - Directly start

```
$ emqx start
emqx v3.2.0 is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

    - systemctl start

```
$ sudo systemctl start emqx
```

    - service start

```
$ sudo service emqx start
```

5. Configuration file path
    - Configuration file path: `/etc/emqx`
    - Log file path: `/var/log/emqx`
    - Data file path: ‘‘/var/lib/emqx‘

3.3.3 Install via zip Package

1. Select the CentOS version via emqx.io and download the zip package for the EMQ X version to be installed.
2. Unzip package

```
$ unzip emqx-centos7-v3.1.0.zip
```

3. Import License file

```
$ cp /path/to/emqx.lic /path/to/emqx/etc/emqx.lic
```

4. Start EMQ X

```
$ ./bin/emqx start
emqx v3.2.0 is started successfully!

$ ./bin/emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

3.4 Ubuntu

- Bionic 18.04 (LTS)
- Xenial 16.04 (LTS)
- Trusty 14.04 (LTS)
- Precise 12.04 (LTS)

3.4.1 Install via Repository

1. Delete the old EMQ X

```
$ sudo apt remove emqx emqx-edge emqx-ee
```

2. Install the required dependency

```
$ sudo apt update && sudo apt install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

3. Add the GPG key for EMQ X

```
$ curl -fsSL https://repos.emqx.io/gpg.pub | sudo apt-key add -
```

Validate key

```
$ sudo apt-key fingerprint 3E640D53

pub   rsa2048 2019-04-10 [SC]
    FC84 1BA6 3775 5CA8 487B  1E3C C0B4 0946 3E64 0D53
uid           [ unknown] emqx team <support@emqx.io>
```

4. Add the EMQ X repository.

```
$ sudo add-apt-repository \
    "deb [arch=amd64] https://repos.emqx.io/emqx-ee/deb/ubuntu/ \
    $(lsb_release -cs) \
    stable"
```

5. Update apt package index

```
$ sudo apt update
```

6. Install the latest version of EMQ X

```
$ sudo apt install emqx-ee
```

: In the case where multiple EMQ X repositories are enabled, and the apt install and apt update commands is not specified with a version number, the latest version of EMQ X is installed. This could be a problem for users with stability needs.

---

7. Install a specific version of EMQ X

    1. Query available version

    ```
    $ sudo apt-cache madison emqx-ee

    emqx-ee |      3.2.0 | https://repos.emqx.io/emqx-ee/deb/ubuntu␣
    ↪bionic/stable amd64 Packages
    ```

    2. Install a specific version using the version string from the second column, such as 3.2.0

    ```
    $ sudo apt install emqx-ee=3.2.0
    ```

8. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

9. Start EMQ X

    - Directly start

```
$ emqx start
emqx v3.2.0 is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

- systemctl start

```
$ sudo systemctl start emqx
```

- service start

```
$ sudo service emqx start
```

10. Configuration file path

- Configuration file path: `/etc/emqx`
- Log file path: `/var/log/emqx`
- Data file path: ‘‘/var/lib/emqx‘

3.4.2 Install via deb Package

1. Select the Ubuntu version via emqx.io and download the deb package for the EMQ X version to be installed.
2. Install EMQ X

```
$ sudo dpkg -i emqx-ee-ubuntu18.04-v3.2.0_amd64.deb
```

3. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

4. Start EMQ X

- Directly start

```
$ emqx start
emqx  is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

- systemctl start

```
$ sudo systemctl start emqx
```

- service start

```
$ sudo service emqx start
```

5. Configuration file path
   - Configuration file path: `/etc/emqx`
   - Log file path: `/var/log/emqx`
   - Data file path: ‘‘/var/lib/emqx‘

3.4.3 Install via zip Package

1. Select the Ubuntu version via emqx.io and download the zip package for the EMQ X version to be installed.
2. Unzip the package

```
$ unzip emqx-ee-ubuntu18.04-v3.2.0.zip
```

3. Import License file

```
$ cp /path/to/emqx.lic /path/to/emqx/etc/emqx.lic
```

4. Start EMQ X

```
$ ./bin/emqx start
emqx v3.2.0 is started successfully!

$ ./bin/emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

3.5 Debian

- Stretch (Debian 9)
- Jessie (Debian 8)

3.5.1 Install via Repository

1. Delete the old EMQ X

```
$ sudo apt remove emqx emqx-edge emqx-ee
```

2. Install the required dependency

```
$ sudo apt update && sudo apt install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

3. Add the GPG key for EMQ X

```
$ curl -fsSL https://repos.emqx.io/gpg.pub | sudo apt-key add -
```

Validate the key

```
$ sudo apt-key fingerprint 3E640D53

pub   rsa2048 2019-04-10 [SC]
    FC84 1BA6 3775 5CA8 487B  1E3C C0B4 0946 3E64 0D53
uid           [ unknown] emqx team <support@emqx.io>
```

4. Add the EMQ X repository.

```
$ sudo add-apt-repository \
    "deb [arch=amd64] https://repos.emqx.io/emqx-ee/deb/debian/ \
    $(lsb_release -cs) \
    stable"
```

5. Update apt package index

```
$ sudo apt update
```

6. Install the latest version of EMQ X

```
$ sudo apt install emqx-ee
```

: In the case where multiple EMQ X repositories are enabled, and the apt install and apt update commands is not specified with a version number, the latest version of EMQ X is installed. This is a problem for users with stability needs.

---

7. Install a specific version of EMQ X

    1. Query available version

```
$ sudo apt-cache madison emqx

emqx-ee |       3.2.0 | https://repos.emqx.io/emqx-ee/deb/ubuntu␣
↪bionic/stable amd64 Packages
```

    2. Install a specific version using the version string from the second column, such as 3.2.0

```
$ sudo apt install emqx-ee=3.2.0
```

8. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

9. Start EMQ X

   - Directly start

```
$ emqx start
emqx v3.2.0 is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

   - systemctl start

```
$ sudo systemctl start emqx
```

   - service start

```
$ sudo service emqx start
```

10. Configuration file path

- Configuration file path: `/etc/emqx`
- Log file path: `/var/log/emqx`
- Data file path: ‘‘/var/lib/emqx‘

3.5.2 Install via deb Package

1. Select the Debian version via emqx.io and download the deb package for the EMQ X version to be installed.
2. Install EMQ X

```
$ sudo dpkg -i emqx-ee-debian9-v3.2.0_amd64.deb
```

3. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

4. Start EMQ X

   - Directly start

```
$ emqx start
emqx v3.2.0 is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

- systemctl start

```
$ sudo systemctl start emqx
```

- service start

```
$ sudo service emqx start
```

5. Configuration file path
   - Configuration file path: `/etc/emqx`
   - Log file path: `/var/log/emqx`
   - Data file path: ‘‘/var/lib/emqx‘

3.5.3 Install via zip Package

1. Select the Debian version via emqx.io and download the zip package for the EMQ X version to be installed.
2. Unzip the package

```
$ unzip emqx-ee-debian9-v3.2.0.zip
```

3. Import License file

```
$ cp /path/to/emqx.lic /path/to/emqx/etc/emqx.lic
```

4. Start EMQ X

```
$ ./bin/emqx start
emqx v3.2.0 is started successfully!

$ ./bin/emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

3.6 macOS

3.6.1 Install via zip Package

1. Select the EMQ X version via ‘emqx.io‘_and download the zip package to install.

2. Unzip the package

```
$ unzip emqx-macos-v3.1.0.zip
```

3. Import License file

```
$ cp /path/to/emqx.lic /path/to/emqx/etc/emqx.lic
```

4. Start EMQ X

```
$ ./bin/emqx start
emqx v3.2.0 is started successfully!

$ ./bin/emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

3.7 Windows

1. Select the Windows version via emqx.io and download the .zip package to install.
2. Unzip the package
3. Import License file

```
$ cp /path/to/emqx.lic /path/to/emqx/etc/emqx.lic
```

4. Open the Windows command line window, change the directory to the program directory, and start EMQ X.

```
cd emqx/bin
emqx start
```

3.8 openSUSE

- openSUSE leap

3.8.1 Install via Repository

1. Delete the old EMQ X

```
$ sudo zypper remove emqx emqx-edge emqx-ee
```

2. Download the GPG public key and import it.

```
$ curl -L -o /tmp/gpg.pub https://repos.emqx.io/gpg.pub
$ sudo rpmkeys --import /tmp/gpg.pub
```

3. Add repository address

```
$ sudo zypper ar -f -c https://repos.emqx.io/emqx-ee/redhat/opensuse/
↪leap/stable emqx-ee
```

4. Install the latest version of EMQ X

```
$ sudo zypper in emqx-ee
```

5. Install a specific version of EMQ X

    1. Query available version

```
$ sudo zypper pa emqx

Loading repository data...
Reading installed packages...
S | Repository | Name | Version  | Arch
--+------------+------+----------+-------
  | emqx-ee    | emqx-ee | 3.2.0-1 | x86_64
```

    2. Use Version column to install a specific version, such as 3.1.0

```
$ sudo zypper in emqx-ee-3.2.0
```

6. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

7. Start EMQ X

    - Directly start

```
$ emqx start
emqx v3.2.0 is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

    - systemctl start

```
$ sudo systemctl start emqx
```

    - service start

```
$ sudo service emqx start
```

8. Configuration file path

    - Configuration file path: `/etc/emqx`
    - Log file path: `/var/log/emqx`
    - Data file path: “/var/lib/emqx‘

3.8.2 Install via rpm Package

1. Select openSUSE via emqx.io and download the rpm package for the EMQ X version to be installed.
2. Install EMQ X and change the path below to the path where you downloaded the EMQ X package.

```
$ sudo rpm -ivh emqx-ee-opensuse-v3.2.0.x86_64.rpm
```

3. Import License file

```
$ cp /path/to/emqx.lic /etc/emqx/emqx.lic
```

4. Start EMQ X
   - Directly start

```
$ emqx start
emqx v3.2.0 is started successfully!

$ emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is runnin
```

   - systemctl start

```
$ sudo systemctl start emqx
```

   - service start

```
$ sudo service emqx start
```

5. Configuration file path
   - Configuration file path: `/etc/emqx`
   - Log file path: `/var/log/emqx`
   - Data file path: ‘‘/var/lib/emqx‘

3.8.3 Install via zip Package

1. Select openSUSE via emqx.io and download the zip package for the EMQ X version to be installed.
2. Unzip the package

```
$ unzip emqx-ee-opensuse-v3.2.0.zip
```

3. Import License file

```
$ cp /path/to/emqx.lic /path/to/emqx/etc/emqx.lic
```

4. Start EMQ X

```
$ ./bin/emqx start
emqx v3.2.0 is started successfully!

$ ./bin/emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

3.9 FreeBSD

- FreeBSD 12

3.9.1 Install via zip Package

1. Select FreeBSD via emqx.io and download the zip package for the EMQ X version to be installed.
2. Unzip the package

```
$ unzip emqx-ee-freebsd12-v3.2.0.zip
```

3. Import License file

```
$ cp /path/to/emqx.lic /path/to/emqx/etc/emqx.lic
```

4. Start EMQ X

```
$ ./bin/emqx start
emqx v3.2.0 is started successfully!

$ ./bin/emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx 3.2.0 is running
```

3.10 Docker

1. Get docker image
   - Through Docker Hub

```
$ docker pull emqx/emqx-ee:v3.2.0
```

   - Download the docker image via emqx.io and load it manually

```
$ wget -O emqx-ee-docker-v3.2.0.zip https://www.emqx.io/
↪downloads/enterprise/v3.2.0/emqx-ee-docker-v3.2.0-amd64.zip
$ unzip emqx-ee-docker.zip
$ docker load < emqx-ee-docker-v3.2.0
```

2. Start the docker container

```
$ docker run -d -\
    -name emqx-ee \
    -p 1883:1883 \
    -p 8083:8083 \
    -p 8883:8883 \
    -p 8084:8084 \
    -p 18083:18083 \
    -v /path/to/emqx.lic:/opt/emqx/etc/emqx.lic
    emqx/emqx-ee:v3.2.0
```

For more information about EMQ X Docker, please check Docker Hub .

CHAPTER 4

User Guide

Suppose an EMQ X Cluster with two Linux nodes deployed on a cloud VPC network or a private network:

Node name	IP
emqx1@192.168.0.10	192.168.0.10
emqx@192.168.0.20	192.168.0.20

4.1 System Parameters

Deployed under Linux, EMQ X allows 100k concurrent connections by default. To achieve this, the system Kernel, Networking, the Erlang VM and EMQ X itself must be tuned.

4.2 System-Wide File Handles

Maximum file handles:

```
# 2 millions system-wide
sysctl -w fs.file-max=262144
sysctl -w fs.nr_open=262144
echo 262144 > /proc/sys/fs/nr_open
```

Maximum file handles for current session:

```
ulimit -n 262144
```

4.3 /etc/sysctl.conf

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

```
fs.file-max = 262144
```

4.4 /etc/security/limits.conf

Persist the maximum number of opened file handles for users in /etc/security/limits.conf:

```
emqx      soft   nofile      262144
emqx      hard   nofile      262144
```

Note: Under Ubuntu, '/etc/systemd/system.conf' is to be modified:

```
DefaultLimitNOFILE=262144
```

4.5 EMQ X Node Name

Set the node name and cookies(required by communication between nodes)

'/etc/emqx/emqx.conf' on emqx1:

```
node.name    = emqx1@192.168.0.10
node.cookie = secret_dist_cookie
```

'/etc/emqx/emqx.conf' on emqx2:

```
node.name    = emqx2@192.168.0.20
node.cookie = secret_dist_cookie
```

4.6 Start EMQ X Broker

The EMQ X Broker can be downloaded at: https://www.emqx.io/downloads#enterprise

After downloading the package, it can be installed or unzipped directly to start running. Taking the zip package for MacOS platform as an example:

```
unzip emqx-ee-macosx-v3.2.0.zip && cd emqx

# Start emqx
./bin/emqx start

# Check running status
./bin/emqx_ctl status
```

4.7 Clustering the EMQ X Nodes

Start the two nodes, then on the emqx1@192.168.0.10 run:

```
$ ./bin/emqx_ctl cluster join emqx2@192.168.0.20

Join the cluster successfully.
Cluster status: [{running_nodes,['emqx1@192.168.0.10','emqx@192.168.0.20']}]
```

or, on the emqx1@192.168.0.20 run:

```
$ ./bin/emqx_ctl cluster join emqx1@192.168.0.10

Join the cluster successfully.
Cluster status: [{running_nodes,['emqx1@192.168.0.10','emqx@192.168.0.20']}]
```

Check the cluster status on any node:

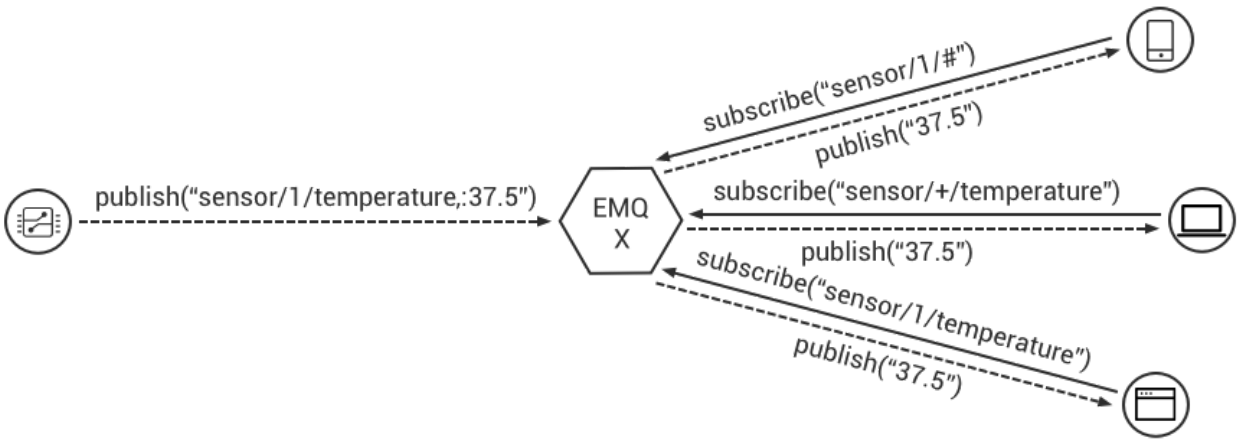
```
$ ./bin/emqx_ctl cluster status

Cluster status: [{running_nodes,['emqx1@192.168.0.10','emqx@192.168.0.20']}]
```

The default TCP ports used by the EMQ X message server include:

1883	MQTT protocol port
8883	MQTT/SSL port
8083	MQTT/WebSocket port
8080	HTTP API port
18083	Dashboard Management Console Port

4.8 MQTT publish and subscription

EMQ X Broker is a lightweight publish-subscribe message broker designed for the mobile Internet and the IoT, it currently supports MQTT v3.1.1 <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>‘_ and v5.0:



After EMQ X is started, devices and clients can connect to the broker through the MQTT protocol and exchange messages via Publish/Subscribe.

MQTT client library: https://github.com/mqtt/mqtt.github.io/wiki/libraries

E.g., using mosquitto_sub/pub CLI to subscribe to topics and publish messages.

```
mosquitto_sub -h 127.0.0.1 -p 1883 -t topic -q 2
mosquitto_pub -h 127.0.0.1 -p 1883 -t topic -q 1 -m "Hello, MQTT!"
```

4.9 Authentication and Access Control

EMQ X Broker provides **Connection Authentication** and **Access Control** using a series of authentication plugins, whose name conforms to the pattern of emqx_auth_<name>.

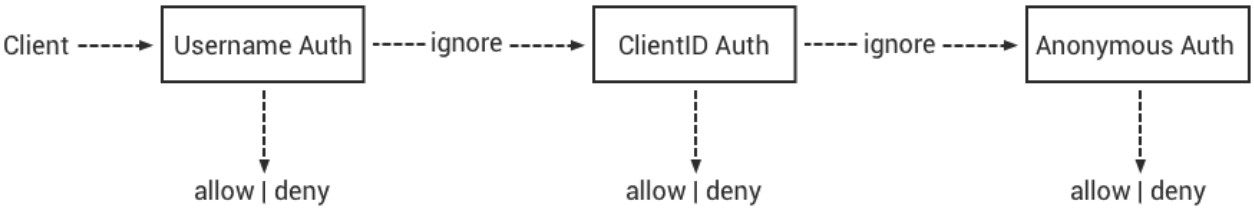
In EMQ X, these two functions are:

1. Connection authentication: EMQ X verifies whether the client on each connection has access to the system. If not, it disconnects the connection
2. Access Control: EMQ X verifies the permissions of each Publish/Subscribe action of a client, and allows/denies the corresponding action

4.9.1 Authentication

EMQ X Message Broker's Authentication is provided by a series of authentication plugins. It supports authentication by username, password, ClientID or anonymous.

By default, anonymous authentication is enabled. Multiple authentication modules can be started by loading the corresponding authentication plug-ins and forming an authentication chain:



** Start anonymous authentication**

Modify the `etc/emqx.conf` file to enable anonymous authentication:

```
## Allow anonymous access
## Value: true | false
allow_anonymous = true
```

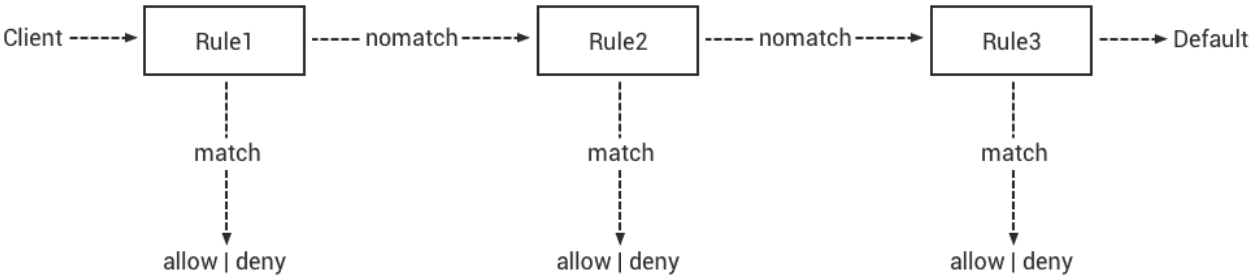
4.9.2 Access Control List

EMQ X Broker implements MQTT client access control through an ACL (Access Control List).

ACL access control rule definition:

```
Allow|Deny Identity Subscribe|Publish Topics
```

When an MQTT client initiates a subscribe/publish request, the access control module of EMQ X Broker will match the ACL rules one by one until the match is successful:



Default access control settings

EMQ X Message Broker's default access control can be set in file `etc/emqx.conf`:

```
## Set whether to allow access when all ACL rules cannot match
## Value: allow | deny
acl_nomatch = allow

## Set the default file for storing ACL rules
## Value: File Name
acl_file = etc/acl.conf
```

The ACL rules are defined in file `etc/acl.conf`, which is loaded into memory when EMQ X starts:

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

%% Allows local user to publish and subscribe to all topics
{allow, {ipaddr, "127.0.0.1"}, pubsub, ["$SYS/#", "#"]}.

%% Deny all the users to subscribe to '$SYS/#' and '#' topics except local␣
↪users
{deny, all, subscribe, ["$SYS/#", {eq, "#"}]}.

%% Allows any situation other than the above rules
{allow, all}.
```

The authentication plugins provided by EMQ X include:

plugins	description
emqx_auth_clientid	ClientId authentication plugin
emqx_auth_username	username and password authentication plugin
emqx_auth_jwt	JWT authentication plugin
emqx_auth_ldap	LDAP authentication plugin
emqx_auth_http	HTTP authentication plugin
emqx_auth_mysql	MySQ Lauthentication plugin
emqx_auth_pgsql	Postgre authentication plugin
emqx_auth_redis	Redis authentication plugin
emqx_auth_mongo	MongoDB authentication plugin

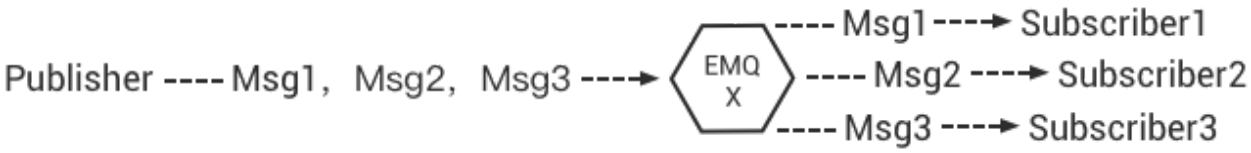
For the configuration and usage of each authentication plug-in, please refer to authentication section of the *Plugins*.

: Multiple auth plug-ins can be started at the same time. The plug-in that starts first checks first.

In addition, EMQ X also supports the use of PSK (Pre-shared Key) for authentication. However, the authentication chain mentioned above is not used in this case. The verification is done during the SSL handshake. For details please refer to Pre-shared Key and emqx_psk_file

4.10 Shared Subscription

The EMQ X R3.0 supports cluster-level shared subscriptions that supports multiple message delivery strategies:



Shared subscriptions support two usage methods:

Subscription prefix	Example
$queue/	mosquitto_sub -t '$queue/topic'
$share/<group>/	mosquitto_sub -t '$share/group/topic'

Example:

```
mosquitto_sub -t '$share/group/topic'

mosquitto_pub -t 'topic' -m msg -q 2
```

The dispatch strategy for shared messages can be configured by the `broker.shared_subscription_strategy` field in the `etc/emqx.conf`

The following strategies are supported by EMQ X to distribute messages:

Strategy	Description
random	Random among all shared subscribers
round_robin	According to subscription order
sticky	The last dispatched subscriber is picked
hash	Hash value of the ClientId of publisher

: When all subscribers are offline, a subscriber will still be picked and stored in the message queue of its Session.

4.11 Bridge

4.11.1 Bridging two EMQ X Nodes

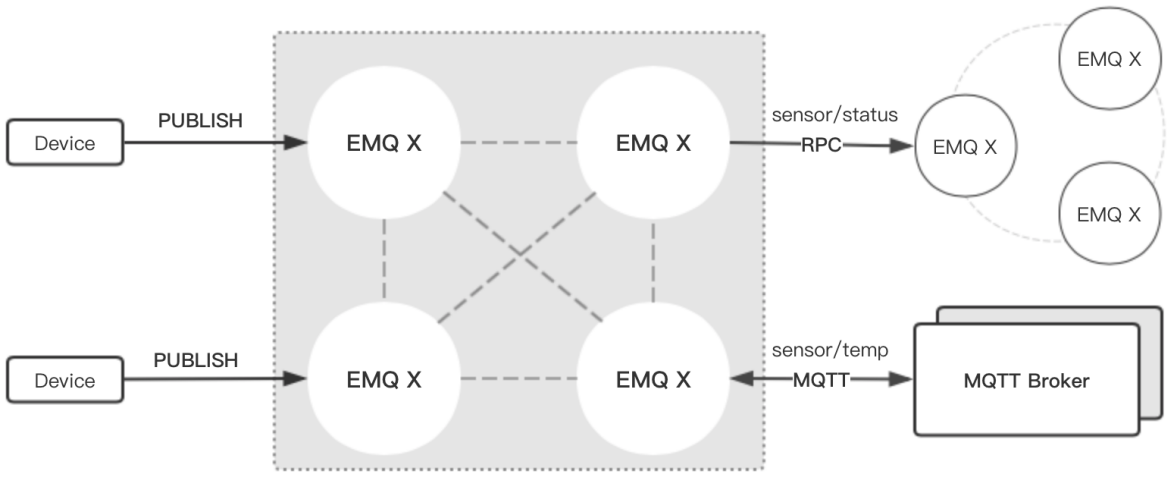
The concept of bridging is that EMQ X forwards messages of some of its topics to another MQTT Broker in some way.

Difference between Bridge and cluster is that bridge does not replicate topic trees and routing tables, a bridge only forwards MQTT messages based on bridging rules.

Currently the bridging methods supported by EMQ X are as follows:

- RPC bridge: RPC Bridge only supports message forwarding and does not support subscribing to the topic of remote nodes to synchronize data.
- MQTT Bridge: MQTT Bridge supports both forwarding and data synchronization through subscription topic

The concept is shown below:



In addition, the EMQ X supports multi-node bridge mode interconnection:

In EMQ X, bridge is configured by modifying `etc/plugins/emqx_bridge_mqtt.conf`. EMQ X distinguishes between different bridges based on different names. E.g:

```
## Bridge address: node name for local bridge, host:port for remote.
bridge.mqtt.aws.address = 127.0.0.1:1883
```

This configuration declares a bridge named `aws` and specifies that it is bridged to the MQTT server of 127.0.0.1:1883 by MQTT mode.

In case of creating multiple bridges, it is convenient to replicate all configuration items of the first bridge, and modify the bridge name and other configuration items if necessary (such as bridge.mqtt.$name.address, where $name refers to the name of bridge)

The next two sections describe how to create a bridge in RPC and MQTT mode respectively and create a forwarding rule that forwards the messages from sensors. Assuming that two EMQ X nodes are running on two hosts:

Name	Node	MQTT port
emqx1	emqx1@192.168.1.1	1883
emqx2	emqx2@192.168.1.2	1883

4.12 EMQ X Node RPC Bridge Configuration

The following is the basic configuration of RPC bridging. The simplest RPC bridging only needs to configure the following three items:

```
## Bridge Address: Use node name (nodename@host) for rpc bridging, and␣
↪host:port for mqtt connection
bridge.mqtt.aws.address = emqx2@192.168.1.2

## Forwarding topics of the message
bridge.mqtt.aws.forwards = sensor1/#,sensor2/#

## bridged mountpoint
bridge.mqtt.aws.mountpoint = bridge/emqx2/${node}/
```

If the message received by the local emqx1 node matches the topic `sersor1/#` or `sensor2/#`, these messages will be forwarded to the `sensor1/#` or `sensor2/#` topic of the remote emqx2 node.

`forwards` is used to specify topics. Messages of the in `forwards` specified topics on local node are forwarded to the remote node.

`mountpoint` is used to add a topic prefix when forwarding a message. To use `mountpoint`, the `forwards` directive must be set. In the above example, a message with the topic sensor1/hello received by the local node will be forwarded to the remote node with the topic `bridge/emqx2/emqx1@192.168.1.1/sensor1/hello`.

Limitations of RPC bridging:

1. The RPC bridge of emqx can only forward local messages to the remote bridge node, and cannot synchronize the messages of the remote bridge node to the local node;
2. RPC bridge can only bridge two EMQ X together and cannot bridge EMQ X to other mqtt brokers.

4.12.1 EMQ X Node MQTT Bridge Configuration

EMQ X 3.0 officially introduced MQTT bridge, so that EMQ X can bridge any MQTT broker. Because of the characteristics of the MQTT protocol, EMQ X can subscribe to the remote mqtt broker's topic through MQTT bridge, and then synchronize the remote MQTT broker's message to the local.

EMQ X MQTT bridging principle: Create an MQTT client on the EMQ X broker, and connect this MQTT client to the remote MQTT broker. Therefore, in the MQTT bridge configuration, following fields may be set for the EMQ X to connect to the remote broker as an mqtt client:

```
## Bridge Address: Use node name for rpc bridging, use host:port for mqtt␣
↪connection
bridge.mqtt.aws.address = 192.168.1.2:1883

## Bridged Protocol Version
## Enumeration value: mqttv3 | mqttv4 | mqttv5
bridge.mqtt.aws.proto_ver = mqttv4

## mqtt client's client_id
bridge.mqtt.aws.client_id = bridge_emq

## mqtt client's clean_start field
## Note: Some MQTT Brokers need to set the clean_start value as `true`
bridge.mqtt.aws.clean_start = true

##  mqtt client's username field
bridge.mqtt.aws.username = user

## mqtt client's password field
bridge.mqtt.aws.password = passwd

## Whether the mqtt client uses ssl to connect to a remote serve or not
bridge.mqtt.aws.ssl = off

## CA Certificate of Client SSL Connection (PEM format)
bridge.mqtt.aws.cacertfile = etc/certs/cacert.pem

## SSL certificate of Client SSL connection
bridge.mqtt.aws.certfile = etc/certs/client-cert.pem
```

()

()

```
## Key file of Client SSL connection
bridge.mqtt.aws.keyfile = etc/certs/client-key.pem

## SSL encryption
bridge.mqtt.aws.ciphers = ECDHE-ECDSA-AES256-GCM-SHA384,ECDHE-RSA-AES256-GCM-
↪SHA384

## TTLS PSK password
## Note 'listener.ssl.external.ciphers' and 'listener.ssl.external.psk_ciphers
↪' cannot be configured at the same time
##
## See 'https://tools.ietf.org/html/rfc4279#section-2'.
## bridge.mqtt.aws.psk_ciphers = PSK-AES128-CBC-SHA,PSK-AES256-CBC-SHA,PSK-
↪3DES-EDE-CBC-SHA,PSK-RC4-SHA

## Client's heartbeat interval
bridge.mqtt.aws.keepalive = 60s

## Supported TLS version
bridge.mqtt.aws.tls_versions = tlsv1.2,tlsv1.1,tlsv1

## Forwarding topics of the message
bridge.mqtt.aws.forwards = sensor1/#,sensor2/#

## Bridged mountpoint
bridge.mqtt.aws.mountpoint = bridge/emqx2/${node}/

## Subscription topic for bridging
bridge.mqtt.aws.subscription.1.topic = cmd/topic1

## Subscription qos for bridging
bridge.mqtt.aws.subscription.1.qos = 1

## Subscription topic for bridging
bridge.mqtt.aws.subscription.2.topic = cmd/topic2

## Subscription qos for bridging
bridge.mqtt.aws.subscription.2.qos = 1

## Bridging reconnection interval
## Default: 30s
bridge.mqtt.aws.reconnect_interval = 30s

## QoS1 message retransmission interval
bridge.mqtt.aws.retry_interval = 20s

## Inflight Size.
bridge.mqtt.aws.max_inflight_batches = 32
```

4.13 EMQ X Bridge Cache Configuration

The bridge of EMQ X has a message caching mechanism. The caching mechanism is applicable to both RPC bridging and MQTT bridging. When the bridge is disconnected (such as when the network connection is unstable), the messages with a topic specified in `forwards` can be cached to the local message queue. Until the bridge is restored, these messages are re-forwarded to the remote node. The configuration of the cache queue is as follows:

```
## emqx_bridge internal number of messages used for batch
bridge.mqtt.aws.queue.batch_count_limit = 32

##  emqx_bridge internal number of message bytes used for batch
bridge.mqtt.aws.queue.batch_bytes_limit = 1000MB

## The path for placing replayq queue. If the item is not specified in the␣
↪configuration, then replayq will run in `mem-only` mode and messages will␣
↪not be cached on disk.
bridge.mqtt.aws.queue.replayq_dir = data/emqx_emqx2_bridge/

## Replayq data segment size
bridge.mqtt.aws.queue.replayq_seg_bytes = 10MB
```

`bridge.emqx2.queue.replayq_dir` is a configuration parameter for specifying the path of the bridge storage queue.

`bridge.mqtt.aws.queue.replayq_seg_bytes` is used to specify the size of the largest single file of the message queue that is cached on disk. If the message queue size exceeds the specified value, a new file is created to store the message queue.

4.14 CLI for EMQ X Bridge

CLI for EMQ X Bridge:

```
$ cd emqx1/ && ./bin/emqx_ctl bridges
bridges list                                    # List bridges
bridges start <Name>                            # Start a bridge
bridges stop <Name>                             # Stop a bridge
bridges forwards <Name>                         # Show a bridge forward topic
bridges add-forward <Name> <Topic>              # Add bridge forward topic
bridges del-forward <Name> <Topic>              # Delete bridge forward topic
bridges subscriptions <Name>                    # Show a bridge subscriptions␣
↪topic
bridges add-subscription <Name> <Topic> <Qos>   # Add bridge subscriptions␣
↪topic
```

List all bridge states

```
$ ./bin/emqx_ctl bridges list
name: emqx     status: Stopped
```

Start the specified bridge

```
$ ./bin/emqx_ctl bridges start emqx
Start bridge successfully.
```

Stop the specified bridge

```
$ ./bin/emqx_ctl bridges stop emqx
Stop bridge successfully.
```

List the forwarding topics for the specified bridge

```
$ ./bin/emqx_ctl bridges forwards emqx
topic:   topic1/#
topic:   topic2/#
```

Add a forwarding topic for the specified bridge

```
$ ./bin/emqx_ctl bridges add-forwards emqx topic3/#
Add-forward topic successfully.
```

Delete the forwarding topic for the specified bridge

```
$ ./bin/emqx_ctl bridges del-forwards emqx topic3/#
Del-forward topic successfully.
```

List subscriptions for the specified bridge

```
$ ./bin/emqx_ctl bridges subscriptions emqx
topic: cmd/topic1, qos: 1
topic: cmd/topic2, qos: 1
```

Add a subscription topic for the specified bridge

```
$ ./bin/emqx_ctl bridges add-subscription emqx cmd/topic3 1
Add-subscription topic successfully.
```

Delete the subscription topic for the specified bridge

```
$ ./bin/emqx_ctl bridges del-subscription emqx cmd/topic3
Del-subscription topic successfully.
```

Note: In case of creating multiple bridges, it is convenient to replicate all configuration items of the first bridge, and modify the bridge name and other configuration items if necessary.

4.15 HTTP Publish API

The EMQ X message server provides an HTTP publish interface through which an application server or web server can publish MQTT messages:

```
HTTP POST http://host:8080/api/v3/mqtt/publish
```

Web servers such as PHP/Java/Python/NodeJS or Ruby on Rails can publish MQTT messages via HTTP POST requests:

```
curl -v --basic -u user:passwd -H "Content-Type: application/json" -d \
'{"qos":1, "retain": false, "topic":"world", "payload":"test" , "client_id":
↪"C_1492145414740"}' \-k http://localhost:8080/api/v3/mqtt/publish
```

HTTP interface parameters:

parameter	description
client_id	MQTT client ID
qos	QoS: 0 | 1 | 2
retain	Retain: true | false
topic	Topic
payload	message payload

: HTTP publishing interface uses authentication of Basic . The user and password in the above example are from the AppId and password in the Applications of Dashboard.

4.16 MQTT WebSocket Connection

EMQ X also supports WebSocket connections, web browsers or applications can connect directly to the broker via WebSocket:

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

The Dashboard plugin provides a test tool for an MQTT WebSocket connection:

```
http://127.0.0.1:18083/#/websocket
```

4.17 $SYS - System topic

The EMQ X Broker periodically publishes its running status, message statistics, client online and offline events to the system topic starting with `$SYS/`.

The `$SYS` topic path begins with `$SYS/brokers/{node}/`. `{node}` is the name of the node where the event/message is generated, for example:

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

$SYS/brokers/emqx@127.0.0.1/uptime
```

: By default, only the MQTT client on localhost is allowed to subscribe to the $SYS topic, this can be changed by modifying the access control rules in file `etc/acl.config`.

$SYS system message publish interval is configured in `etc/emqx.conf`:

```
## System interval of publishing $SYS messages.
##
## Value: Duration
## Default: 1m, 1 minute
broker.sys_interval = 1m
```

4.17.1 Cluster status information

Topic	Description
$SYS/brokers	cluster node list
$SYS/brokers/${node}/version	EMQ X broker version
$SYS/brokers/${node}/uptime	EMQ X broker startup time
$SYS/brokers/${node}/datetime	EMQ X broker time
$SYS/brokers/${node}/sysdescr	EMQ X broker Description

4.17.2 Client Online and Offline Events

$SYS topic prefix: $SYS/brokers/${node}/clients/

Topic	Description
${clientid}/connected	Online event. This message is published when a client goes online.
${clientid}/disconnected	Offline event. This message is published when a client is offline

The Payload of the 'connected' event message can be parsed into JSON format:

```
{
    "clientid":"id1",
    "username":"u",
    "ipaddress":"127.0.0.1",
    "connack":0,
    "ts":1554047291,
    "proto_ver":3,
    "proto_name":"MQIsdp",
    "clean_start":true,
```

()

()

```
    "keepalive":60
}
```

The Payload of the 'disconnected' event message can be parsed into JSON format:

```
{
    "clientid":"id1",
    "username":"u",
    "reason":"normal",
    "ts":1554047291
}
```

4.17.3 Statistics

System topic prefix: $SYS/brokers/${node}/stats/

Client statistics

Topic	Description
connections/count	Total number of current clients
connections/max	Maximum number of clients

Session statistics

Topic	Description
sessions/count	Total number of current sessions
sessions/max	maximum number of sessions
sessions/persistent/count	Total number of persistent sessions
sessions/persistent/max	maximum number of persistent sessions

Subscription statistics

Topic	Description
suboptions/count	number of current subscription options
suboptions/max	total number of maximum subscription options
subscribers/max	total number of maximum subscribers
subscribers/count	number of current subscribers
subscriptions/max	maximum number of subscriptions
subscriptions/count	total number of current subscription
subscriptions/shared/count	total number of current shared subscriptions
subscriptions/shared/max	maximum number of shared subscriptions

Topic statistics

Topic	Description
topics/count	total number of current topics
topics/max	maximum number of topics

Routes statistics

Topic	Description
routes/count	total number of current Routes
routes/max	maximum number of Routes

: The topics/count and topics/max are numerically equal to routes/count and routes/max.

4.17.4 Throughput (bytes/packets/message) statistics

System Topic Prefix: $SYS/brokers/${node}/metrics/

sent and received bytes statistics

Topic	Description
bytes/received	Accumulated received bytes
bytes/sent	Accumulated sent bytes

sent and received MQTT packets statistics

Topic	Description
packets/received	Accumulative received MQTT packets
packets/sent	Accumulative sent MQTT packets
packets/connect	Accumulative received packets of MQTT CONNECT
packets/connack	Accumulative sent packets of MQTT CONNACK
packets/publish/received	Accumulative received packets of MQTT PUBLISH
packets/publish/sent	Accumulative sent packets of MQTT PUBLISH
packets/puback/received	Accumulative received packets of MQTT PUBACK
packets/puback/sent	Accumulative sent packets of MQTT PUBACK
packets/puback/missed	Accumulative missed packets of MQTT PUBACK
packets/pubrec/received	Accumulative received packets of MQTT PUBREC
packets/pubrec/sent	Accumulative sent packets of MQTT PUBREC
packets/pubrec/missed	Accumulative missed packets of MQTT PUBREC
packets/pubrel/received	Accumulative received packets of MQTT PUBREL
packets/pubrel/sent	Accumulative sent packets of MQTT PUBREL
packets/pubrel/missed	Accumulative missed packets of MQTT PUBREL
packets/pubcomp/received	Accumulative received packets of MQTT PUBCOMP
packets/pubcomp/sent	Accumulative sent packets of MQTT PUBCOMP
packets/pubcomp/missed	Accumulative missed packets of MQTT PUBCOMP
packets/subscribe	Accumulative received packets of MQTT SUBSCRIBE
packets/suback	Accumulative sent packets of MQTT SUBACK
packets/unsubscribe	Accumulative received packets of MQTT UNSUBSCRIBE
packets/unsuback	Accumulative sent packets of MQTT UNSUBACK
packets/pingreq	Accumulative received packets of MQTT PINGREQ
packets/pingresp	Accumulative sent packets of MQTT PINGRESP
packets/disconnect/received	Accumulative received packets of MQTT DISCONNECT
packets/disconnect/sent	Accumulative sent packets of MQTT MQTT DISCONNECT
packets/auth	Accumulative received packets of MQTT Auth

MQTT sent and received messages statistics

Topic	Description
messages/received	Accumulative received messages
messages/sent	Accumulative sent messages
messages/expired	Accumulative expired messages
messages/retained	Accumulative retained messages
messages/dropped	Total number of dropped messages
messages/forward	Total number of messages forwarded by the node
messages/qos0/received	Accumulative received messages of QoS0
messages/qos0/sent	Accumulative sent messages of QoS0
messages/qos1/received	Accumulative received messages QoS1
messages/qos1/sent	Accumulative sent messages QoS1
messages/qos2/received	Accumulative received messages of QoS2
messages/qos2/sent	Accumulative sent messages of QoS2
messages/qos2/expired	Total number of expired messages of QoS2
messages/qos2/dropped	Total number of dropped messages of QoS2

4.17.5 Alarms - system alarms

System Topic Prefix: $SYS/brokers/${node}/alarms/

Topic	Description
alert	newly generated alarm
clear	cleared alarm

4.17.6 Sysmon - system monitoring

System Topic Prefix: $SYS/brokers/${node}/sysmon/

Topic	Description
long_gc	GC Overtime alarm
long_schedule	Alarm for Excessive Scheduling Time
large_heap	ALarm for Heap Memory Occupancy
busy_port	Alarm for Port busy
busy_dist_port	Alarm for Dist Port busy

4.18 Trace

EMQ X message server supports tracing all messages from a client or published to a topic.

Trace messages from the client:

```
$ ./bin/emqx_ctl log primary-level debug

$ ./bin/emqx_ctl trace start client "clientid" "trace_clientid.log" debug
```

Trace messages published to a topic:

```
$ ./bin/emqx_ctl log primary-level debug

$ ./bin/emqx_ctl trace start topic "t/#" "trace_topic.log" debug
```

Query trace:

```
$ ./bin/emqx_ctl trace list
```

Stop trace:

```
$ ./bin/emqx_ctl trace stop client "clientid"

$ ./bin/emqx_ctl trace stop topic "topic"
```

CHAPTER 5

Configuration

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

File	Description
etc/emqx.conf	*EMQ X* 3.0 Configuration File
etc/acl.conf	The default ACL File
etc/plugins/*.conf	Config Files of Plugins

5.1 EMQ X Configuration Change History

The EMQ X configuration file has been adjusted four times for the convenience of users and plug-in developers.

1. The EMQ X 1.x version uses the Erlang native configuration file format etc/emqttd.config:

```
{emqttd, [
  %% Authentication and Authorization
  {access, [
    %% Authetication. Anonymous Default
    {auth, [
        %% Authentication with username, password
        %{username, []},

        %% Authentication with clientid
        %{clientid, [{password, no}, {file, "etc/clients.config"}]},
```

Erlang's native configuration format is multi-level nested, which is very unfriendly to non-Erlang developer users.

2. EMQ X 2.0-beta.x version simplifies the native Erlang configuration file in a format similar to rebar.config or relx.config:

```
%% 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
```

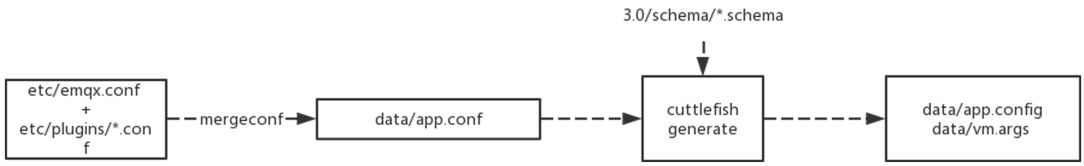
The simplified Erlang native configuration format is user-friendly, but plug-in developers have to rely on the gen_conf library instead of appliaton: get_env to read configuration parameters.

3. The EMQ X 2.0-rc.2 integrates the cuttlefish library, adopts the k = V universal format similar to sysctl, and translates it into Erlang native configuration format at system startup:

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

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

4. From EMQ X 3.0-beta.1 beta version, the emqttd is renamed to emqx, and configuration names and configuration values/information are changed in corresponding:

```
   ## Profile
   etc/emq.config  ==> etc/emqx.config

   ## Node name
   Original:
   node.name = emq@127.0.0.1
   Now:
   node.name = emqx@127.0.0.1


Configuration file processing flow during EMQ X start-up::
```



5.2 OS Environment Variables

EMQX_NODE_NAME	Erlang node name
EMQX_NODE_COOKIE	Cookie for distributed erlang node
EMQX_MAX_PORTS	Maximum number of opened sockets
EMQX_TCP_PORT	MQTT TCP Listener Port, Default: 1883
EMQX_SSL_PORT	MQTT SSL Listener Port, Default: 8883
EMQX_WS_PORT	MQTT/WebSocket Port, Default: 8083
EMQX_WSS_PORT	MQTT/WebSocket/SSL Port, Default: 8084

5.3 EMQ X Cluster Setup

Cluster name:

```
cluster.name = emqxcl
```

Cluster discovery strategy:

```
cluster.discovery = manual
```

Cluster Autoheal:

```
cluster.autoheal = on
```

Cluster Autoclean:

```
cluster.autoclean = 5m
```

5.3.1 EMQ X Autodiscovery Strategy

EMQ X 3.0 supports node discovery and autocluster with various strategies:

Strategy	Description
manual	Create cluster manually
static	Autocluster by static node list
mcast	Autocluster by UDP Multicast
dns	Autocluster by DNS A Record
etcd	Autocluster using etcd
k8s	Autocluster on Kubernetes

Create cluster manually

This is the default configuration of clustering, nodes join a cluster by executing *./bin/emqx_ctl join <Node>* CLI command:

```
cluster.discovery = manual
```

Autocluster by static node list

Configuration of static cluster discovery:

```
cluster.discovery = static
```

Static node list:

```
cluster.static.seeds = emqx1@127.0.0.1,emqx2@127.0.0.1
```

Autocluster by IP Multicast

Configuration of cluster discovery by IP multicast:

```
cluster.discovery = mcast
```

IP multicast address:

```
cluster.mcast.addr = 239.192.0.1
```

Multicast port range:

```
cluster.mcast.ports = 4369,4370
```

Network adapter address:

```
cluster.mcast.iface = 0.0.0.0
```

Multicast TTL:

```
cluster.mcast.ttl = 255
```

Whether to send multicast packets cyclically:

```
cluster.mcast.loop = on
```

Autocluster by DNS A Record

Configuration of cluster discovery by DNS A record:

```
cluster.discovery = dns
```

dns name:

```
cluster.dns.name = localhost
```

Application name is used to build the node name with the IP address:

```
cluster.dns.app  = emqx
```

Autocluster using etcd

Configure cluster discovery by etcd:

```
cluster.discovery = etcd
```

List of etcd servers, separated by , :

```
cluster.etcd.server = http://127.0.0.1:2379
```

The prefix of the node's path in etcd. Each node in the cluster creates the following path in etcd: v2/keys/<prefix>/<cluster.name>/<node.name>:

```
cluster.etcd.prefix = emqxcl
```

The TTL of the node in etcd:

```
cluster.etcd.node_ttl = 1m
```

Path containing the client's private PEM encoded key file:

```
cluster.etcd.ssl.keyfile = etc/certs/client-key.pem
```

Path containing the client certificate file:

```
cluster.etcd.ssl.certfile = etc/certs/client.pem
```

Path containing the PEM-encoded CA certificate file:

```
cluster.etcd.ssl.cacertfile = etc/certs/ca.pem
```

Autocluster on Kubernetes

Cluster discovery strategy is k8s:

```
cluster.discovery = k8s
```

List of Kubernetes API servers, separated by , :

```
cluster.k8s.apiserver = http://10.110.111.204:8080
```

The service name of the EMQ X node in the cluster:

```
cluster.k8s.service_name = emqx
```

Address type used to extract hosts from k8s services:

```
cluster.k8s.address_type = ip
```

EMQ X node name:

```
cluster.k8s.app_name = emqx
```

Kubernetes namespace:

```
cluster.k8s.namespace = default
```

5.4 EMQ X Node and Cookie

Erlang node name:

```
node.name = emqx@127.0.0.1
```

Erlang communication cookie within distributed nodes:

```
node.cookie = emqxsecretcookie
```

Erlang Maximum number of clients allowed by a node:

```
node.max_clients = 1024000
```

: The Erlang/OTP platform application is composed of distributed Erlang nodes (processes). Each Erlang node (process) needs to be assigned a node name for mutual communication between nodes. All Erlang nodes (processes) in communication are authenticated by a shared cookie.

5.5 EMQ X Node Connection Method

The EMQ X node is based on IPv4, IPv6 or TLS protocol of Erlang/OTP platform for connections:

```
##  Specify the Erlang Distributed Communication Protocol: inet_tcp | inet6_
↪tcp | inet_tls
node.proto_dist = inet_tcp

## Specify the parameter configuration of Erlang Distributed Communication SSL
## node.ssl_dist_optfile = etc/ssl_dist.conf
```

5.6 Erlang VM Parameters

Erlang system heartbeat monitoring during running. Comment this line to disable heartbeat monitoring, or set the value as `on` to enable the function :

```
node.heartbeat = on
```

The number of threads in the asynchronous thread pool, with the valid range: 0-1024:

```
node.async_threads = 32
```

The maximum number of processes allowed by the Erlang VM. An MQTT connection consumes 2 Erlang processes:

```
node.process_limit = 2048000
```

The maximum number of ports allowed by the Erlang VM. One MQTT connection consumes 1 Port:

```
node.max_ports = 1024000
```

Allocate buffer busy limit:

```
node.dist_buffer_size = 8MB
```

The maximum number of ETS tables. Note that mnesia and SSL will create a temporary ETS table:

```
node.max_ets_tables = 256000
```

GC frequency:

```
node.fullsweep_after = 1000
```

Crash dump log file location:

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

Specify the Erlang distributed protocol:

```
node.proto_dist = inet_tcp
```

Files for storing SSL/TLS options when Erlang distributed using TLS:

```
node.ssl_dist_optfile = etc/ssl_dist.conf
```

Tick time of distributed nodes:

```
node.dist_net_ticktime = 60
```

Port range of TCP connections for communication between Erlang distributed nodes:

```
node.dist_listen_min = 6396
node.dist_listen_max = 6396
```

5.7 RPC Parameter Configuration

RPC Mode (sync | async):

```
rpc.mode = async
```

Max batch size of async RPC requests:

```
rpc.async_batch_size = 256
```

TCP port for RPC (local):

```
rpc.tcp_server_port = 5369
```

TCP port for RPC(remote):

```
rpc.tcp_client_port = 5369
```

Number of outgoing RPC connections.

```
rpc.tcp_client_num = 32
```

RPC connection timeout:

```
rpc.connect_timeout = 5s
```

RPC send timeout:

```
rpc.send_timeout = 5s
```

Authentication timeout:

```
rpc.authentication_timeout = 5s
```

Synchronous call timeout:

```
rpc.call_receive_timeout = 15s
```

Maximum keep-alive time when socket is idle:

```
rpc.socket_keepalive_idle = 900
```

Socket keep-alive detection interval:

```
rpc.socket_keepalive_interval = 75s
```

The maximum number of heartbeat detection failures before closing the connection:

```
rpc.socket_keepalive_count = 9
```

Size of TCP send buffer:

```
rpc.socket_sndbuf = 1MB
```

Size of TCP receive buffer:

```
rpc.socket_recbuf = 1MB
```

Size of user-level software socket buffer:

```
rpc.socket_buffer = 1MB
```

5.8 Log Parameter Configuration

Log output location, it can be set to write to the terminal or write to the file:

```
log.to = both
```

Set the log level:

```
log.level = error
```

Set the primary logger level and the log level of all logger handlers to the file and terminal.

Set the storage path of the log file:

```
log.dir = log
```

Set the file name for storing the "log.level":

```
log.file = emqx.log
```

Set the maximum size of each log file:

```
log.rotation.size = 10MB
```

Set the maximum number of files for log rotation:

```
log.rotation.count = 5
```

The user can write a level of log to a separate file by configuring additional file logger handlers in the format log.$level.file = $filename.

For example, the following configuration writes all logs higher than or equal to the info level to the info.log file:

```
log.info.file = info.log
```

5.9 Anonymous Authentication and ACL Files

Whether to allow the client to pass the authentication as an anonymous identity:

```
allow_anonymous = true
```

EMQ X supports ACLs based on built-in ACLs and plugins such as MySQL and PostgreSQL.

Set whether to allow access when all ACL rules cannot match:

```
acl_nomatch = allow
```

Set the default file for storing ACL rules:

```
acl_file = etc/acl.conf
```

Set whether to allow ACL caching:

```
enable_acl_cache = on
```

Set the maximum number of ACL caches for each client:

```
acl_cache_max_size = 32
```

Set the effective time of the ACL cache:

```
acl_cache_ttl = 1m
```

Etc/acl.conf access control rule definition:

```
Allow|Deny User|IP Address|ClientID Publish|Subscribe Topic List
```

The access control rules are in the Erlang tuple format, and the access control module matches the rules one by one:

```
.. image:: _static/images/zone.png
```

`etc/acl.conf` default access rule settings:

Allow `dashboard` users to subscribe to `$SYS/#`:

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

Allow local users to publish and subscribe all topics:

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

Deny users other than local users to subscribe to topics `$SYS/#` and `#`:

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

Allow any access other than the above rules:

```
{allow, all}.
```

: The default rule only allows local users to subscribe to $SYS/# and #.

When the EMQ X broker receives an Publish or Subscribe request from MQTT client, it will match the ACL rule one by one until the match returns to allow or deny.

5.10 MQTT Protocol Parameter Configuration

MQTT maximum packet size:

```
   mqtt.max_packet_size = 1MB

Maximum length of ClientId:
```

```
   mqtt.max_clientid_len = 65535

Maximum level of Topic, 0 means no limit:
```

```
mqtt.max_topic_levels = 0
```

Maximum allowed QoS:

```
mqtt.max_qos_allowed = 2
```

Maximum number of Topic Alias , 0 means Topic Alias is not supported:

```
mqtt.max_topic_alias = 0
```

Whether to support MQTT messages retain:

```
mqtt.retain_available = true
```

Whether to support MQTT wildcard subscriptions:

```
mqtt.wildcard_subscription = true
```

Whether to support MQTT shared subscriptions:

```
mqtt.shared_subscription = true
```

Whether to allow the loop deliver of the message:

```
mqtt.ignore_loop_deliver = false
```

This configuration is mainly used to implement (backporting) the No Local feature in MQTT v3.1.1. This feature is standardized in MQTT 5.

5.11 MQTT Zones Parameter Configuration

EMQ X uses **Zone** to manage configuration groups. A Zone defines a set of configuration items (such as the maximum number of connections, etc.), and Listener can specify to use a Zone to use all the configurations under that Zone. Multiple Listeners can share the same Zone.

Listeners are configured as follows, with priority Zone > Global > Default::

```
                    ---------              ----------              -----------
Listeners --------> | Zone  | --nomatch--> | Global | --nomatch--> | Default |
                    ---------              ----------              -----------
                        |                      |                       |
                      match                  match                   match
                       \|/                    \|/                     \|/
                 Zone Configs           Global Configs           Default␣
↪Configs
```

A zone config has a form `zone.$name.xxx`, here the `$name` is the zone name. Such as `zone.internal.xxx` and `zone.external.xxx`. User can also define customized zone name.

5.11.1 External Zone Parameter Settings

The maximum time to wait for MQTT CONNECT packet after the TCP connection is established:

```
zone.external.idle_timeout = 15s
```

Message Publish rate limit:

```
## zone.external.publish_limit = 10,100
```

Enable blacklist checking:

```
zone.external.enable_ban = on
```

Enable ACL check:

```
zone.external.enable_acl = on
```

Whether to statistics the information of each connection:

```
zone.external.enable_stats = on
```

MQTT maximum packet size:

```
## zone.external.max_packet_size = 64KB
```

ClientId maximum length:

```
## zone.external.max_clientid_len = 1024
```

Topic maximum level, 0 means no limit:

```
## zone.external.max_topic_levels = 7
```

Maximum allowed QoS:

```
## zone.external.max_qos_allowed = 2
```

Maximum number of Topic Alias , 0 means Topic Alias is not supported:

```
## zone.external.max_topic_alias = 0
```

Whether to support MQTT messages retain:

```
## zone.external.retain_available = true
```

Whether to support MQTT wildcard subscriptions:

```
## zone.external.wildcard_subscription = false
```

Whether to support MQTT shared subscriptions:

```
## zone.external.shared_subscription = false
```

The connection time allowed by the server, Commenting this line means that the connection time is determined by the client:

```
## zone.external.server_keepalive = 0
```

Keepalive * backoff * 2 is the actual keep-alive time:

```
zone.external.keepalive_backoff = 0.75
```

The maximum number of allowed topic subscriptions , 0 means no limit:

```
zone.external.max_subscriptions = 0
```

Whether to allow QoS upgrade:

```
zone.external.upgrade_qos = off
```

The maximum size of the in-flight window:

```
zone.external.max_inflight = 32
```

Resend interval for QoS 1/2 messages:

```
zone.external.retry_interval = 20s
```

The maximum number of QoS2 messages waiting for PUBREL (Client -> Broker), 0 means no limit:

```
zone.external.max_awaiting_rel = 100
```

Maximum time to wait for PUBREL before QoS2 message (Client -> Broker) is deleted

```
zone.external.await_rel_timeout = 300s
```

Default session expiration time used in MQTT v3.1.1 connections:

```
zone.external.session_expiry_interval = 2h
```

Message queue type:

```
zone.external.mqueue_type = simple
```

Maximum length of the message queue:

```
zone.external.max_mqueue_len = 1000
```

Topic priority:

```
## zone.external.mqueue_priorities = topic/1=10,topic/2=8
```

Whether the message queue stores QoS0 messages:

```
zone.external.mqueue_store_qos0 = true
```

Whether to enable flapping detection:

```
zone.external.enable_flapping_detect = off
```

The maximum number of state changes allowed during the specified time:

```
zone.external.flapping_threshold = 10, 1m
```

Flapping prohibited time:

```
zone.external.flapping_banned_expiry_interval = 1h
```

5.11.2 Internal Zone Parameter Settings

Allow anonymous access:

```
zone.internal.allow_anonymous = true
```

Whether to Statistics the information of each connection:

```
zone.internal.enable_stats = on
```

Close ACL checking:

```
zone.internal.enable_acl = off
```

Whether to support MQTT wildcard subscriptions:

```
## zone.internal.wildcard_subscription = true
```

Whether to support MQTT shared subscriptions:

```
## zone.internal.shared_subscription = true
```

The maximum number of allowed topic subscriptions, 0 means no limit:

```
zone.internal.max_subscriptions = 0
```

The maximum size of the in-flight window:

```
zone.internal.max_inflight = 32
```

The maximum number of QoS2 messages waiting for PUBREL (Client -> Broker), 0 means no limit:

```
zone.internal.max_awaiting_rel = 100
```

Maximum length of the message queue:

```
zone.internal.max_mqueue_len = 1000
```

Whether the message queue stores QoS0 messages:

```
zone.internal.mqueue_store_qos0 = true
```

Whether to enable flapping detection:

```
zone.internal.enable_flapping_detect = off
```

The maximum number of state changes allowed during the specified time:

```
zone.internal.flapping_threshold = 10, 1m
```

Flapping banned time:

```
zone.internal.flapping_banned_expiry_interval = 1h
```

5.12 MQTT Listeners Parameter Description

The EMQ X message server supports the MQTT, MQTT/SSL, and MQTT/WS protocol, the port, maximum allowed connections, and other parameters are configurable through `listener.tcp|ssl|ws|wss|.*`.

The TCP ports of the EMQ X broker that are enabled by default include:

1883	MQTT TCP protocol port
8883	MQTT/TCP SSL port
8083	MQTT/WebSocket port
8084	MQTT/WebSocket with SSL port

Listener parameter description:

listener.tcp.${name}.acceptors	TCP Acceptor pool
listener.tcp.${name}.max_connections	Maximum number of allowed TCP connections
listener.tcp.${name}.max_conn_rate	Connection limit configuration
listener.tcp.${name}.zone	To which zone the listener belongs
listener.tcp.${name}.rate_limit	Connection rate configuration

5.13 MQTT/TCP Listener - 1883

The EMQ X supports the configuration of multiple MQTT protocol listeners, for example, two listeners named `external` and `internal`:

TCP listeners:

```
listener.tcp.external = 0.0.0.0:1883
```

Receive pool size:

```
listener.tcp.external.acceptors = 8
```

Maximum number of concurrent connections:

```
listener.tcp.external.max_connections = 1024000
```

Maximum number of connections created per second:

```
listener.tcp.external.max_conn_rate = 1000
```

Zone used by the listener:

```
listener.tcp.external.zone = external
```

Mountpoint:

```
## listener.tcp.external.mountpoint = devicebound/
```

TCP data receive rate limit:

```
## listener.tcp.external.rate_limit = 1024,4096
```

Access control rules:

```
## listener.tcp.external.access.1 = allow 192.168.0.0/24

listener.tcp.external.access.1 = allow all
```

Whether the proxy protocol V1/2 is enabled when the EMQ X cluster is deployed with HAProxy or Nginx:

```
## listener.tcp.external.proxy_protocol = on
```

Timeout of the proxy protocol:

```
## listener.tcp.external.proxy_protocol_timeout = 3s
```

Enable the X.509 certificate-based authentication option. EMQ X will use the common name of the certificate as the MQTT username:

```
## listener.tcp.external.peer_cert_as_username = cn
```

The maximum length of the queue of suspended connection:

```
listener.tcp.external.backlog = 1024
```

TCP send timeout:

```
listener.tcp.external.send_timeout = 15s
```

Whether to close the TCP connection when the sent is timeout:

```
listener.tcp.external.send_timeout_close = on
```

TCP receive buffer (os kernel) for MQTT connections:

```
#listener.tcp.external.recbuf = 2KB
```

TCP send buffer (os kernel) for MQTT connections:

```
#listener.tcp.external.sndbuf = 2KB
```

The size of the user-level software buffer used by the driver should not to be confused with the options of sndbuf and recbuf, which correspond to the kernel socket buffer. It is recommended to use val(buffer) >= max(val(sndbuf), val(recbuf)) to avoid performance problems caused by unnecessary duplication. When the sndbuf or recbuf value is set, val(buffer) is automatically set to the maximum value abovementioned:

```
#listener.tcp.external.buffer = 2KB
```

Whether to set buffer = max(sndbuf, recbuf):

```
## listener.tcp.external.tune_buffer = off
```

Whether to set the TCP_NODELAY flag. If this flag is set, it will attempt to send data once there is data in the send buffer.

```
listener.tcp.external.nodelay = true
```

Whether to set the SO_REUSEADDR flag:

```
listener.tcp.external.reuseaddr = true
```

5.14 MQTT/SSL Listener - 8883

SSL listening port:

```
listener.ssl.external = 8883
```

Acceptor size:

```
listener.ssl.external.acceptors = 16
```

Maximum number of concurrent connections:

```
listener.ssl.external.max_connections = 102400
```

Maximum number of connections created per second:

```
listener.ssl.external.max_conn_rate = 500
```

Zone used by the listener:

```
listener.ssl.external.zone = external
```

Mountpoint:

```
## listener.ssl.external.mountpoint = devicebound/
```

Access control rules:

```
listener.ssl.external.access.1 = allow all
```

TCP data receive rate limit:

```
## listener.ssl.external.rate_limit = 1024,4096
```

Whether the proxy protocol V1/2 is enabled when the EMQ X cluster is deployed with HAProxy or Nginx:

```
## listener.ssl.external.proxy_protocol = on
```

Timeout of the proxy protocol:

```
## listener.ssl.external.proxy_protocol_timeout = 3s
```

TLS version to prevent POODLE attacks:

```
## listener.ssl.external.tls_versions = tlsv1.2,tlsv1.1,tlsv1
```

TLS handshake timeout:

```
listener.ssl.external.handshake_timeout = 15s
```

Path of the file containing the user's private key:

```
listener.ssl.external.keyfile = etc/certs/key.pem
```

Path of the file containing the user certificate:

```
listener.ssl.external.certfile = etc/certs/cert.pem
```

Path of the file containing the CA certificate:

```
## listener.ssl.external.cacertfile = etc/certs/cacert.pem
```

Path of the file containing dh-params:

```
## listener.ssl.external.dhfile = etc/certs/dh-params.pem
```

Configure verify mode, and the server only performs x509 path verification in verify_peer mode and sends a certificate request to the client:

```
## listener.ssl.external.verify = verify_peer
```

When the server is in the verify_peer mode, whether the server returns a failure if the client does not have a certificate to send:

```
## listener.ssl.external.fail_if_no_peer_cert = true
```

SSL cipher suites:

```
listener.ssl.external.ciphers = ECDHE-ECDSA-AES256-GCM-SHA384,ECDHE-RSA-
↪AES256-GCM-SHA384,ECDHE-ECDSA-AES256-SHA384,ECDHE-RSA-AES256-SHA384,ECDHE-
↪ECDSA-DES-CBC3-SHA,ECDH-ECDSA-AES256-GCM-SHA384,ECDH-RSA-AES256-GCM-SHA384,
↪ECDH-ECDSA-AES256-SHA384,ECDH-RSA-AES256-SHA384,DHE-DSS-AES256-GCM-SHA384,
↪DHE-DSS-AES256-SHA256,AES256-GCM-SHA384,AES256-SHA256,ECDHE-ECDSA-AES128-
↪GCM-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-ECDSA-AES128-SHA256,ECDHE-RSA-
↪AES128-SHA256,ECDH-ECDSA-AES128-GCM-SHA256,ECDH-RSA-AES128-GCM-SHA256,ECDH-
↪ECDSA-AES128-SHA256,ECDH-RSA-AES128-SHA256,DHE-DSS-AES128-GCM-SHA256,DHE-
↪DSS-AES128-SHA256,AES128-GCM-SHA256,AES128-SHA256,ECDHE-ECDSA-AES256-SHA,
↪ECDHE-RSA-AES256-SHA,DHE-DSS-AES256-SHA,ECDH-ECDSA-AES256-SHA,ECDH-RSA-
↪AES256-SHA,AES256-SHA,ECDHE-ECDSA-AES128-SHA,ECDHE-RSA-AES128-SHA,DHE-DSS-
↪AES128-SHA,ECDH-ECDSA-AES128-SHA,ECDH-RSA-AES128-SHA,AES128-SHA
```

Whether to start a more secure renegotiation mechanism:

```
## listener.ssl.external.secure_renegotiate = off
```

Whether to allow the client to reuse an existing session:

```
## listener.ssl.external.reuse_sessions = on
```

Whether to force ciphers to be set in the order specified by the server, not by the client:

```
## listener.ssl.external.honor_cipher_order = on
```

Use the CN, EN, or CRT field in the client certificate as the username. Note that "verify" should be set to "verify_peer":

```
## listener.ssl.external.peer_cert_as_username = cn
```

The maximum length of the queue of suspended connection:

```
## listener.ssl.external.backlog = 1024
```

TCP send timeout:

```
## listener.ssl.external.send_timeout = 15s
```

Whether to close the TCP connection when the sent is timeout:

```
## listener.ssl.external.send_timeout_close = on
```

TCP receive buffer (os kernel) for MQTT connections:

```
#listener.ssl.external.recbuf = 2KB
```

TCP send buffer (os kernel) for MQTT connections:

```
## listener.ssl.external.sndbuf = 4KB
```

The size of the user-level software buffer used by the driver should not to be confused with the options of sndbuf and recbuf, which correspond to the kernel socket buffer. It is recommended to use val(buffer) >= max(val(sndbuf), val(recbuf)) to avoid performance problems caused by unnecessary duplication. When the sndbuf or recbuf value is set, val(buffer) is automatically set to the maximum value abovementioned:

```
## listener.ssl.external.buffer = 4KB
```

Whether to set buffer = max(sndbuf, recbuf):

```
## listener.ssl.external.tune_buffer = off
```

Whether to set the TCP_NODELAY flag. If this flag is set, it will attempt to send data once there is data in the send buffer:

```
## listener.ssl.external.nodelay = true
```

Whether to set the SO_REUSEADDR flag:

```
listener.ssl.external.reuseaddr = true
```

5.15 MQTT/WebSocket Listener - 8083

MQTT/WebSocket listening port:

```
listener.ws.external = 8083
```

Acceptors size:

```
listener.ws.external.acceptors = 4
```

Maximum number of concurrent connections:

```
listener.ws.external.max_connections = 102400
```

Maximum number of connections created per second:

```
listener.ws.external.max_conn_rate = 1000
```

TCP data receive rate limit:

```
## listener.ws.external.rate_limit = 1024,4096
```

Zone used by the listener:

```
listener.ws.external.zone = external
```

Mountpoint:

```
## listener.ws.external.mountpoint = devicebound/
```

Access control rules:

```
listener.ws.external.access.1 = allow all
```

Whether to verify that the protocol header is valid:

```
listener.ws.external.verify_protocol_header = on
```

Uses X-Forward-For to identify the original IP after the EMQ X cluster is deployed with NGINX or HAProxy:

```
## listener.ws.external.proxy_address_header = X-Forwarded-For
```

Uses X-Forward-For to identify the original port after the EMQ X cluster is deployed with NGINX or HAProxy:

```
## listener.ws.external.proxy_port_header = X-Forwarded-Port
```

Whether the proxy protocol V1/2 is enabled when the EMQ X cluster is deployed with HAProxy or Nginx:

```
## listener.ws.external.proxy_protocol = on
```

Proxy protocol timeout:

```
## listener.ws.external.proxy_protocol_timeout = 3s
```

The maximum length of the queue of suspended connection:

```
listener.ws.external.backlog = 1024
```

TCP send timeout:

```
listener.ws.external.send_timeout = 15s
```

Whether to close the TCP connection when the send is timeout:

```
listener.ws.external.send_timeout_close = on
```

TCP receive buffer (os kernel) for MQTT connections:

```
## listener.ws.external.recbuf = 2KB
```

TCP send buffer (os kernel) for MQTT connections:

```
## listener.ws.external.sndbuf = 2KB
```

The size of the user-level software buffer used by the driver should not to be confused with the options of sndbuf and recbuf, which correspond to the kernel socket buffer. It is recommended to use val(buffer) >= max(val(sndbuf), val(recbuf)) to avoid performance problems caused by unnecessary duplication. When the sndbuf or recbuf value is set, val(buffer) is automatically set to the maximum value abovementioned:

```
## listener.ws.external.buffer = 2KB
```

Whether to set buffer = max(sndbuf, recbuf):

```
## listener.ws.external.tune_buffer = off
```

Whether to set the TCP_NODELAY flag. If this flag is set, it will attempt to send data once there is data in the send buffer:

```
listener.ws.external.nodelay = true
```

Whether to compress Websocket messages:

```
## listener.ws.external.compress = true
```

Websocket deflate option:

```
## listener.ws.external.deflate_opts.level = default
## listener.ws.external.deflate_opts.mem_level = 8
## listener.ws.external.deflate_opts.strategy = default
## listener.ws.external.deflate_opts.server_context_takeover = takeover
## listener.ws.external.deflate_opts.client_context_takeover = takeover
## listener.ws.external.deflate_opts.server_max_window_bits = 15
## listener.ws.external.deflate_opts.client_max_window_bits = 15
```

Maximum idle time:

```
## listener.ws.external.idle_timeout = 2h
```

Maximum packet size, 0 means no limit:

```
## listener.ws.external.max_frame_size = 0
```

5.16 MQTT/WebSocket with SSL Listener - 8084

MQTT/WebSocket with SSL listening port:

```
listener.wss.external = 8084
```

Acceptors size:

```
listener.wss.external.acceptors = 4
```

Maximum number of concurrent connections:

```
listener.wss.external.max_connections = 16
```

Maximum number of connections created per second:

```
listener.wss.external.max_conn_rate = 1000
```

TCP data receive rate limit:

```
## listener.wss.external.rate_limit = 1024,4096
```

Zone used by the listener:

```
listener.wss.external.zone = external
```

Mountpoint:

```
## listener.wss.external.mountpoint = devicebound/
```

Access control rules:

```
listener.wss.external.access.1 = allow all
```

Whether to verify that the protocol header is valid:

```
listener.wss.external.verify_protocol_header = on
```

Uses X-Forward-For to identify the original IP after the EMQ X cluster is deployed with NGINX or HAProxy:

```
## listener.wss.external.proxy_address_header = X-Forwarded-For
```

Uses X-Forward-For to identify the original port after the EMQ X cluster is deployed with NGINX or HAProxy:

```
## listener.wss.external.proxy_port_header = X-Forwarded-Port
```

Whether the proxy protocol V1/2 is enabled when the EMQ X cluster is deployed with HAProxy or Nginx:

```
## listener.wss.external.proxy_protocol = on
```

Proxy protocol timeout:

```
## listener.wss.external.proxy_protocol_timeout = 3s
```

TLS version to prevent POODLE attacks:

```
## listener.wss.external.tls_versions = tlsv1.2,tlsv1.1,tlsv1
```

Path of the file containing the user's private key:

```
listener.wss.external.keyfile = etc/certs/key.pem
```

Path of the file containing the user certificate:

```
listener.wss.external.certfile = etc/certs/cert.pem
```

Path of the file containing the CA certificate:

```
## listener.wss.external.cacertfile = etc/certs/cacert.pem
```

Path of the file containing dh-params:

```
## listener.ssl.external.dhfile = etc/certs/dh-params.pem
```

Configure verify mode, and the server only performs x509 path verification in verify_peer mode and sends a certificate request to the client:

```
## listener.wss.external.verify = verify_peer
```

When the server is in the verify_peer mode, whether the server returns a failure if the client does not have a certificate to send :

```
## listener.wss.external.fail_if_no_peer_cert = true
```

SSL cipher suites:

```
## listener.wss.external.ciphers = ECDHE-ECDSA-AES256-GCM-SHA384,ECDHE-RSA-
↪AES256-GCM-SHA384,ECDHE-ECDSA-AES256-SHA384,ECDHE-RSA-AES256-SHA384,ECDHE-
↪ECDSA-DES-CBC3-SHA,ECDH-ECDSA-AES256-GCM-SHA384,ECDH-RSA-AES256-GCM-SHA384,
↪ECDH-ECDSA-AES256-SHA384,ECDH-RSA-AES256-SHA384,DHE-DSS-AES256-GCM-SHA384,
↪DHE-DSS-AES256-SHA256,AES256-GCM-SHA384,AES256-SHA256,ECDHE-ECDSA-AES128-
↪GCM-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-ECDSA-AES128-SHA256,ECDHE-RSA-
↪AES128-SHA256,ECDH-ECDSA-AES128-GCM-SHA256,ECDH-RSA-AES128-GCM-SHA256,ECDH-
↪ECDSA-AES128-SHA256,ECDH-RSA-AES128-SHA256,DHE-DSS-AES128-SHA256,DHE-RSA-
↪DSS-AES128-SHA256,AES128-GCM-SHA256,AES128-SHA256,ECDHE-ECDSA-AES256-SHA,
↪ECDHE-RSA-AES256-SHA,DHE-DSS-AES256-SHA,ECDH-ECDSA-AES256-SHA,ECDH-RSA-
↪AES256-SHA,AES256-SHA,ECDHE-ECDSA-AES128-SHA,ECDHE-RSA-AES128-SHA,DHE-DSS-
```

()

Whether to enable a more secure renegotiation mechanism:

```
## listener.wss.external.secure_renegotiate = off
```

Whether to allow the client to reuse an existing session:

```
## listener.wss.external.reuse_sessions = on
```

Whether to force ciphers to be set in the order specified by the server, not by the client:

```
## listener.wss.external.honor_cipher_order = on
```

Use the CN, EN, or CRT field in the client certificate as the username. Note that "verify" should be set to "verify_peer":

```
## listener.wss.external.peer_cert_as_username = cn
```

The maximum length of the queue that suspends the connection:

```
listener.wss.external.backlog = 1024
```

TCP send timeout:

```
listener.wss.external.send_timeout = 15s
```

Whether to close the TCP connection when the send is timeout :

```
listener.wss.external.send_timeout_close = on
```

TCP receive buffer (os kernel) for MQTT connections:

```
## listener.wss.external.recbuf = 4KB
```

TCP send buffer (os kernel) for MQTT connections:

```
## listener.wss.external.sndbuf = 4KB
```

The size of the user-level software buffer used by the driver should not to be confused with the options of sndbuf and recbuf, which correspond to the kernel socket buffer. It is recommended to use val(buffer) >= max(val(sndbuf), val(recbuf)) to avoid performance problems caused by unnecessary duplication. When the sndbuf or recbuf value is set, val(buffer) is automatically set to the maximum value abovementioned:

```
## listener.wss.external.buffer = 4KB
```

Whether to set the TCP_NODELAY flag. If this option is enabled, it will attempt to send data once there is data in the send buffer :

```
## listener.wss.external.nodelay = true
```

Whether to compress Websocket messages:

```
## listener.wss.external.compress = true
```

Websocket deflate option:

```
## listener.wss.external.deflate_opts.level = default
## listener.wss.external.deflate_opts.mem_level = 8
## listener.wss.external.deflate_opts.strategy = default
## listener.wss.external.deflate_opts.server_context_takeover = takeover
## listener.wss.external.deflate_opts.client_context_takeover = takeover
## listener.wss.external.deflate_opts.server_max_window_bits = 15
## listener.wss.external.deflate_opts.client_max_window_bits = 15
```

Maximum idle time:

```
## listener.wss.external.idle_timeout = 2h
```

Maximum packet size, 0 means no limit:

```
## listener.wss.external.max_frame_size = 0
```

5.17 Bridges

5.17.1 Bridges Parameter Setting

Bridge address, use node name for RPC bridging, and use host:port for MQTT connection:

```
bridge.aws.address = 127.0.0.1:1883
```

Bridged protocol version:

```
bridge.aws.proto_ver = mqttv4
```

Client's client_id:

```
bridge.aws.client_id = bridge_aws
```

The clean_start field of the client:

```
bridge.aws.clean_start = true
```

The username field of the client:

```
bridge.aws.username = user
```

The password field of the client:

```
bridge.aws.password = passwd
```

Bridge mount points:

```
bridge.aws.mountpoint = bridge/aws/${node}/
```

The topic of the message to be forwarded:

```
bridge.aws.forwards = topic1/#,topic2/#
```

Whether the client uses SSL to connect to the remote server:

```
bridge.aws.ssl = off
```

SSL certificate for CA connection (PEM format)

```
bridge.aws.cacertfile = etc/certs/cacert.pem
```

SSL certificate for SSL connection:

```
bridge.aws.certfile = etc/certs/client-cert.pem
```

Key file for SSL connection:

```
bridge.aws.keyfile = etc/certs/client-key.pem
```

SSL cipher suites:

```
#bridge.aws.ciphers = ECDHE-ECDSA-AES256-GCM-SHA384,ECDHE-RSA-AES256-GCM-
↪SHA384
```

TLS PSK Password:

```
#bridge.aws.psk_ciphers = PSK-AES128-CBC-SHA,PSK-AES256-CBC-SHA,PSK-3DES-EDE-
↪CBC-SHA,PSK-RC4-SHA
```

Client's heartbeat interval:

```
bridge.aws.keepalive = 60s
```

Supported TLS versions:

```
bridge.aws.tls_versions = tlsv1.2,tlsv1.1,tlsv1
```

subscription topics of bridge:

```
bridge.aws.subscription.1.topic = cmd/topic1
```

subscription qos of bridge:

```
bridge.aws.subscription.1.qos = 1
```

Bridge start type:

```
bridge.aws.start_type = manual
```

Bridge reconnection interval:

```
bridge.aws.reconnect_interval = 30s
```

Resending interval for QoS 1/2 messages:

```
bridge.aws.retry_interval = 20s
```

In-flight window size:

```
bridge.aws.max_inflight_batches = 32
```

The number of messages used internally by emqx_bridge for batch:

```
bridge.aws.queue.batch_count_limit = 32
```

The bytes of messages used internally by emqx_bridge for batch:

```
bridge.aws.queue.batch_bytes_limit = 1000MB
```

The path where the replayq queue is placed. If the item is not specified in the configuration, replayq will run in mem-only mode and the message will not be cached on disk:

```
bridge.aws.queue.replayq_dir = {{ platform_data_dir }}/emqx_aws_bridge/
```

Replayq data segment size:

```
bridge.aws.queue.replayq_seg_bytes = 10MB
```

5.18 Modules

EMQ X supports module expansion. The three default modules are the online and offline status message publishing module, the proxy subscription module, and the topic rewriting module.

5.18.1 Online and offline status message publishing module

Whether to enable the online and offline status message publishing module:

```
module.presence = on
```

QoS used by the online and offline status message publishing module to publish MQTT messages:

```
module.presence.qos = 1
```

5.18.2 Proxy Subscription Module

Whether to enable the proxy subscription module:

```
module.subscription = off
```

Topics and QoS that are automatically subscribed when the client connects:

```
## Subscribe the Topics's qos
## module.subscription.1.topic = $client/%c
## module.subscription.1.qos = 0
## module.subscription.2.topic = $user/%u
## module.subscription.2.qos = 1
```

5.18.3 Topic Rewriting Module

Whether to enable the topic rewriting module:

```
module.rewrite = off
```

Topic rewriting rule:

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

5.19 Configuration Files for Plugins

The directory where the plugin configuration file is stored:

```
plugins.etc_dir = etc/plugins/
```

Path of the file to store list of plugins that needs to be automatically loaded at startup

```
plugins.loaded_file = data/loaded_plugins
```

The EMQ X plugin configuration file, which is in the directory of `etc/plugins/` by default, and can be adjusted by modification of plugins.etc_dir.

5.20 Broker Parameter Settings

System message publishing interval:

```
broker.sys_interval = 1m
```

Whether to register the session globally:

```
broker.enable_session_registry = on
```

Session locking strategy:

```
broker.session_locking_strategy = quorum
```

Dispatch strategy for shared subscriptions:

```
broker.shared_subscription_strategy = random
```

Whether an ACK is required when dispatching the sharing subscription:

```
broker.shared_dispatch_ack_enabled = false
```

Whether to enable route batch cleaning:

```
broker.route_batch_clean = on
```

5.21 Erlang VM Monitoring

Whether to enable long_gc monitoring and how long garbage collection lasts that can trigger the long_gc event:

```
sysmon.long_gc = false
```

How long does a process or port in the system keep running to trigger long_schedule event:

```
sysmon.long_schedule = 240
```

How big is the size of allocated heap caused by garbage collection to trigger the large_heap event:

```
sysmon.large_heap = 8MB
```

Whether a process in the system triggers a busy_port event when it hangs because it is sent to a busy port:

```
sysmon.busy_port = false
```

Whether to listen Erlang distributed port busy events:

```
sysmon.busy_dist_port = true
```

Cpu occupancy check interval:

```
os_mon.cpu_check_interval = 60s
```

An alarm is generated when the CPU usage is higher than:

```
os_mon.cpu_high_watermark = 80%
```

Clear the alarm when the CPU usage is lower than:

```
os_mon.cpu_low_watermark = 60%
```

Memory usage check interval:

```
os_mon.mem_check_interval = 60s
```

An alarm is generated when the system memory usage is higher than:

```
os_mon.sysmem_high_watermark = 70%
```

An alarm is generated when the memory usage of a single process is higher than:

```
os_mon.procmem_high_watermark = 5%
```

The check interval of the number of processes:

```
vm_mon.check_interval = 30s
```

An alarm is generated when the ratio of the current number of processes to the maximum number of processes reached:

```
vm_mon.process_high_watermark = 80%
```

Clear the alarm when the ratio of the current number of processes to the maximum number of processes reached:

```
vm_mon.process_low_watermark = 60%
```

CHAPTER 6

EMQ X Dashboard

6.1 Introduction

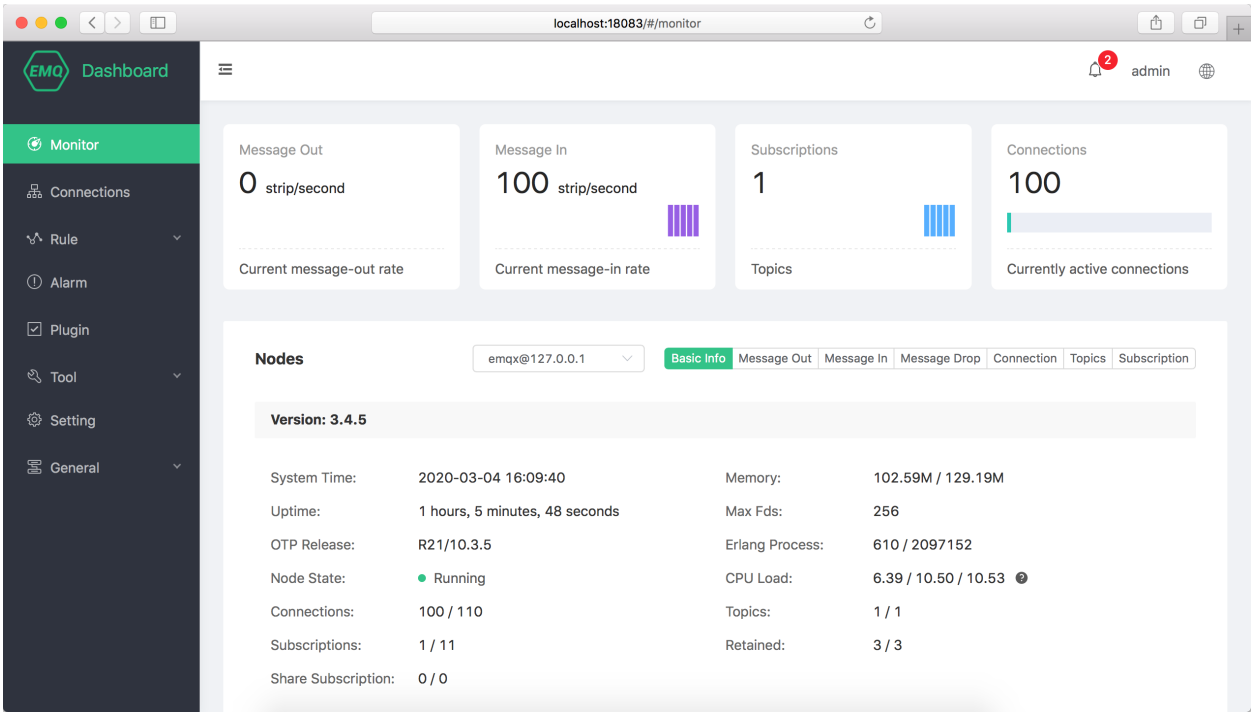
EMQ X Dashboard is a web console provided by EMQ X. You can view the running status of nodes and cluster, metrics, online status and subscriptions of clients through the Dashboard. You can also configure the plug-in on the Dashboard, stop and start the specified plug-in, manage the HTTP API key and most configurations of the EMQ X cluster.

6.2 Quick Start

If EMQ X is installed on this machine, use your browser to open the address http://127.0.0.1:18083. To log in, enter the default user name `admin` and the default password `public` to log in to Dashboard. If you forget to account information, click the **Forgot Password** button on the login page and follow the instructions or use management commands to reset or create a new account.

The Dashboard interface is shown in the following figure. It contains the left navigation bar, top control bar, and content area. The top control bar (red frame area) has three functions:

- Alarm: EMQ X alarm info. The number of alarms triggered by excessive resource usage and EMQ X internal errors is displayed. Click to view the alarm list.
- User: the currently logged in Dashboard user, which can be used to log out and change passwords;
- I18n: Dashboard displays Chinese / English by default according to the user's browser language. Click to switch languages.



6.3 Monitor

On the monitoring page, you can view the running status of the current cluster. The functional area from top to bottom of the interface is as follows:

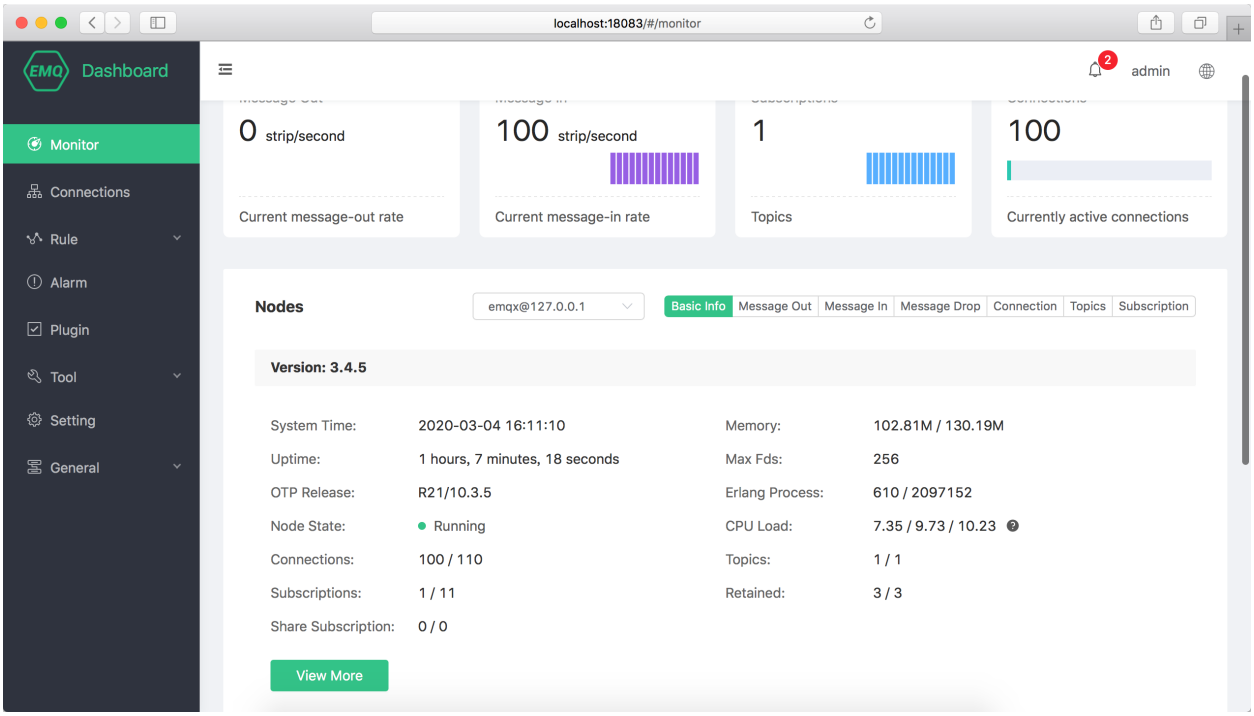
6.3.1 Running Status

There are four cards at the top of the page, which includes the message-out rate of the cluster, the message-in rate, the number of subscriptions, and the current number of connections.

6.3.2 Node

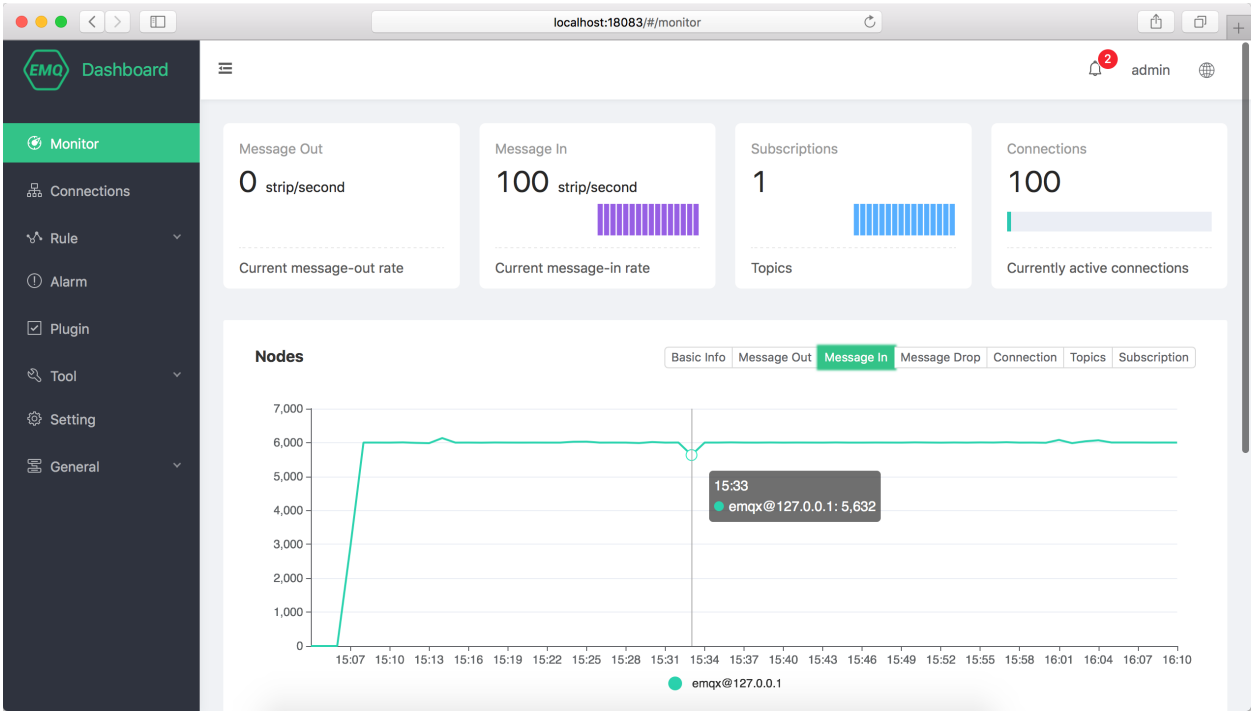
Click the node drop-down list to switch to view the basic information of the node, including EMQ X version, runtime, resource occupation, connection, and subscription data. Some data is defined as follows:

- Memory: The current memory/maximum memory used by the Erlang VM, **where the maximum memory is automatically applied to the system by EMQ X depending on the resource usage**.
- Max Fds: Allow the current session/process to open the number of file handles. If this value is too small, it will limit the EMQ X concurrency performance. When it is far less than the maximum number of connections authorized by the license, please refer to the tuning or contact the EMQ technical staff to modify;
- Erlang ProcessConnectionsTopicsSubscriptionsRetainedShare Subscription: It is divided into two groups by / which are the current value and the maximum value.



6.3.3 Recent status

Click the button group on the right side of the node data area to switch to the recent cluster data graph. The graph values are the actual values during the sampling period:



6.3.4 Nde details

Click the **View More** button below the node data to open the node details page, view the **basic information** of the current node, the **listener** and connection status, and **metrics**.

Listener

The listener is the list of the current EMQ X listening network ports. The field information is as follows:

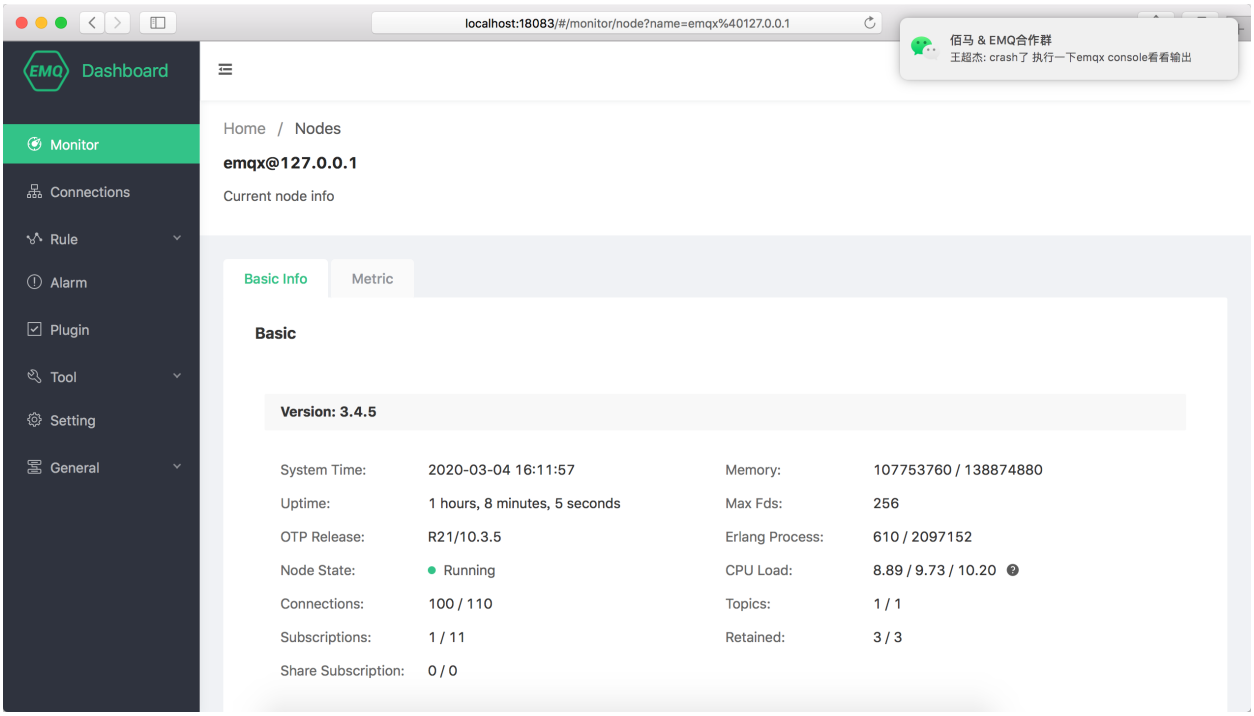
- Protocol: listening network/application protocols, including protocol and function info:
  - mqtt:ssl: MQTT TCP TLS protocols, the default is 102400
  - mqtt:tcp: MQTT TCP protocols, the default is 102400
  - http:dashboard: HTTP protocol used by Dashboard, the default is 512
  - http:management: HTTP protocol used by EMQ X REST API, the default is 512
  - mqtt:ws :MQTT over WebSocket, the default is 102400
  - mqtt:wss: MQTT over WebSocket TLS, the default is 102400
- Address: Listen to the bound network address and port. Listen to all IP addresses by default;
- Acceptors: listening the processor thread pool;
- Connect: It contains a set of current/maximum values. The current value is the actual number of established connections. The maximum value is the maximum number of connections configured in the configuration file. **If any listener exceeds the maximum value, a new connection cannot be established.**

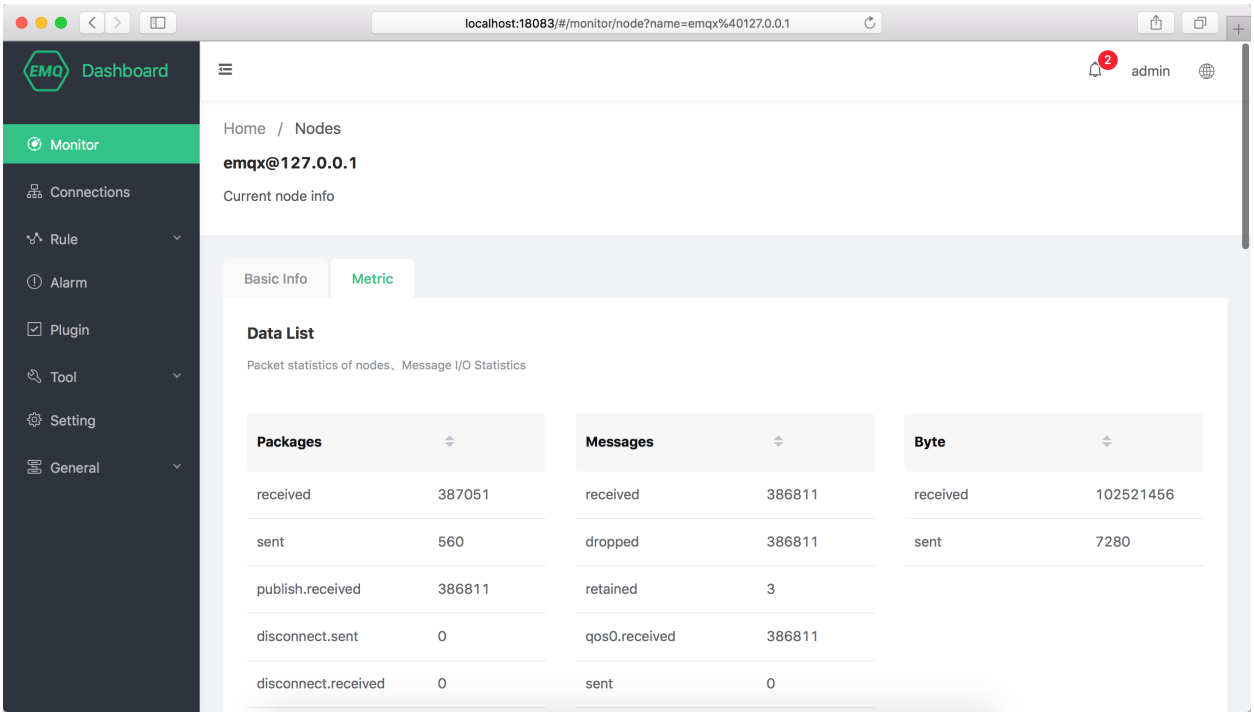
About the maximum number of connections

The actual maximum connection depends on the license and configuration:

1. The number of connections per listening protocol in the node cannot exceed the maximum number of connections in the configuration file;
2. The total number of MQTT/MQTT over WebSocket protocol connections in the cluster cannot exceed the upper limit of the license.

Of course, system tuning and hardware will also affect the maximum number of connections. Please refer to tuning or contact EMQ technicians for confirmation.





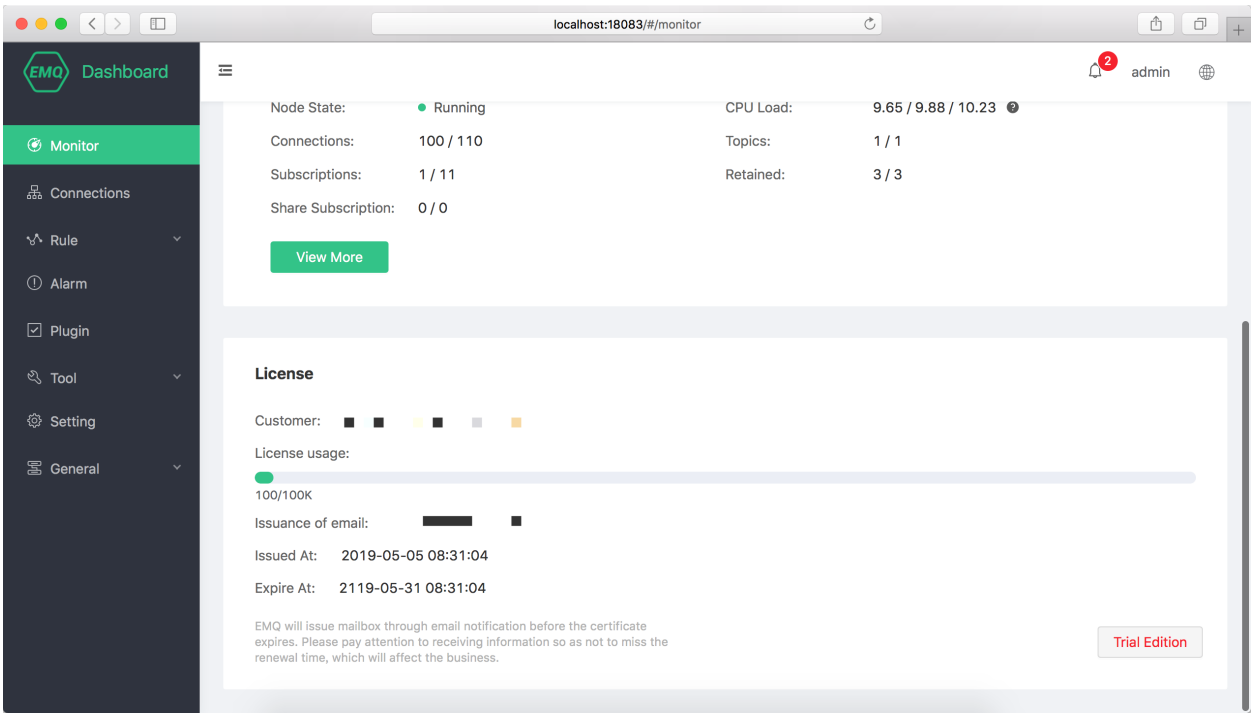
6.3.5 License

You can view the license information of the cluster by monitoring the license card at the bottom of the page:

- Customer: Name of the company or department of the same business contract customer.
- License usage: License specification and current usage.

- Issuance of email: Same email address as a business contract customer.
- License Edition: License edition, trial or official.

EMQ will issue a mailbox through email notification before the certificate expires. Please pay attention to receiving information so as not to miss the renewal time, which will affect the business.

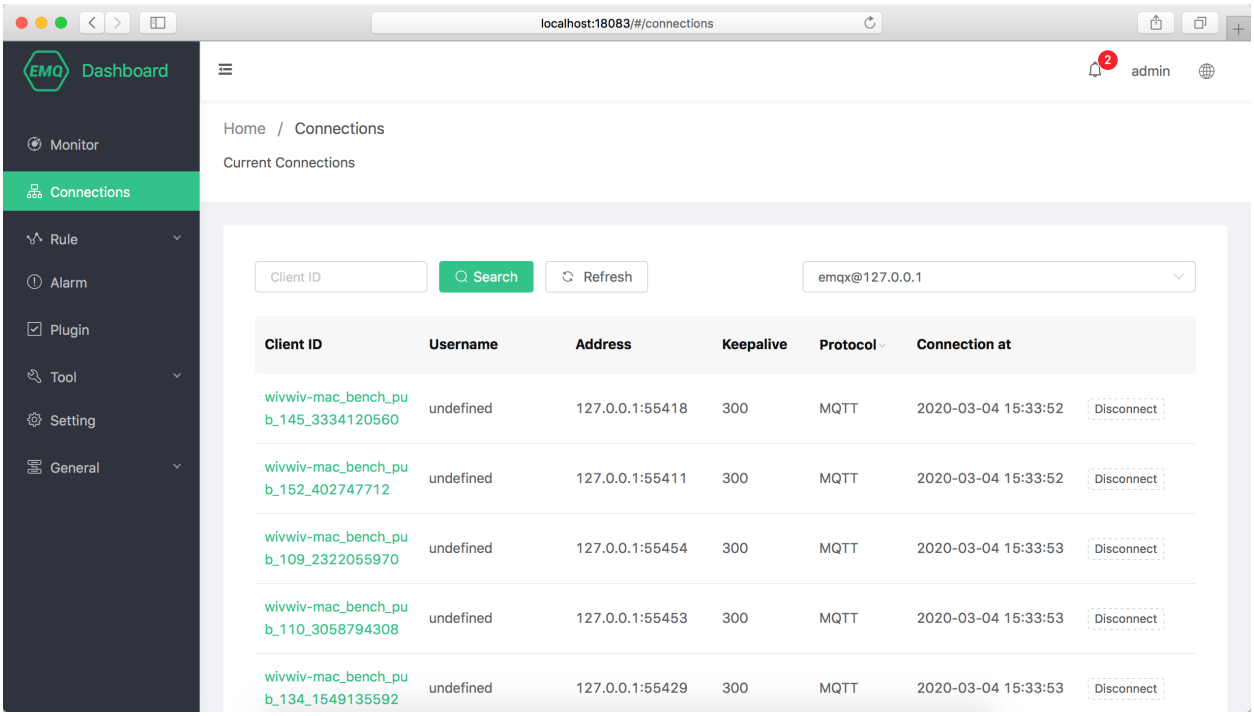


6.4 Connections

6.4.1 Current Connections

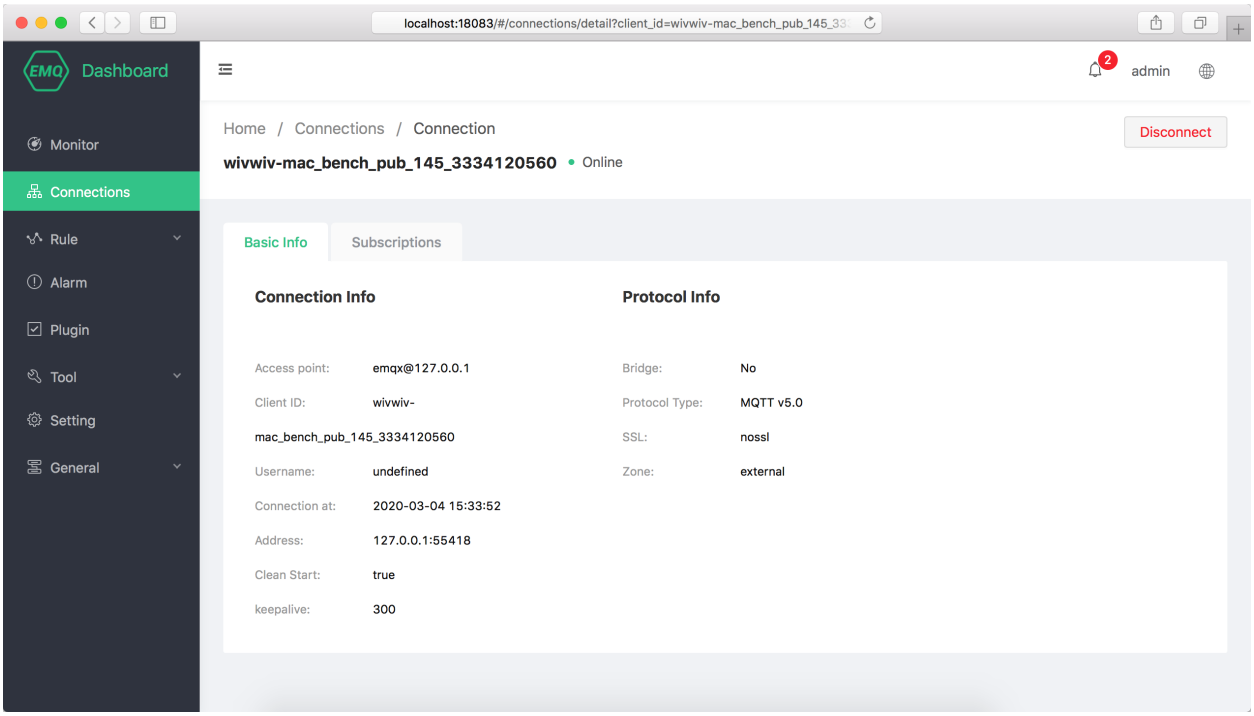
The client list page displays a list of currently connected clients.Several important information in the list is as follows:

- Client IDUsername: MQTT client ID and MQTT username, respectively. Click the **Client ID** to view the client details and subscription info.
- IP Address: The client address + port.
- Disconnect/Clean Session: For an online client, the connection will be disconnected and the session will be cleared. If the client is not online, clicking Clear Session will clear the session such as the client's subscription relationship.



6.4.2 Basic Info

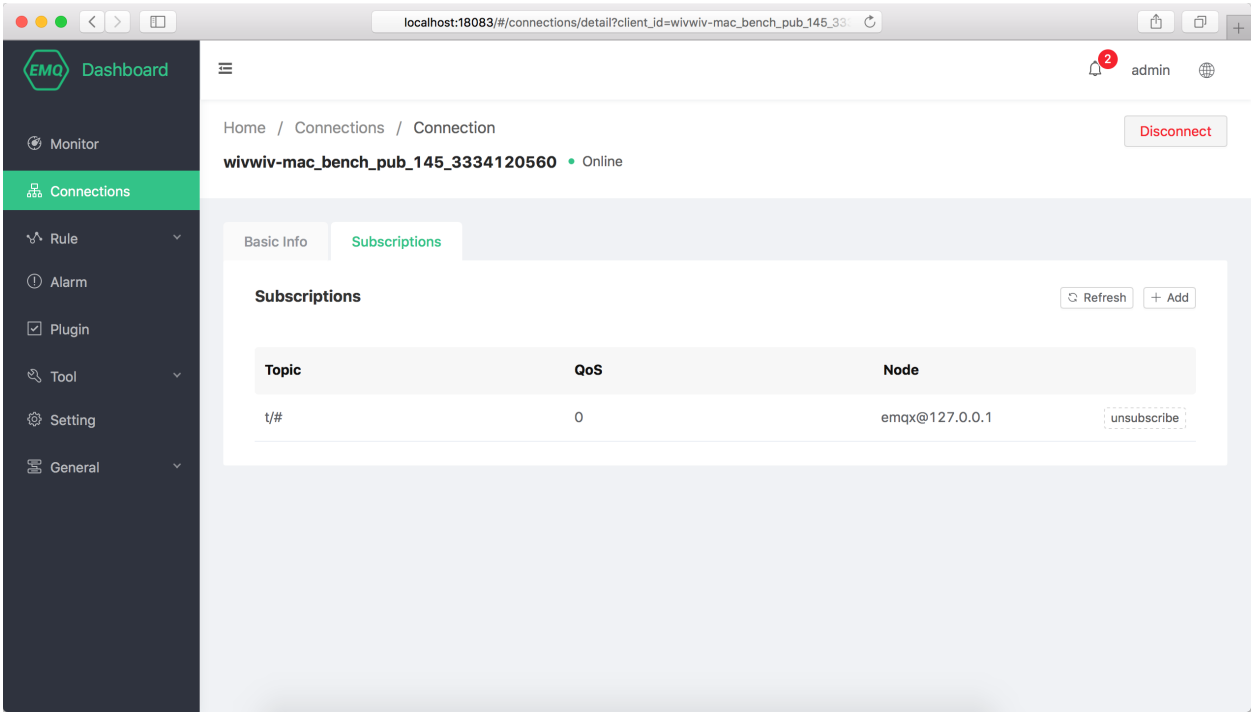
Click the **Client ID** to view the client details and subscription list. The basic information includes the selected client connection info and session info and includes key business information such as message traffic and message statistics.



6.4.3 Subscription

The subscription contains information about the topics to which the selected client has subscribed:

- Unsubscribe: Clicking the Unsubscribe button will delete the subscription relationship between the device and the topic. This operation is insensitive to the device.
- Add: Specify a topic for the selected client proxy subscription.



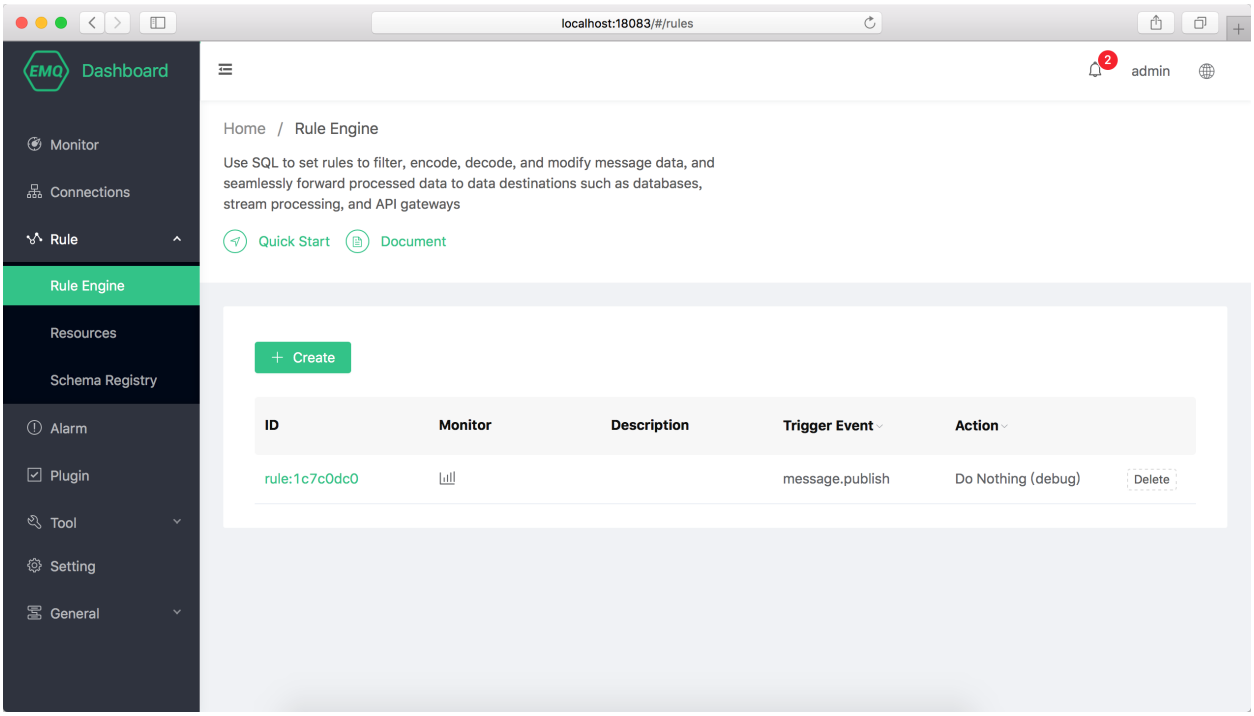
6.5 Rule

6.5.1 Rule Engine

Use SQL to set rules to filter, encode, decode, and modify message data, and seamlessly forward processed data to data destinations such as databases, stream processing, and API gateways.

The Rule Engine not only provides a clear and flexible configurable business integration solution, but also simplifies the business development process, improves user usability, and reduces the degree of coupling between business systems and EMQ X. Excellent infrastructure.

- ID: Unique ID within the cluster, which can be used in CLI and REST API.
- Topic: The MQTT topic or EMQ X event topic that the Rule matches.
- Monitor: Click to display the execution statistics of the selected Rule, including the number of rule hits and executions, and the number of success/failed actions triggered.



6.5.2 Create Rule

EMQ X will trigger the Rule Engine when the message is published and the event is triggered, and the rules meeting the triggering conditions will execute their respective SQL statements to filter and process the context information of the message and event.

With the Actions, the Rule Engine can store the message processing results of a specified topic to the database, send them to the HTTP Server, forward them to the Kafka or RabbitMQ, and republish them to a new topic or another broker cluster like Azure IoT Hub. Each rule can allocate multiple Actions.

1. Select the messages published to t/# and select all fields:

```
SELECT * FROM "message.publish" WHERE topic =~ 't/#'
```

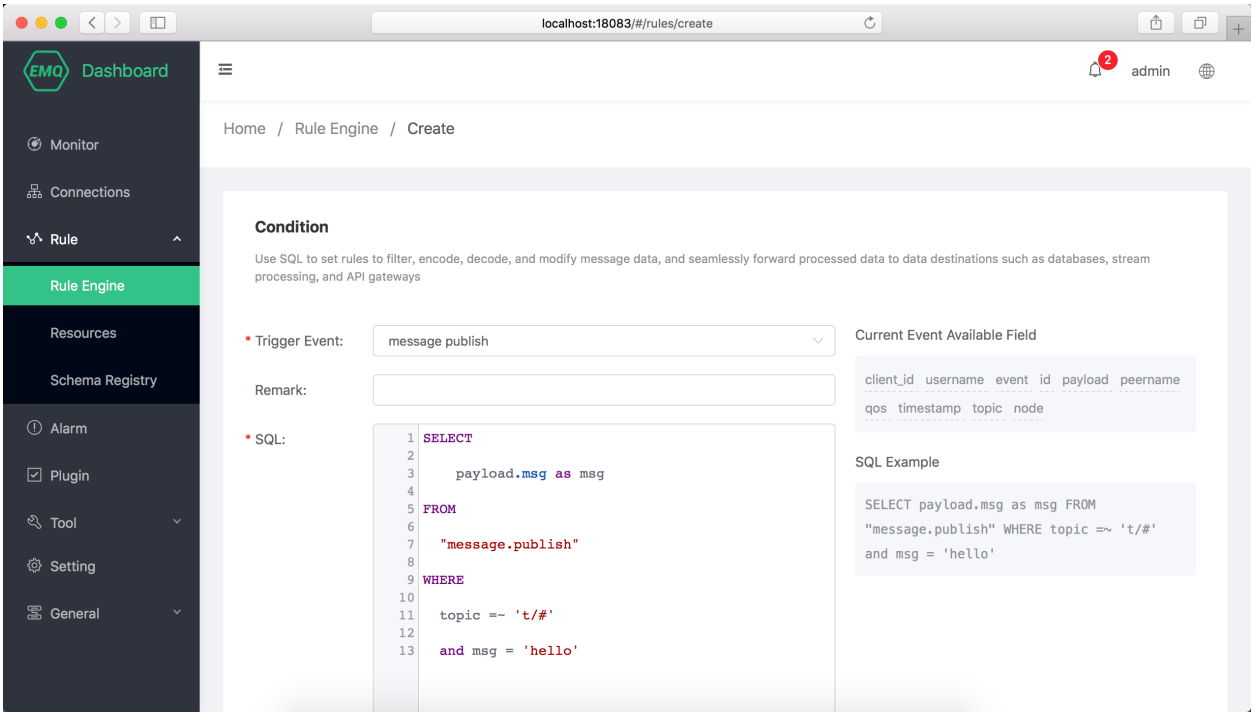
2. Select the message published to the t/a topic, and select the "x" field from the message payload in JSON format:

```
SELECT payload.x as x FROM "message.publish" WHERE topic =~ 't/a'
```

The Rule Engine uses the **Events** to process the built-in events of EMQ X. the built-in events provide more sophisticated message control and client action processing capabilities, which can be used in the message arrival records of QoS 1 and QoS 2, the device up and down line records and other businesses.

1. Select the client connected event, filter the device with Username 'emqx' and select the connection information:

```
SELECT clientid, connected_at FROM "client.connected" WHERE username = 'emqx'
```

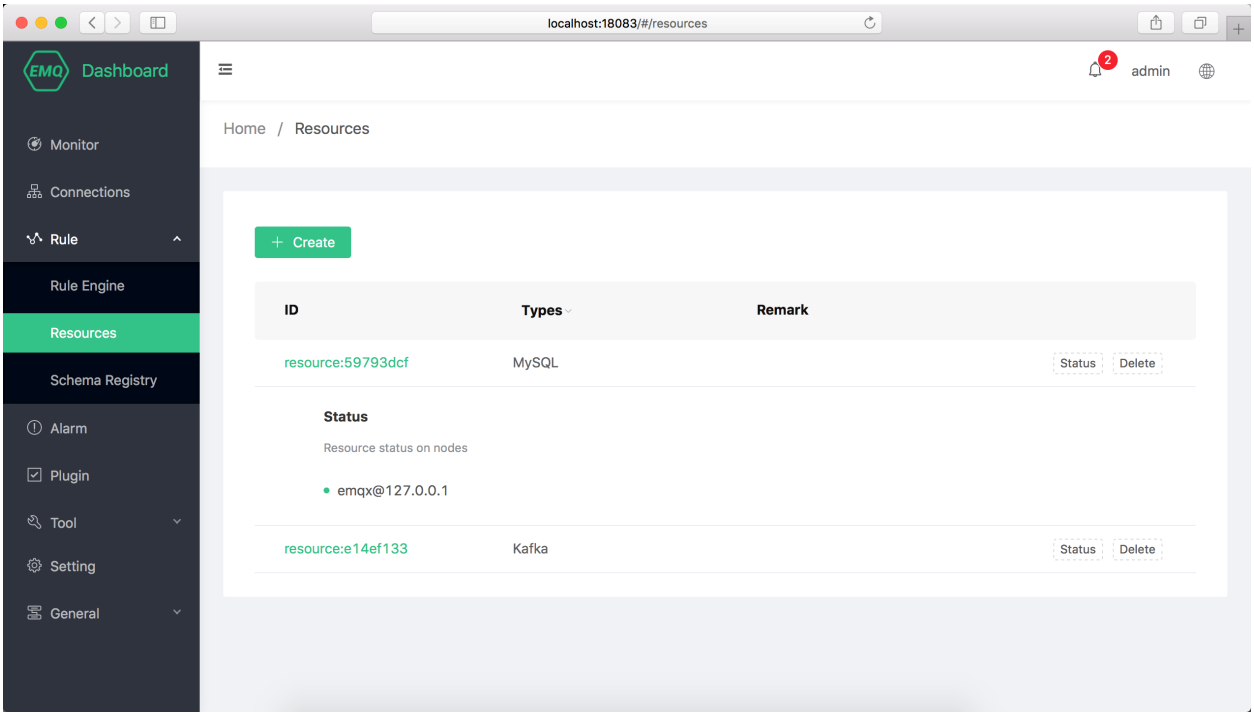


6.6 Resource

The resource instances (such as database instance and Web Server) required by the Rule Engine action. Before creating a rule, you need to create the resources required for the relevant action and ensure that the resources are available.

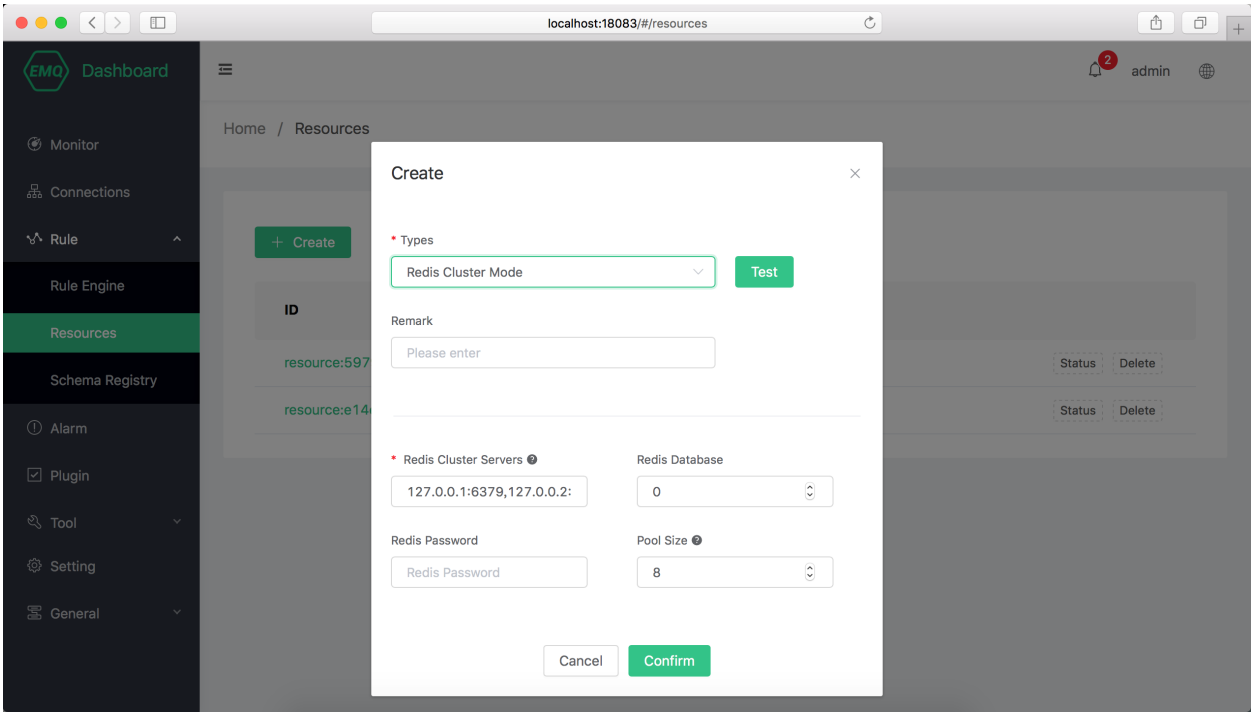
6.6.1 Resource list

- ID: Unique ID within the cluster, which can be used in CLI and REST API.
- Status: After the resource is created, each node in the cluster will establish a connection with the resource, click to expand the resource status on the node.
- Delete: The resources being used by the Rule Engine cannot be deleted. Please delete the rules that depend on the selected resource before deleting.



6.6.2 Create Resource

Click the **Create** to open the resource creation dialog. Select the resource type and enter the corresponding connection information to create the resource. Click **Test** to check the resource connectivity before creation.

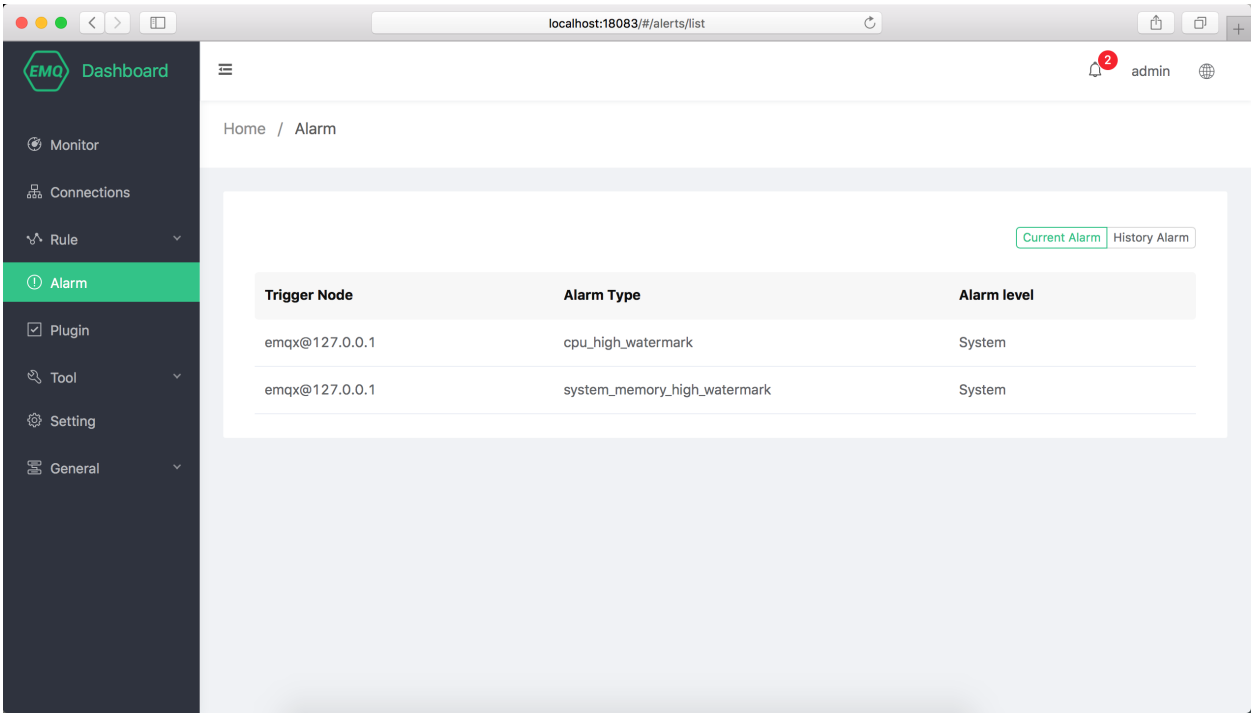


6.7 Schema Registry

Schema Registry supports Protobuf, Avro, and private message encoding parsing and processing, and can implement complex operations like message encryption, message compression, and binary-JSON message conversion.

6.8 Alarm

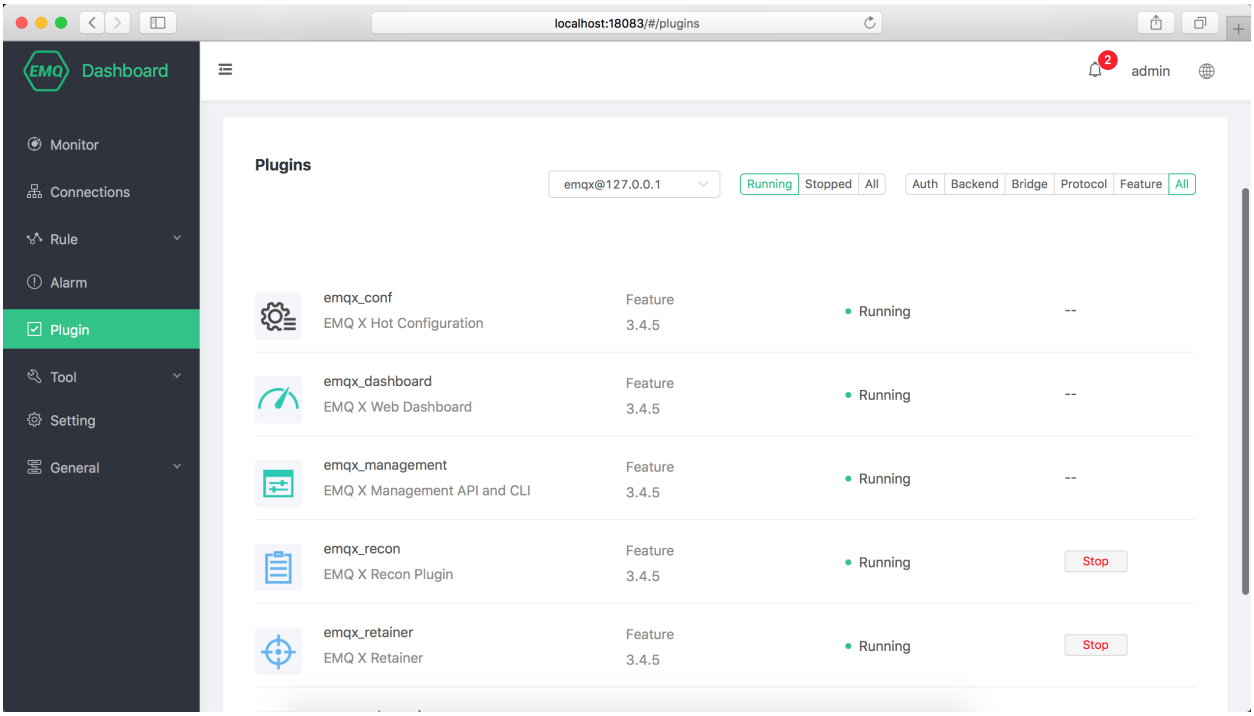
The alarm shows the basic alarm information of EMQ X, including current alarm and historical alarm. More advanced alarm, log and monitoring management is provided by EMQ X Control Center, please contact EMQ technicians if necessary.



6.9 Plugin

View the list of EMQ X built-in plugins.

Unlike the command line plugin management, the plugin starts and stop operations on the Dashboard are synchronized to the cluster. If the plugin fails to start, check whether the configuration of each node in the cluster is correct. If any node fails to start, the plugin cannot be successfully started.



6.10 Tool

It provides MQTT over WebScoket client test tool, which can realize the publish and subscribe test of multiple mqtt connections at the same time.

6.11 Setting

Provides parameter configuration for the EMQ X cluster and supports hot configuration. You can join and leave the cluster on the Dashboard.

6.11.1 Basic

Some basic configuration items that can be hot updated in`emqx.conf` are opened in the settings. You can complete most configuration items such as whether to enable anonymous authentication, ACL cache events, and ACL cache switches without restarting EMQ X.

The basic settings are organized in zones. By default, the external zone is associated with the listener on port 1883.

_static/images/dashboard/image-20200201165159596.png

6.11.2 Cluster

The cluster setting cannot change the cluster mode, but it can be used for manual cluster invitation nodes to join the cluster, and change the cluster parameter parameters such as static cluster and DNS cluster.

6.12 General

6.12.1 Application

In order to invoke the certificate of REST API, the application can query and adjust EMQ X cluster information through REST API, and manage and operate the equipment.

After the application is created successfully, click the Application ID in the **AppID** column of the application list to view the AppID and Secret. You can edit the application status and expiration time, and create or delete an application.

6.12.2 Users

Dashboard user account management, you can create, edit, delete users, if you forget the user password, you can reset the password through CLI.

CHAPTER 7

Rule Engine

7.1 EMQ X Rule Engine Introduction

Messages and events can be handled flexibly by the rule engine of EMQ X. Using the rule engine, it is convenient to implement such things as converting a message to a specified format, then saving it to a database table, or sending it to a message queue.

The concepts associated with the EMQ X rules engine include rule, action, resource, and resource-type.

relationship between rule, action and resource.

```
Rule: {
    SQL statement,
    Action list: [
        {
            Action1,
            Action parameter,
            Binding resource: {
              Resource configuration
            }
        },
        {
            Action2,
            Action parameter,
            Binding resource: {
                Resource configuration
            }
        }, ...
    ]
}
```

- Rule: Rule consists of SQL statements and action list. SQL statement is used to filter or transform data in events. An action is a task that is performed after a SQL statement is matched. The action list contains one or more actions and their parameters.
- Action: Action defines an operation for data. Actions can be bound to resources or not. For example, the "inspect" action does not need to bind resources, but simply prints the data content and action parameters. The "data_to_webserver" action needs to bind a resource of type web_hook with a URL configured in it.
- Resource: Resource is an object instantiated by a resource type as a template, and stores resource-related configurations (such as database connection address and port, user name, and password).
- Resource Type: Resource type is a static definition of a resource that describes the configuration items required for this type of resource.

: Action and resource type are provided by the code of emqx or plugins and cannot be dynamically created through the API and CLI.

7.2 SQL Statement

7.2.1 SQL Syntax

SQL statement is used to filter out the fields from the original data according to the conditions and perform preprocessing and conversion. The basic format is:

```
SELECT <field name> FROM <trigger event> [WHERE <condition>]
```

FROMSELECT and WHERE clause:

- `FROM` clause mounts the rule to a trigger event, such as "message publish", "connection completed", "connection disconnected", etc.
- `SELECT` clause is used to filter or convert fields in an event.
- `WHERE` clause is used to filter events based on criteria.

7.2.2 SQL Statement Example:

- Extract all fields from the message with topic "t/a":

```
SELECT * FROM "message.publish" WHERE topic = 't/a'
```

- All fields extracted from message of 't/#' can be matched by topic. Note that the **'=~'** operator is used here for topic matching with wildcards:

```
SELECT * FROM "message.publish" WHERE topic =~ 't/#'
```

- THe fields of qosusername and client_id extracted from message of 't/#' can be matched by topic:

```
SELECT qos, username, client_id FROM "message.publish" WHERE topic =~ 't/
↪#'
```

- Extract the username field from any topic message with the filter condition of username = 'Steven':

```
SELECT username FROM "message.publish" WHERE username='Steven'
```

- Extract the x field from the payload of message with any topic and create the alias x for use in the WHERE clause. The WHERE clause is restricted as x = 1. Note that the payload must be in JSON format. Example: This SQL statement can match the payload {"x": 1}, but not to the payload {"x": 2}:

```
SELECT payload.x as x FROM "message.publish" WHERE x=1
```

- Similar to the SQL statement above, but extract the data in the payload nestedly, and this SQL statement can match the payload {"x": {"y": 1}}:

```
SELECT payload.x.y as a FROM "message.publish" WHERE a=1
```

- When client_id = 'c1' tries to connect, extract its source IP address and port number:

```
SELECT peername as ip_port FROM "client.connected" WHERE client_id = 'c1'
```

- Filter all client_ids that subscribe to the 't/#' topic and subscription level is QoS1. Note that the strict equality operator **'='** is used here, so it does not match subscription requests with the subject 't' or 't/+/a':

```
SELECT client_id FROM "client.subscribe" WHERE topic = 't/#' and qos = 1
```

- In fact, the topic and qos fields in the above example are aliases that are set for ease of use when the subscription request contains only one pair (Topic, QoS). However, if the Topic Filters in the subscription request contain multiple (Topic, QoS) combination pairs, then the contains_topic() or contains_topic_match() function must be explicitly used to check if the Topic Filters contain the specified (Topic, QoS):

```
SELECT client_id FROM "client.subscribe" WHERE contains_topic(topic_
↪filters, 't/#')

SELECT client_id FROM "client.subscribe" WHERE contains_topic(topic_
↪filters, 't/#', 1)
```

:

- The trigger event after the FROM clause needs to be enclosed in double quotes `""`.
- The WHERE clause is followed by the filter condition. If the string is used, it needs to be enclosed in single quotes `''`.
- In the SELECT clause, if the `"."` symbol is sed to nest the payload, the payload must be in JSON format.

7.2.3 Trigger Event Available for the FROM Clause

Event name	Interpretation
message.publish	Message publish
message.deliver	Message delivery
message.acked	Message acked
message.dropped	Message dropped
client.connected	Connect
client.disconnected	Disconnect
client.subscribe	Subscribe
client.unsubscribe	Unsubscribe

7.2.4 Fields Available for the SELECT Clause

The fields available to the SELECT clause are related to the type of trigger event, where `client_id`, `username` and `event` are common fields that is available for each event type .

message.publish

client_id	Client ID
username	User name
event	Fixed to "message.publish"
id	MQTT message ID
topic	MQTT topic
payload	MQTT payload
peername	Client IPAddress and Port
qos	MQTT message QoS
timestamp	Timestamp

message.deliver

client_id	Client ID
username	User name
event	Fixed to "message.deliver"
id	MQTT message ID
topic	MQTT topic
payload	MQTT payload
peername	Client IPAddress and Port
qos	MQTT message QoS
timestamp	Timestamp
auth_result	Authentication result
mountpoint	Message topic mountpoint

message.acked

client_id	Client ID
username	User name
event	Fixed to "message.acked"
id	MQTT message ID
topic	MQTT topic
payload	MQTT payload
peername	Client IPAddress and Port
qos	MQTT message QoS
timestamp	Timestamp

message.dropped

client_id	Client ID
username	User name
event	Fixed to "message.dropped"
id	MQTT message ID
topic	MQTT topic
payload	MQTT payload
peername	Client IPAddress and Port
qos	MQTT message QoS
timestamp	Timestamp
node	Node name

client.connected

client_id	Client ID
username	User name
event	Fixed to "client.connected"
auth_result	Authentication result
clean_start	MQTT clean start Flag position
connack	MQTT CONNACK result
connected_at	Connection timestamp
is_bridge	Bridge or not
keepalive	MQTT keepalive interval
mountpoint	Message topic mountpoint
peername	Client IPAddress and Port
proto_ver	MQTT Protocol version

client.disconnected

client_id	Client ID
username	User name
event	Fixed to "client.disconnected"
auth_result	Authentication result
mountpoint	Message topic mountpoint
peername	Client IPAddress and Port
reason_code	Reason code for disconnection

client.subscribe

client_id	Client ID
username	User name
event	Fixed to "client.subscribe"
auth_result	Authentication result
mountpoint	Message topic mountpoint
peername	Client IPAddress and Port
topic_filters	MQTT subscription list
topic	MQTT first topic of subscription list
Qos	MQTT first QoS of subscription list

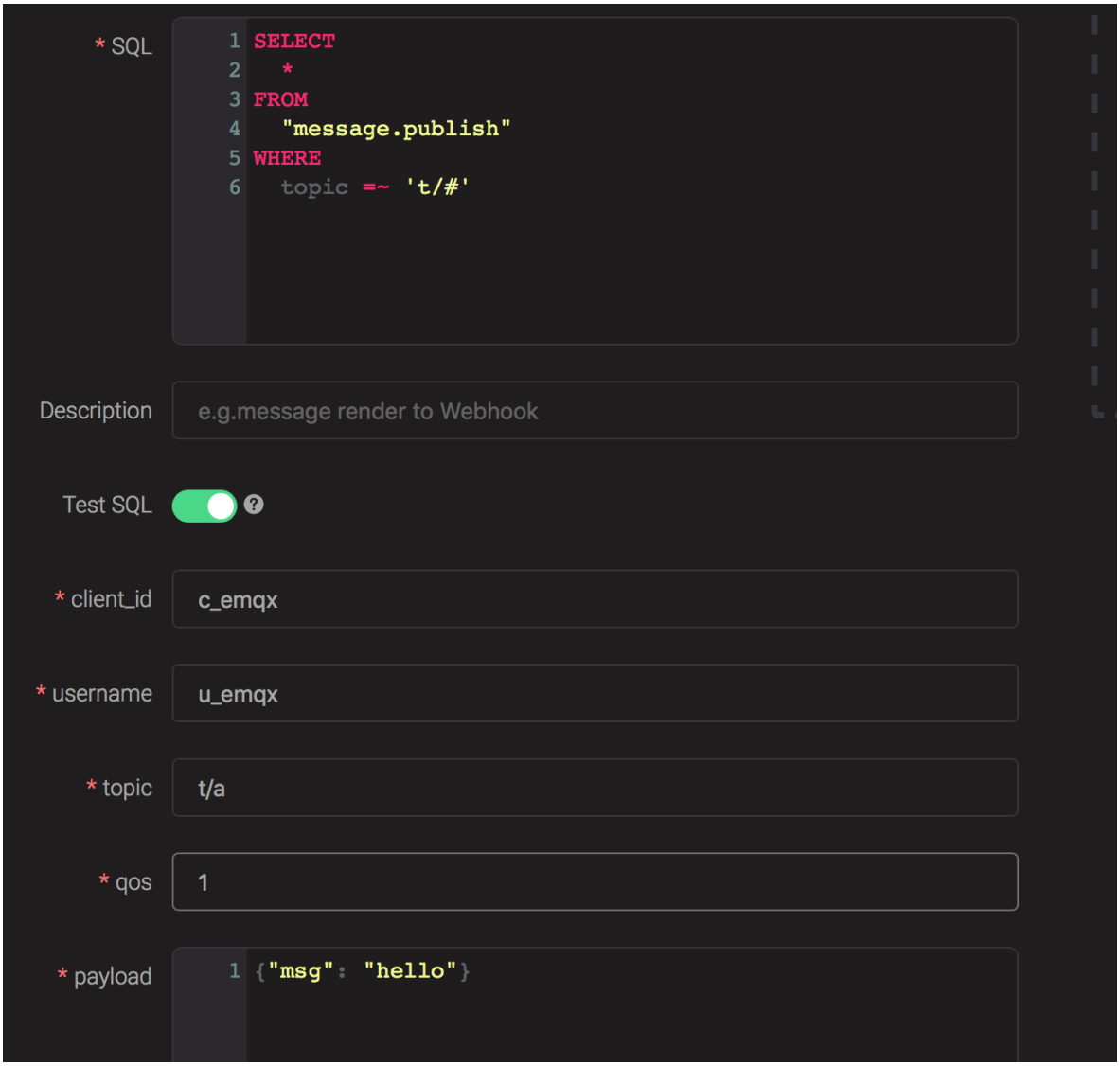
client.unsubscribe

client_id	Client ID
username	User name
event	Fixed to "client.unsubscribe"
auth_result	Authentication result
mountpoint	Message topic mountpoint
peername	Client IPAddress and Port
topic_filters	MQTT subscription list
topic	MQTT first topic of subscription list
QoS	MQTT first QoS of subscription list

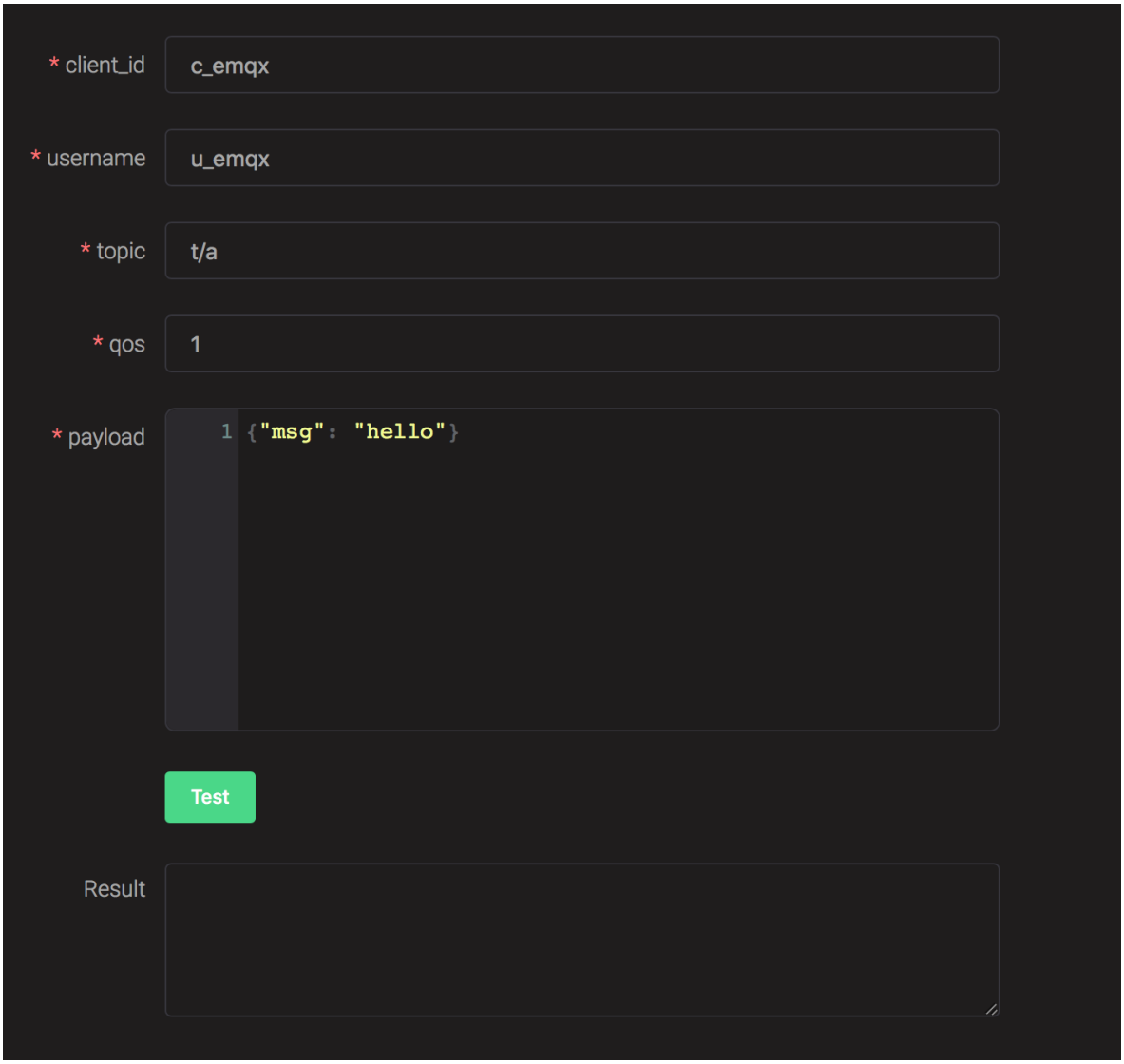
7.2.5 Test SQL Statements in Dashboard

The Dashboard interface provides SQL statement testing capabilities to present SQL test results with given SQL statements and event parameters.

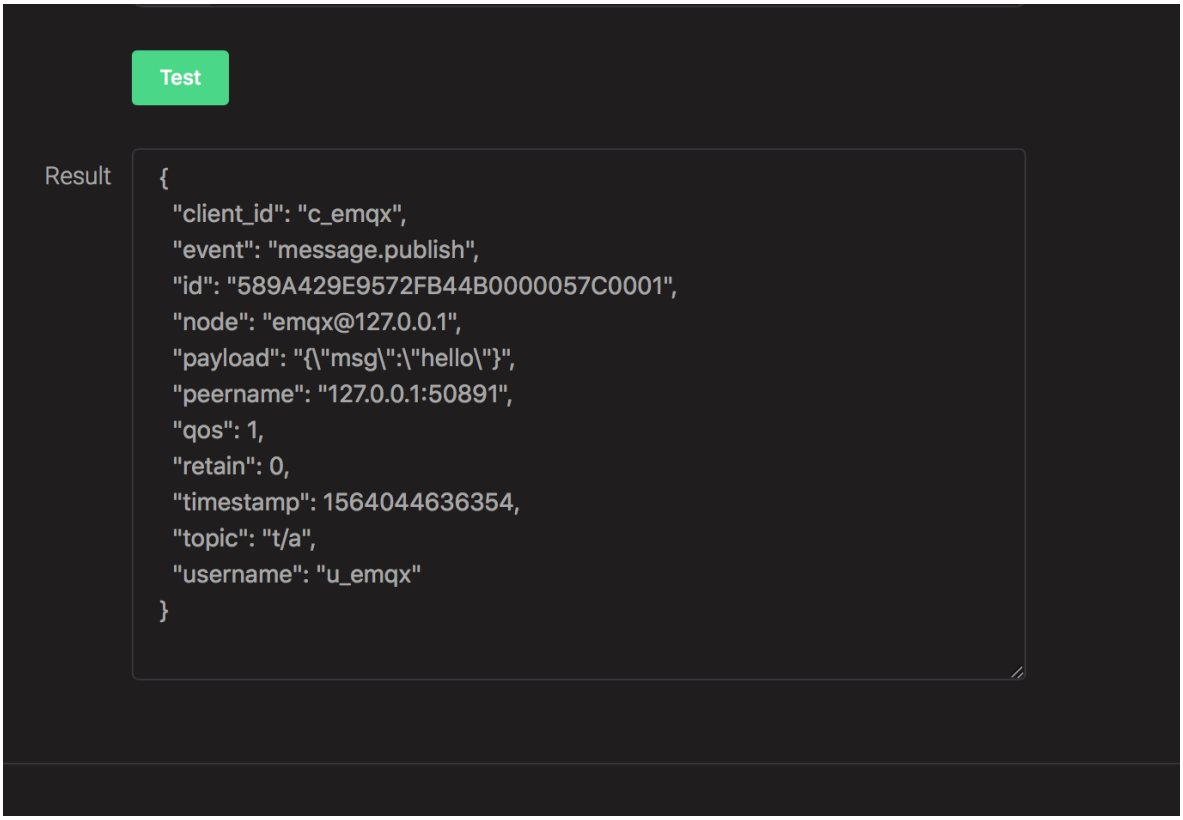
1. In the Rule Creation interface, enter **Rules SQL** and enable the **SQL Test** switch:



2. Modify the fields of the simulated event, or use the default configuration, click the **Test** button:



3. The processed results of the SQL will be displayed in the **Test Output** text box:



7.3 Rule Engine Management Commands and HTTP API

7.3.1 Rule Engine Command

Rules Command

rules list	List all rules
rules show <RuleId>	Show a rule
emqx_ctl rules create <sql> <actions> [-d [<descr>]]	Create a rule
rules delete <RuleId>	Delete a rule

rules create

Create a new rule. The parameters are as follows:

- <sql>: rule SQL
- <actions>: Action list in JSON format
- -d <descr>: Optional, rule description information

Example:

```
## Create a test rule that simply prints all message content sent to the 't/a
↪' topic
$ ./bin/emqx_ctl rules create \
  'select * from "message.publish"' \
  '[{"name":"inspect", "params": {"a": 1}}]' \
  -d 'Rule for debug'

Rule rule:9a6a725d created
```

The above example creates a rule with the ID `rule:9a6a725d`. There is only one action in the action list whose name is inspect, and the action parameter is `{"a": 1}`.

rules list

List all current rules:

```
$ ./bin/emqx_ctl rules list

rule(id='rule:9a6a725d', for='['message.publish']', rawsql='select * from
↪"message.publish"', actions=[{"metrics":...,"name":"inspect","params":...}],
↪ metrics=..., enabled='true', description='Rule for debug')
```

rules show

Query rule:

```
## Query the rule with RuleID 'rule:9a6a725d'
$ ./bin/emqx_ctl rules show 'rule:9a6a725d'

rule(id='rule:9a6a725d', for='['message.publish']', rawsql='select * from
↪"message.publish"', actions=[{"metrics":...,"name":"inspect","params":...}],
↪ metrics=..., enabled='true', description='Rule for debug')
```

rules delete

Delete rule:

```
## Delete rule with RuleID 'rule:9a6a725d'
$ ./bin/emqx_ctl rules delete 'rule:9a6a725d'

ok
```

Rule Actions Command

rule-actions list [-k [<eventype>]]	List actions
rule-actions show <ActionId>	Show a rule action

: Actions can be built in by emqx (called system built-in actions) or written by the emqx plugin, but they cannot be added or removed via the CLI/API.

rule-actions show

Query action:

```
## Query the action named 'inspect'
$ ./bin/emqx_ctl rule-actions show 'inspect'

action(name='inspect', app='emqx_rule_engine', for='$any', types=[], title =
↪'Inspect (debug)', description='Inspect the details of action params for␣
↪debug purpose')
```

rule-actions list

List eligible actions:

```
## List all current actions
$ ./bin/emqx_ctl rule-actions list

action(name='data_to_rabbit', app='emqx_bridge_rabbit', for='$any',␣
↪types=[bridge_rabbit], title ='Data bridge to RabbitMQ', description='Store␣
↪Data to Kafka')
action(name='data_to_timescaledb', app='emqx_backend_pgsql', for='$any',␣
↪types=[timescaledb], title ='Data to TimescaleDB', description='Store data␣
↪to TimescaleDB')
...

## List all actions that match the EventType type 'client.connected'
## '$any' indicates that this action can be bound to all types of events.
$ ./bin/emqx_ctl rule-actions list -k 'client.connected'

action(name='data_to_cassa', app='emqx_backend_cassa', for='$any',␣
↪types=[backend_cassa], title ='Data to Cassandra', description='Store data␣
↪to Cassandra')
action(name='data_to_dynamo', app='emqx_backend_dynamo', for='$any',␣
↪types=[backend_dynamo], title ='Data to DynamoDB', description='Store Data␣
↪to DynamoDB')
...
```

resources command

resources create <type> [-c [<config>]] [-d [<descr>]]	Create a resource
resources list [-t <ResourceType>]	List resources
resources show <ResourceId>	Show a resource
resources delete <ResourceId>	Delete a resource

resources create

Create a new resource with parameters as follows:

- type: Resource Type
- -c config: JSON format configuration
- -d descr: Optional, description of the resource

```
$ ./bin/emqx_ctl resources create 'web_hook' -c '{"url": "http://host-name/
↪chats"}' -d 'forward msgs to host-name/chats'

Resource resource:a7a38187 created
```

resources list

List all current resources:

```
$ ./bin/emqx_ctl resources list

resource(id='resource:a7a38187', type='web_hook', config=#{<<"url">> => <<
↪"http://host-name/chats">>}, status=#{is_alive => false}, description=
↪'forward msgs to host-name/chats')
```

resources list by type

List all current resources:

```
$ ./bin/emqx_ctl resources list --type='web_hook'

resource(id='resource:a7a38187', type='web_hook', config=#{<<"url">> => <<
↪"http://host-name/chats">>}, status=#{is_alive => false}, description=
↪'forward msgs to host-name/chats')
```

resources show

Query resource:

```
$ ./bin/emqx_ctl resources show 'resource:a7a38187'

resource(id='resource:a7a38187', type='web_hook', config=#{<<"url">> => <<
↪"http://host-name/chats">>}, status=#{is_alive => false}, description=
↪'forward msgs to host-name/chats')
```

resources delete

Delete resource:

```
$ ./bin/emqx_ctl resources delete 'resource:a7a38187'

ok
```

resource-types command

resource-types list	List all resource-types
resource-types show <Type>	Show a resource-type

: Resource types can be built in by emqx (called system built-in resource types) or written by the emqx plugin, but cannot be added or removed via the CLI/API.

resource-types list

List all current resource types:

```
./bin/emqx_ctl resource-types list

resource_type(name='backend_mongo_rs', provider='emqx_backend_mongo', title =
↪'MongoDB Replica Set Mode', description='MongoDB Replica Set Mode')
resource_type(name='backend_cassa', provider='emqx_backend_cassa', title =
↪'Cassandra', description='Cassandra Database')
...
```

resource-types show

Query resource type:

```
$ ./bin/emqx_ctl resource-types show backend_mysql

resource_type(name='backend_mysql', provider='emqx_backend_mysql', title =
↪'MySQL', description='MySQL Database')
```

7.3.2 Rule Engine HTTP API

Rule API

Create rule

API definition:

```
POST api/v3/rules
```

Parameter definition:

rawsql	StringSQL statement for filtering and converting raw data
actions	JSON Arrayaction list
• actions.name	String, action name
• actions.params	JSON Object, action parameter
description	Stringoptional, rule description

API request example:

```
GET http://localhost:8080/api/v3/rules
```

API request payload:

```
{
  "rawsql": "select * from \"message.publish\"",
  "actions": [{
      "name": "inspect",
      "params": {
          "a": 1
      }
  }],
  "description": "test-rule"
}
```

API returned data example:

```
{
  "code": 0,
  "data": {
      "actions": [{
          "name": "inspect",
          "params": {
              "a": 1
          }
```

()

()

```
      }],
      "description": "test-rule",
      "enabled": true,
      "for": "message.publish",
      "id": "rule:34476883",
      "rawsql": "select * from \"message.publish\""
  }
}
```

Query rule

API definition:

```
GET api/v3/rules/:id
```

API request example:

```
GET api/v3/rules/rule:34476883
```

API returned data example:

```
{
  "code": 0,
  "data": {
      "actions": [{
          "name": "inspect",
          "params": {
              "a": 1
          }
      }],
      "description": "test-rule",
      "enabled": true,
      "for": "message.publish",
      "id": "rule:34476883",
      "rawsql": "select * from \"message.publish\""
  }
}
```

Get the current rule list

API definition:

```
GET api/v3/rules
```

API returned data example:

```
{
  "code": 0,
  "data": [{
      "actions": [{
          "name": "inspect",
          "params": {
              "a": 1
          }
      }],
      "description": "test-rule",
      "enabled": true,
      "for": "message.publish",
      "id": "rule:34476883",
      "rawsql": "select * from \"message.publish\""
  }]
}
```

Delete rule

API definition:

```
DELETE api/v3/rules/:id
```

Request parameter example:

```
DELETE api/v3/rules/rule:34476883
```

API returned data example:

```
{
  "code": 0
}
```

Action API

Get the current action list

API definition:

```
GET api/v3/actions?for=${hook_type}
```

API request example:

```
GET api/v3/actions
```

API returned data example:

```
{
  "code": 0,
  "data": [{
      "app": "emqx_rule_engine",
      "description": "Republish a MQTT message to another topic",
      "for": "message.publish",
      "name": "republish",
      "params": {
          "target_topic": {
              "description": "To which topic the message will be republished",
              "format": "topic",
              "required": true,
              "title": "To Which Topic",
              "type": "string"
          }
      },
      "types": []
  }]
}
```


API request example:

```
GET 'api/v3/actions?for=client.connected'
```

API returned data example:


```
{
  "code": 0,
  "data": [{
      "app": "emqx_rule_engine",
      "description": "Inspect the details of action params for debug purpose",
      "for": "$any",
      "name": "inspect",
      "params": {},
      "types": []
  }]
}
```


Query action

API definition:

```
GET api/v3/actions/:action_name
```

API request example:

```
GET 'api/v3/actions/inspect'
```

API returned data example:

```
{
  "code": 0,
  "data": {
      "app": "emqx_rule_engine",
      "description": "Inspect the details of action params for debug purpose",
      "for": "$any",
      "name": "inspect",
      "params": {},
      "types": []
  }
}
```

Resource Type API

Get the current resource type list

API definition:

```
GET api/v3/resource_types
```

Returned data example:

```
{
  "code": 0,
  "data": [{
      "config": {
          "url": "http://host-name/chats"
      },
      "description": "forward msgs to host-name/chats",
      "id": "resource:a7a38187",
      "type": "web_hook"
  }]
}
```

Query resource type

API definition:

```
GET api/v3/resource_types/:type
```

Returned data example:

```
GET api/v3/resource_types/web_hook
```

```
{
  "code": 0,
  "data": {
```

()

()

```
      "description": "WebHook",
      "name": "web_hook",
      "params": {},
      "provider": "emqx_web_hook"
  }
}
```

Get certain kind of resource

API definition:

```
GET api/v3/resource_types/:type/resources
```

API request example:

```
GET api/v3/resource_types/web_hook/resources
```

API returned data example:

```
{
  "code": 0,
  "data": [{
      "config": {"url":"http://host-name/chats"},
      "description": "forward msgs to host-name/chats",
      "id": "resource:6612f20a",
      "type": "web_hook"
  }]
}
```

Resource API

Create resource

API definition:

```
POST api/v3/resources
```

API parameter definition:

type	String, resource type
config	JSON Object, resource configuration
description	StringOptional, rule description

API request parameter example:

```
{
  "type": "web_hook",
  "config": {
      "url": "http://127.0.0.1:9910",
      "headers": {"token":"axfw34y235wrq234t4ersgw4t"},
      "method": "POST"
  },
  "description": "web hook resource-1"
}
```

API returned data example:

```
{
  "code": 0,
  "data": {
      "config": {
          "headers":{"token":"axfw34y235wrq234t4ersgw4t"},
          "method":"POST",
          "url":"http://127.0.0.1:9910"
      },
      "description": "web hook resource-1",
      "id": "resource:62763e19",
      "type": "web_hook"
  }
}
```

Get resource list

API definition:

```
GET api/v3/resources
```

API returned data example:

```
{
  "code": 0,
  "data": [{
      "config": {
          "headers":{"token":"axfw34y235wrq234t4ersgw4t"},
          "method":"POST",
          "url":"http://127.0.0.1:9910"
      },
      "description": "web hook resource-1",
      "id": "resource:62763e19",
      "type": "web_hook"
  }]
}
```

Query resource

API definition:

```
GET api/v3/resources/:resource_id
```

API returned data example:

```
GET 'api/v3/resources/resource:62763e19'
```


```
{
  "code": 0,
  "data": {
      "config": {
          "headers":{"token":"axfw34y235wrq234t4ersgw4t"},
          "method":"POST",
          "url":"http://127.0.0.1:9910"
      },
      "description": "web hook resource-1",
      "id": "resource:62763e19",
      "type": "web_hook"
  }
}
```


Delete resource

API definition:

```
DELETE api/v3/resources/:resource_id
```

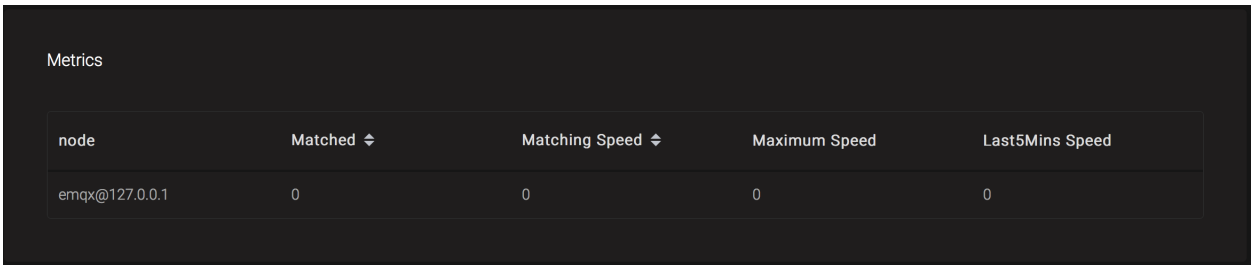
API returned data example:

```
DELETE 'api/v3/resources/resource:62763e19'
```


```
{
  "code": 0
}
```

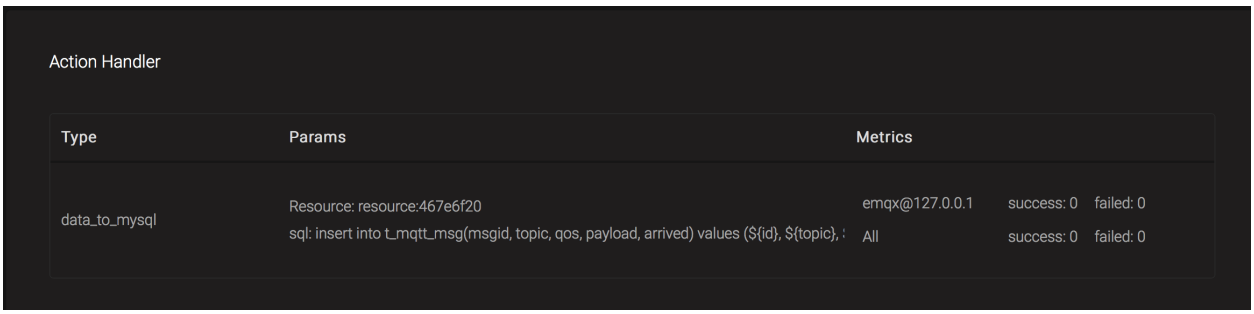
7.4 Status, Statistical Indicator, and Alerts Related to the Rules Engine

7.4.1 Rule Status and Statistical Indicators



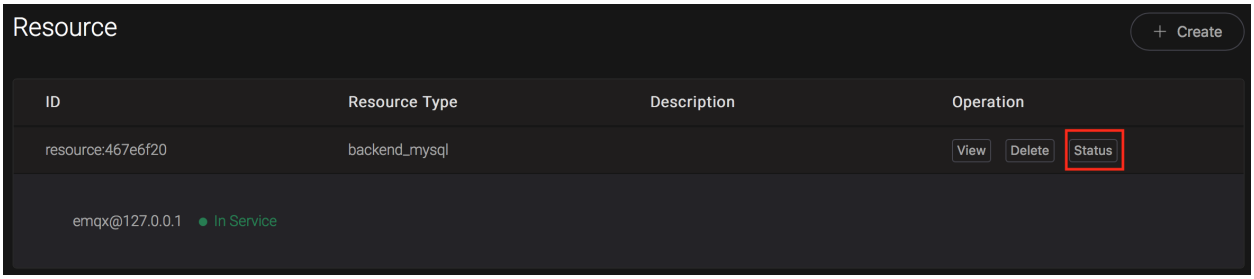
- Hit: The hit number of rule (rule SQL matches successfully),
- Hit speed: the speed of the rule hit (times / second)
- Maximum hit speed: The peak of the rule hit speed (times / second)
- Average speed in five minutes: average hit speed of the rule within 5 minutes (times/second)

7.4.2 Action Status and Statistical Indicators



- Success: Number of successful executions of action
- Failure: Number of failed executions of action

7.4.3 Resource Status and Alarm



- Available: Resources is available

- Not available: Resource is not available (such as database disconnection)

7.5 Example of Rule Creation

7.5.1 Create Database and Bridge Rules through the CLI

Create Inspect Rules

Create WebHook Rule

7.5.2 Create Database and Bridge Rules through DashBoard

Create MySQL Rules

Create PostgreSQL Rules

Create Cassandra Rules

Create MongoDB Rules

Create DynamoDB Rules

Create Redis Rules

Create OpenTSDB Rules

Create TimescaleDB Rules

Create InfluxDB Rules

Creat WebHook Rules

Create Kafka Rules

Create Pulsar Rules

Create RabbitMQ Rules

Create BridgeMQTT Rules

Create EMQX Bridge Rules

CHAPTER 8

Rule Engine Tutorial

8.1 Create MySQL Rules

0. Setup a MySQL database, and changes the username/password to root/public, taking Mac OSX for instance:

```
$ brew install mysql

$ brew services start mysql

$ mysql -u root -h localhost -p

  ALTER USER 'root'@'localhost' IDENTIFIED BY 'public';
```

1. Initiate MySQL table:

```
$ mysql -u root -h localhost -ppublic
```

   create "test" database:

```
CREATE DATABASE test;
```

   create "t_mqtt_msg" table:

```
USE test;

CREATE TABLE `t_mqtt_msg` (
`id` int(11) unsigned NOT NULL AUTO_INCREMENT,
`msgid` varchar(64) DEFAULT NULL,
`topic` varchar(255) NOT NULL,
```

()

()

```
`qos` tinyint(1) NOT NULL DEFAULT '0',
`payload` blob,
`arrived` datetime NOT NULL,
PRIMARY KEY (`id`),
INDEX topic_index(`id`, `topic`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8MB4;
```

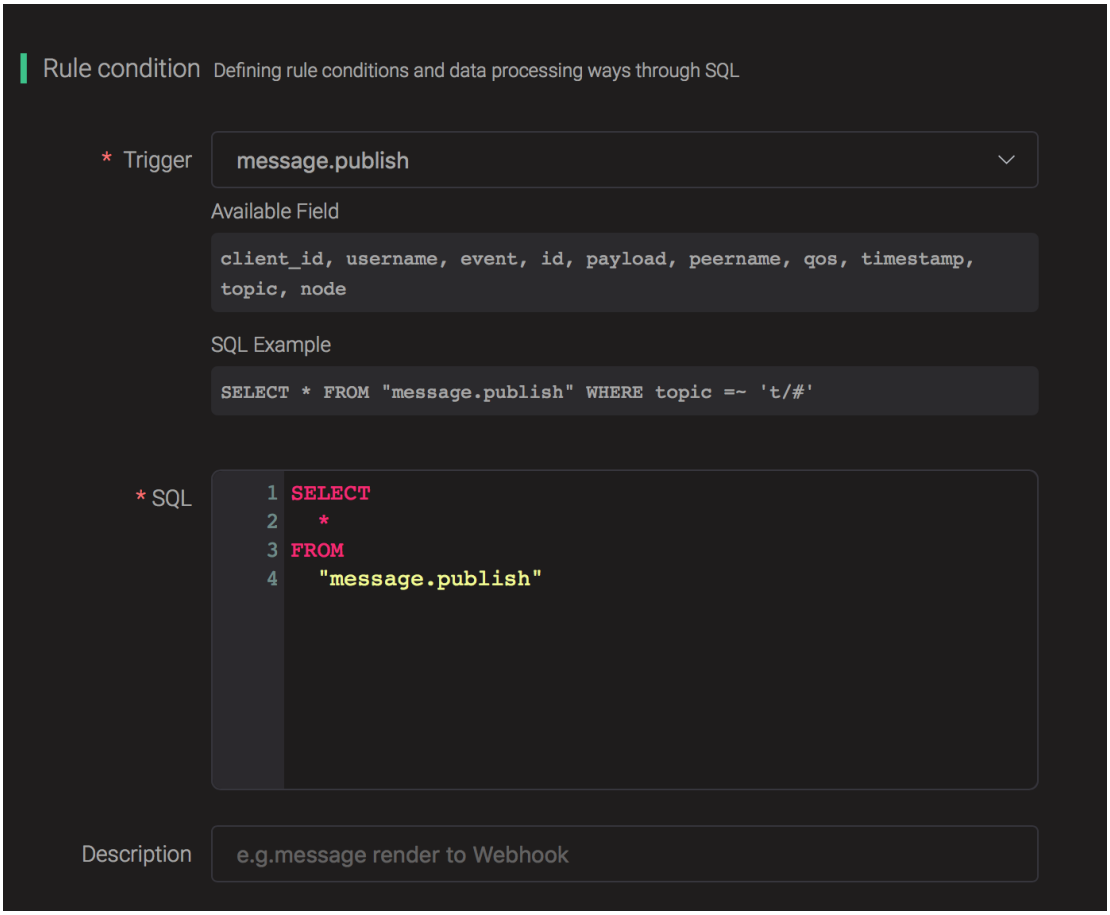
```
mysql> CREATE DATABASE test;
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> USE test;
Database changed
mysql> DROP TABLE IF EXISTS `t_mqtt_msg`;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE `t_mqtt_msg` (
    ->   `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
    ->   `msgid` varchar(64) DEFAULT NULL,
    ->   `topic` varchar(255) NOT NULL,
    ->   `qos` tinyint(1) NOT NULL DEFAULT '0',
    ->   `payload` blob,
    ->   `arrived` datetime NOT NULL,
    ->   PRIMARY KEY (`id`),
    ->   INDEX topic_index(`id`, `topic`)
    -> ) ENGINE=InnoDB DEFAULT CHARSET=utf8MB4;
Query OK, 0 rows affected (0.07 sec)

mysql> describe t_mqtt_msg;
+---------+------------------+------+-----+---------+----------------+
| Field   | Type             | Null | Key | Default | Extra          |
+---------+------------------+------+-----+---------+----------------+
| id      | int(11) unsigned | NO   | PRI | NULL    | auto_increment |
| msgid   | varchar(64)      | YES  |     | NULL    |                |
| topic   | varchar(255)     | NO   |     | NULL    |                |
| qos     | tinyint(1)       | NO   |     | 0       |                |
| payload | blob             | YES  |     | NULL    |                |
| arrived | datetime         | NO   |     | NULL    |                |
+---------+------------------+------+-----+---------+----------------+
6 rows in set (0.02 sec)
```

2. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

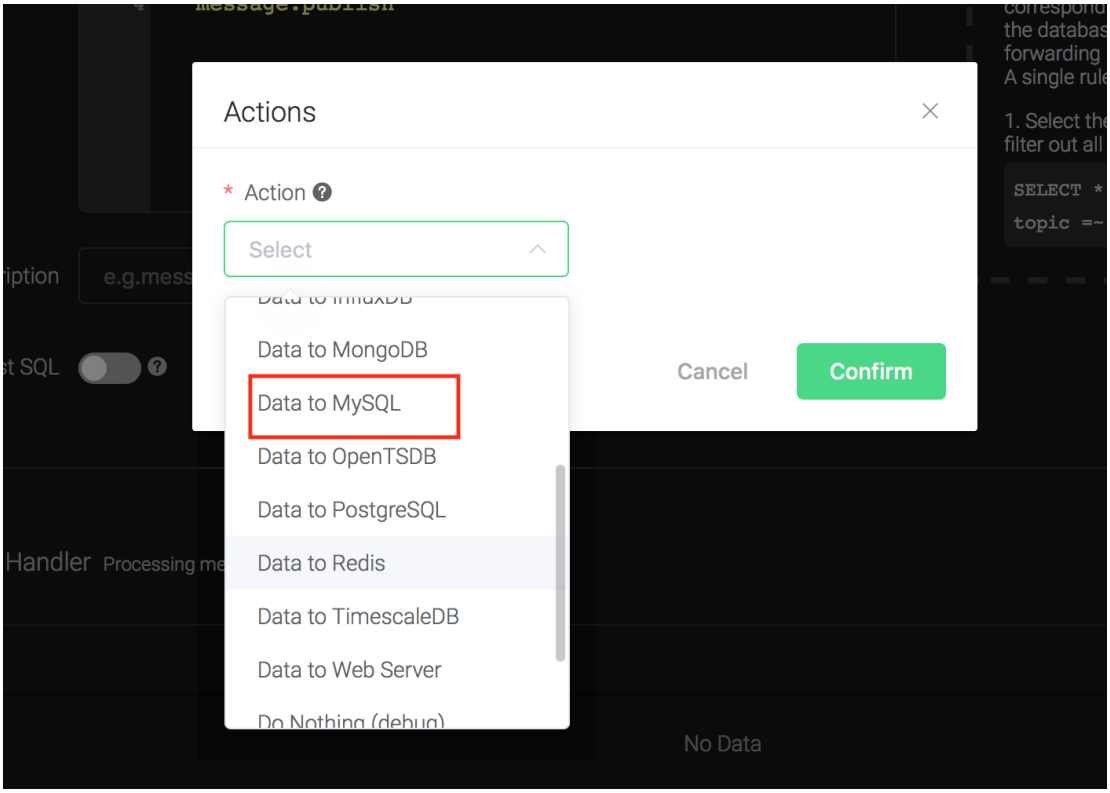
   Select "message.publish", then type in the following SQL:

```
SELECT * FROM "message.publish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to MySQL" in the pop-up dialog window.



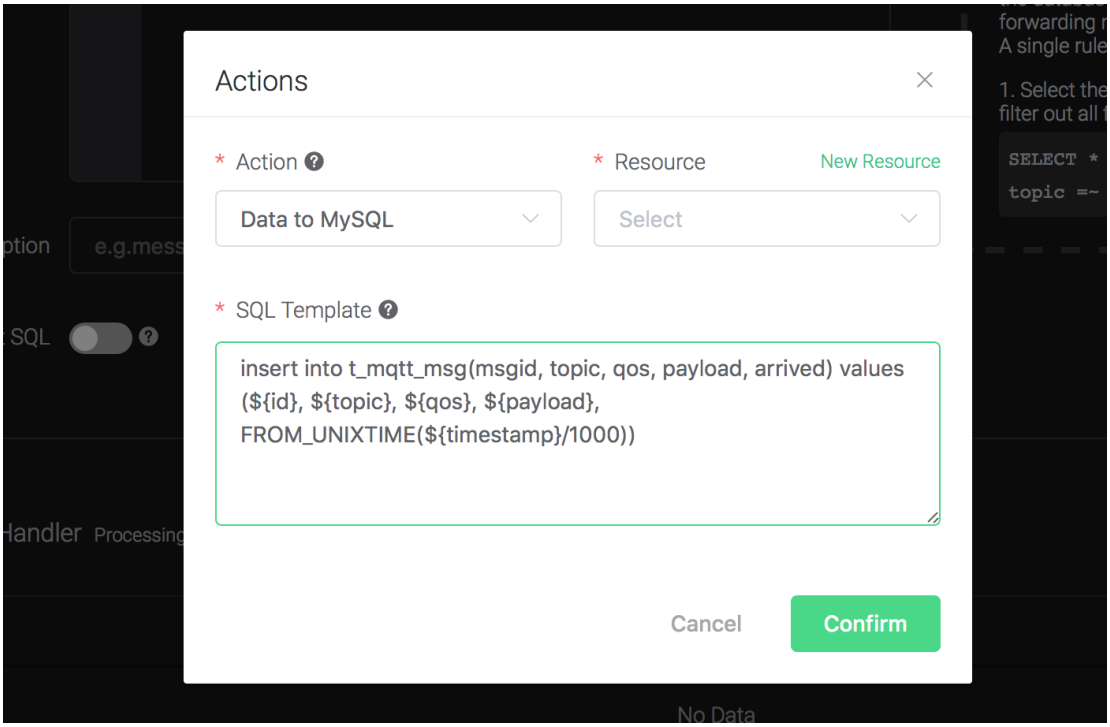
4. Fill in the parameters required by the action:

   Two parameters is required by action "Data to MySQL":

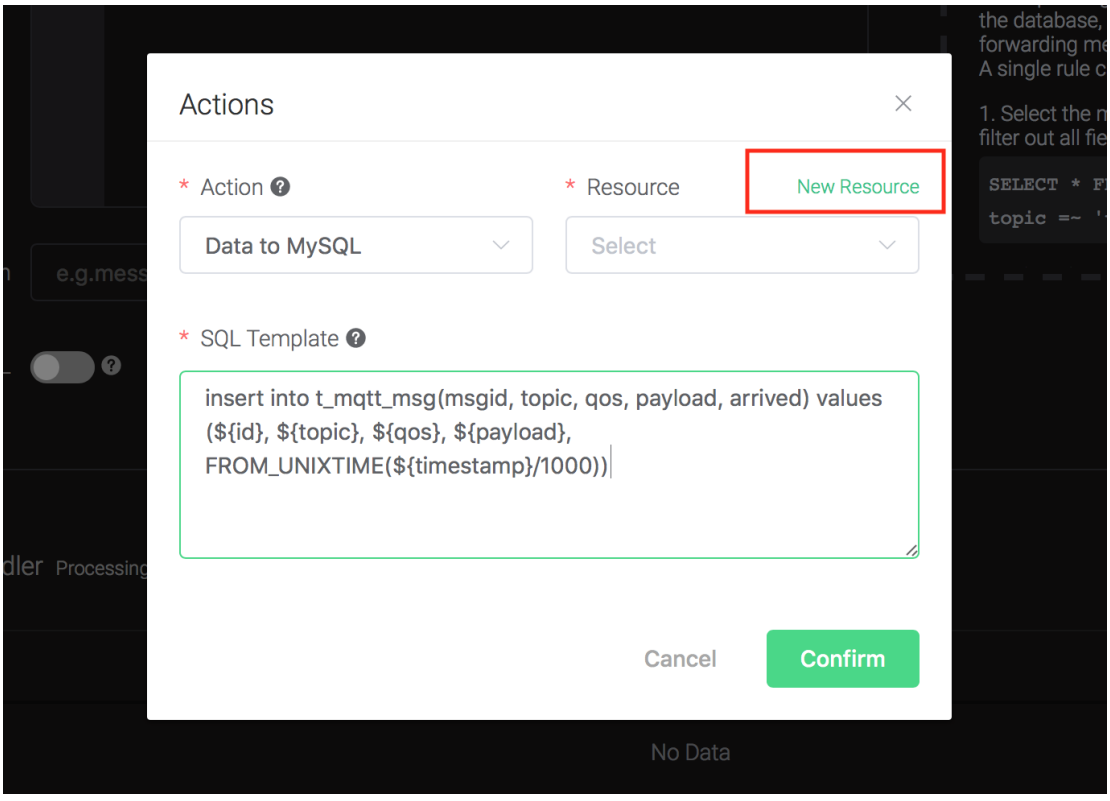
   1). SQL template. SQL template is the sql command you'd like to run when the action is triggered. In this example we'll insert a message into mysql, so type in the following sql template:

```
insert into t_mqtt_msg(msgid, topic, qos, payload, arrived) values ($
↪{id}, ${topic}, ${qos}, ${payload}, FROM_UNIXTIME(${timestamp}/
↪1000))
```

   Before data is inserted into the table, placeholders like ${key} will be replaced by the corresponding values.

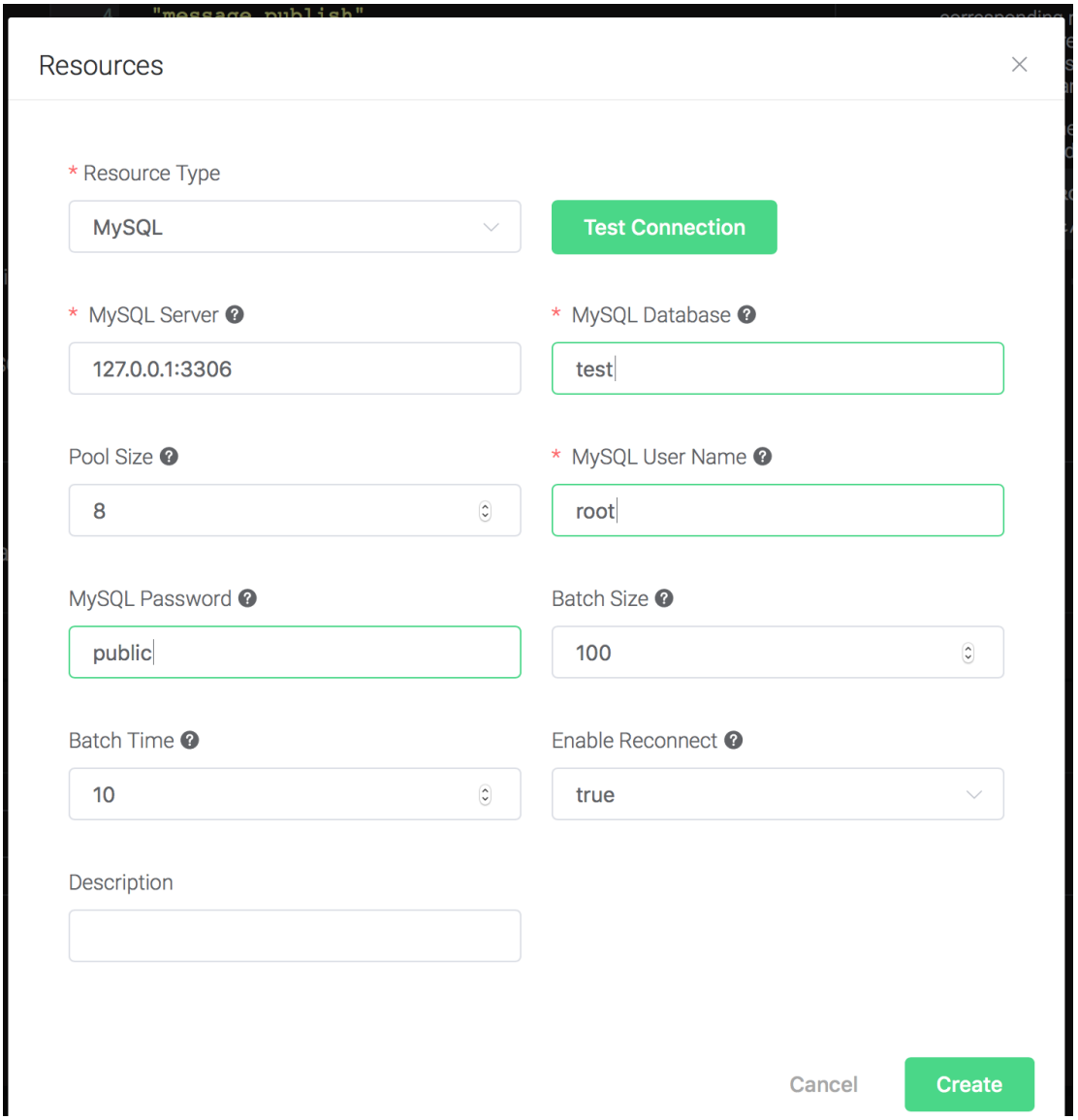


2). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "MySQL":

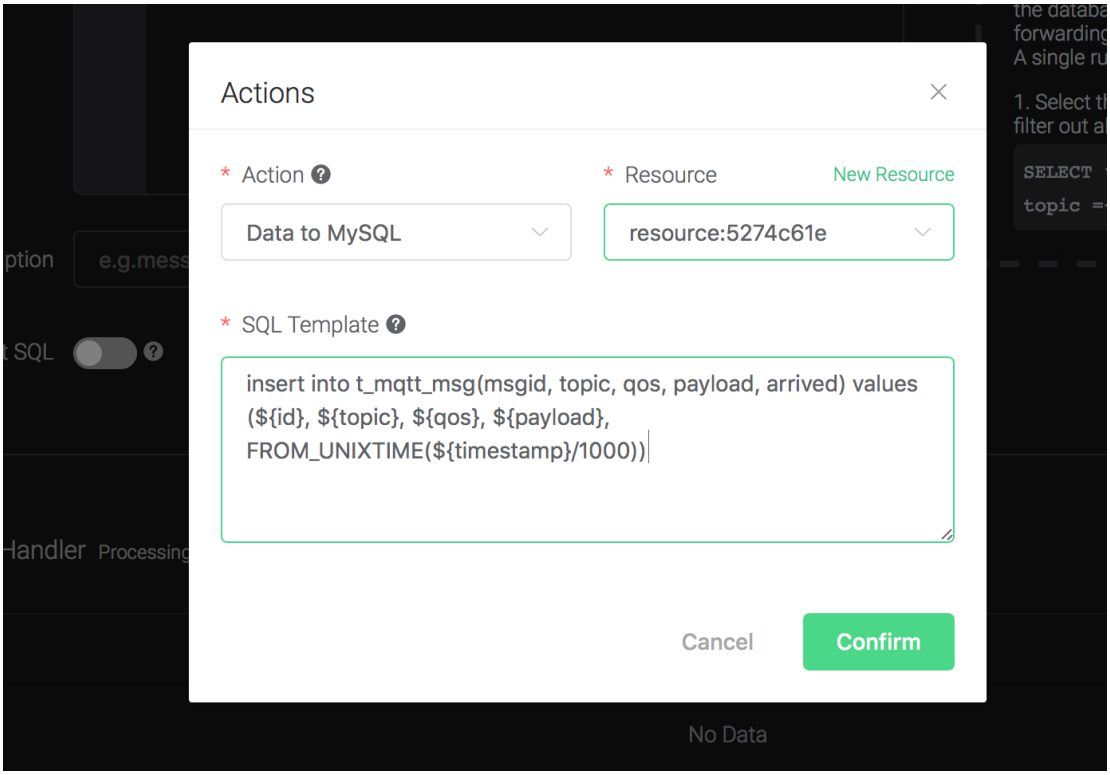


5. Configure the resource:

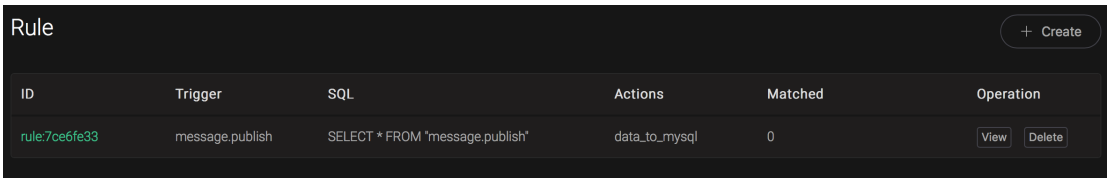
Set "MySQL Database" to "test", "MySQL Username" to "root", "MySQL Password" to "public", and "Description" to "MySQL resource to 127.0.0.1:3306 db=test", and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/a"

   QoS: 1

   Payload: "hello"

   Then inspect the MySQL table, verify a new record has been inserted:

```
mysql> select * from t_mqtt_msg;
Empty set (0.00 sec)

mysql>
mysql> select * from t_mqtt_msg;
+----+----------------------------+-------+-----+---------+---------------------+
| id | msgid                      | topic | qos | payload | arrived             |
+----+----------------------------+-------+-----+---------+---------------------+
|  1 | 589886299C168F442000008E50002 | t/a   |   1 | hello   | 2019-05-23 14:42:26 |
+----+----------------------------+-------+-----+---------+---------------------+
1 row in set (0.00 sec)

mysql>
```

8.2 Create PostgreSQL Rules

0. Setup a PostgreSQL database, taking Mac OSX for instance:

```
$ brew install postgresql

$ brew services start postgresql

## create root user
$ createuser --interactive --pwprompt
  Enter name of role to add: root
  Enter password for new role: public
  Enter it again: public
  Shall the new role be a superuser? (y/n) y

## create database named 'mqtt' using root
$ createdb -U root mqtt

$ psql -U root mqtt

  mqtt=> \dn;
  List of schemas
    Name  | Owner
  --------+-------
   public | shawn
  (1 row)
```

1. Initiate PgSQL table:

   $ psql -U root mqtt

   create `t_mqtt_msg` table:

```
CREATE TABLE t_mqtt_msg (
id SERIAL primary key,
msgid character varying(64),
sender character varying(64),
topic character varying(255),
```

()

()

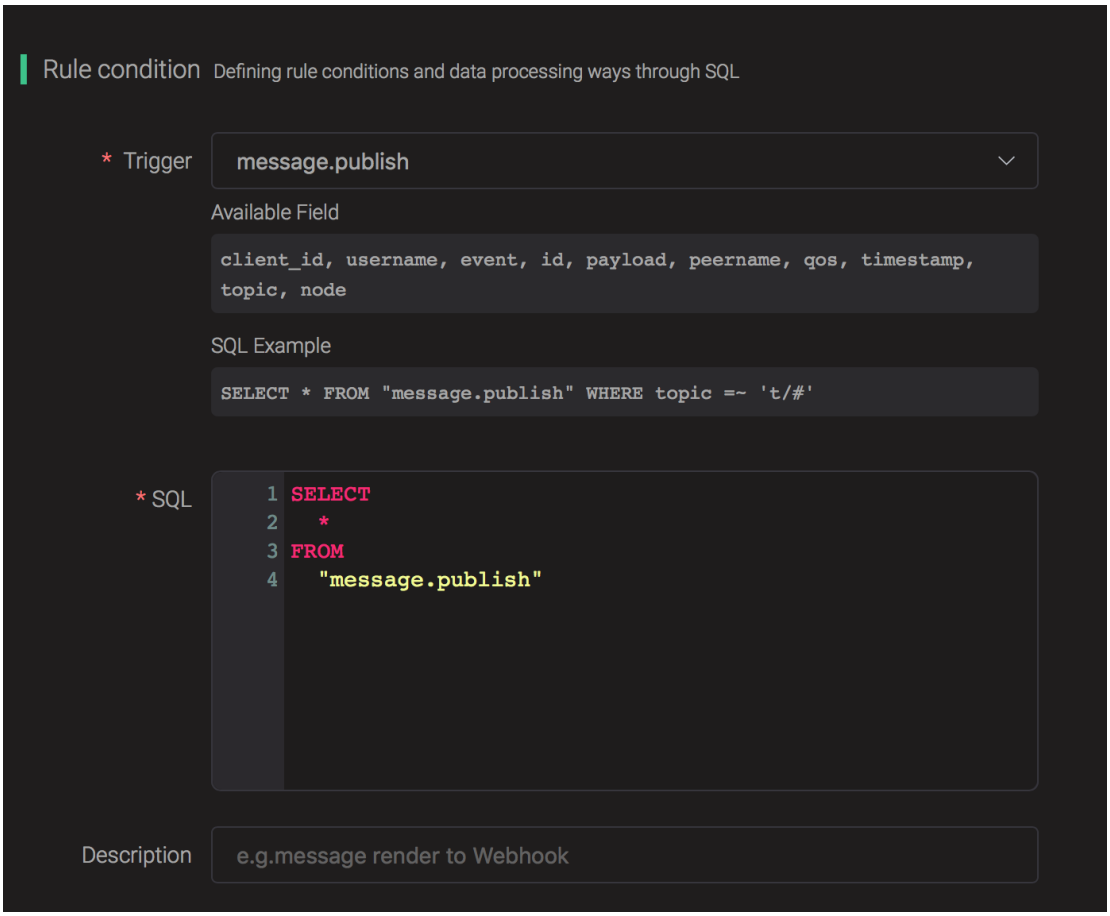
```
qos integer,
retain integer,
payload text,
arrived timestamp without time zone
);
```

2. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

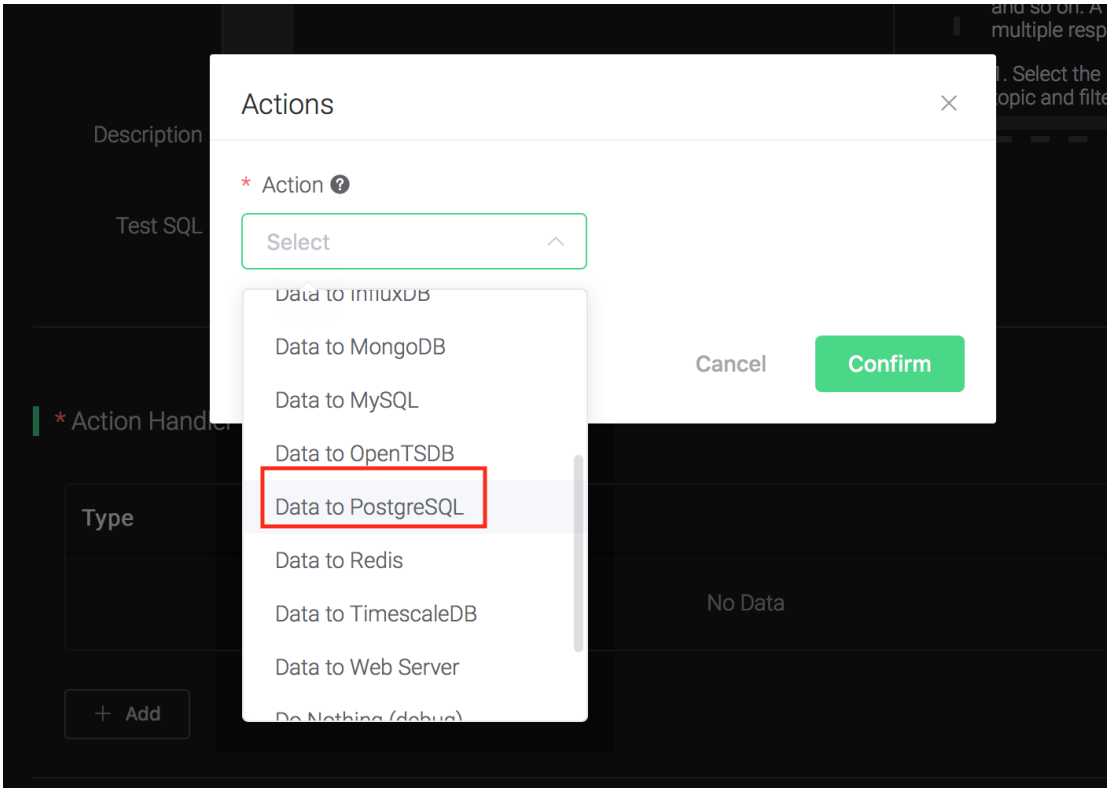
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to PostgreSQL" in the pop-up dialog window.



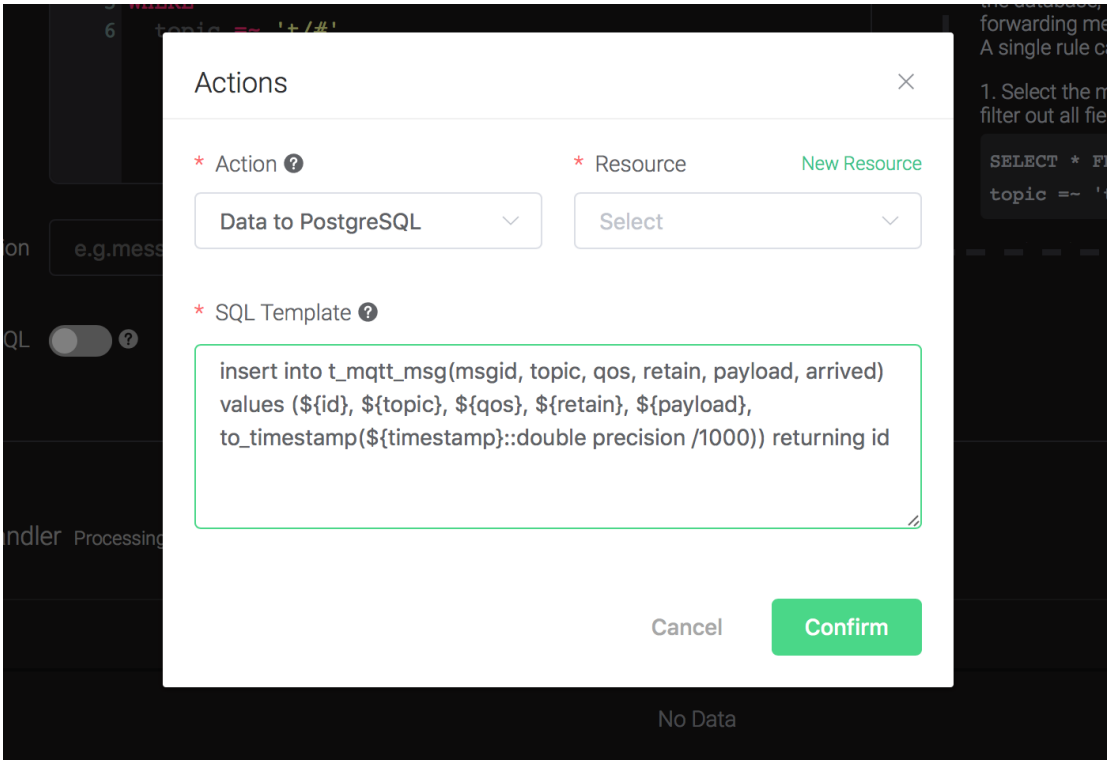
4. Fill in the parameters required by the action:

   Two parameters is required by action "Data to PostgreSQL":

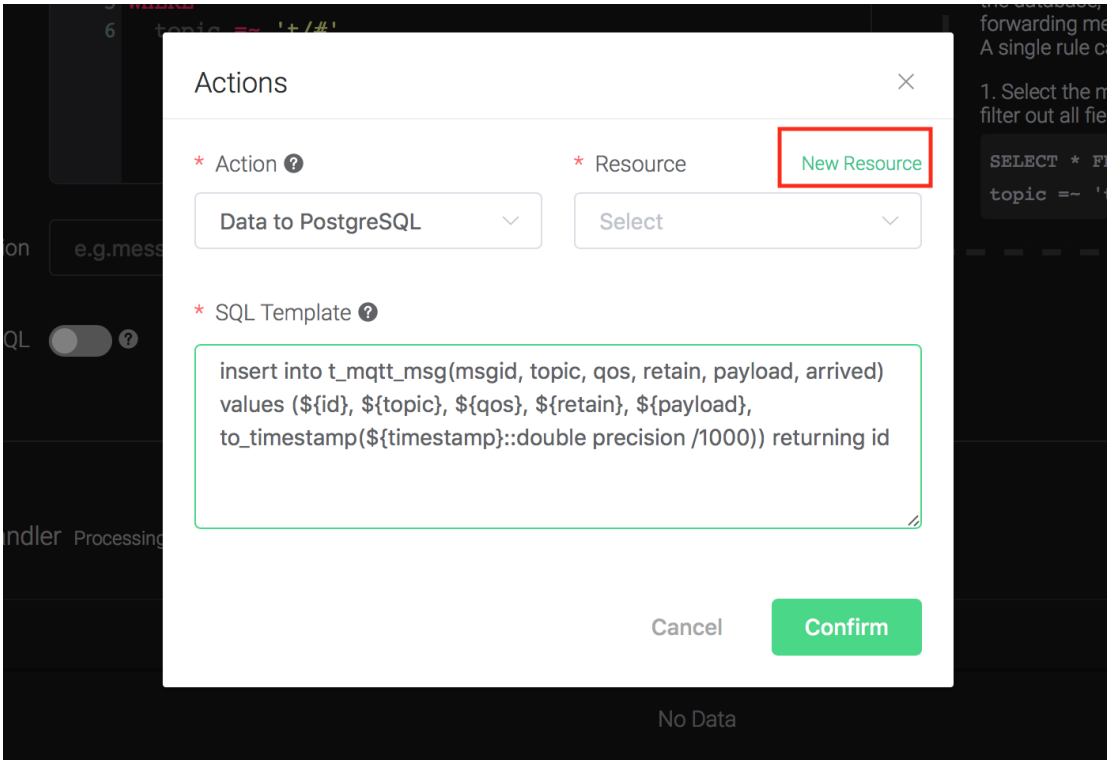
   1). SQL template. SQL template is the sql command you'd like to run when the action is triggered. In this example we'll insert a message into pgsql, so type in the following sql template:

```
insert into t_mqtt_msg(msgid, topic, qos, retain, payload, arrived)␣
↪values (${id}, ${topic}, ${qos}, ${retain}, ${payload}, to_
↪timestamp(${timestamp}::double precision /1000)) returning id
```

   Before data is inserted into the table, placeholders like ${key} will be replaced by the corresponding values.



2). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "PostgreSQL":

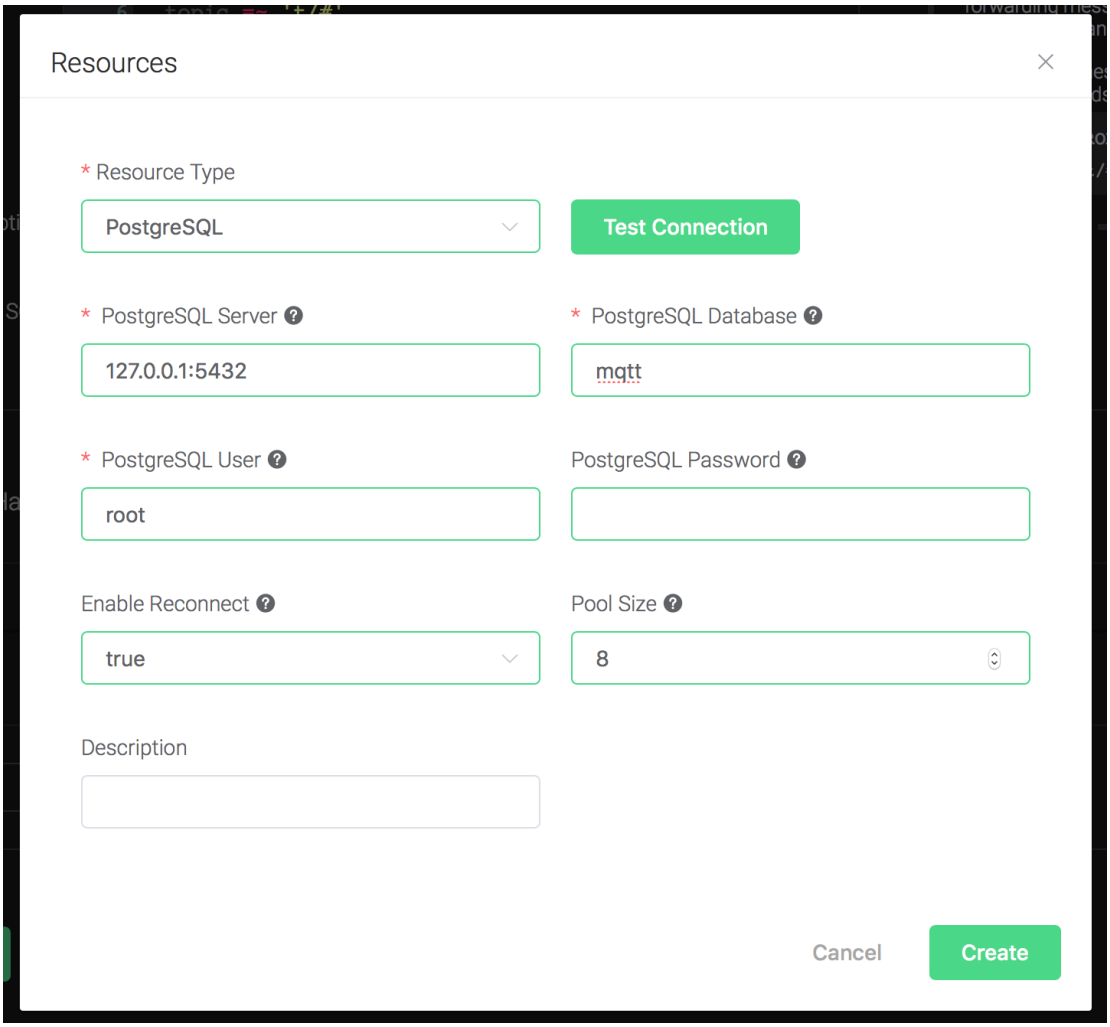


Select "PostgreSQL Resource".

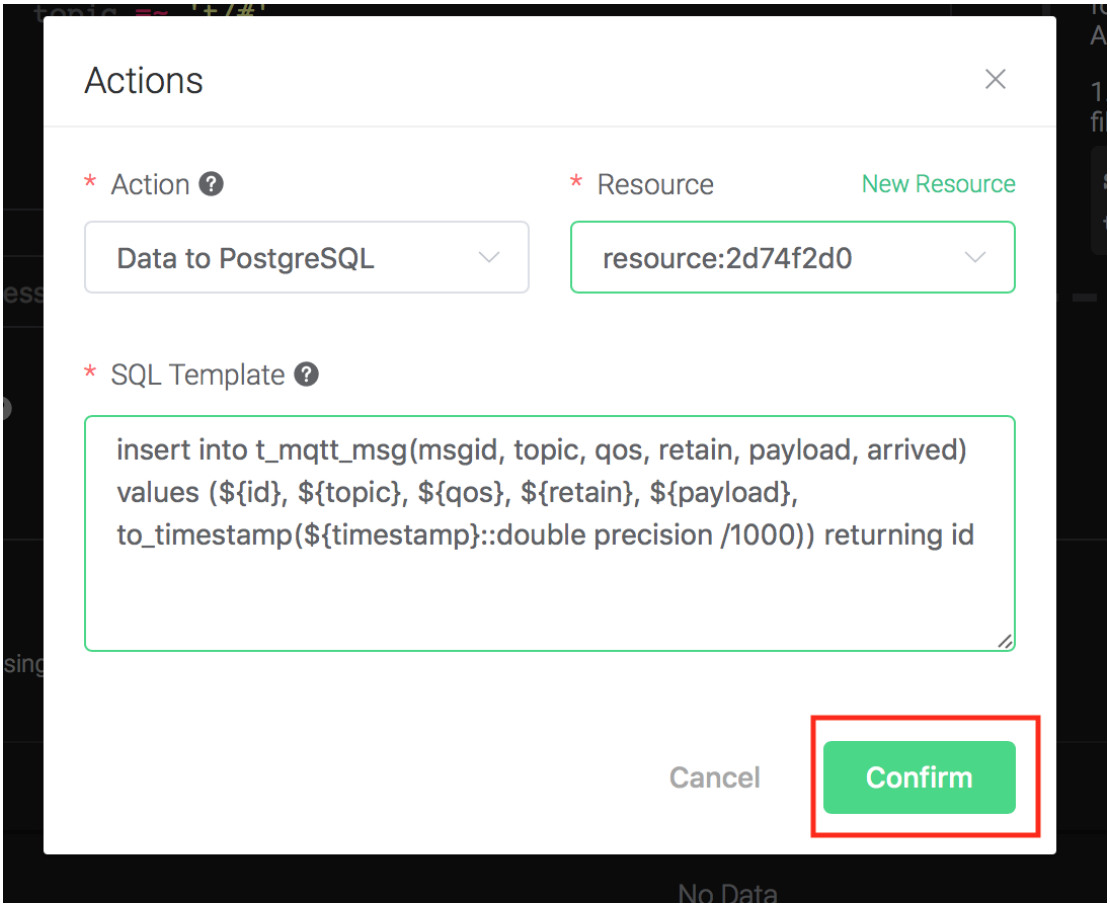
5. Configure the resource:

   Set "PostgreSQL Database" to "mqtt", "PostgreSQL User" to "root", and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully.

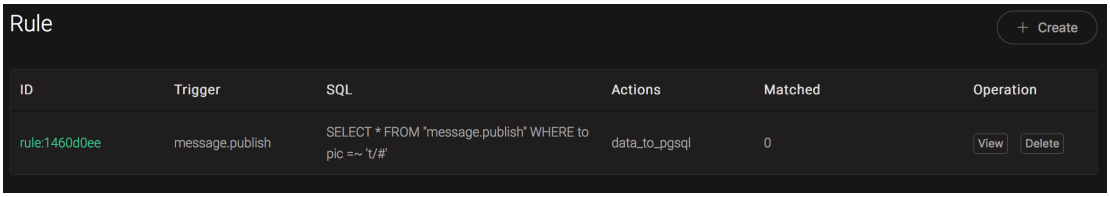
   Finally click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

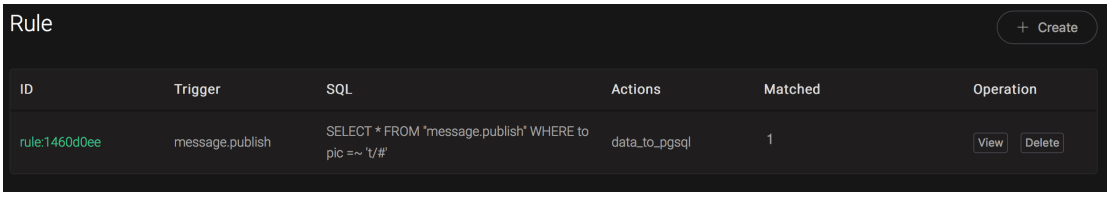
   Retained: false

   Payload: "hello1"

   Then inspect the PgSQL table, verify a new record has been inserted:

```
 id |             msgid              | sender | topic | qos | retain | payload |         arrived
----+--------------------------------+--------+-------+-----+--------+---------+-------------------------
  3 | 58C2292328ECBF442000008BC0001 |        | t/1   |   0 | f      | hello1  | 2019-06-25 17:29:53.117
(1 row)
```

   And from the rule list, verify that the "Matched" column has increased to 1:



8.3 Create Cassandra Rules

0. Setup a Cassandra database, and changes the root/password to root/public, taking Mac OSX for instance:

```
$ brew install cassandra

## change the config file to enable authentication
$  vim /usr/local/etc/cassandra/cassandra.yaml

   authenticator: PasswordAuthenticator
   authorizer: CassandraAuthorizer

$ brew services start cassandra

## login to cql shell and then create the root user
$ cqlsh -ucassandra -pcassandra

cassandra@cqlsh> create user root with password 'public' superuser;
```

1. Initiate Cassandra Table:

```
$ cqlsh -uroot -ppublic
```

Create Keyspace "test":

```
CREATE KEYSPACE test WITH replication = {'class': 'SimpleStrategy',
↪'replication_factor': '1'}  AND durable_writes = true;
```

Create "t_mqtt_msg" table:

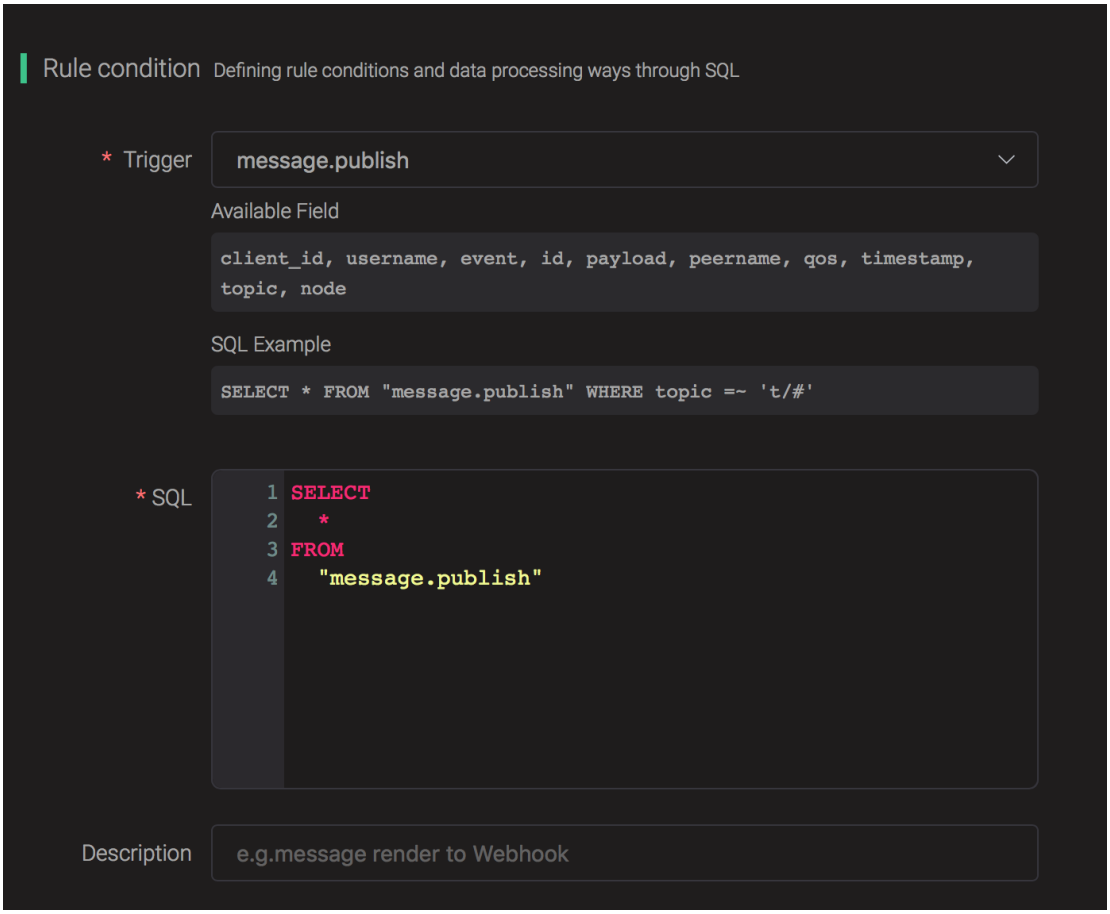
```
USE test;

CREATE TABLE t_mqtt_msg (
  msgid text,
  topic text,
  qos int,
  payload text,
  retain int,
  arrived timestamp,
  PRIMARY KEY (msgid, topic)
);
```

2. Create a rule:

Go to emqx dashboard, select the "rule" tab on the menu to the left.

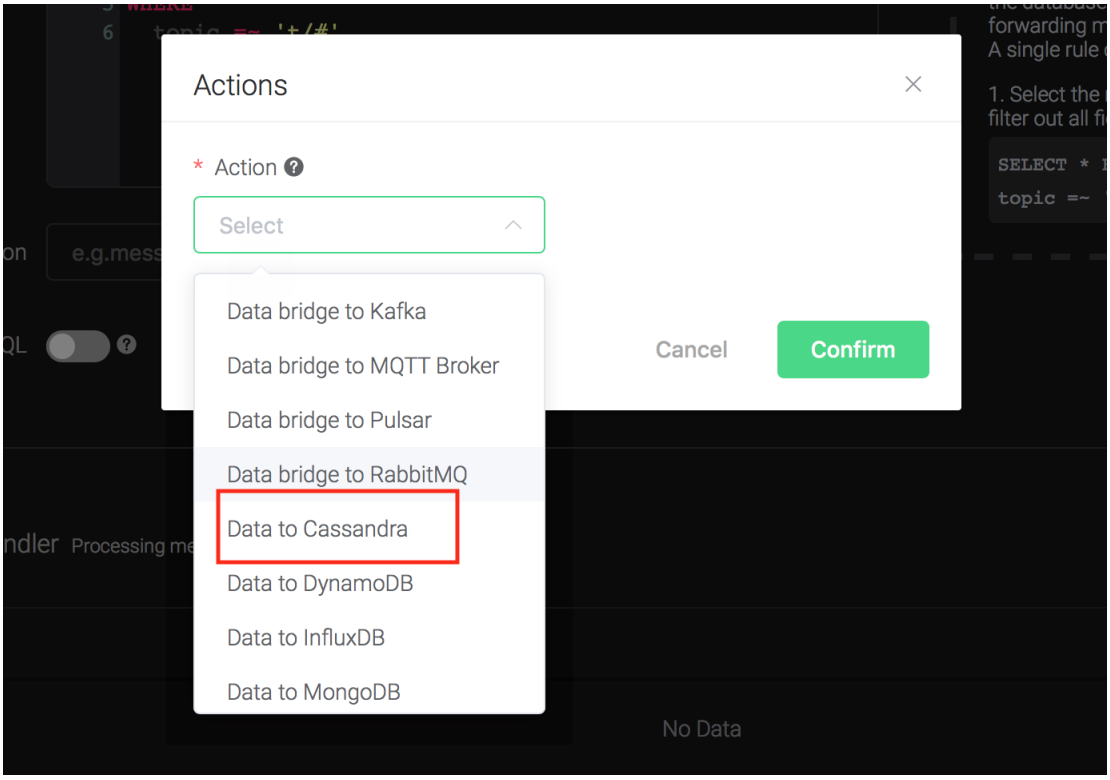
Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to Cassandra" in the pop-up dialog window.



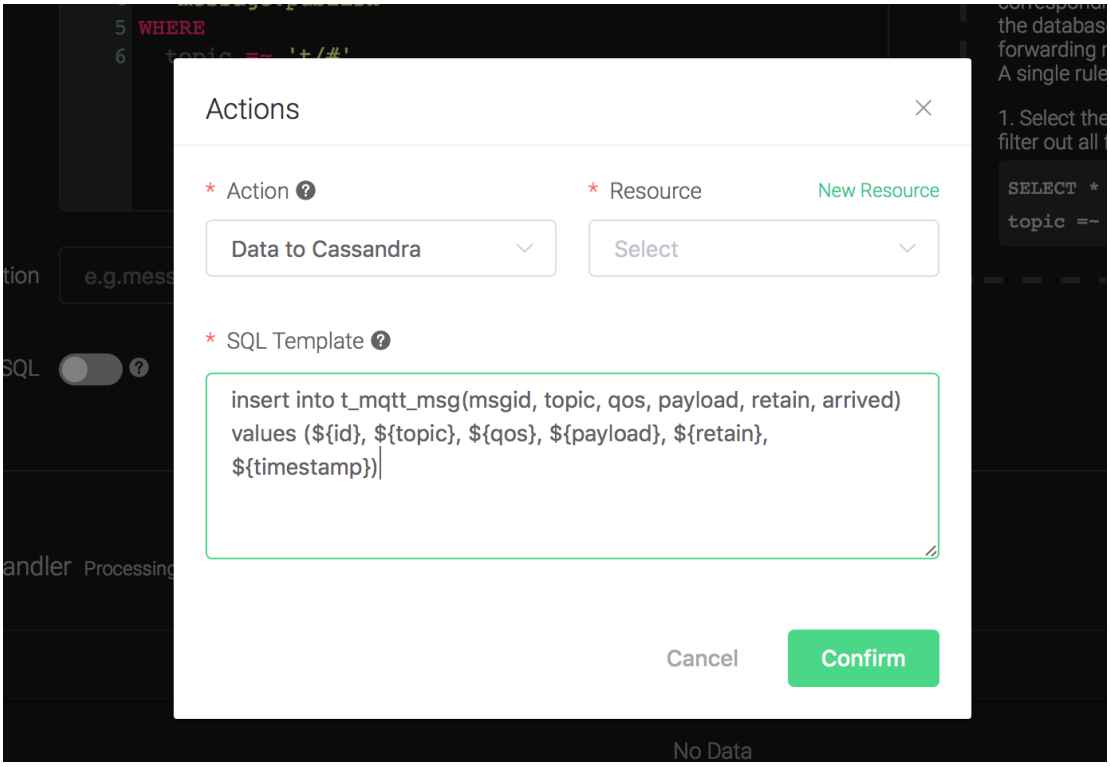
4. Fill in the parameters required by the action:

   Two parameters is required by action "Data to Cassandra":

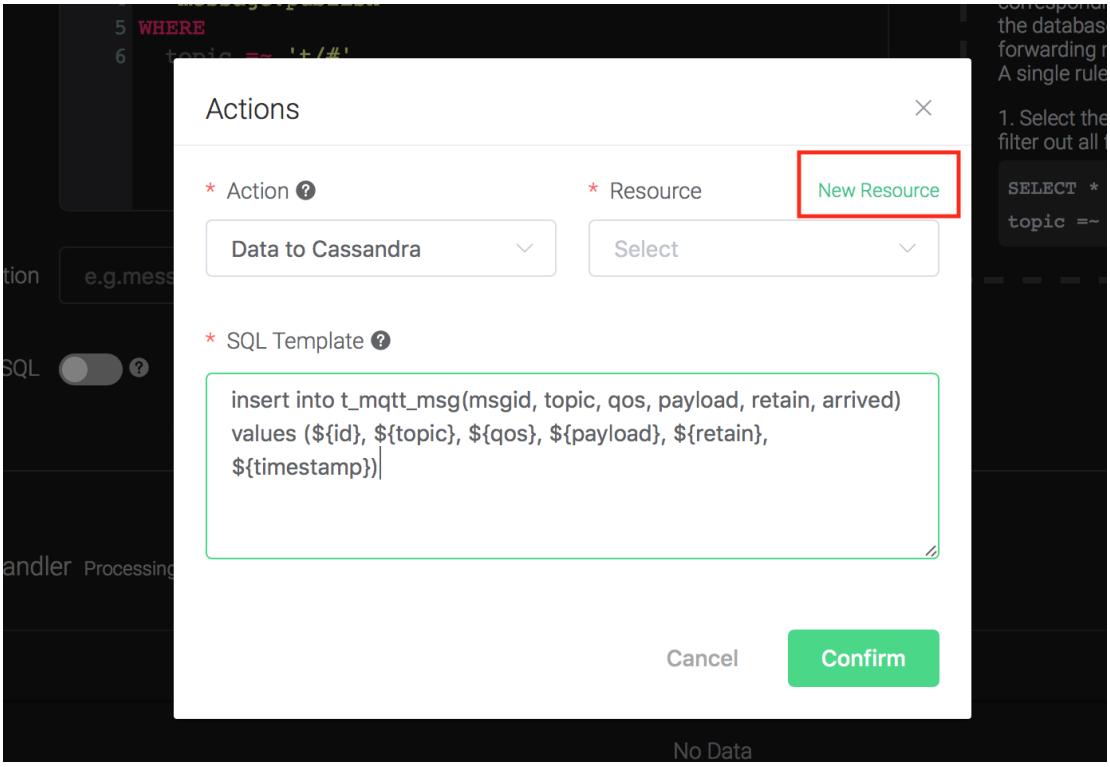
   1). SQL template. SQL template is the sql command you'd like to run when the action is triggered. In this example we'll insert a message into Cassandra, so type in the following sql template:

```
insert into t_mqtt_msg(msgid, topic, qos, payload, retain, arrived)␣
↪values (${id}, ${topic}, ${qos}, ${payload}, ${retain}, $
↪{timestamp})
```

   Before data is inserted into the table, placeholders like ${key} will be replaced by the corresponding values.

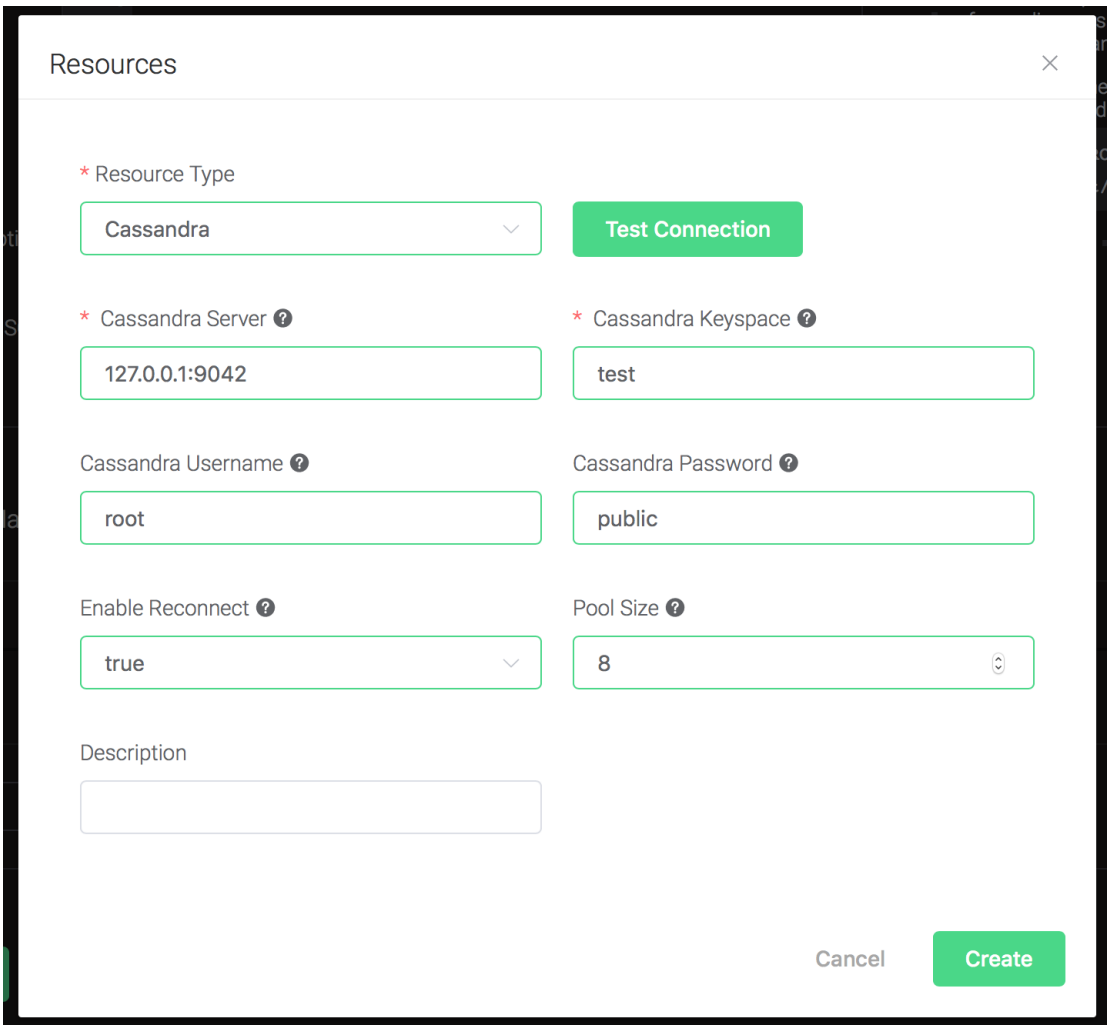


2). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "Cassandra":



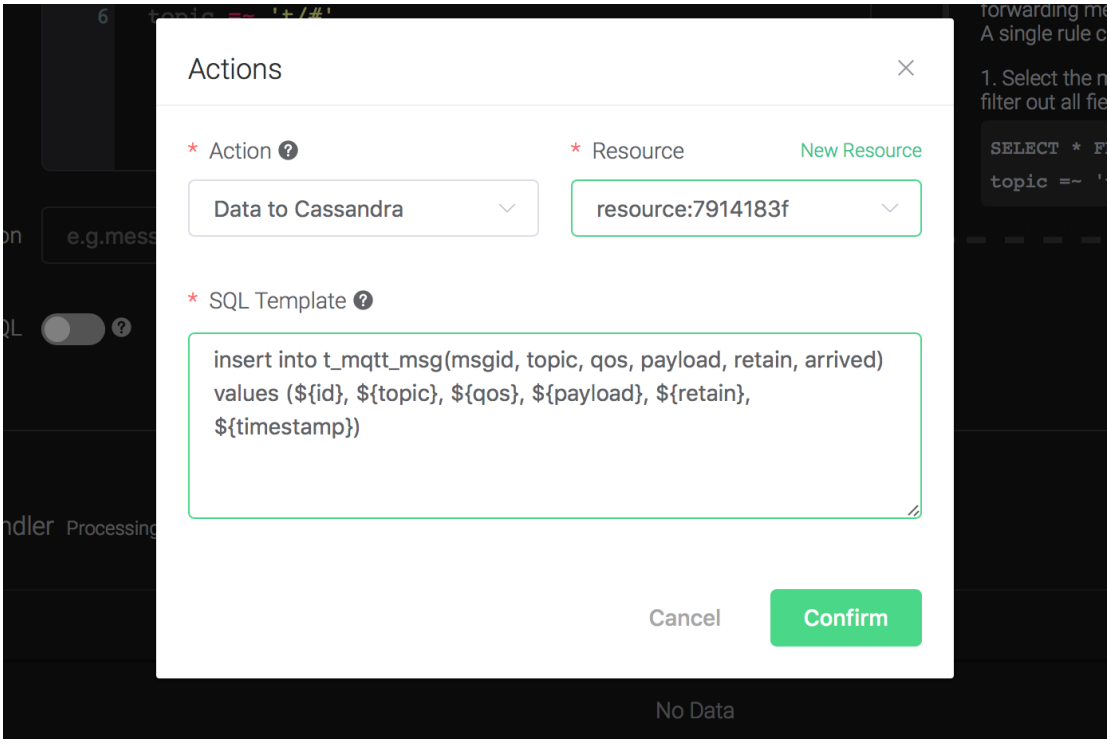
5. Configure the resource:

Set "Cassandra Keyspace" to "test", "Cassandra Username" to "root", "Cassandra Password" to "public", and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully.

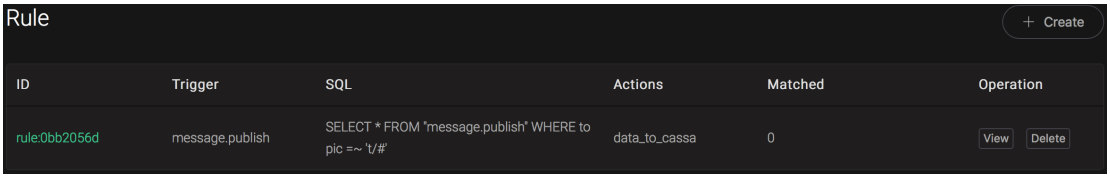


Then click on the "Create" button.

6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



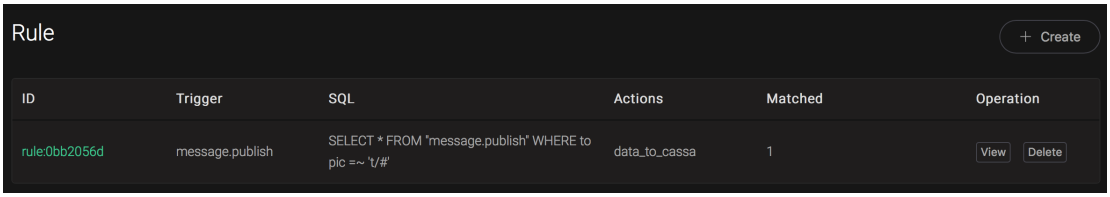
8. We have finished, testing the rule by sending an MQTT message to emqx:

```
Topic: "t/cass"
QoS: 1
Retained: true
Payload: "hello"
```

Then inspect the Cassandra table, verify a new record has been inserted:

```
root@cqlsh:test> select * from t_mqtt_msg;

 msgid                            | topic  | arrived                         | payload | qos | retain
----------------------------------+--------+---------------------------------+---------+-----+--------
 58C4A92E6A7ACF44000000A9F0001 | t/cass | 2019-06-27 09:13:23.612000+0000 |   hello |   1 |      1
```

And from the rule list, verify that the "Matched" column has increased to 1:



8.4 Create MongoDB Rules

0. Setup a MongoDB database, and changes the username/password to root/public, taking Mac OSX for instance:

```
$ brew install mongodb
$ brew services start mongodb

## add user root
$ use mqtt;
$ db.createUser({user: "root", pwd: "public", roles: [{role: "readWrite",
↪ db: "mqtt"}]});

## change the config file to enable authentication
$ vim /usr/local/etc/mongod.conf

  security:
    authorization: enabled

$ brew services restart mongodb
```

1. Initiate the MongoDB table:

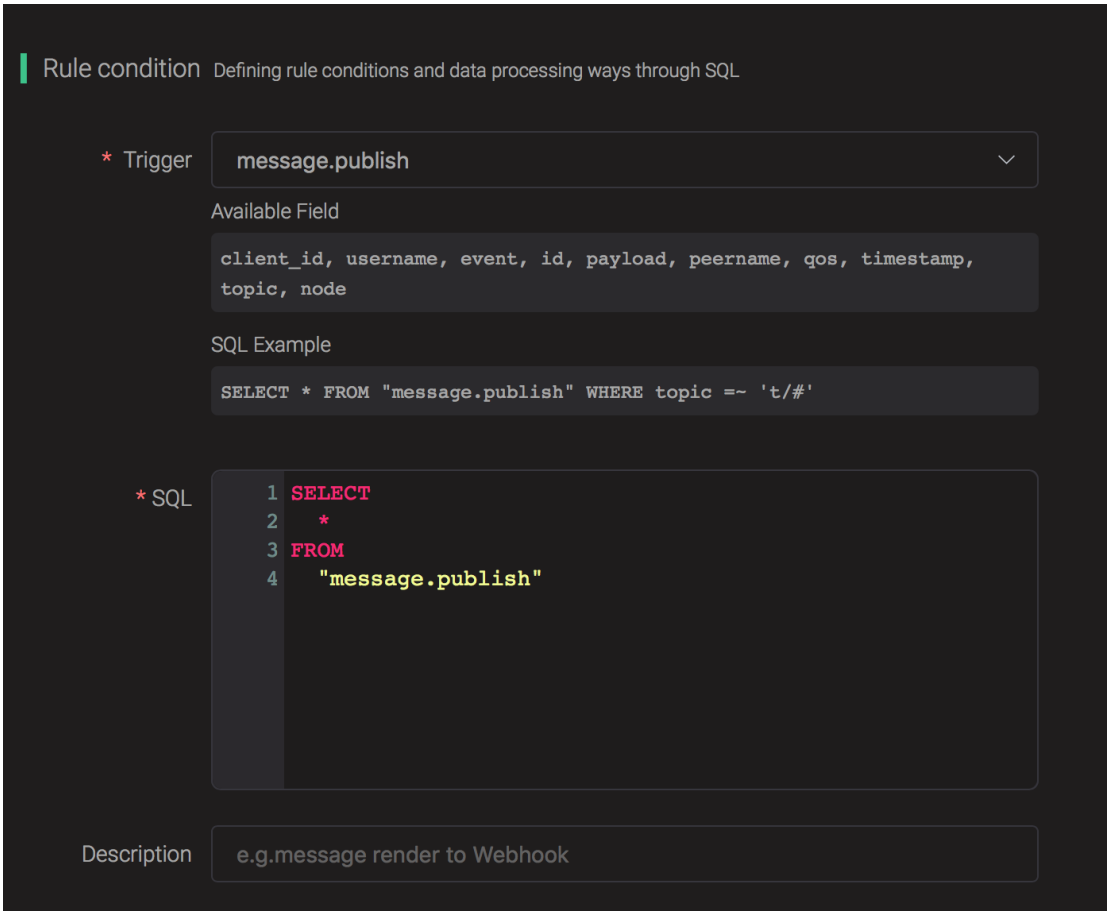
```
$ mongo 127.0.0.1/mqtt -uroot -ppublic

  db.createCollection("t_mqtt_msg");
```

2. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

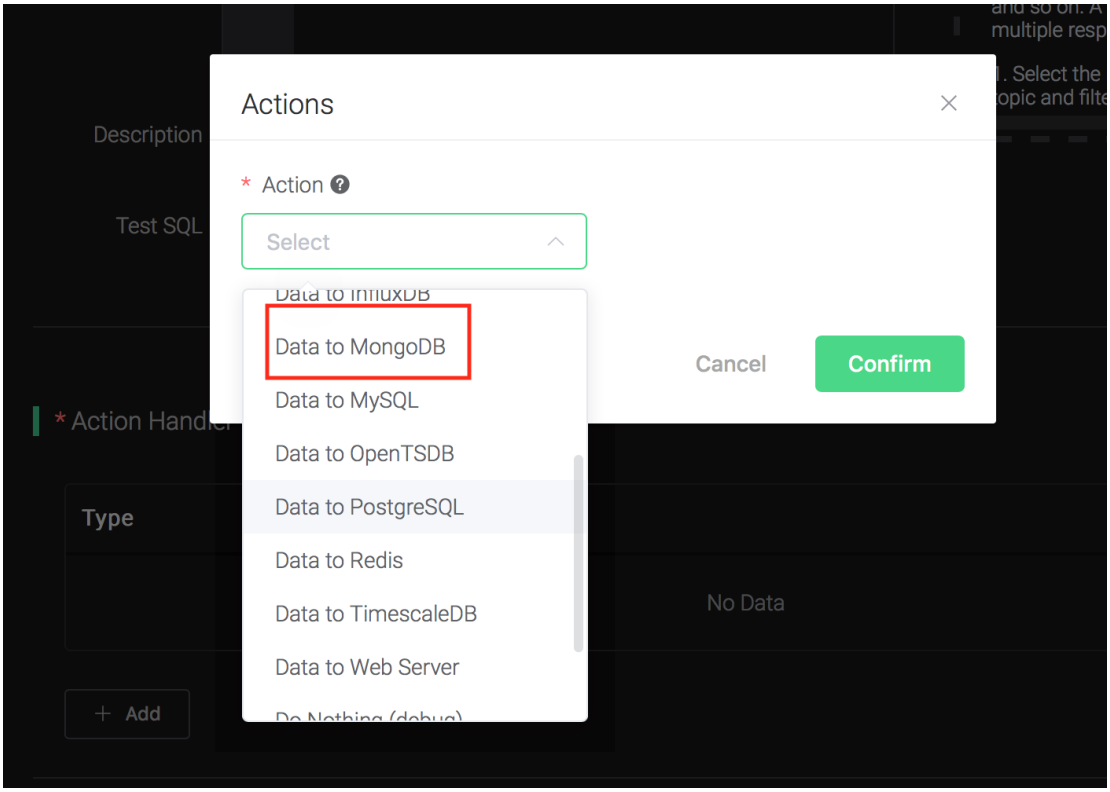
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to MongoDB" in the pop-up dialog window.



4. Fill in the parameters required by the action:

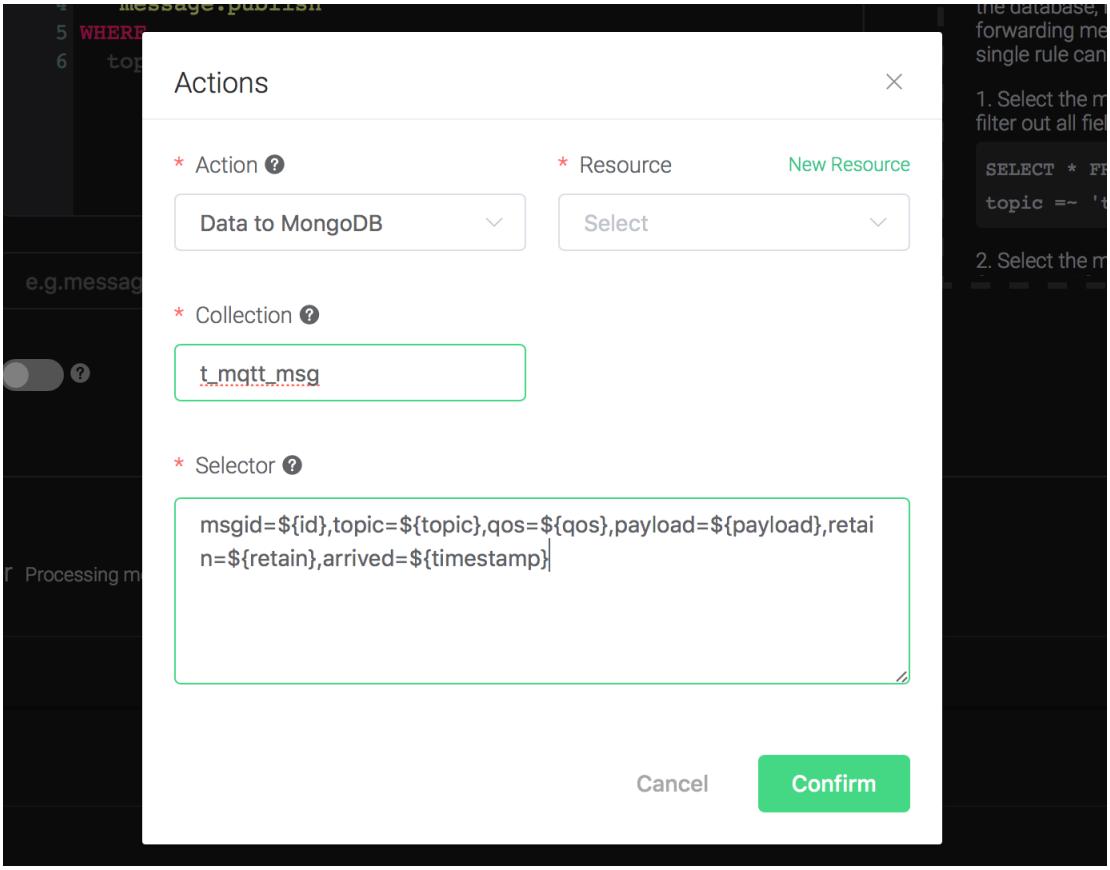
   Two parameters is required by action "Data to MongoDB":

   1). The mongodb collection. Set it to "t_mqtt_msg" we just created.

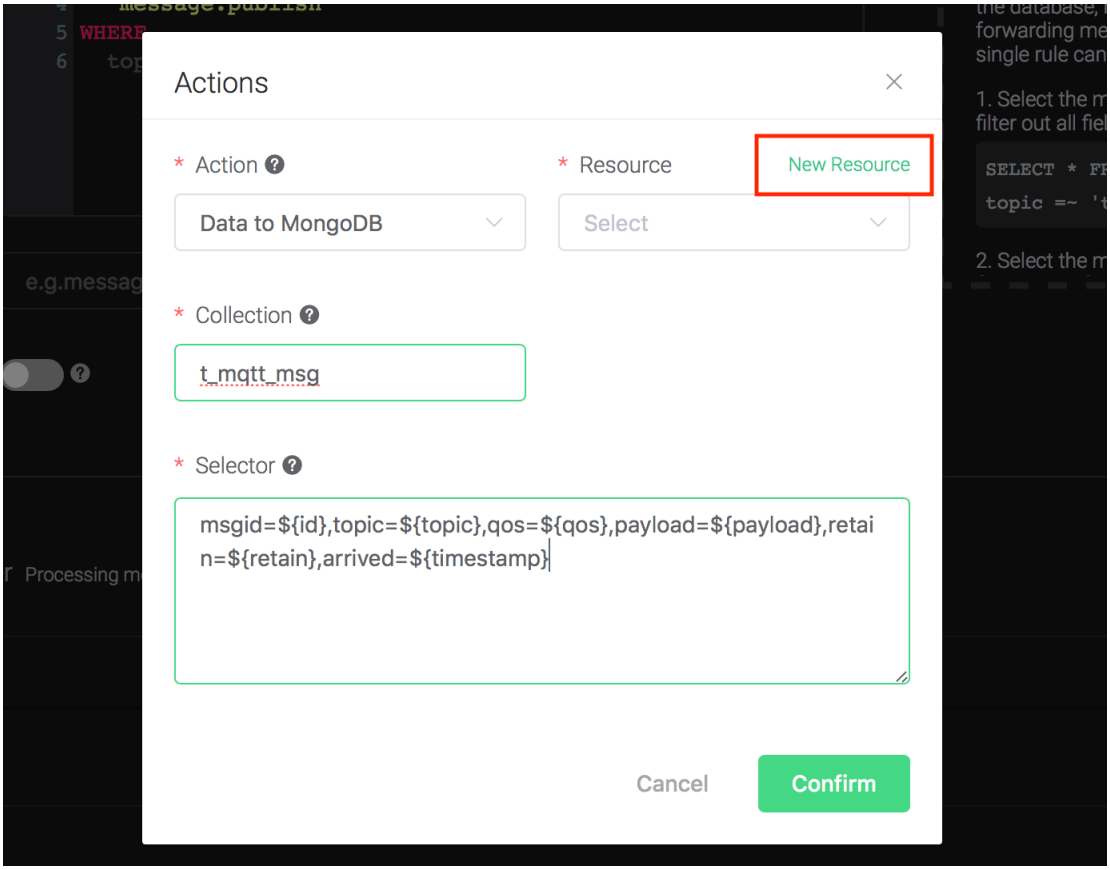
   2). Selector template. Selector template is the keys and values you'd like to insert into mongodb when the action is triggered. In this example we'll insert a message into mongodb, so type in the following sql template:

```
msgid=${id},topic=${topic},qos=${qos},payload=${payload},retain=$
↪{retain},arrived=${timestamp}
```

   Before data is inserted into the table, placeholders like ${key} will be replaced by the corresponding values.

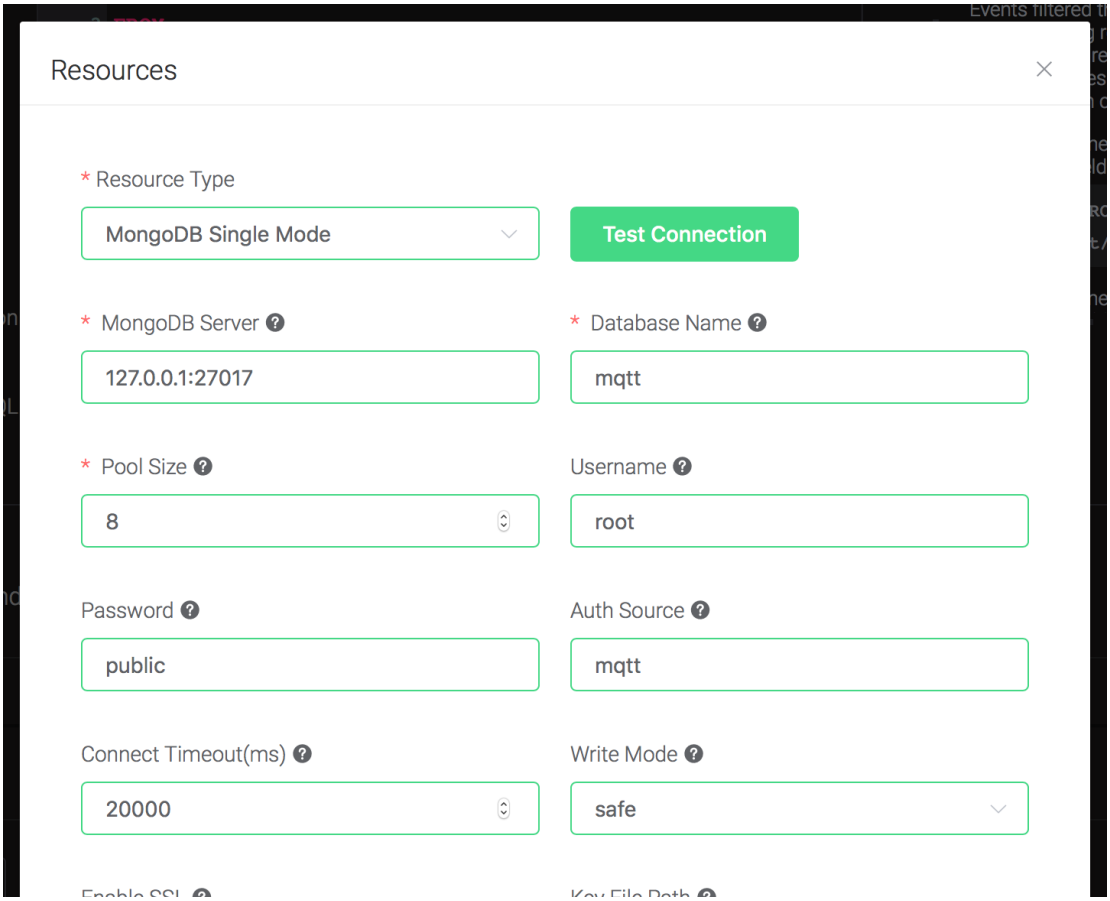


3). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "MongoDB Single Mode":

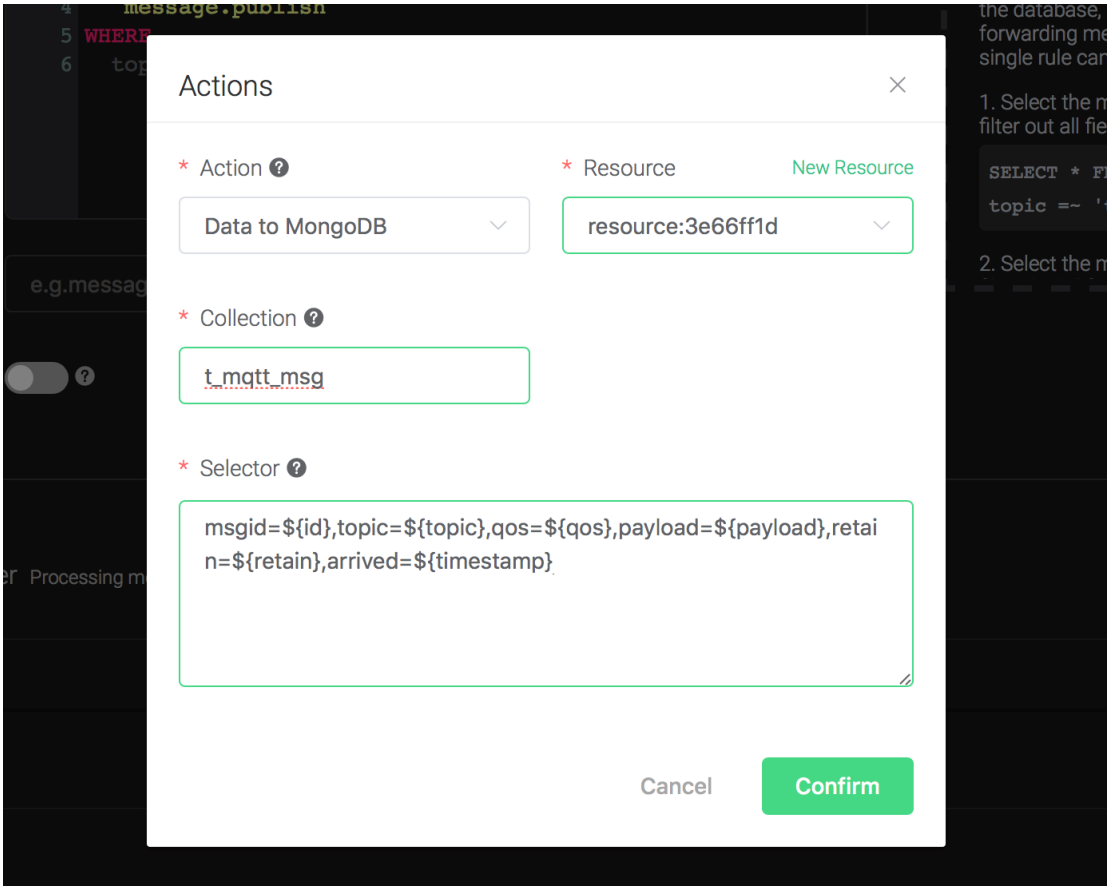


5. Configure the resource:

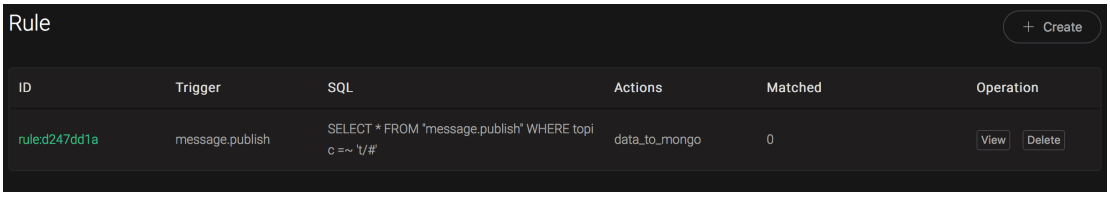
   Set "Database Name" to "mqtt", "Username" to "root", "Password" to "public", "Auth Source" to "mqtt", and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button..



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



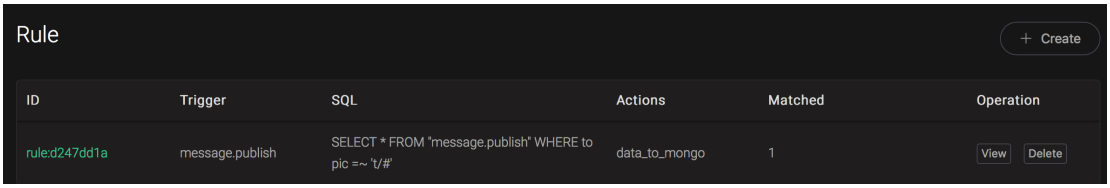
8. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/mongo" QoS: 1 Retained: true Payload: "hello"

   Then inspect the MongoDB table, verify a new record has been inserted:

```
> use mqtt;
switched to db mqtt
>
> db.t_mqtt_msg.find()
{ "_id" : ObjectId("5d14ac204a38b8f2f0000002"), "msgid" : "58C4CAF712AB0F440000009C30001", "topic" :
"t/mongo", "qos" : "1", "payload" : "hello", "retain" : "true", "arrived" : "1561635872386" }
>
```

   And from the rule list, verify that the "Matched" column has increased to 1:



8.5 Create DynamoDB Rules

0. Setup a DynamoDB database, taking Mac OSX for instance:

```
$ brew install dynamodb-local

$ dynamodb-local
```

1. Initiate the DynamoDB table:

   1). Create table definition file "mqtt_msg.json" for DynamoDB:

```
{
    "TableName": "mqtt_msg",
    "KeySchema": [
        { "AttributeName": "msgid", "KeyType": "HASH" }
    ],
    "AttributeDefinitions": [
        { "AttributeName": "msgid", "AttributeType": "S" }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

   2). Create the DynamoDB table:

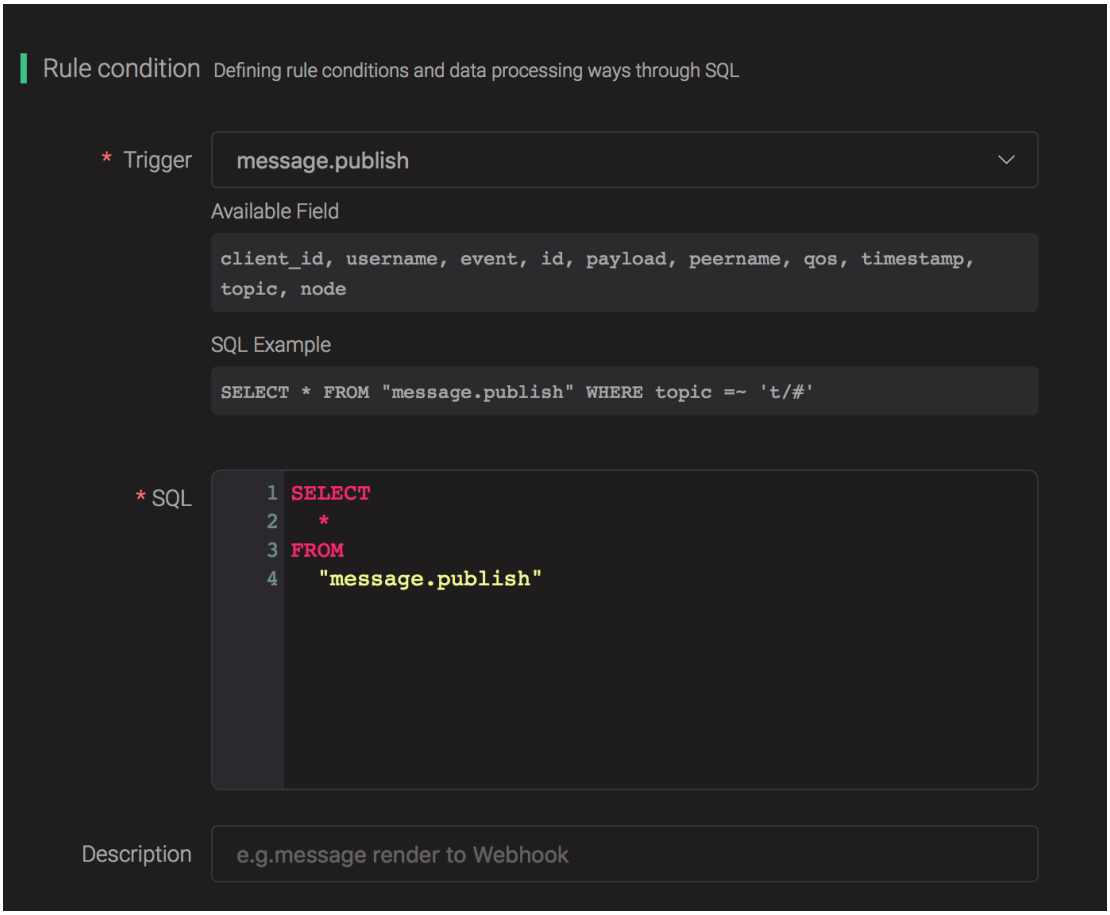
```
$ aws dynamodb create-table --cli-input-json file://mqtt_msg.json --
↪endpoint-url http://localhost:8000
```

2. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

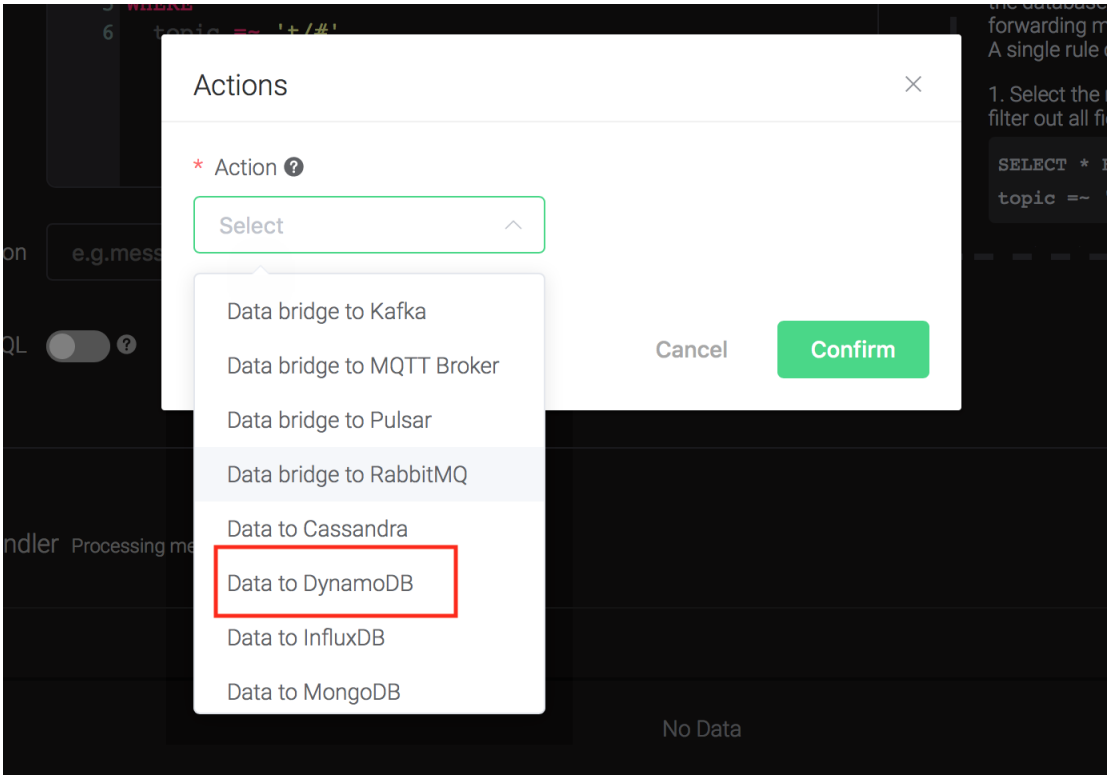
   Select "message.publish", then type in the following SQL:

```
SELECT
 msgid as id, topic, payload
FROM
  "message.pubish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to DynamoDB" in the pop-up dialog window.



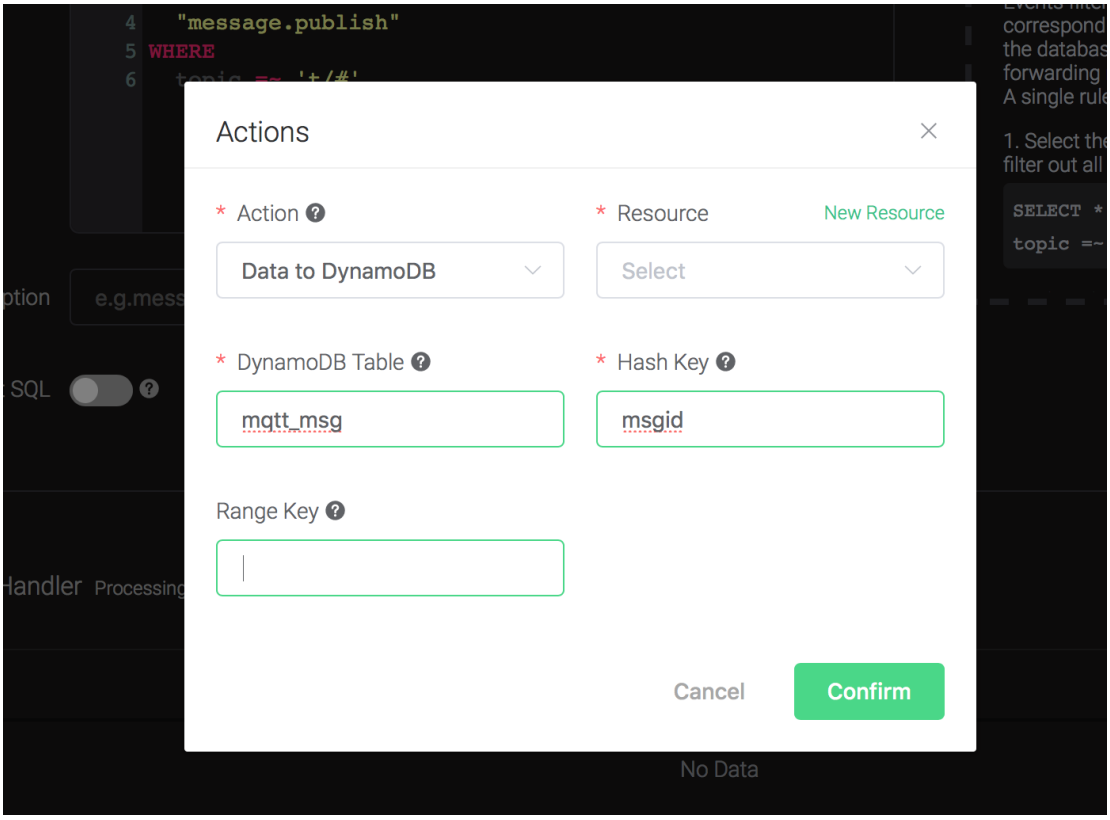
4. Fill in the parameters required by the action:

   Four parameters is required by action "Data to DynamoDB":

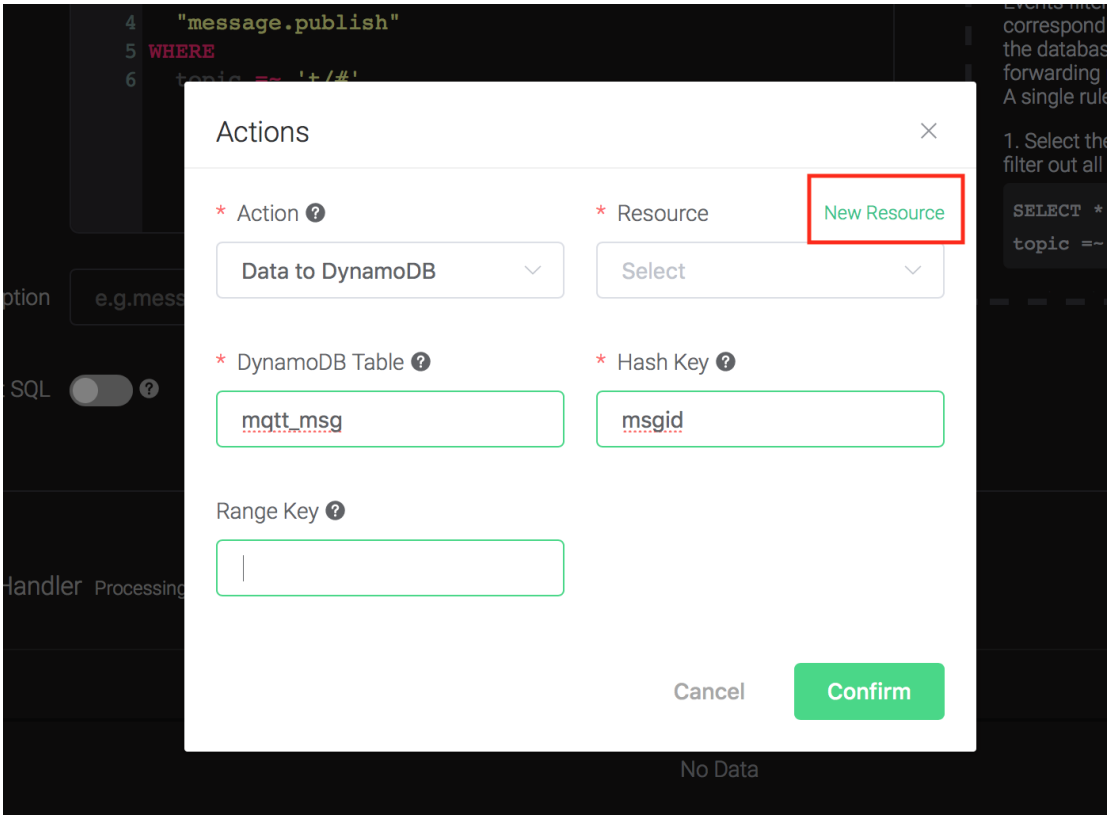
   1). DynamoDB Table. Here set it to "mqtt_msg".

   2). Hash Key. Here set it to "msgid".

   3). DynamoDB Range Key. Leave it empty as we didn't define a range key in the dynamodb definition file.



4). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "DynamoDB":

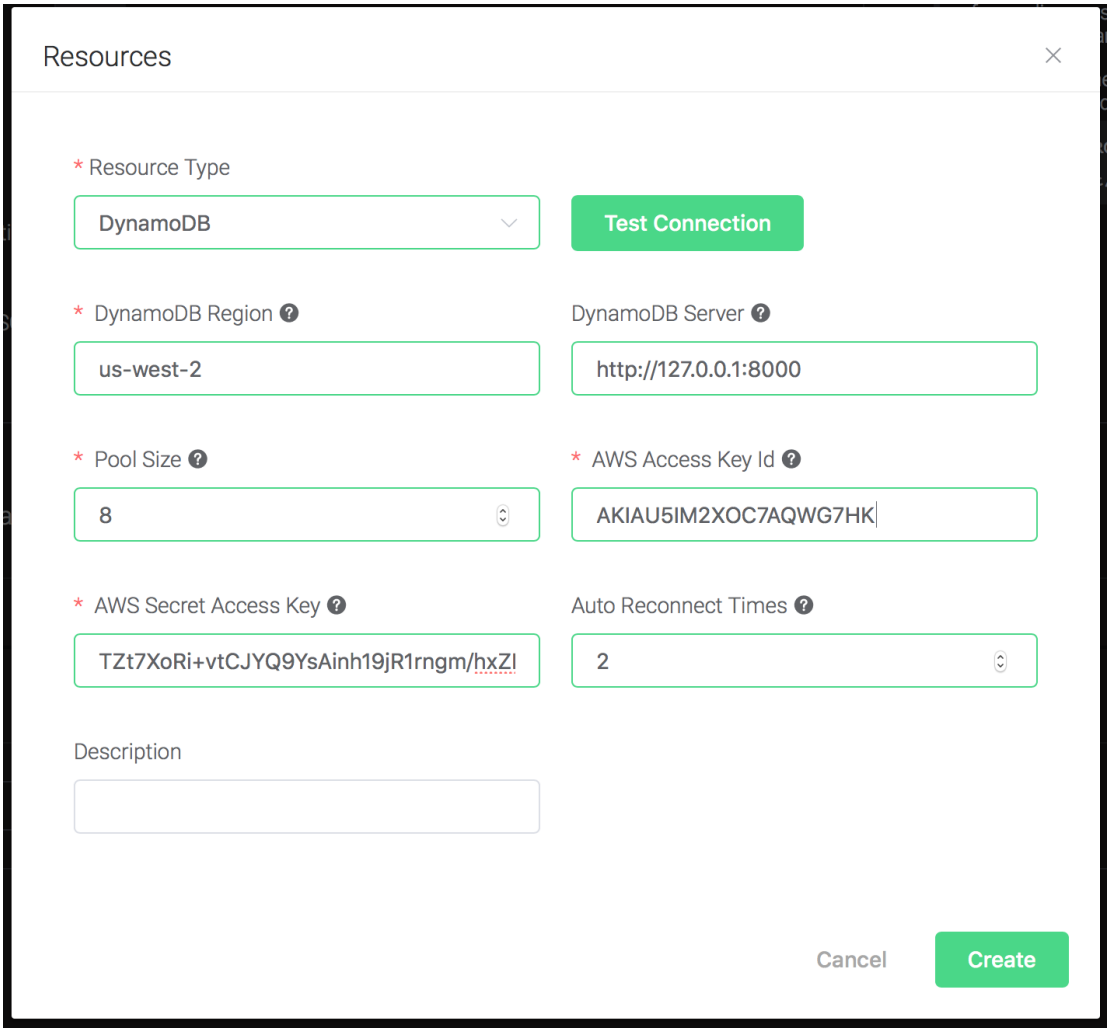


5. Configure the resource:

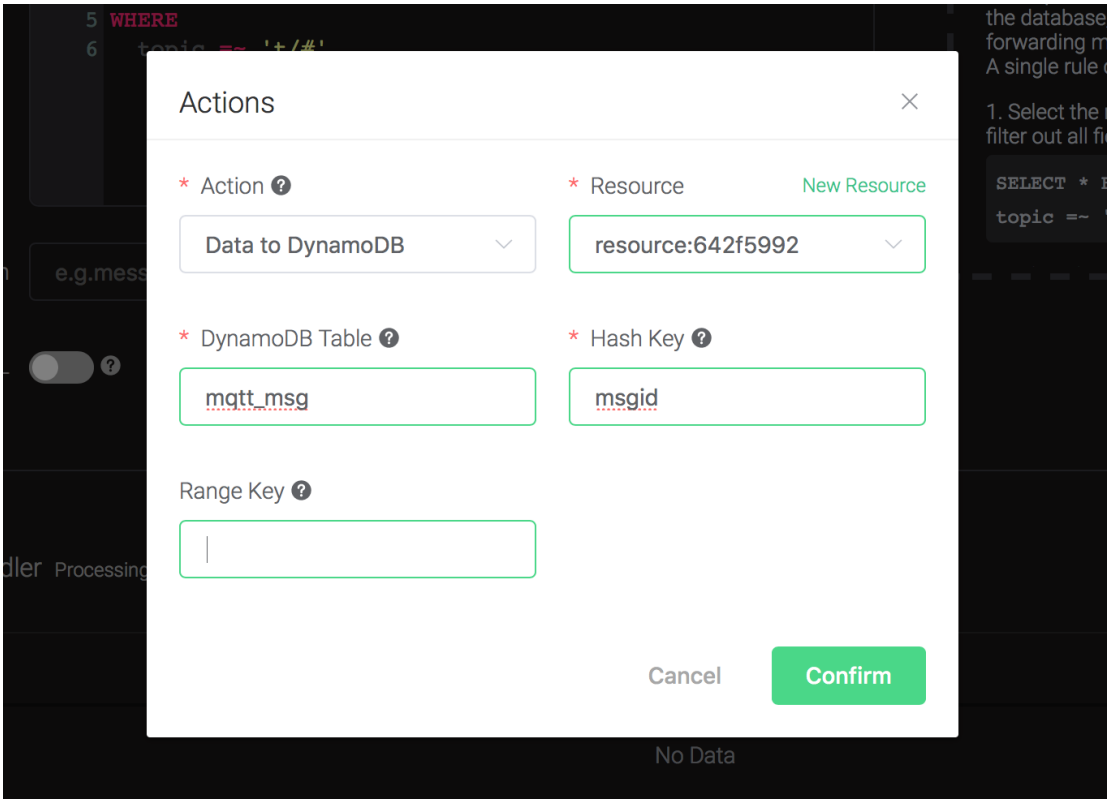
   Fill in the configs as following:

   - DynamoDB Region: us-west-2
   - DynamoDB Server: http://localhost:8000
   - AWS Access Key Id: "AKIAU5IM2XOC7AQWG7HK"
   - AWS Secret Access Key: "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"

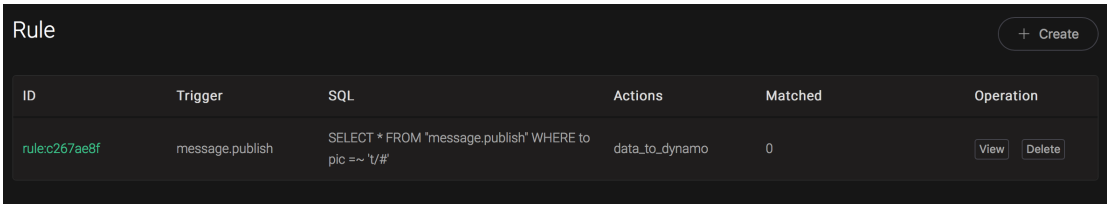
   And then click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/a"

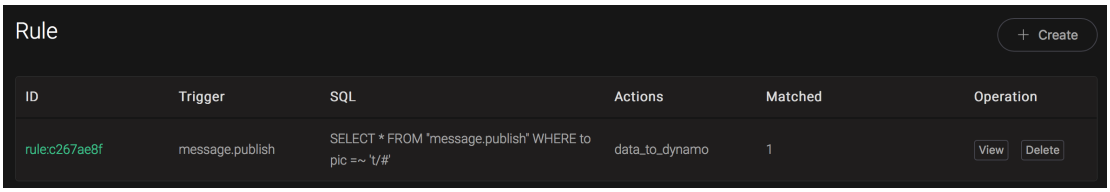
   QoS: 1

   Payload: "hello"

   Then inspect the DynamoDB table, verify a new record has been inserted:

```
└➤ aws dynamodb scan --table-name mqtt_msg --region us-west-2  --endpoint-url http://localhost:8000
{
    "Items": [
        {
            "topic": {
                "S": "t/1"
            },
            "payload": {
                "S": "{ \"msg\": \"Hello, World!\" }"
            },
            "msgid": {
                "S": "58D64DDA6C3F5F44300000ED60002"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

And from the rule list, verify that the "Matched" column has increased to 1:



8.6 Create Redis Rules

0. Setup a Redis database, taking Mac OSX for instance:

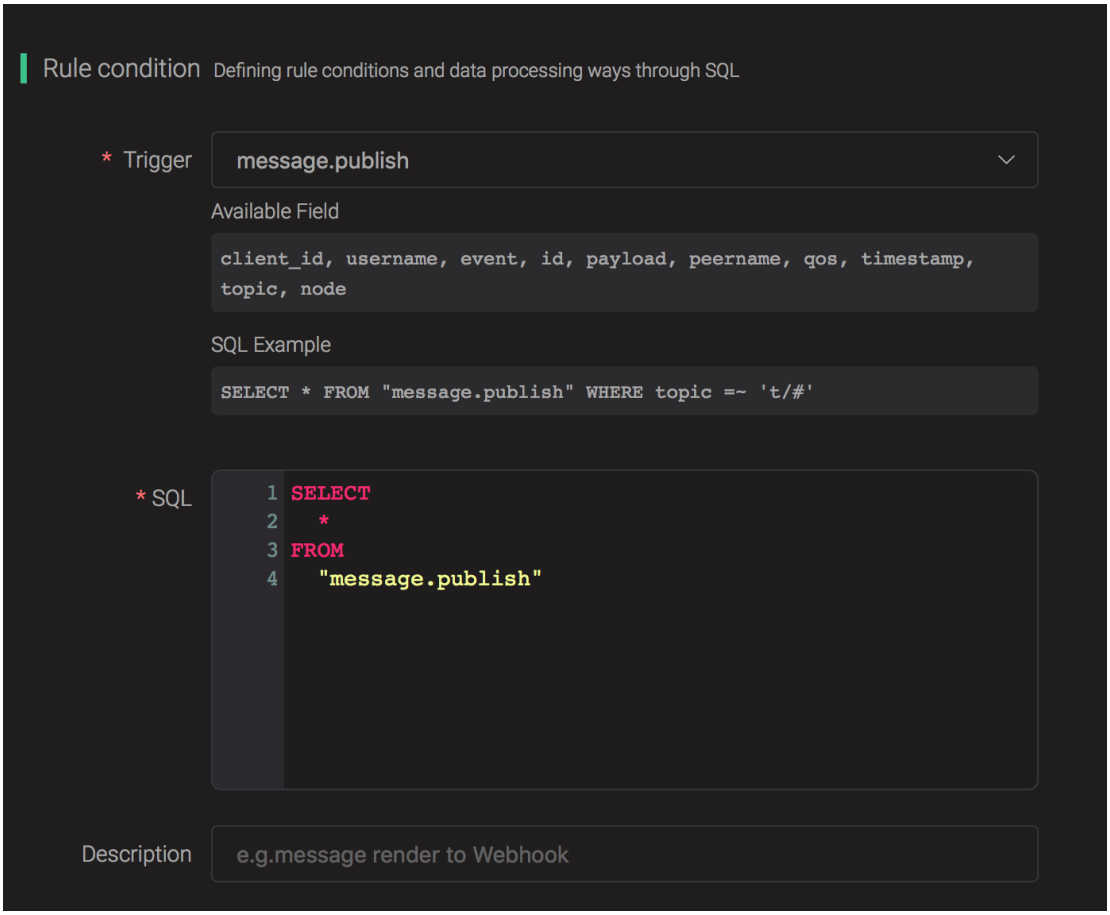
```
$ wget http://download.redis.io/releases/redis-4.0.14.tar.gz
$ tar xzf redis-4.0.14.tar.gz
$ cd redis-4.0.14
$ make && make install

# start redis
$ redis-server
```

1. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

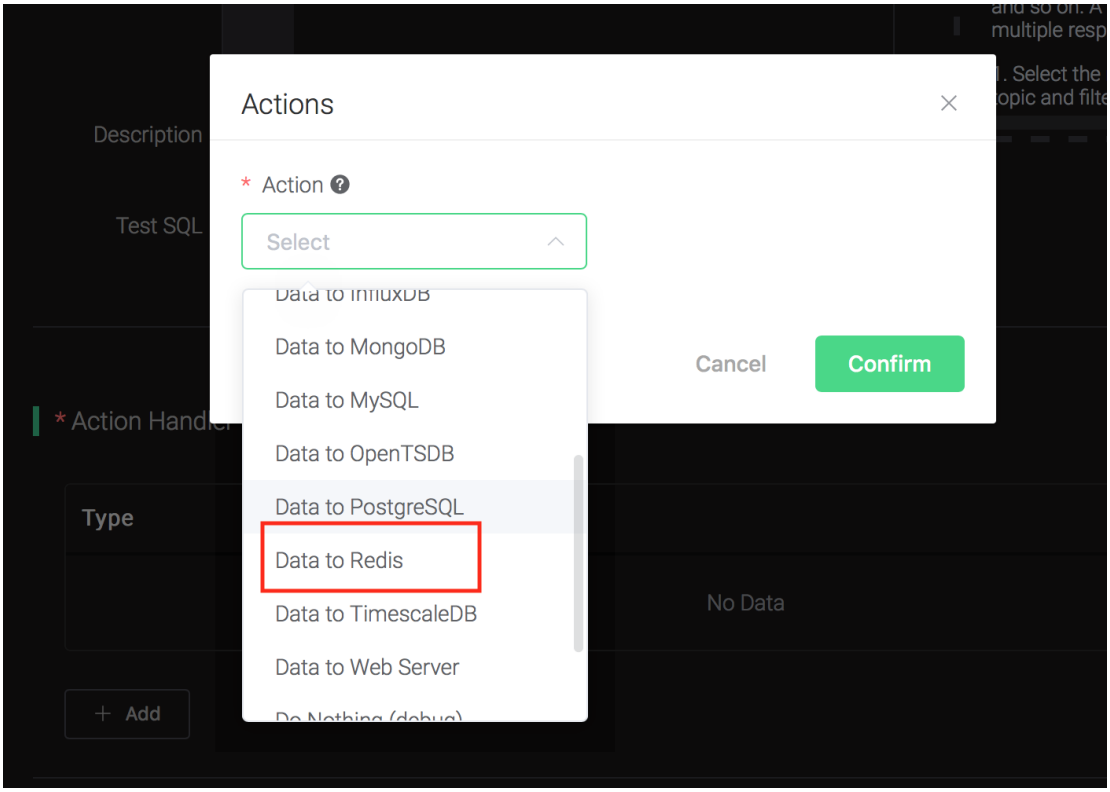
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



2. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to MySQL" in the pop-up dialog window.

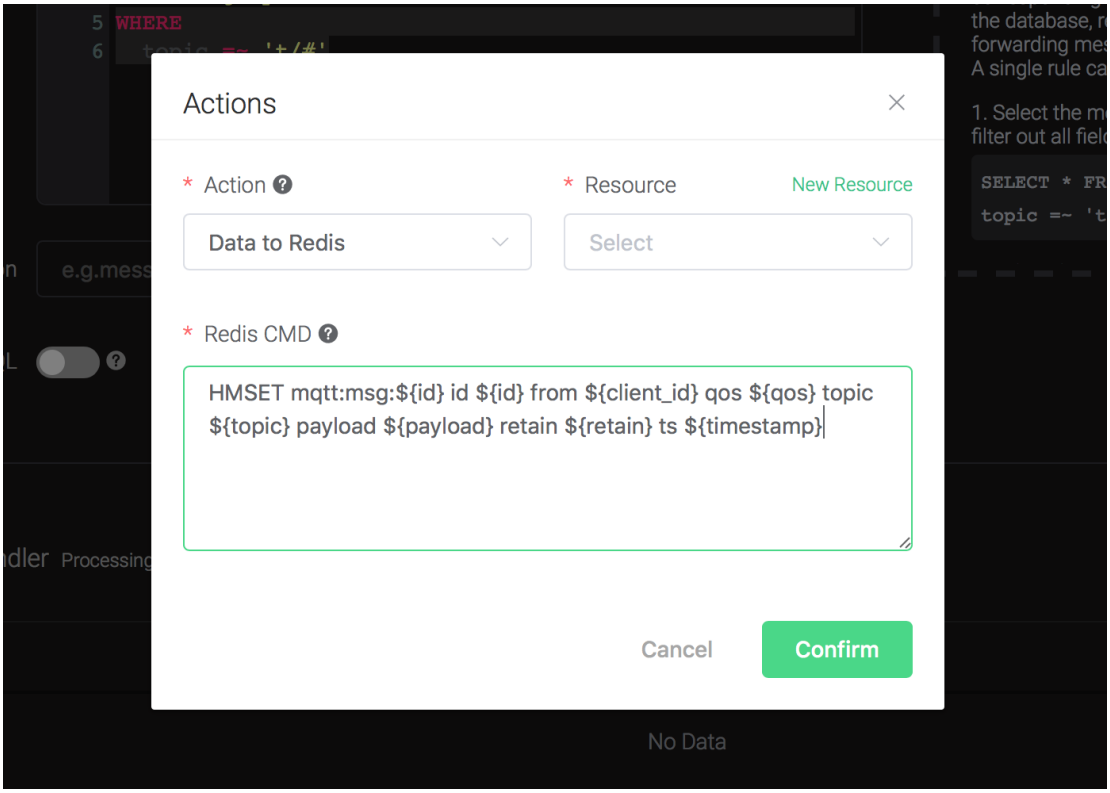


3. Fill in the parameters required by the action:

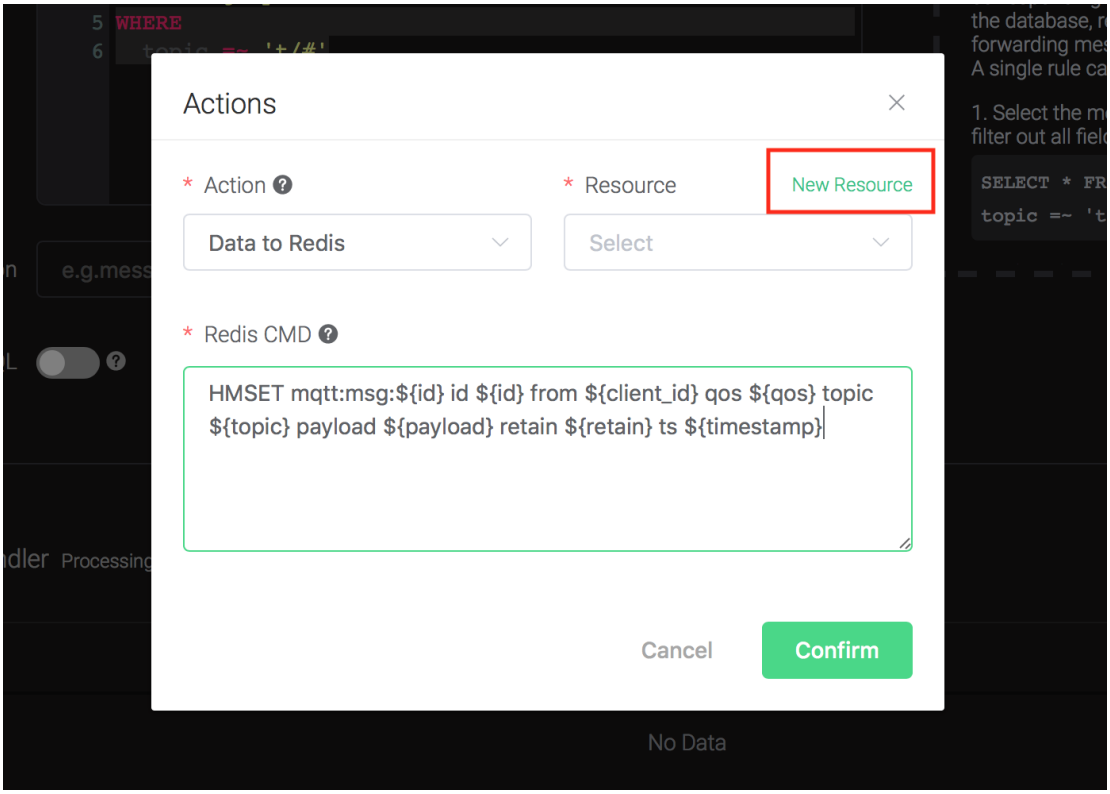
   Two parameters is required by action "Data to Redis":

   1). Redis CMD. The redis command you'd like to run when the action is triggered. In this example we'll insert a message into redis, so type in the following command:

```
HMSET mqtt:msg:${id} id ${id} from ${client_id} qos ${qos} topic $
↪{topic} payload ${payload} retain ${retain} ts ${timestamp}
```

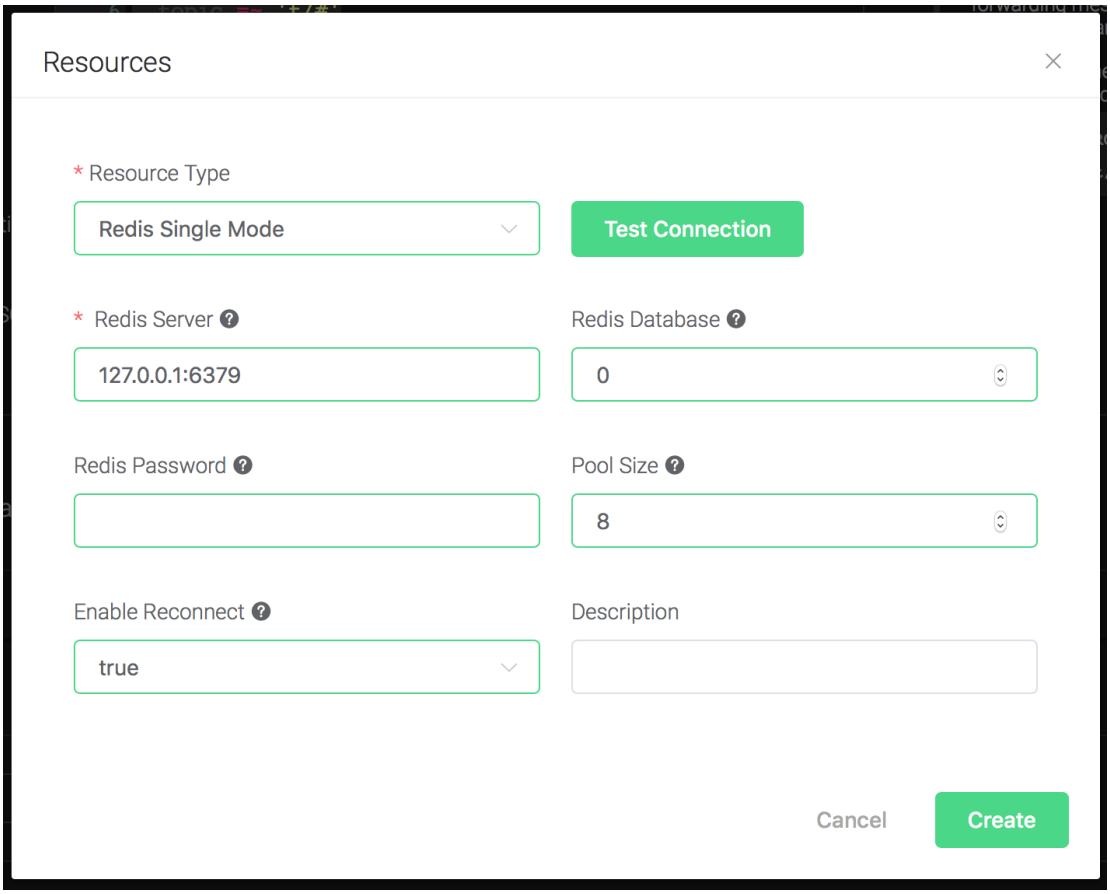


2). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "Redis Single Mode":

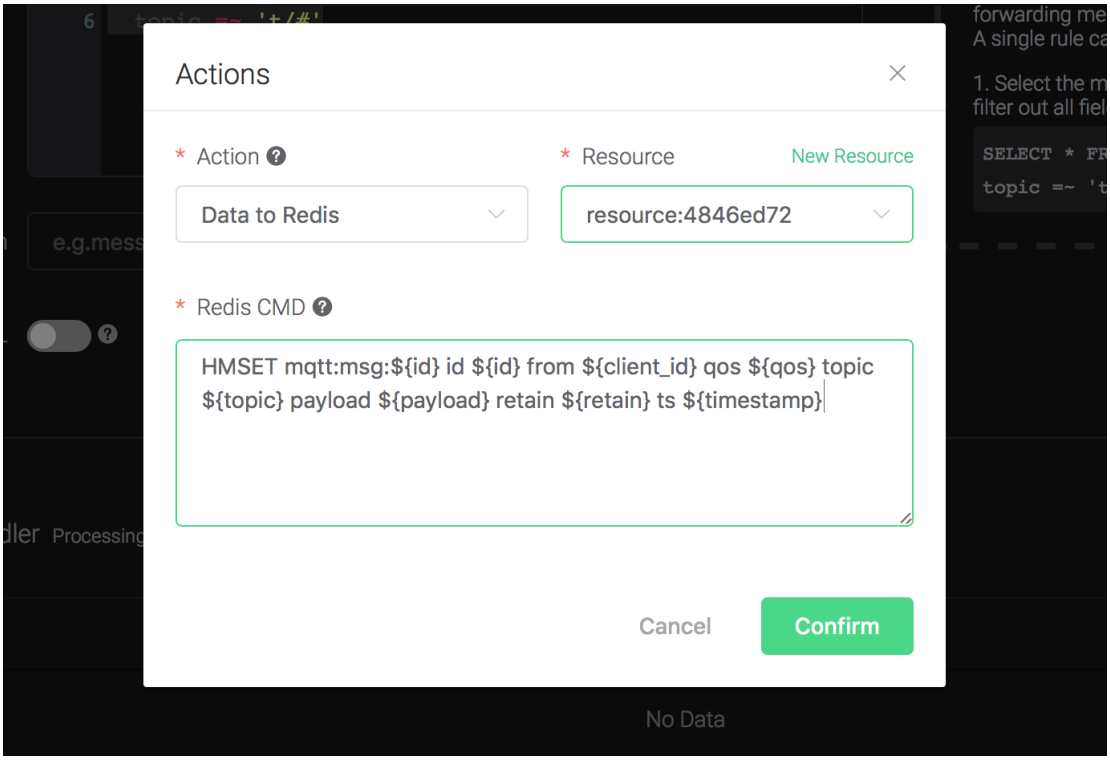


4. Configure the resource:

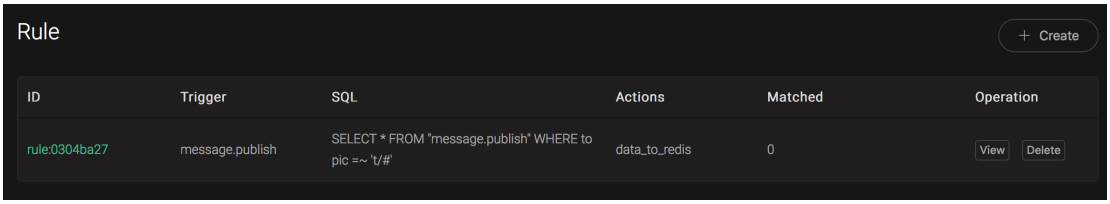
   Fill in the "Redis Server", and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



5. Back to the "Actions" dialog, and then click on the "Confirm" button.



6. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



7. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

   Retained: false

   Payload: "hello"

   Then inspect the Redis table, verify a new record has been inserted:

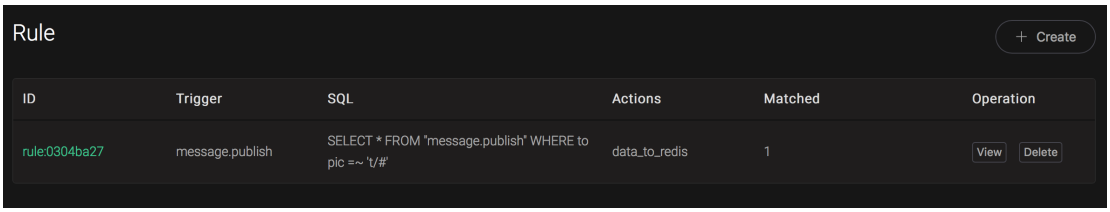
   $ redis-cli

   127.0.0.1:6379> KEYS mqtt:msg*

   127.0.0.1:6379> hgetall <key>

```
127.0.0.1:6379>
127.0.0.1:6379> KEYS mqtt:msg*
1) "mqtt:msg:58CBE39D13232F4420000198F0001"
127.0.0.1:6379> hgetall mqtt:msg:58CBE39D13232F4420000198F0001
 1) "id"
 2) "58CBE39D13232F4420000198F0001"
 3) "from"
 4) "mqttjs_b3719cdb09"
 5) "qos"
 6) "0"
 7) "topic"
 8) "t/1"
 9) "payload"
10) "hello"
11) "retain"
12) "0"
13) "ts"
14) "1562123525239"
127.0.0.1:6379>
```

And from the rule list, verify that the "Matched" column has increased to 1:



8.7 Create OpenTSDB Rules

0. Setup a OpenTSDB database, taking Mac OSX for instance:

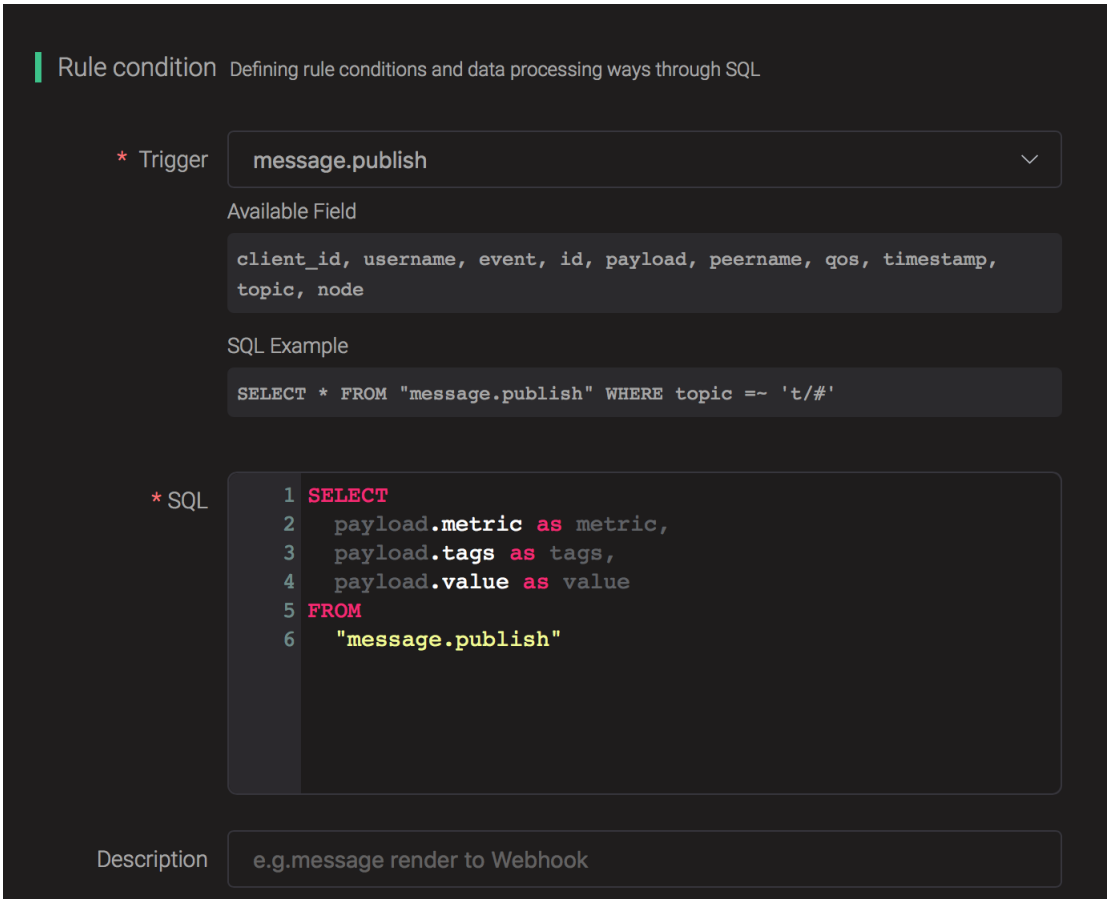
```
$ docker pull petergrace/opentsdb-docker

$ docker run -d --name opentsdb -p 4242:4242 petergrace/opentsdb-docker
```

1. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

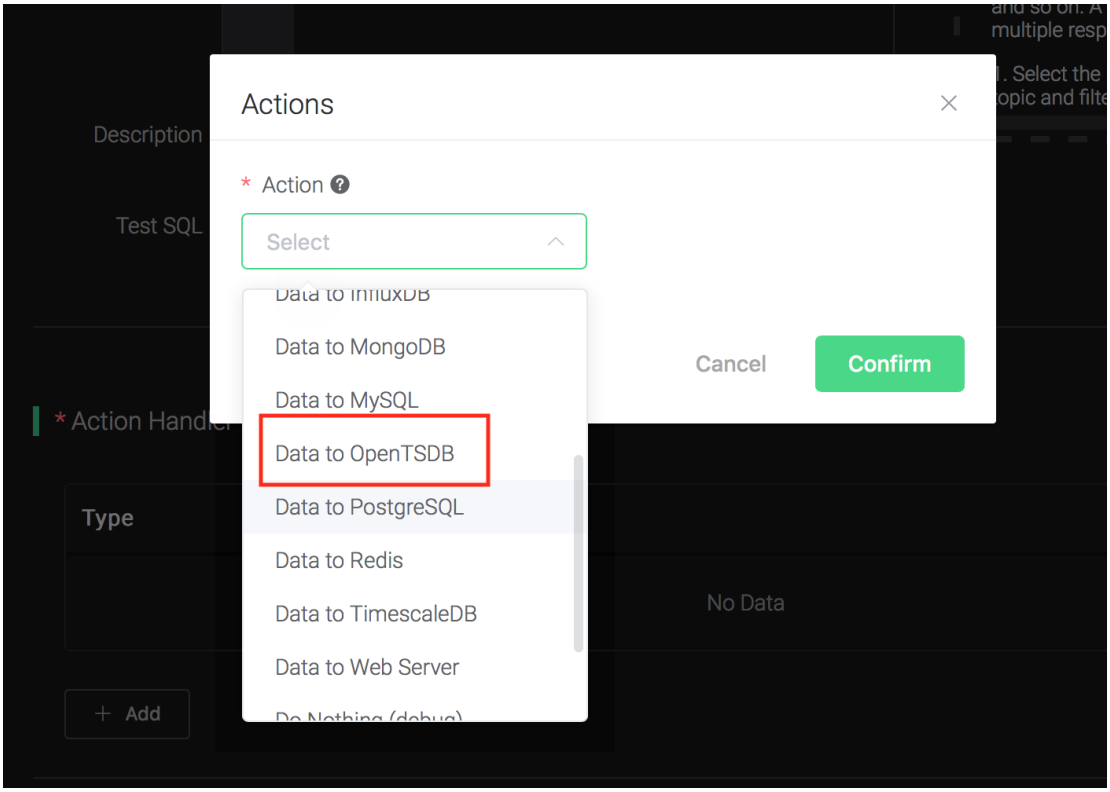
   Select "message.publish", then type in the following SQL:

```
SELECT
  payload.metric as metric, payload.tags as tags, payload.value as␣
↪value
FROM
  "message.publish"
```



2. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to OpenTSDB" in the pop-up dialog window.



3. Fill in the parameters required by the action:

   Six parameters is required by action "Data to OpenTSDB":

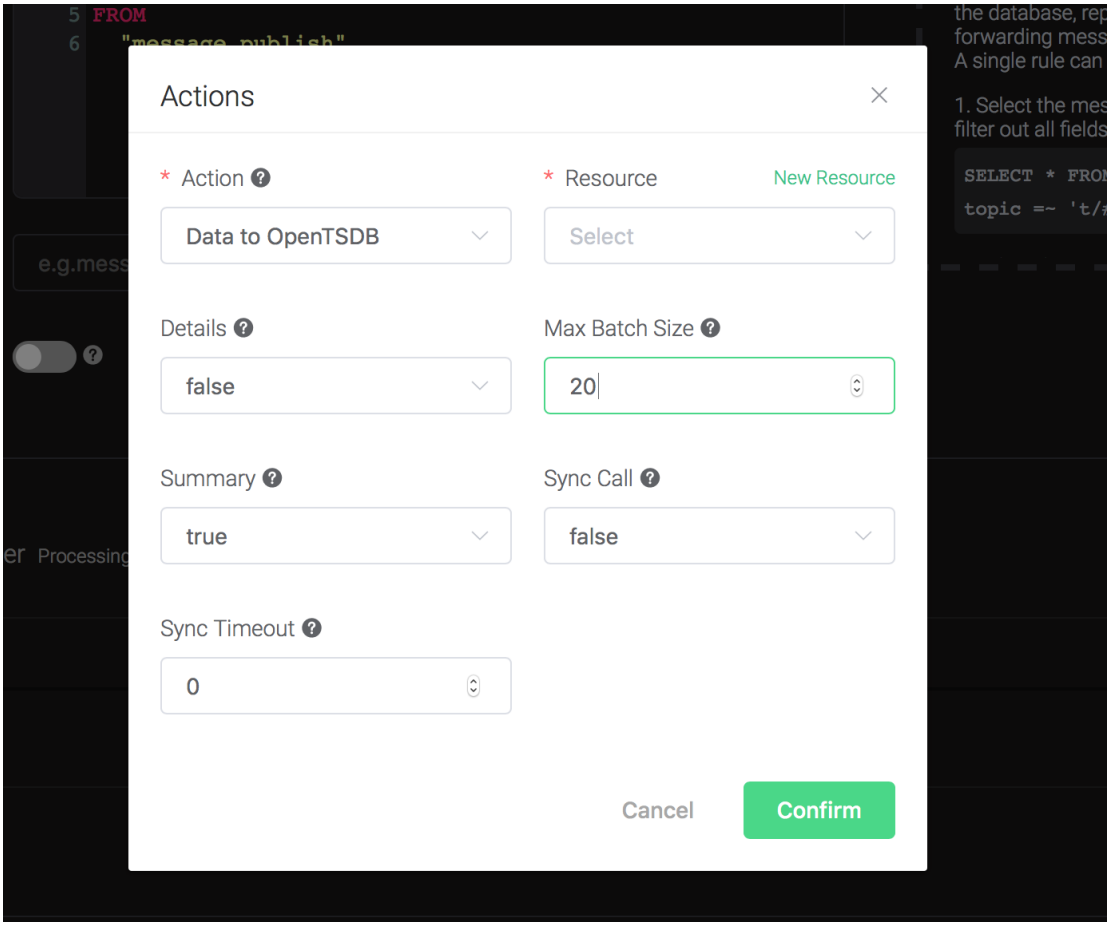
   1). Details. Whether let OpenTSDB Server return the failed data point and their error reason, defaults to false.

   2). Summary. Whether let OpenTSDB Server return data point success/failure count, defaults to true.

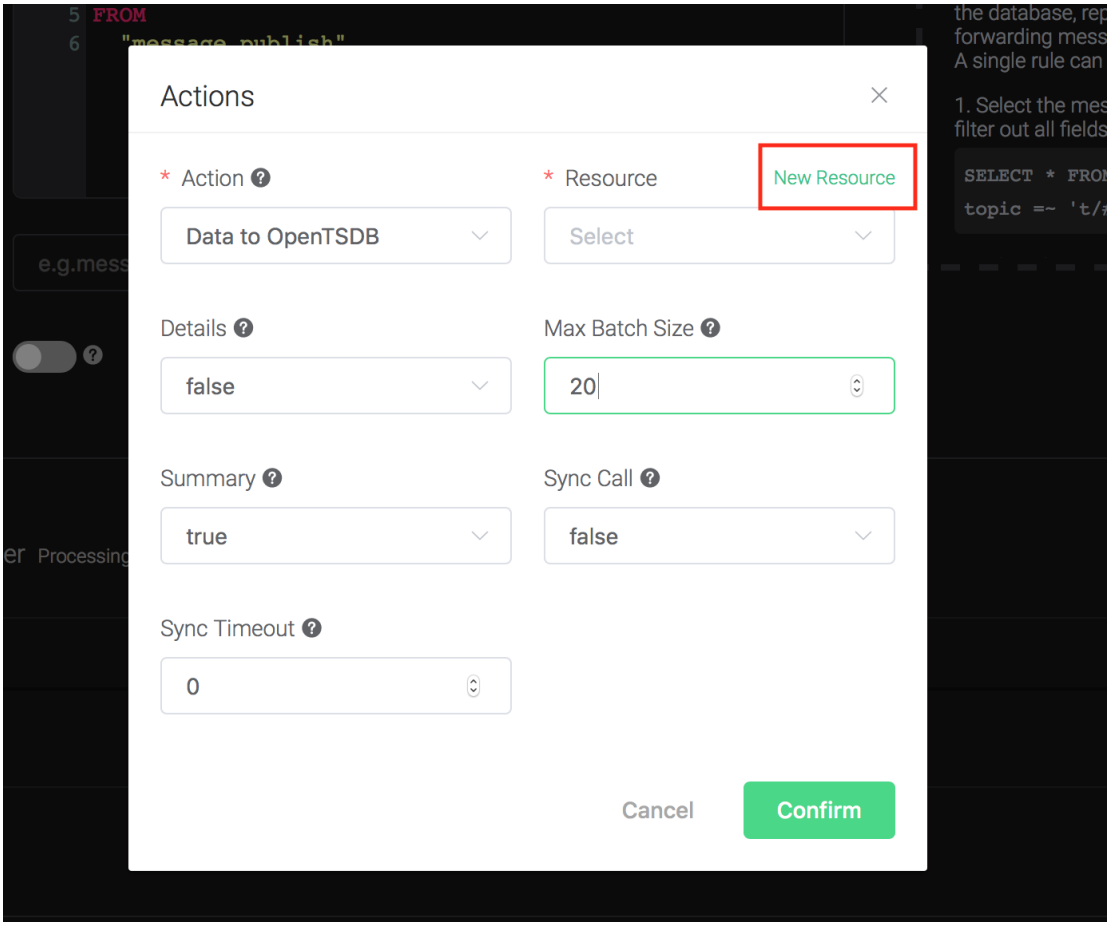
   3). Max Batch Size. In case of heavy traffic, how many data points are allowed to be included in a single request. Default to 20.

   4). Sync Call. Defaults to false.

   5). Sync Timeout. Defaults to 0.

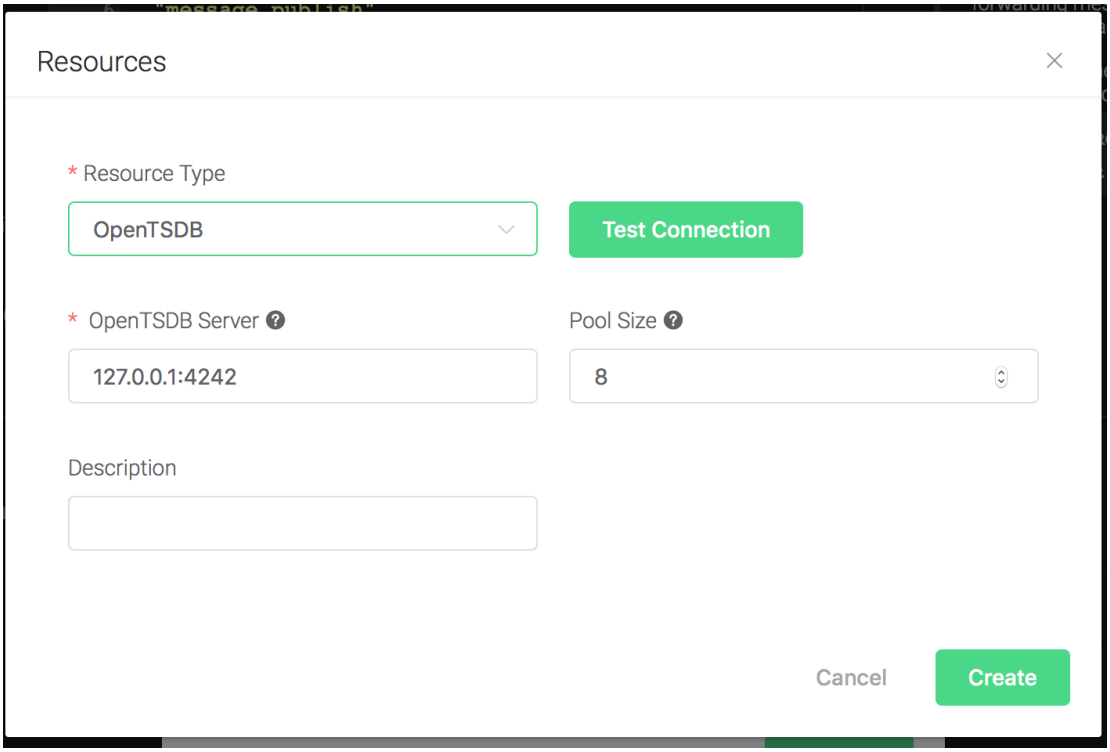


6). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "OpenTSDB":

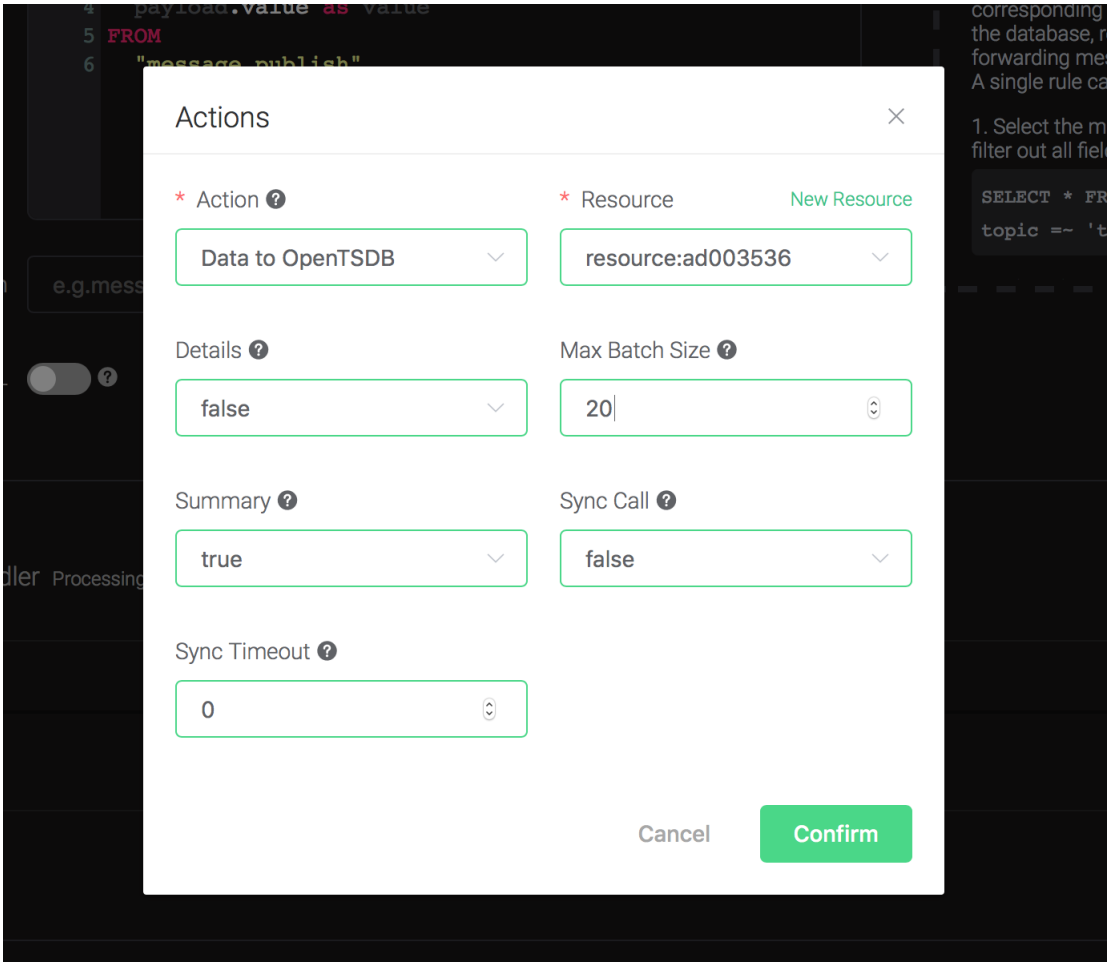


4. Configure the resource:

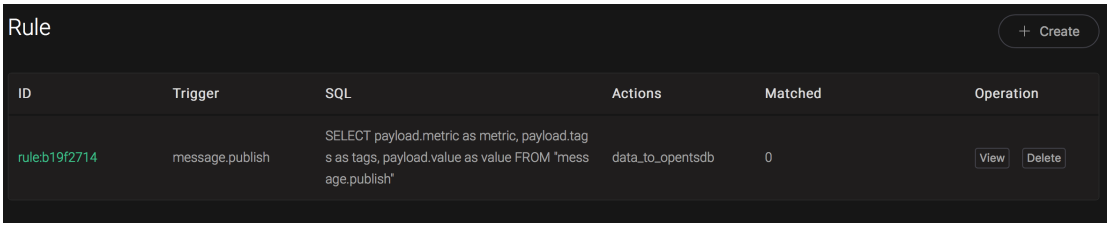
   Keep all the default configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



5. Back to the "Actions" dialog, and then click on the "Confirm" button.



6. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



7. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

   Retained: false

   Payload: {"metric":"cpu","tags":{"host":"serverA"},"value":12}

   Then inspect the OpenTSDB table, verify a new record has been inserted:

```
## Use postman to send an HTTP request to the opentsdb server:
POST /api/query HTTP/1.1
Host: 127.0.0.1:4242
Content-Type: application/json
cache-control: no-cache
Postman-Token: 69af0565-27f8-41e5-b0cd-d7c7f5b7a037
{
  "start": 1560409825000,
  "queries": [
      {
          "aggregator": "last",
          "metric": "cpu",
          "tags": {
              "host": "*"
          }
      }
  ],
  "showTSUIDs": "true",
  "showQuery": "true",
  "delete": "false"
}
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

The response should look like following:

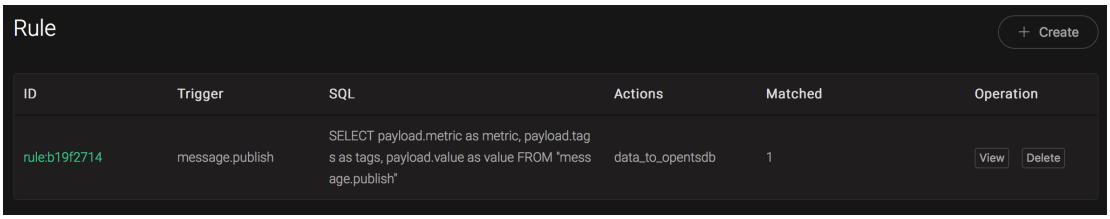
```
[
  {
      "metric": "cpu",
      "tags": {
          "host": "serverA"
      },
      "aggregateTags": [],
      "query": {
          "aggregator": "last",
          "metric": "cpu",
          "tsuids": null,
          "downsample": null,
          "rate": false,
          "filters": [
              {
                  "tagk": "host",
                  "filter": "*",
                  "group_by": true,
                  "type": "wildcard"
              }
          ],
          "index": 0,
          "tags": {
              "host": "wildcard(*)"
          },
          "rateOptions": null,
          "filterTagKs": [
```

()

()

```
                "AAAC"
            ],
            "explicitTags": false
        },
        "tsuids": [
            "000002000002000007"
        ],
        "dps": {
            "1561532453": 12
        }
    }
]
```

And from the rule list, verify that the "Matched" column has increased to 1:



8.8 Create TimescaleDB Rules

0. Setup a TimescaleDB database, taking Mac OSX for instance:

```
$ docker pull timescale/timescaledb

$ docker run -d --name timescaledb -p 5432:5432 -e POSTGRES_
↪PASSWORD=password timescale/timescaledb:latest-pg11

$ docker exec -it timescaledb psql -U postgres

## create tutorial database
> CREATE database tutorial;

> \c tutorial

> CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

1. Initiate the TimescaleDB table:

```
$ docker exec -it timescaledb psql -U postgres -d tutorial

CREATE TABLE conditions (
  time        TIMESTAMPTZ       NOT NULL,
  location    TEXT              NOT NULL,
  temperature DOUBLE PRECISION  NULL,
  humidity    DOUBLE PRECISION  NULL
```

()

()

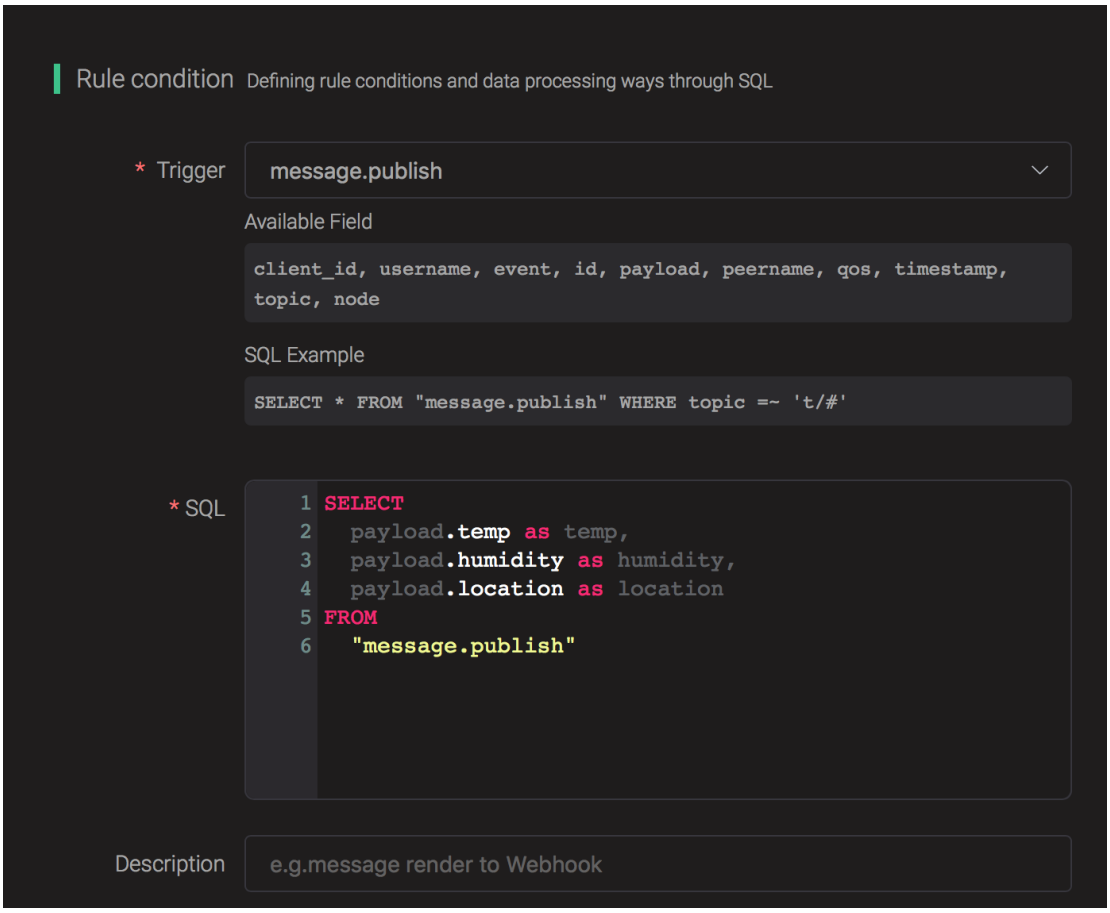
```
);

SELECT create_hypertable('conditions', 'time');
```

2. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

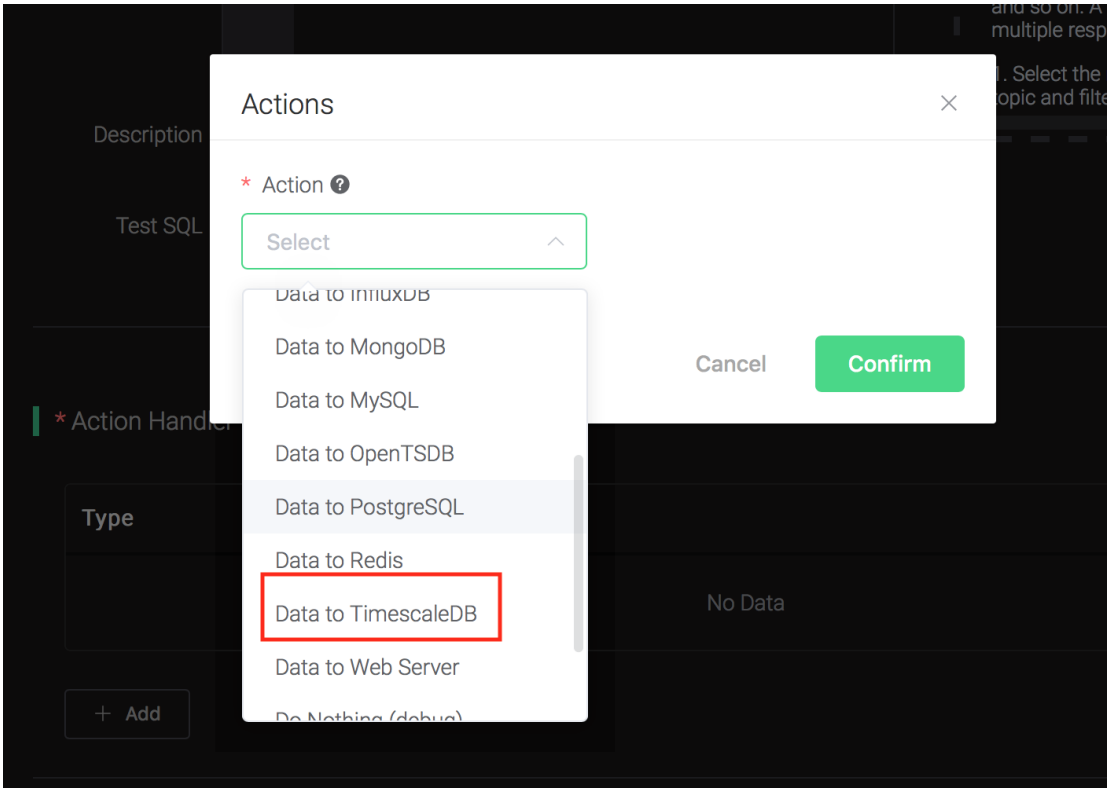
   Select "message.publish", then type in the following SQL:

```
SELECT
  payload.temp as temp,
  payload.humidity as humidity,
  payload.location as location
FROM
  "message.publish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to TimescaleDB" in the pop-up dialog window.



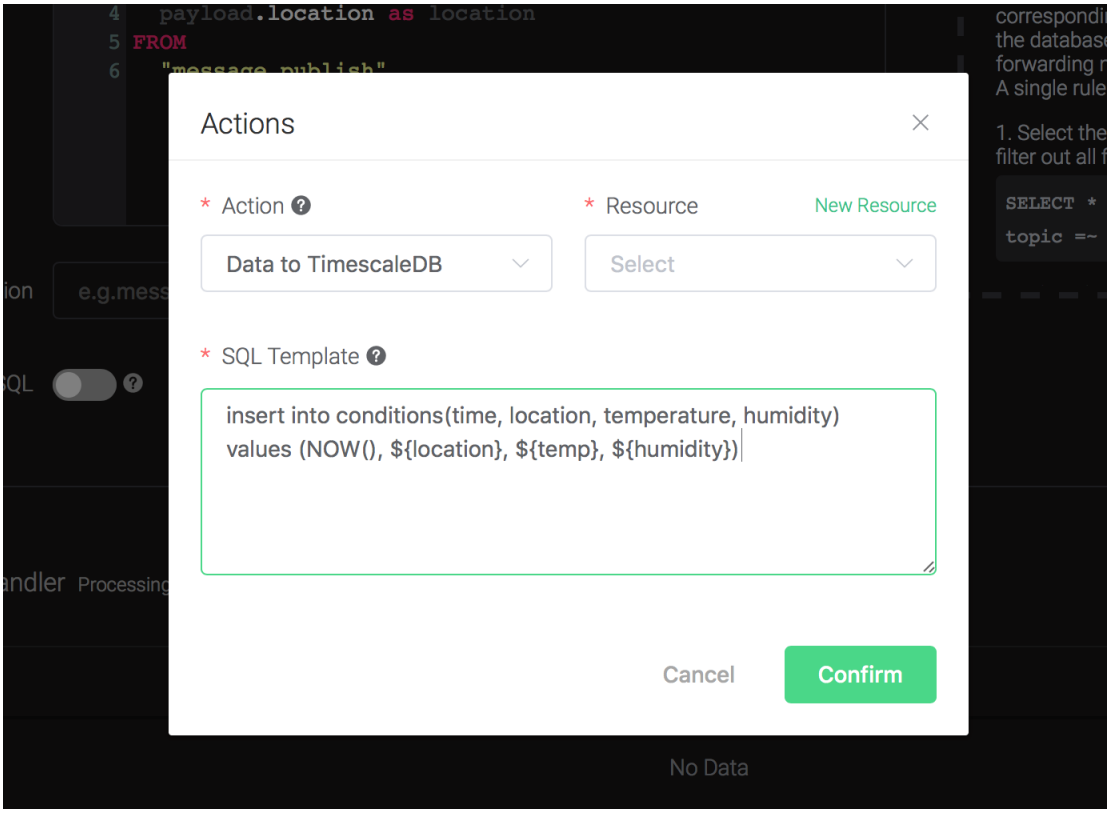
4. Fill in the parameters required by the action:

   Two parameters is required by action "Data to TimescaleDB":

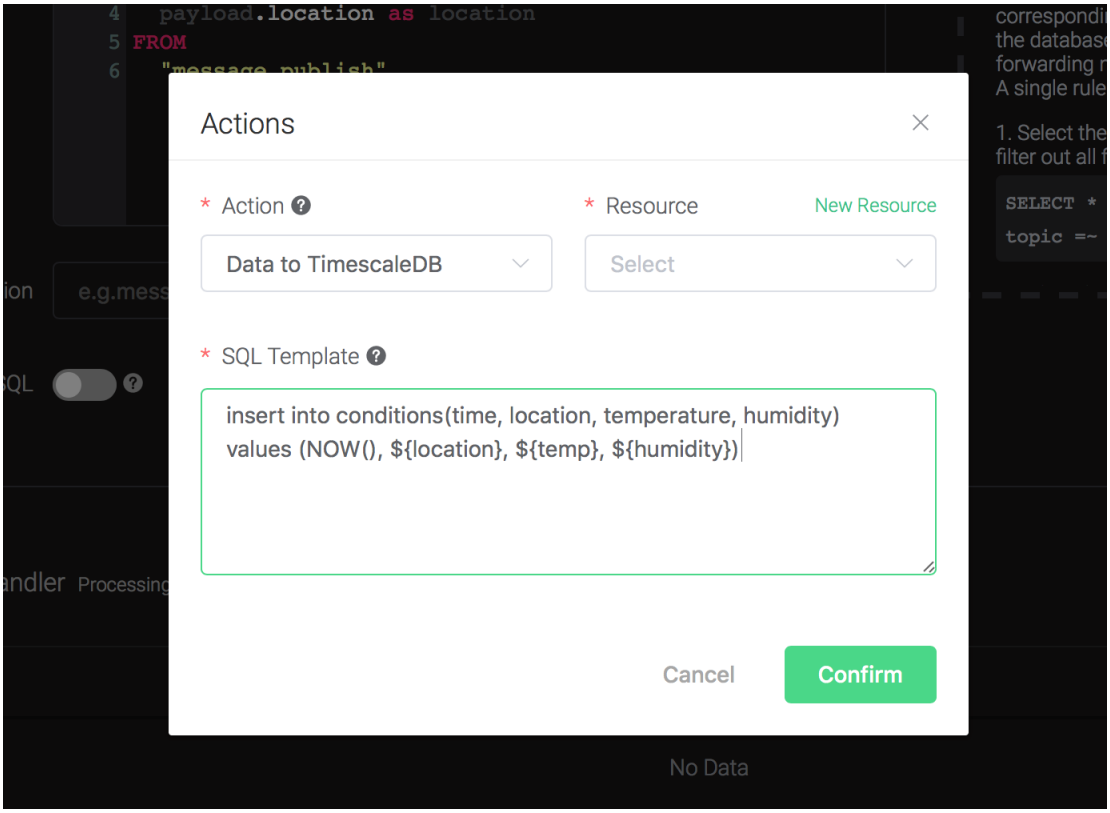
   1). SQL template. SQL template is the sql command you'd like to run when the action is triggered. In this example we'll insert a message into timescaledb, so type in the following sql template:

```
insert into conditions(time, location, temperature, humidity) values␣
↪(NOW(), ${location}, ${temp}, ${humidity})
```

   Before data is inserted into the table, placeholders like ${key} will be replaced by the corresponding values.

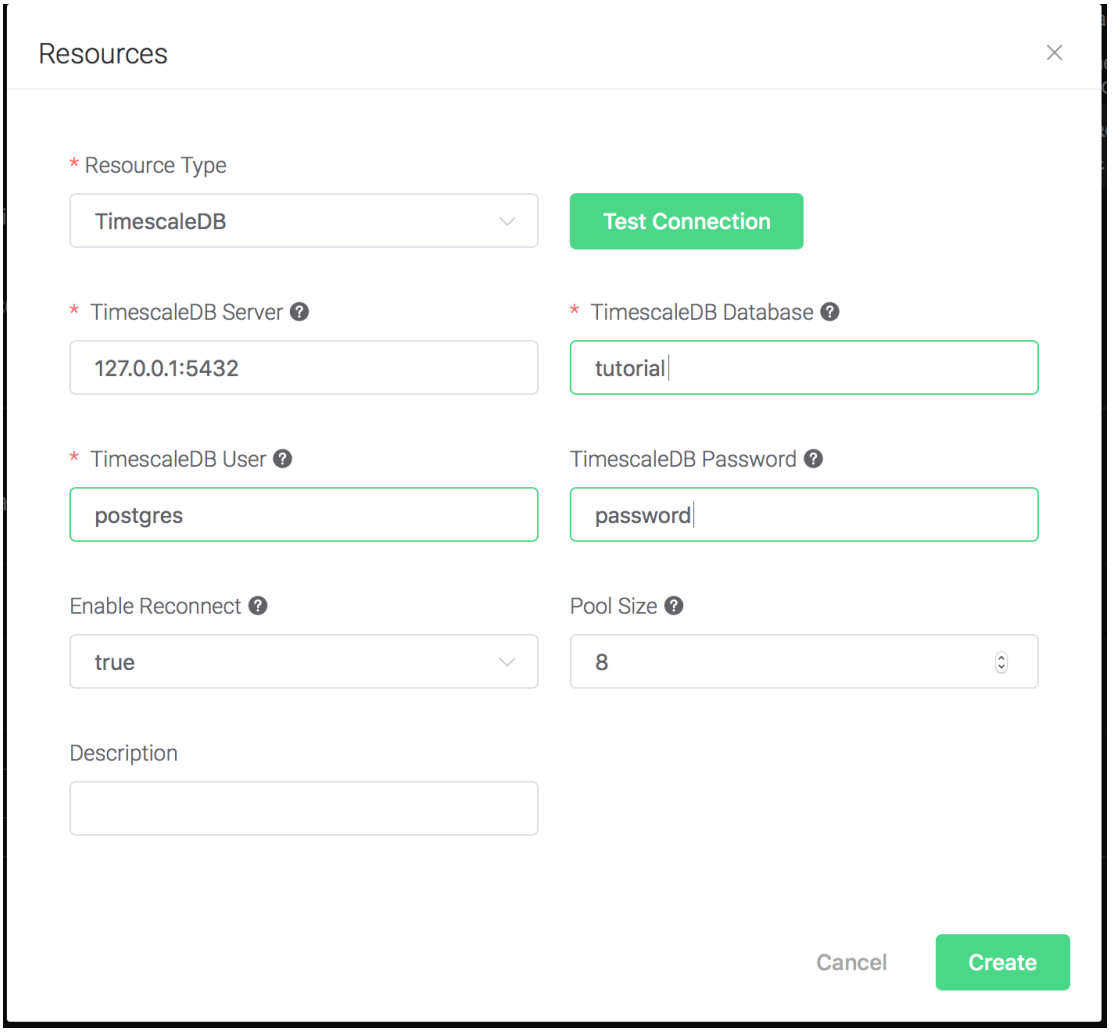


2). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "TimescaleDB":

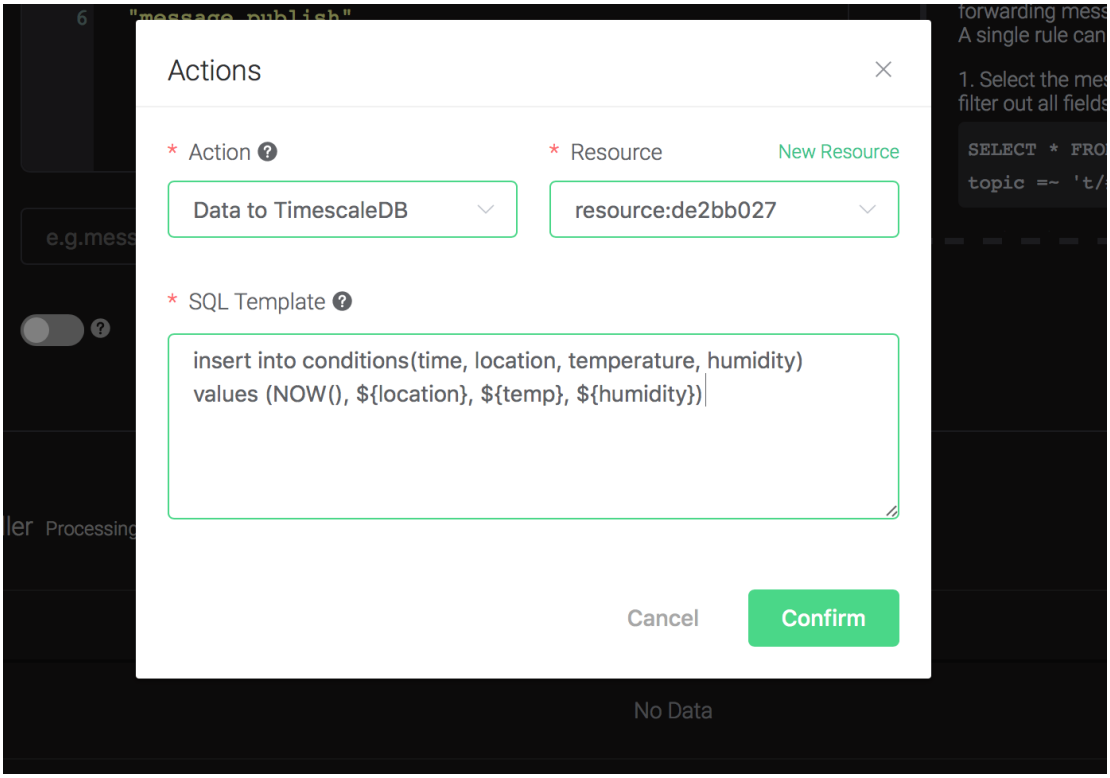


5. Configure the resource:

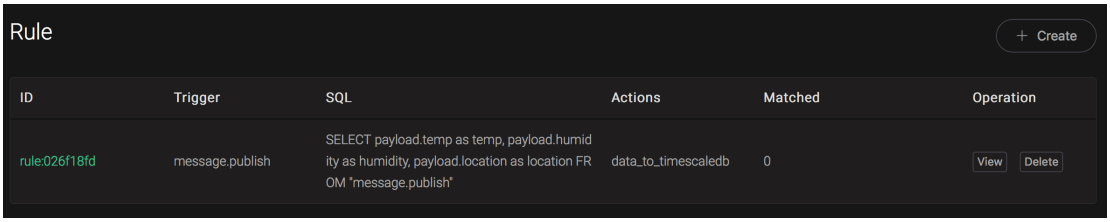
   Set "TimescaleDB Database" to "tutorial", "TimescaleDB User" to "postgres", "TimescaleDB Password" to "password", and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

   Retained: false

   Payload: {"temp":24,"humidity":30,"location":"hangzhou"}

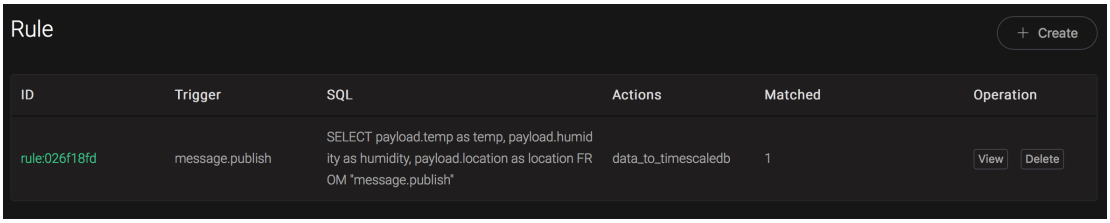
   Then inspect the TimescaleDB table, verify a new record has been inserted:

   tutorial=# SELECT * FROM conditions LIMIT 100;

   The output data could look like this:

```
            time               | location | temperature | humidity
-------------------------------+----------+-------------+----------
2019-06-27 01:41:08.752103+00 | hangzhou |          24 |       30
```

   And from the rule list, verify that the "Matched" column has increased to 1:



8.9 Create InfluxDB Rules

0. Setup a InfluxDB database, taking Mac OSX for instance:

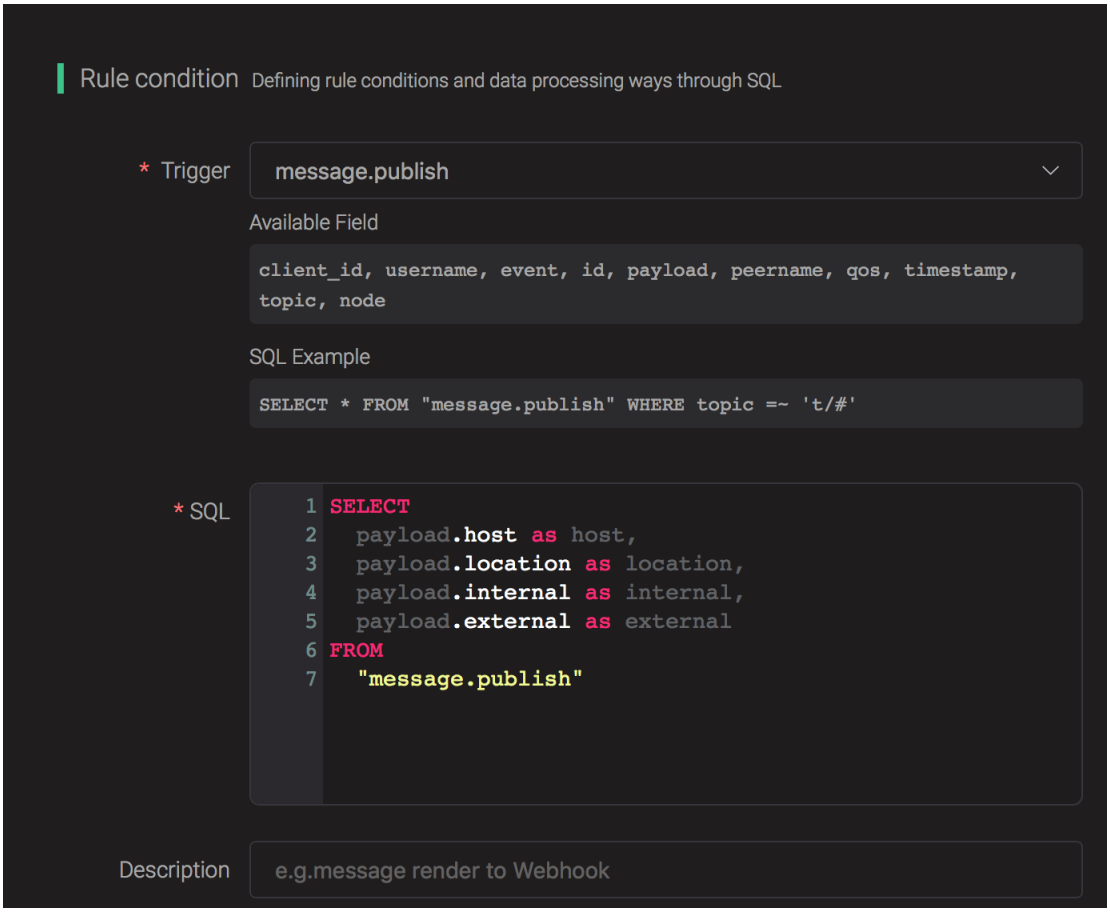
```
$ docker pull influxdb

$ git clone -b v1.0.0 https://github.com/palkan/influx_udp.git

$ cd influx_udp

$ docker run --name=influxdb --rm -d -p 8086:8086 -p 8089:8089/udp -v $
↪{PWD}/files/influxdb.conf:/etc/influxdb/influxdb.conf:ro -e INFLUXDB_
↪DB=db influxdb:latest
```

1. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

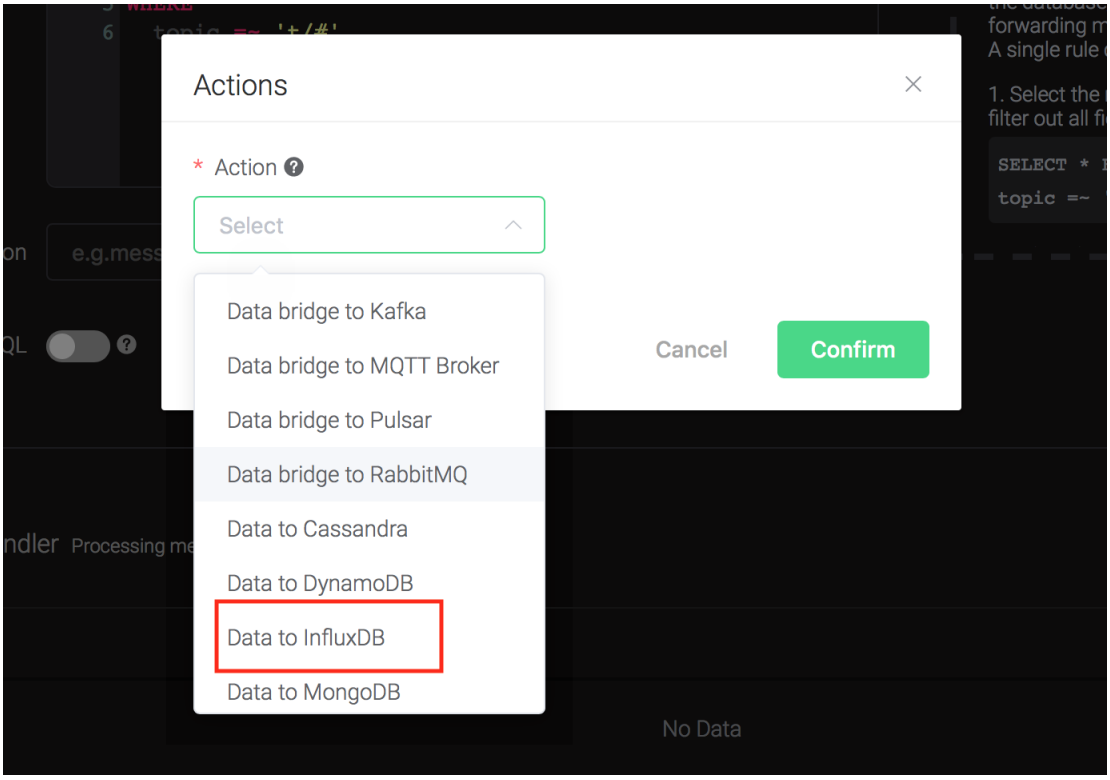
   Select "message.publish", then type in the following SQL:

```
SELECT
  payload.host as host,
  payload.location as location,
  payload.internal as internal,
  payload.external as external
FROM
  "message.publish"
```



2. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to InfluxDB" in the pop-up dialog window.



3. Fill in the parameters required by the action:

   Six parameters is required by action "Data to InfluxDB" :

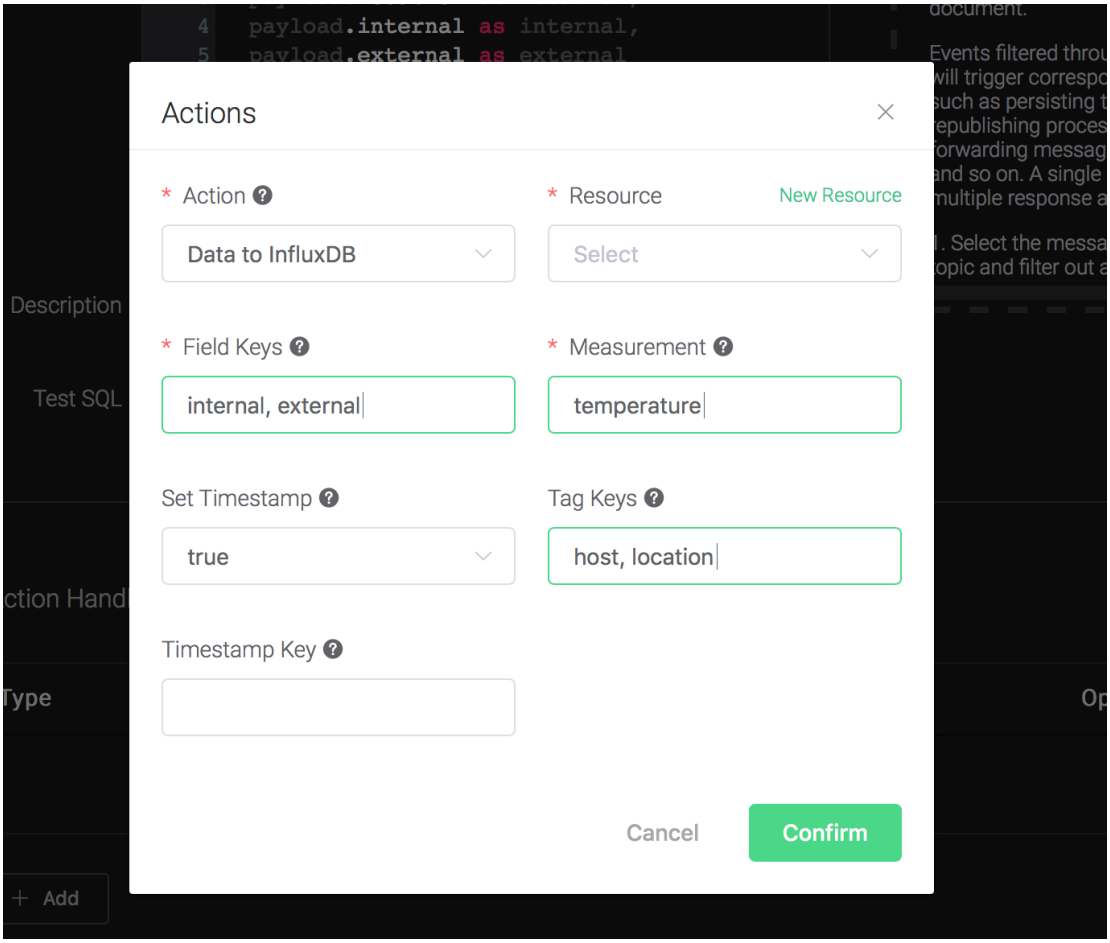
   1). Measurement: Measurement of the data point.

   2). Field Keys: Which fields can be used as fields of data point.

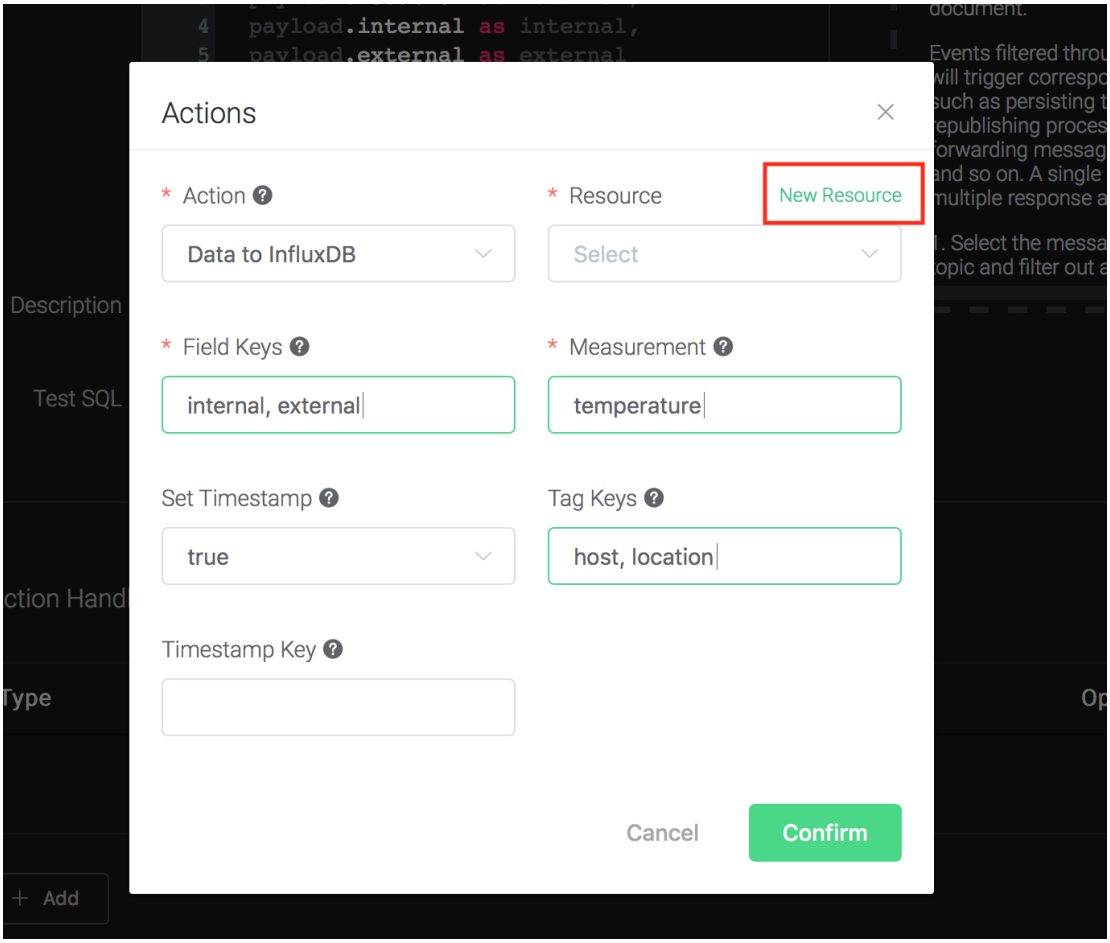
   3). Tags Keys. Which fields can be used as tags of data point.

   4). Timestamp Key. Which fields can be used as timestamp of data point.

   5). Set Timestamp. Whether to generate a timestamp if 'Timestamp Key' is not configured.

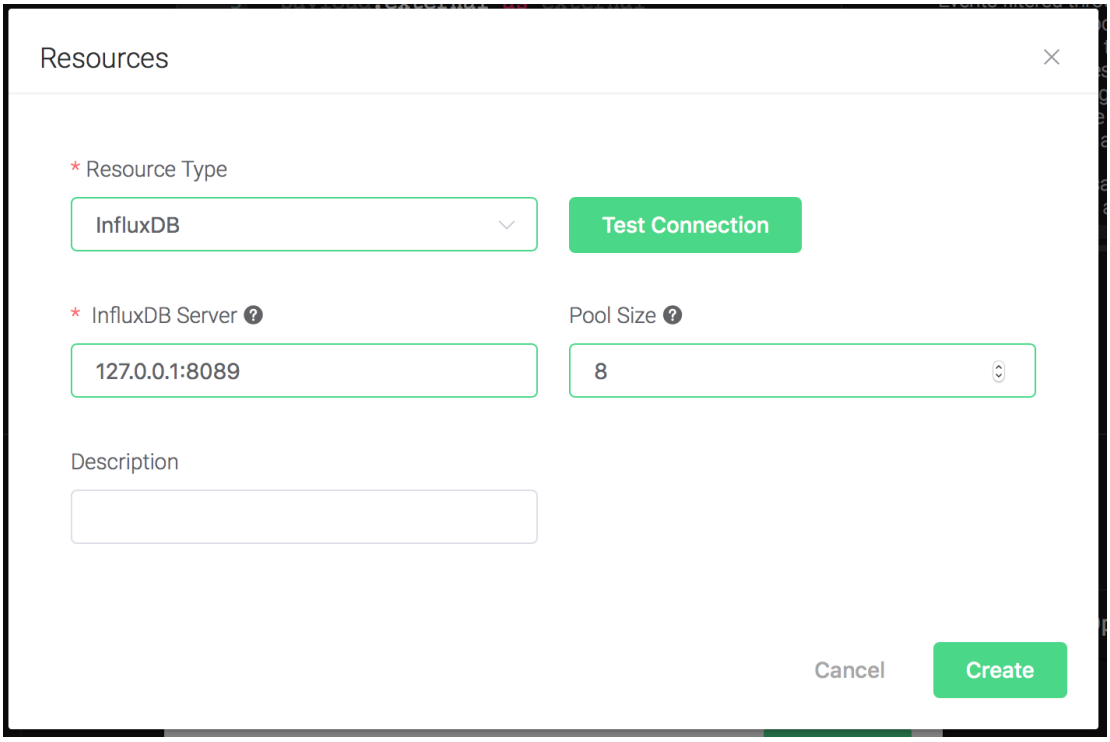


6). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "InfluxDB":

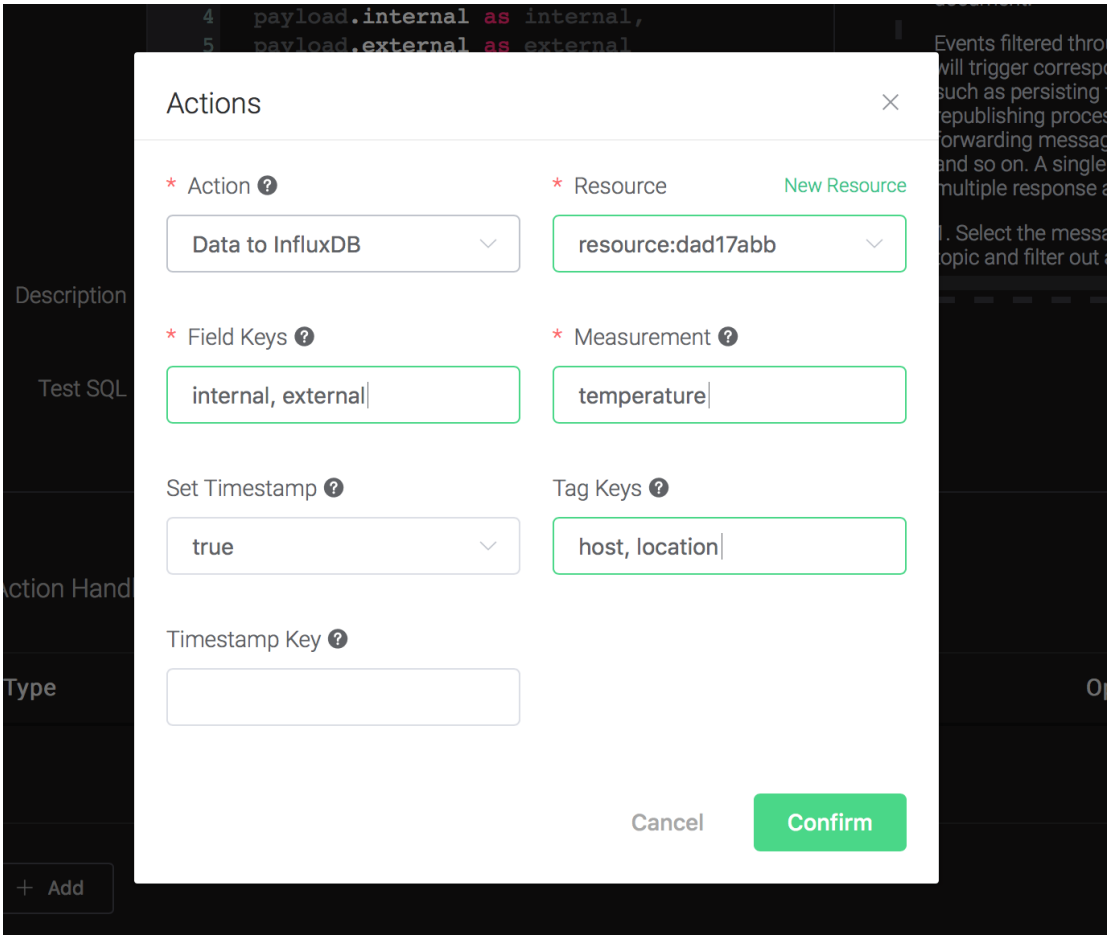


4. Configure the resource:

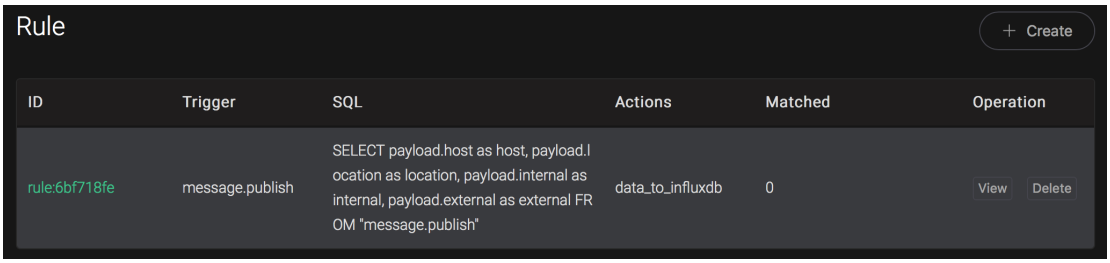
   Keep all the configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



5. Back to the "Actions" dialog, and then click on the "Confirm" button.



6. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



7. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

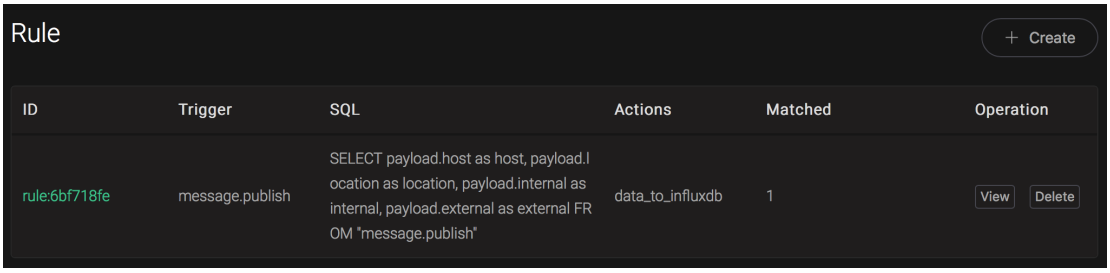
   Retained: false

   Payload: {"host":"serverA","location":"roomA","internal":25,"external":37}

   Then inspect the InfluxDB table, verify a new record has been inserted:

```
$ docker exec -it influxdb influx

> use db
Using database db
> select * from "temperature"
name: temperature
time                external host    internal location
----                -------- ----    -------- --------
1561535778444457348 35       serverA 25       roomA
```

And from the rule list, verify that the "Matched" column has increased to 1:



8.10 Creat WebHook Rules

0. Setup a Web Service, here we setup a simple web service using the linux tool `nc`:

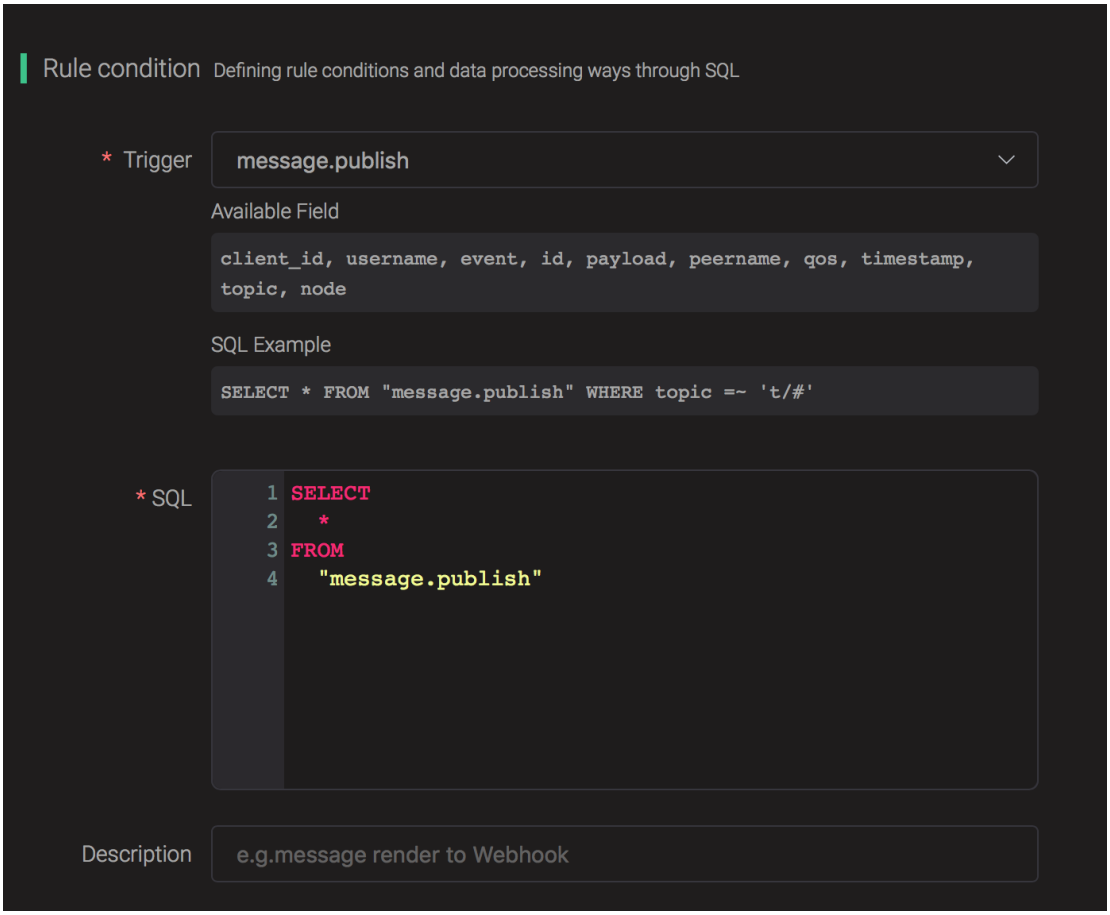
```
$ while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l 127.0.0.1␣
↪9901; done;
```

1. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

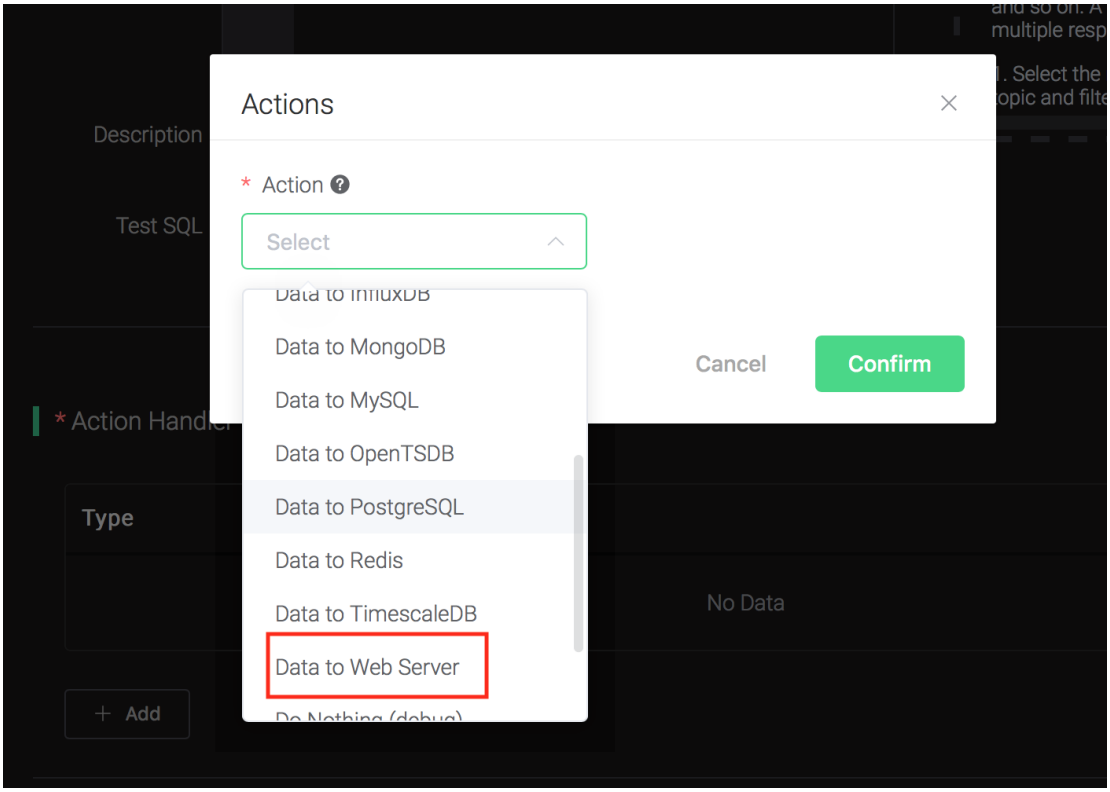
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



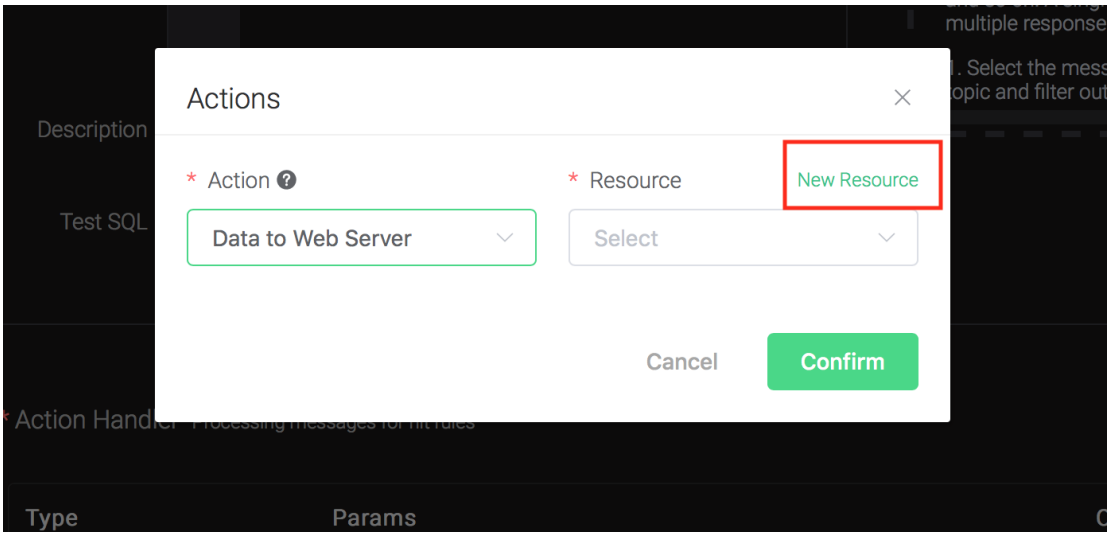
2. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data to Web Server" in the pop-up dialog window.



3. Bind a resource to the action:

   Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "WebHook":



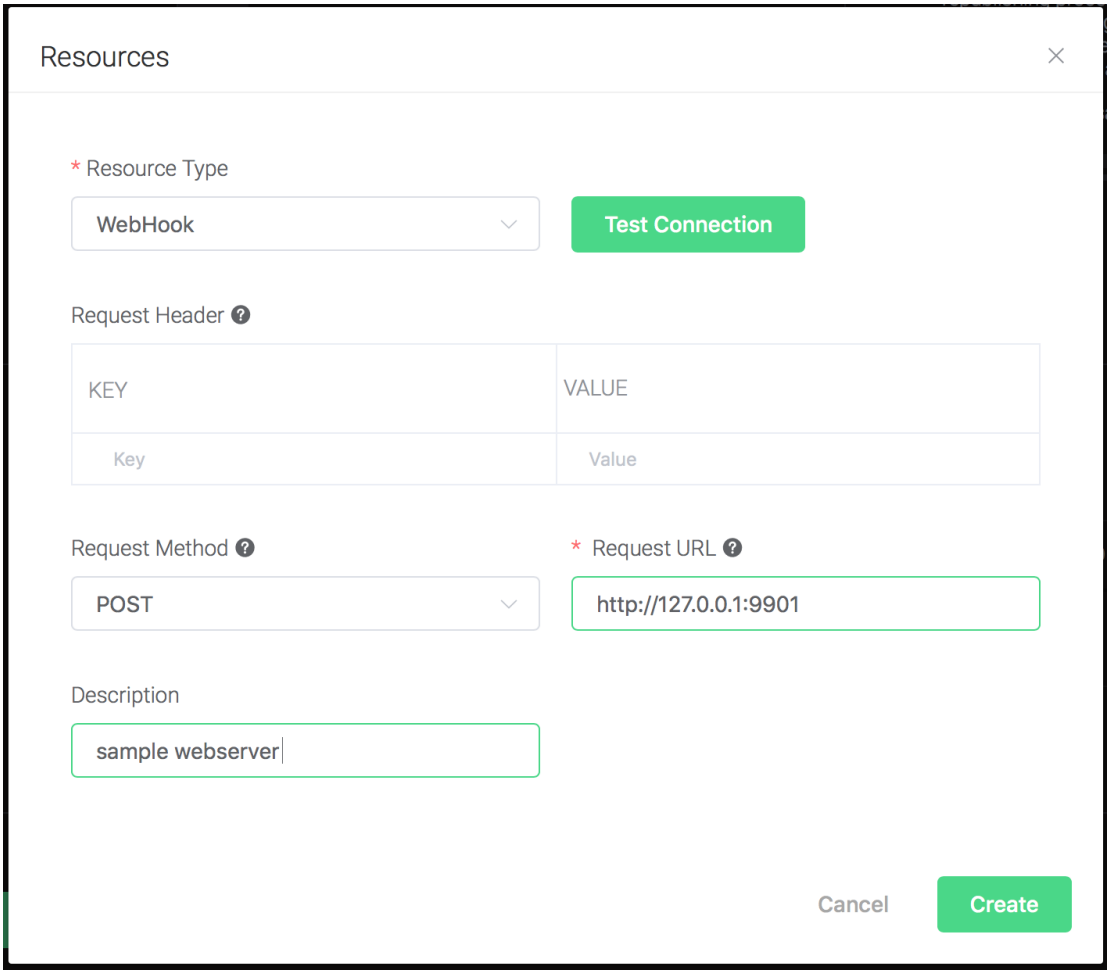
4. Configure the resource:

   Fill in the "Request URL" and "Request Header"(Optional):

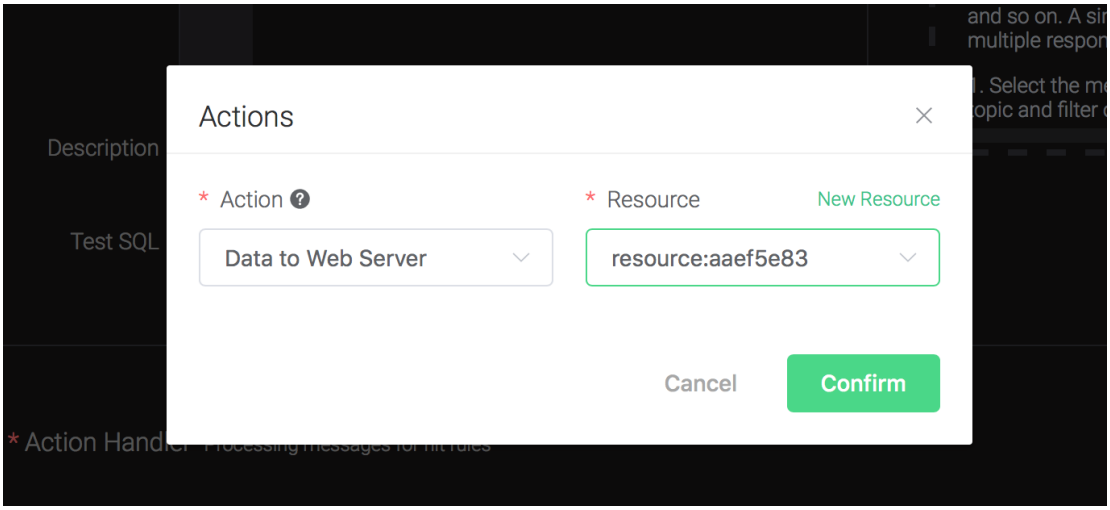
```
http://127.0.0.1:9901
```

   And click on the "Testing Connection" button to make sure the connection can be created suc-

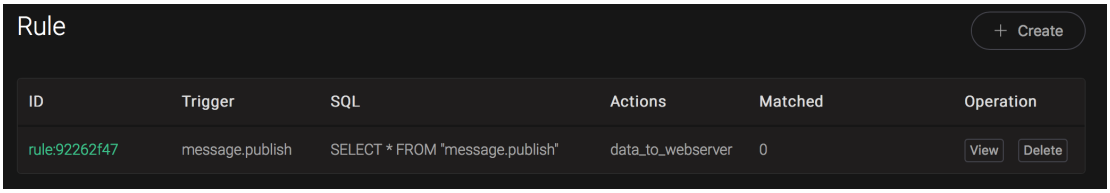
cessfully, and then click on the "Create" button.



5. Back to the "Actions" dialog, and then click on the "Confirm" button.



6. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



7. We have finished, testing the rule by sending an MQTT message to emqx:

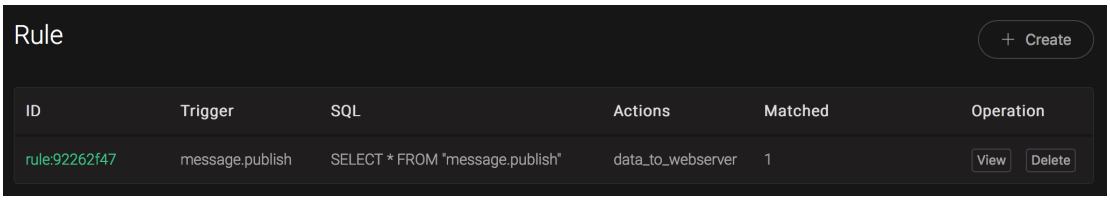
   Topic: "t/1"

   QoS: 1

   Payload: "Hello web server"

   Then inspect the Web Service table, verify a new record has been received:

   And from the rule list, verify that the "Matched" column has increased to 1:



8.11 Create Kafka Rules

0. Setup a Kafka, taking Mac OSX for instance:

```
$ wget http://apache.claz.org/kafka/2.3.0/kafka_2.12-2.3.0.tgz

$ tar -xzf  kafka_2.12-2.3.0.tgz

$ cd kafka_2.12-2.3.0

# start Zookeeper
$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
# start Kafka
$ ./bin/kafka-server-start.sh config/server.properties
```

1. Create topics for Kafka:

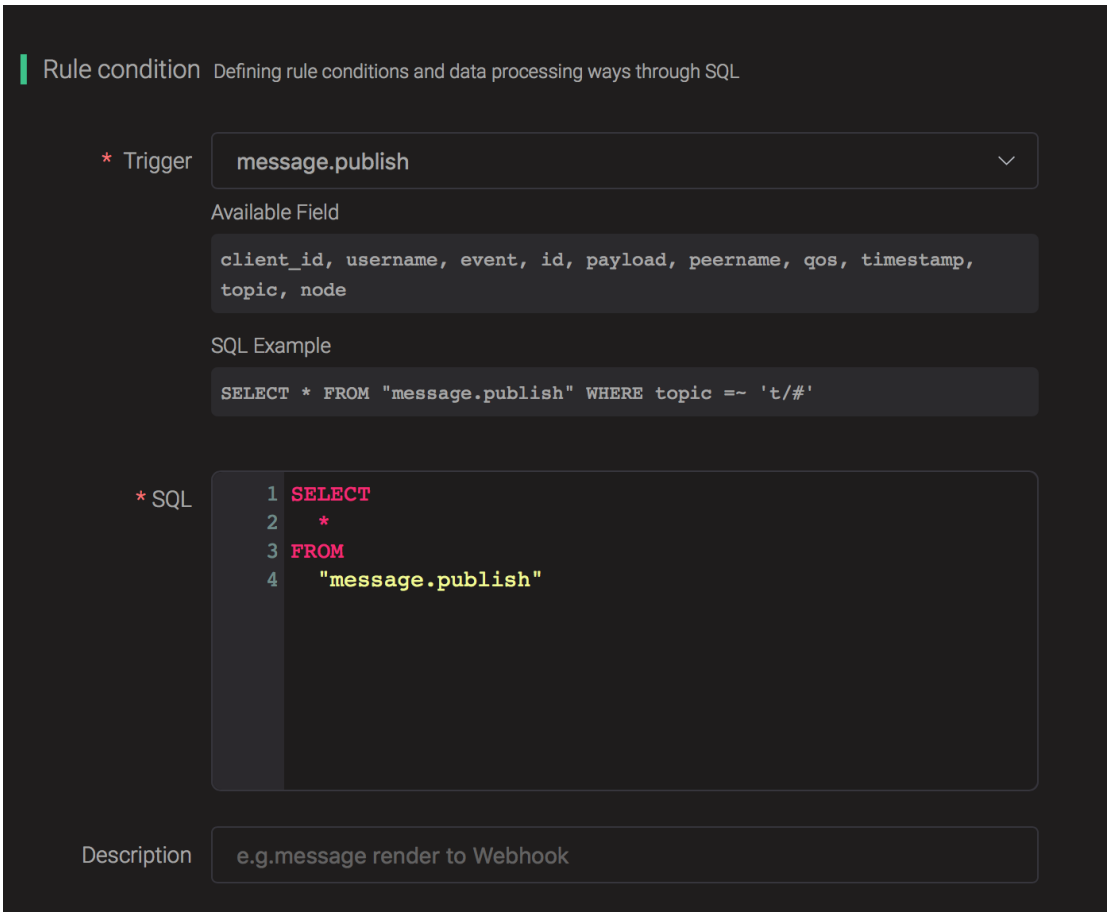
```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --replication-factor␣
↪1 --partitions 1 --topic testTopic --create

Created topic testTopic.

.. note:: Kafka topics should be created before creating the kafka rule,␣
↪or the rule creation would not success.
```

2. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

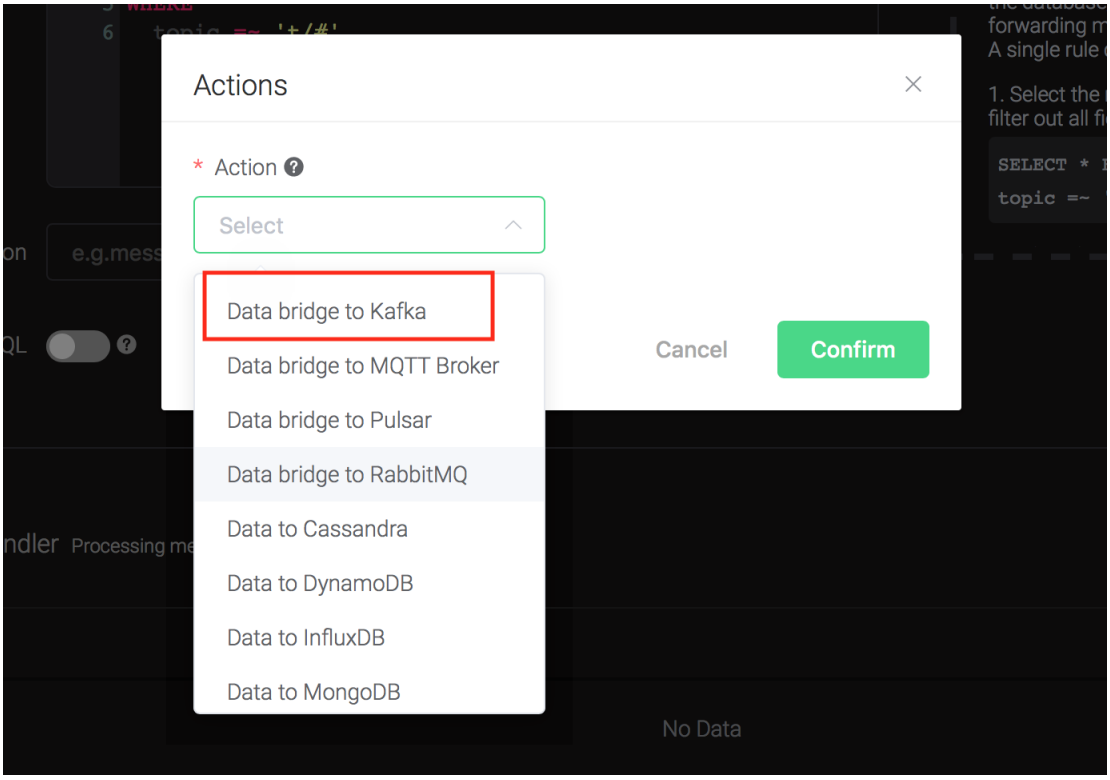
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data bridge to Kafka" in the pop-up dialog window.

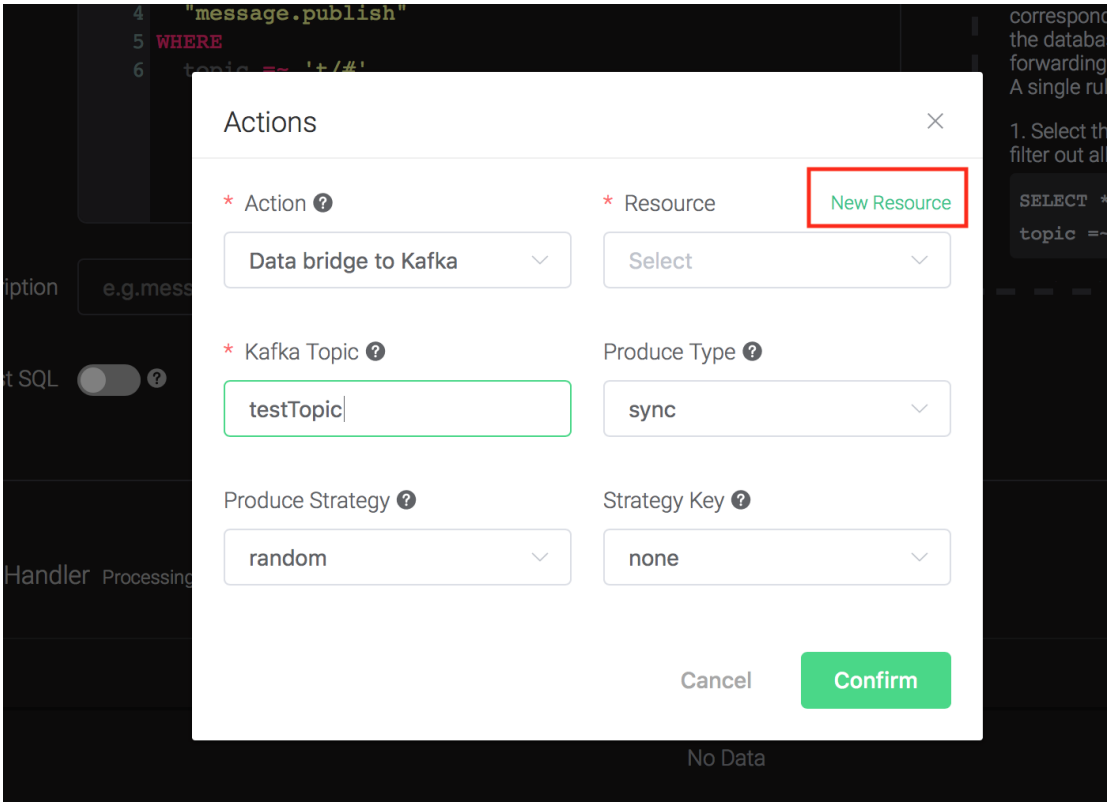


4. Fill in the parameters required by the action:

   Two parameters is required by action "Data to Kafka":

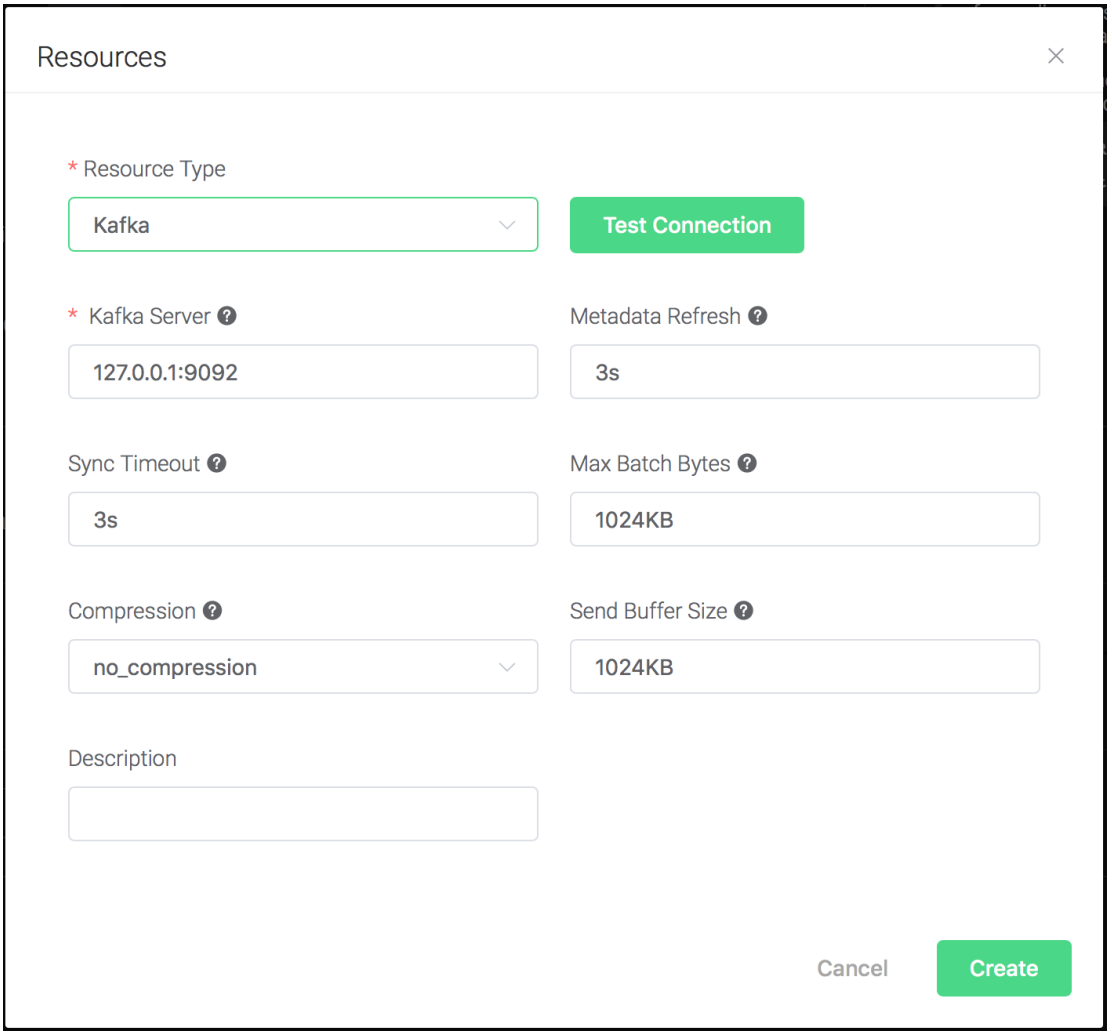
   1). Kafka Topic

   2). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "Kafka":

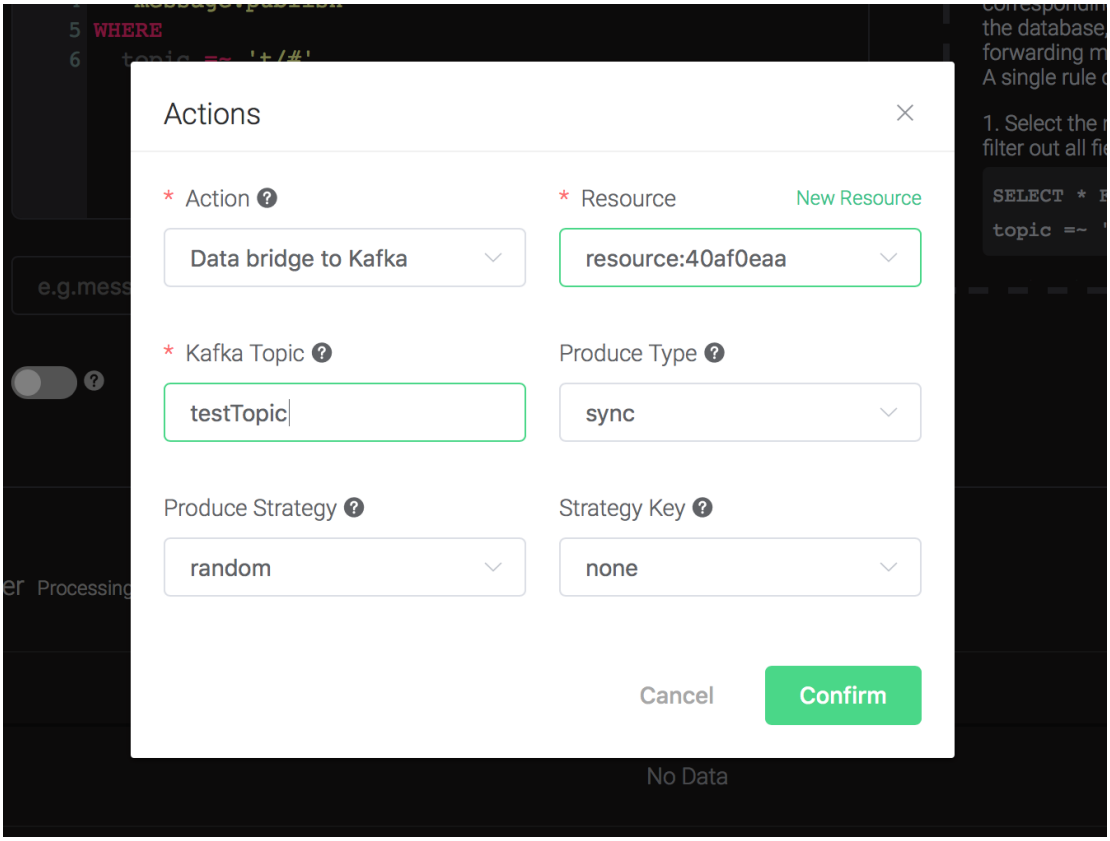


5. Configure the resource:

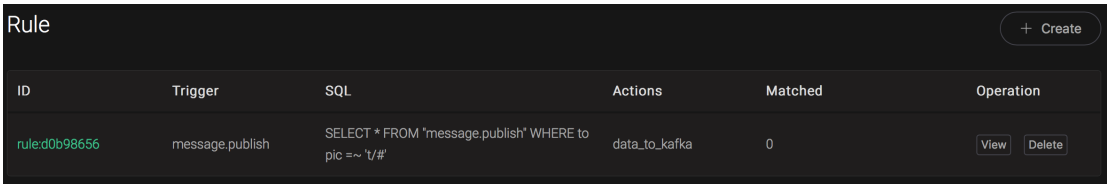
   Set the "Kafka Server" to "127.0.0.1:9092" (multiple servers should be separated by comma), and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

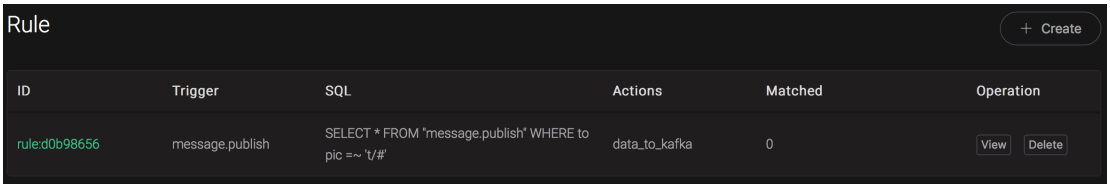
   Retained: false

   Payload: "hello"

   Then inspect Kafka by consume from the topic:

```
$ ./bin/kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092
↪--topic testTopic --from-beginning
```

   And from the rule list, verify that the "Matched" column has increased to 1:



8.12 Create Pulsar Rules

0. Setup a Pulsar, taking Mac OSX for instance:

```
$ wget http://apache.mirrors.hoobly.com/pulsar/pulsar-2.3.2/apache-
↪pulsar-2.3.2-bin.tar.gz

$ tar xvfz apache-pulsar-2.3.2-bin.tar.gz

$ cd apache-pulsar-2.3.2

# Start Pulsar
$ ./bin/pulsar standalone
```

1. Create Pulsar Topics:

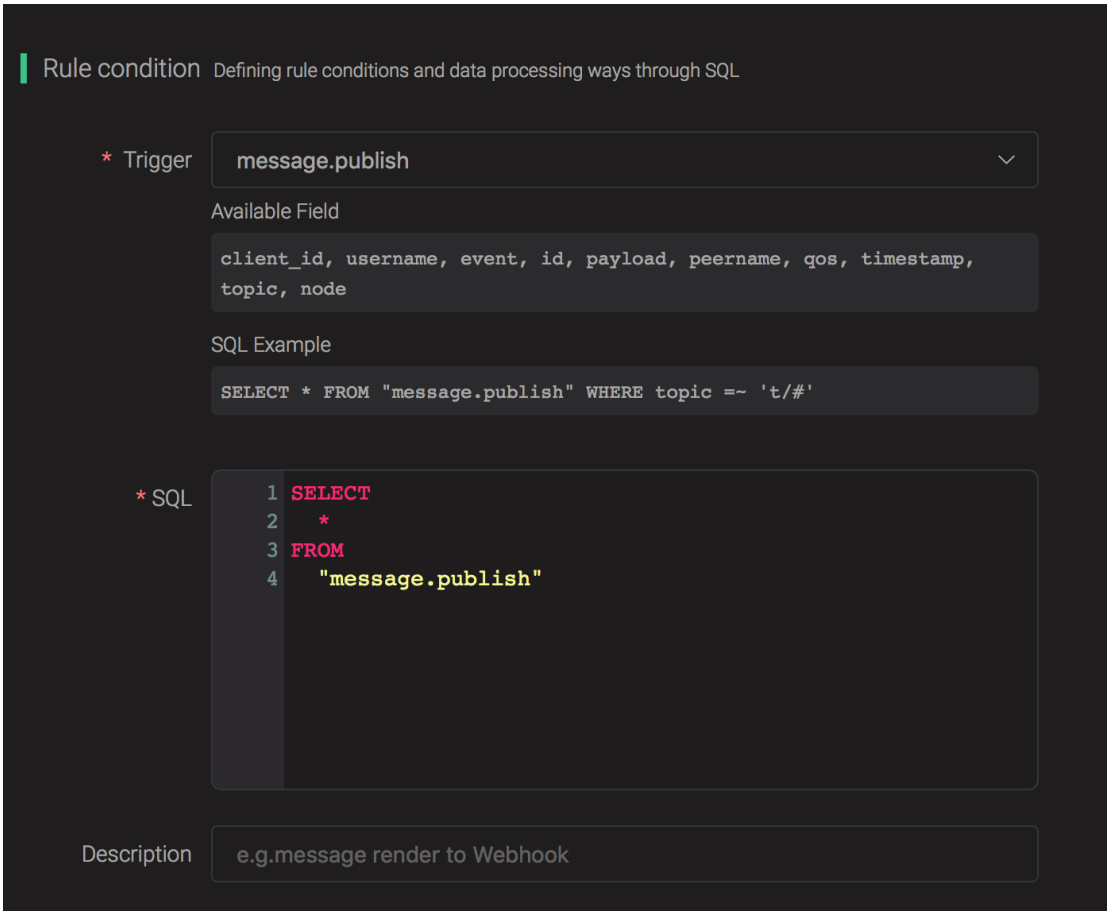
```
$ ./bin/pulsar-admin topics create-partitioned-topic -p 5 testTopic
```

2. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

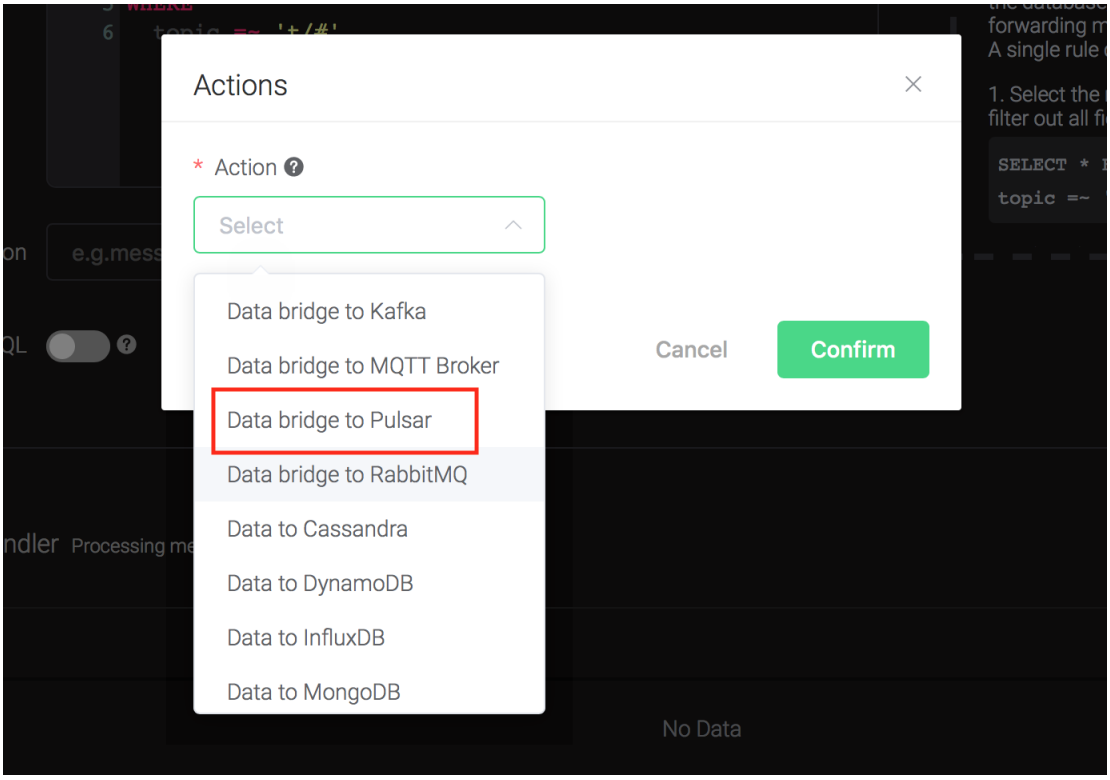
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



3. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data bridge to Pulsar" in the pop-up dialog window.

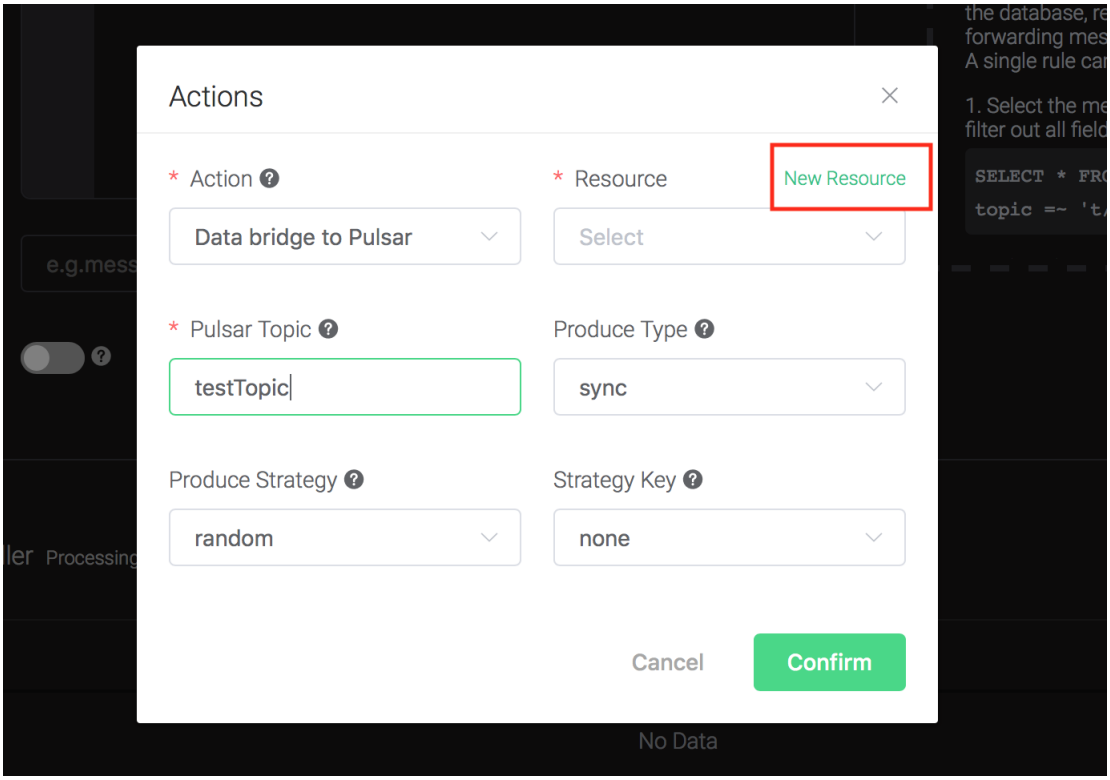


4. Fill in the parameters required by the action:

   Two parameters is required by action "Data bridge to Pulsar":

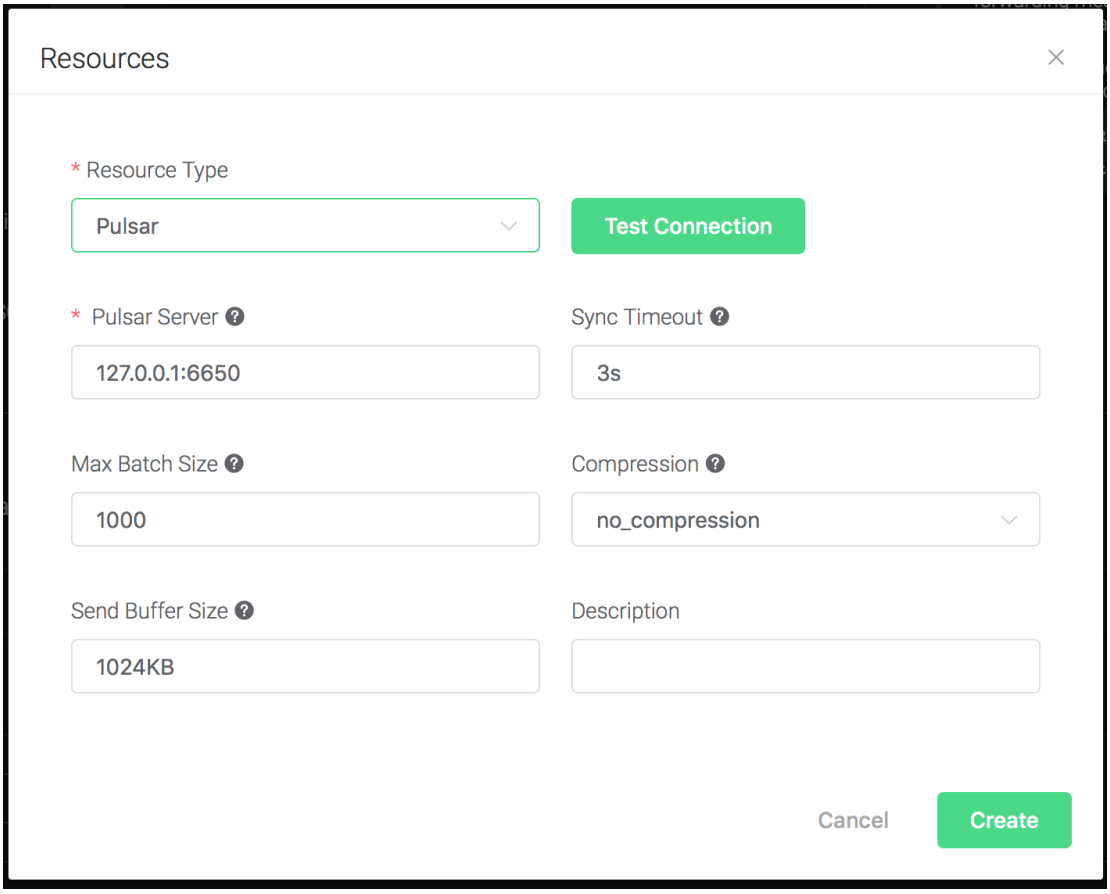
   1). Pulsar Topic

   2). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "Pulsar":

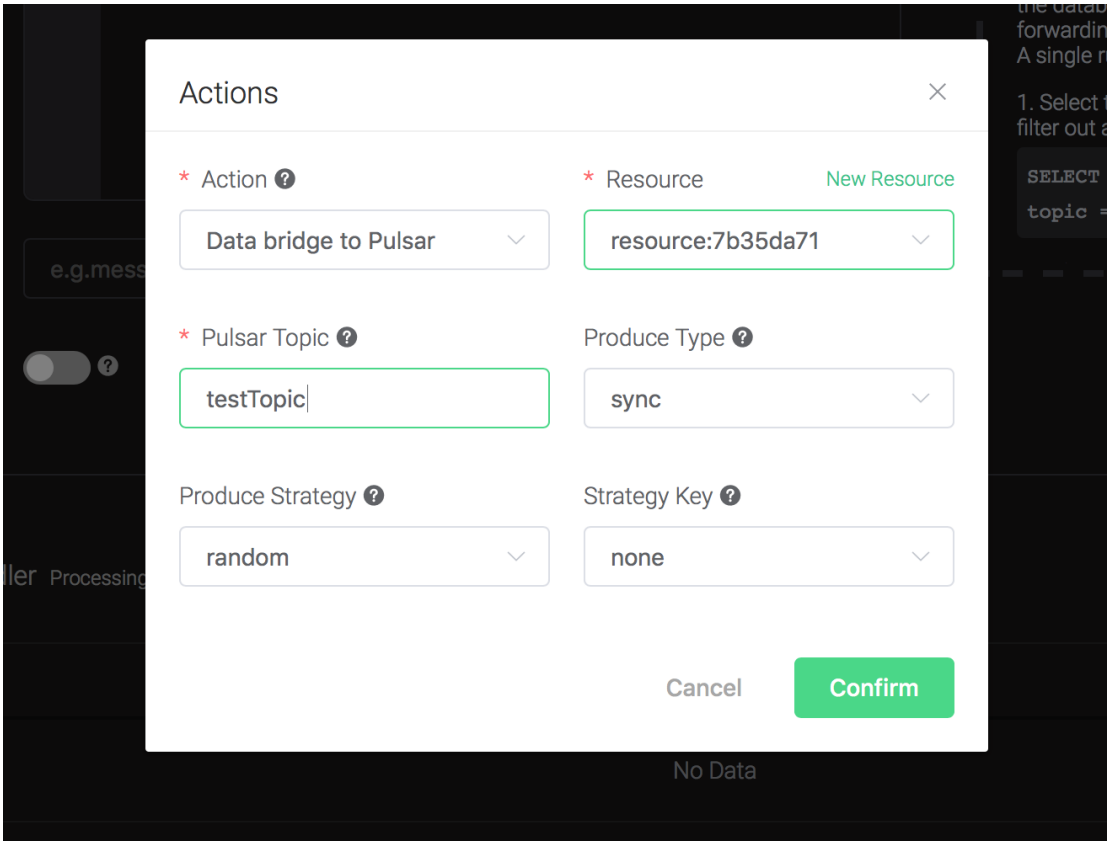


5. Configure the resource:

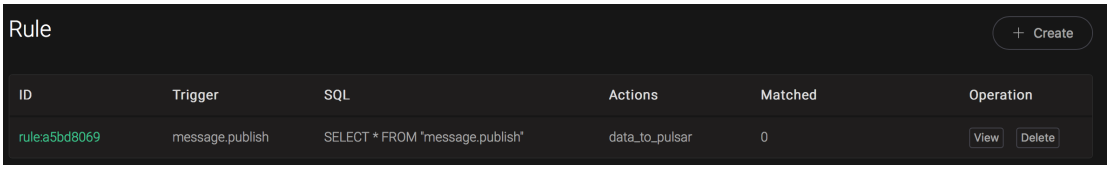
   Set the " Pulsar Server" to "127.0.0.1:6650"(multiple servers should be separated by comma), and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

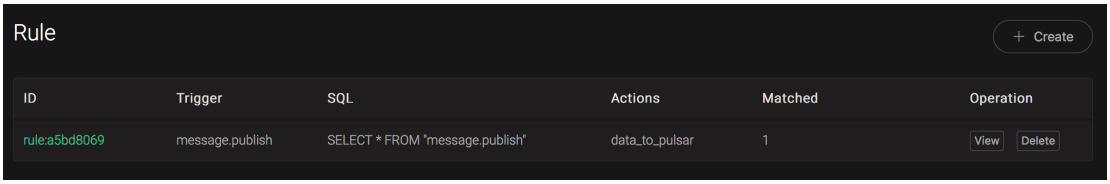
   Retained: false

   Payload: "hello"

   Then inspect the Pulsar by consume the topic, verify a new record has been produced:

   $ ./bin/pulsar-client consume testTopic -s "sub-name" -n 1000

   And from the rule list, verify that the "Matched" column has increased to 1:



8.13 Create RabbitMQ Rules

0. Setup a RabbitMQ, taking Mac OSX for instance:

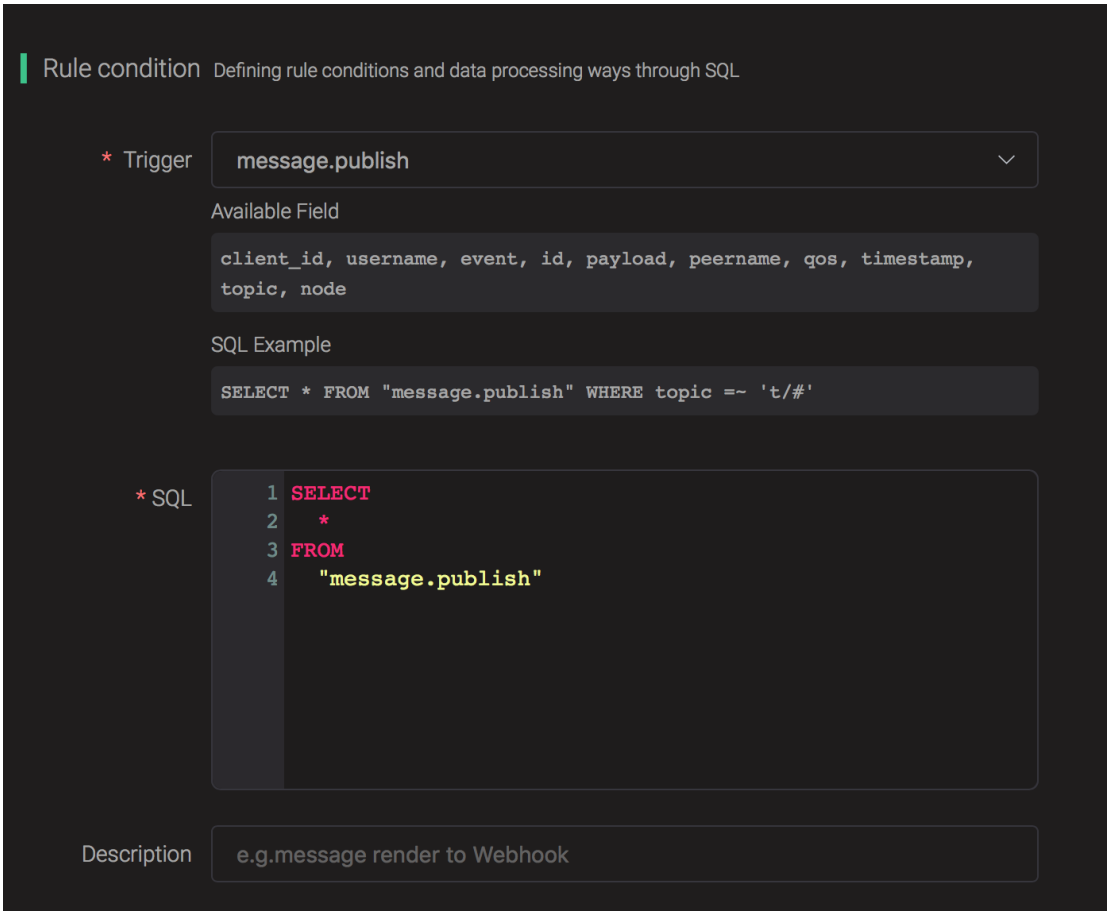
```
$ brew install rabbitmq

# start rabbitmq
$ rabbitmq-server
```

1. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

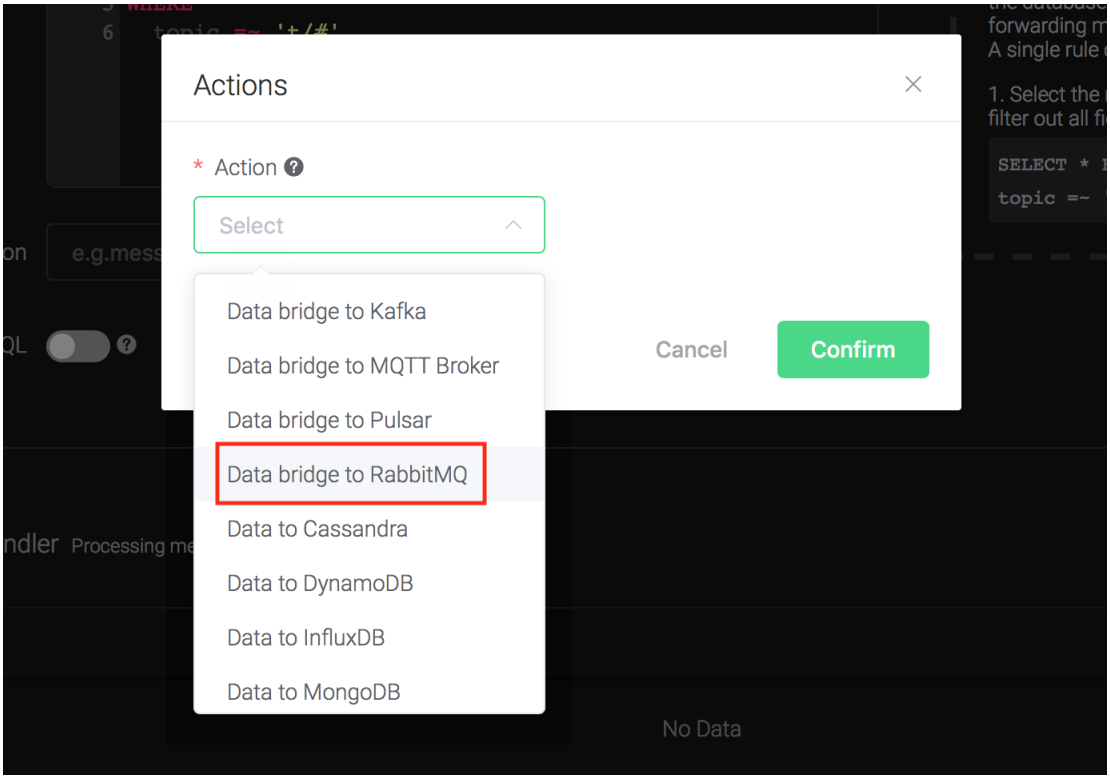
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```



2. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data bridge to RabbitMQ" in the pop-up dialog window.



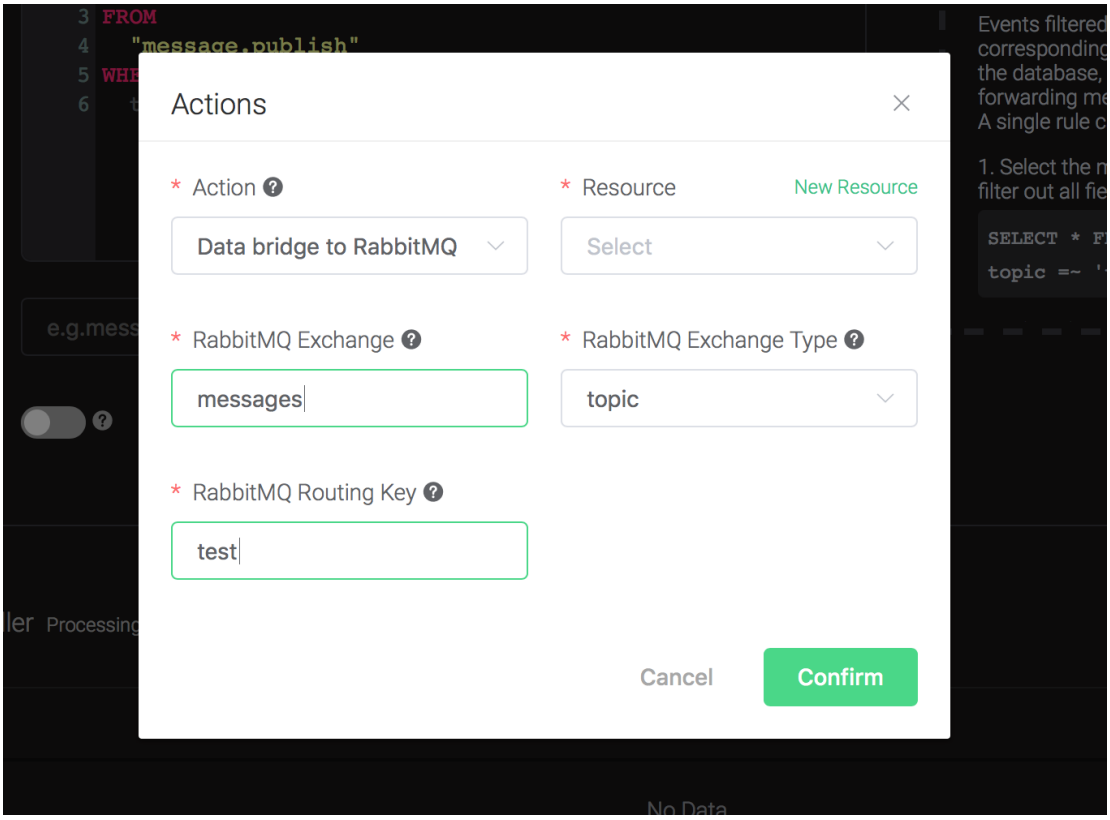
3. Fill in the parameters required by the action:

   Two parameters is required by action "Data bridge to RabbitMQ":

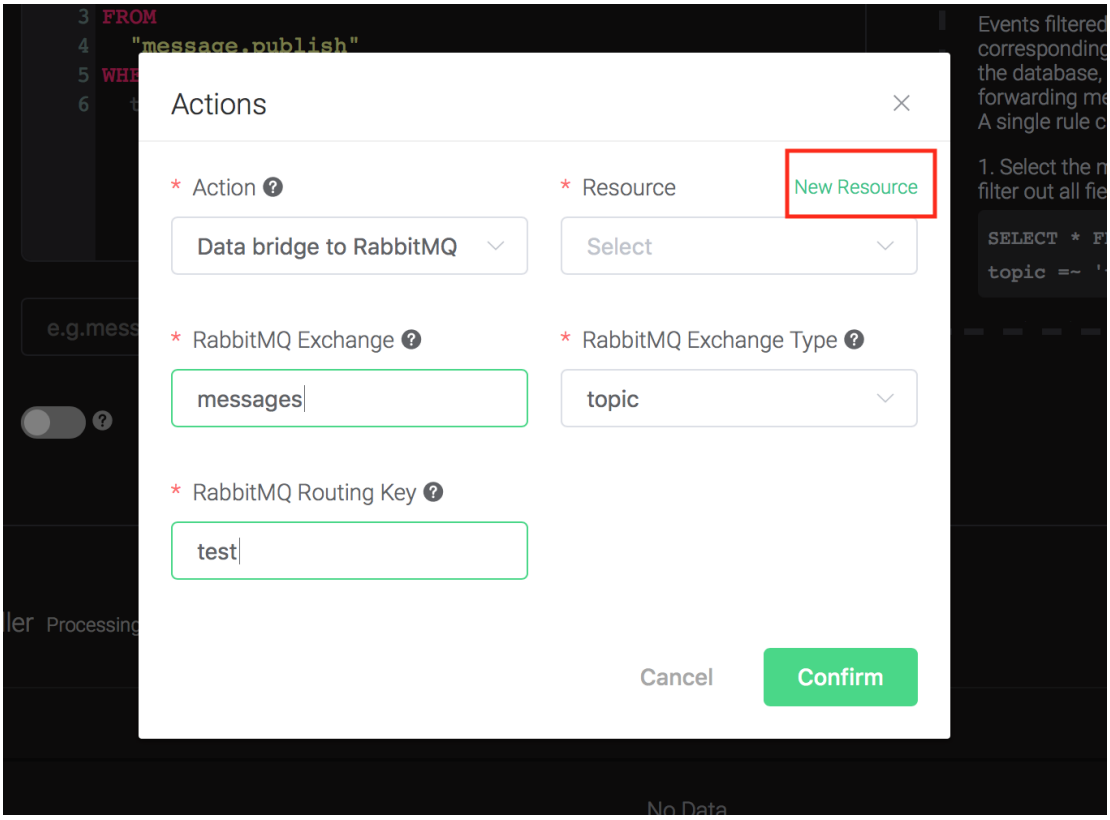
   1). RabbitMQ Exchange. Here set it to "messages"

   2). RabbitMQ Exchange Type. Here set it to "topic"

   3). RabbitMQ Routing Key. Here set it to "test"

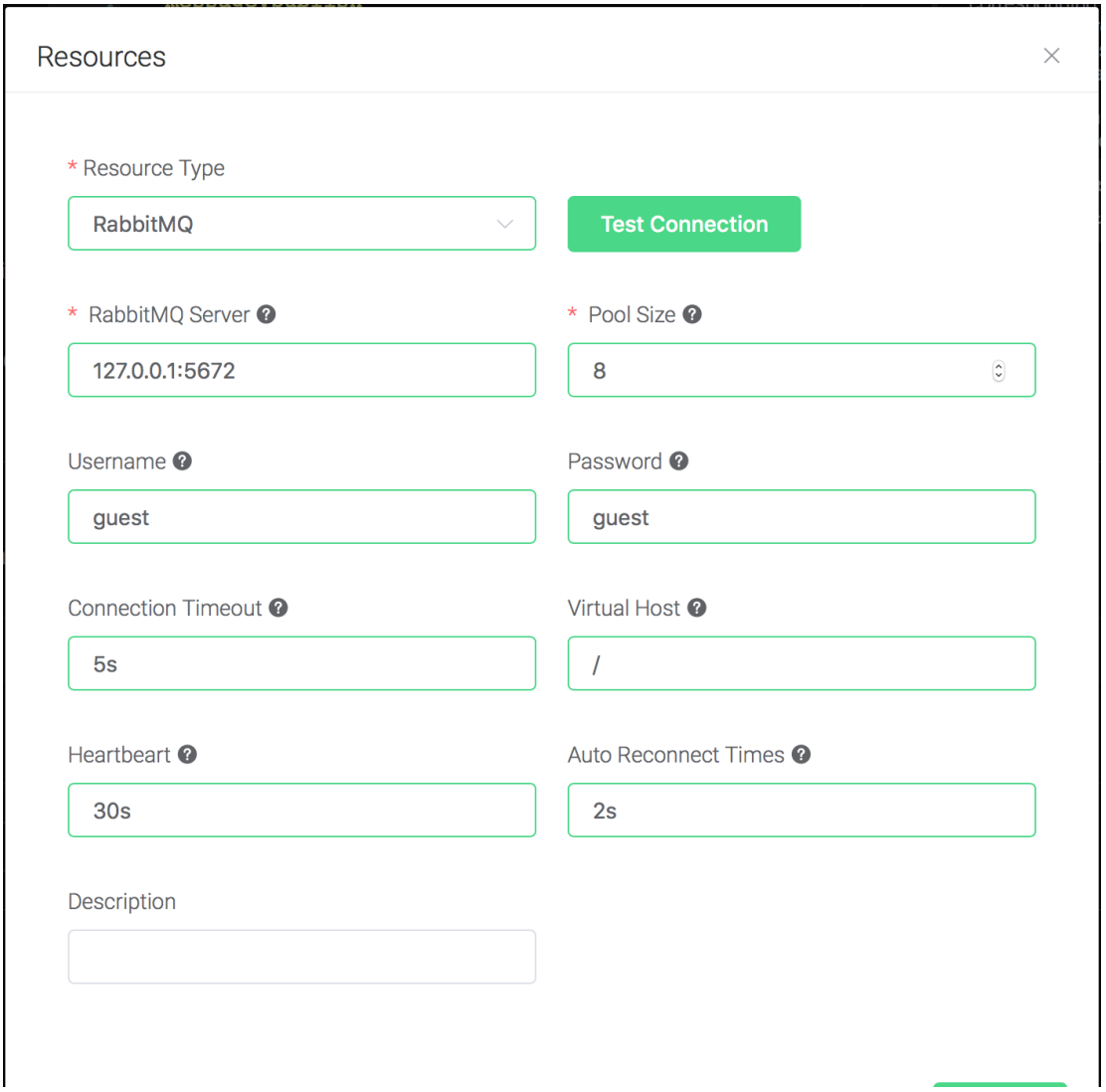


4). Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "RabbitMQ":

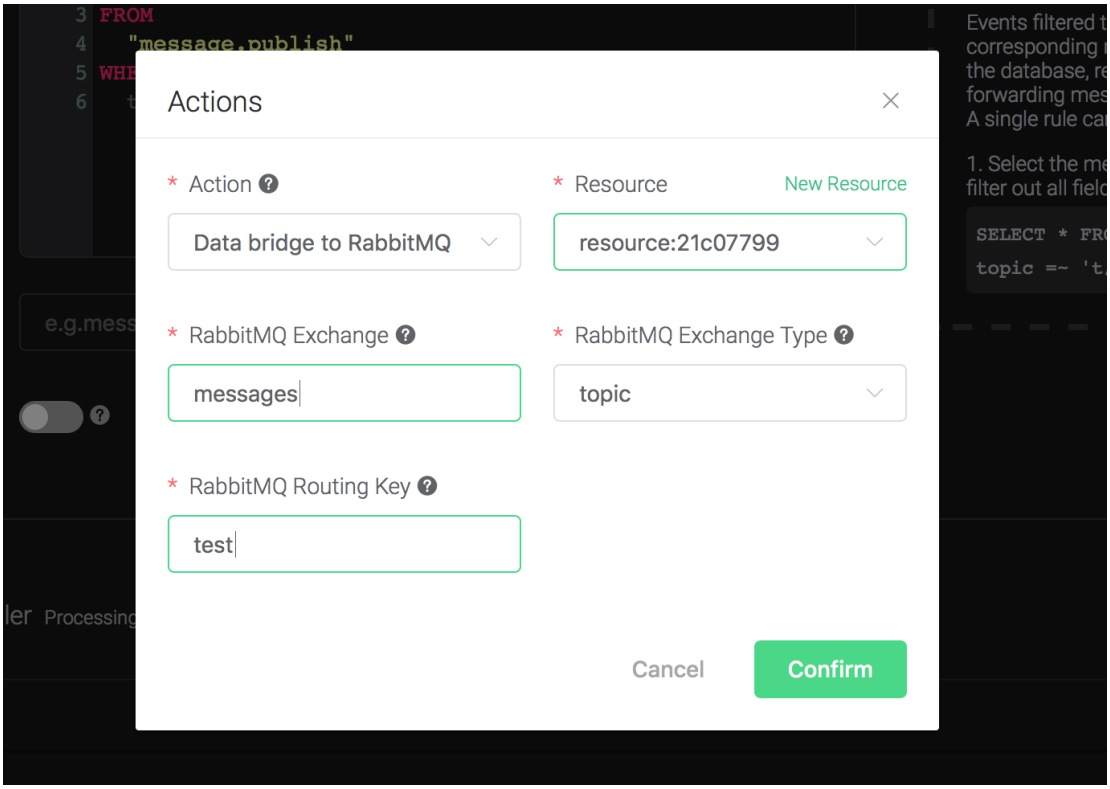


4. Configure the resource:

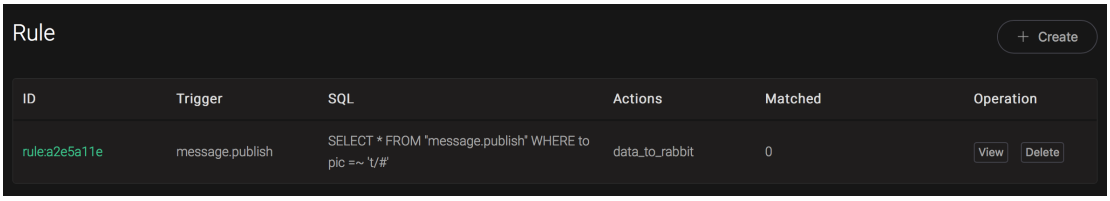
   Set "RabbitMQ Server" to "127.0.0.1:5672", and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



5. Back to the "Actions" dialog, and then click on the "Confirm" button.



6. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



7. We have finished, testing the rule by sending an MQTT message to emqx:

   Topic: "t/1"

   QoS: 0

   Retained: false

   Payload: "Hello, World!"

   Write an AMQP Client to consume the messages, following is the one written in python:

```
#!/usr/bin/env python
import pika

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='messages', exchange_type='topic')
```

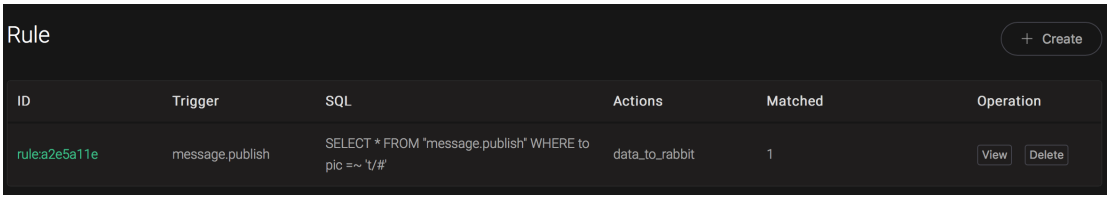
()

()

```
result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue

channel.queue_bind(exchange='messages', queue=queue_name, routing_
↪key='test')

print('[*] Waiting for messages. To exit press CTRL+C')

def callback(ch, method, properties, body):
    print(" [x] %r" % body)

channel.basic_consume(
    queue=queue_name, on_message_callback=callback, auto_ack=True)

channel.start_consuming()
```

```
└─➤ python amqp_client.py
[*] Waiting for messages. To exit press CTRL+C
 [x] '{"client_id":"mqttjs_8d8b9b6c7b","event":"message.publish","id":"58D7899C4EBEBF44300
000A7B0004","node":"emqx@127.0.0.1","payload":"{ \\"msg\\": \\"Hello, World!\\" }","peerna
me":"127.0.0.1:63998","qos":1,"retain":0,"timestamp":1562923998965,"topic":"t/1","username
":""}'
```

And from the rule list, verify that the "Matched" column has increased to 1:



8.14 Create BridgeMQTT Rules

0. Setup anther MQTT broker using mosquitto, change the port to 2883. Taking Mac OSX for instance:

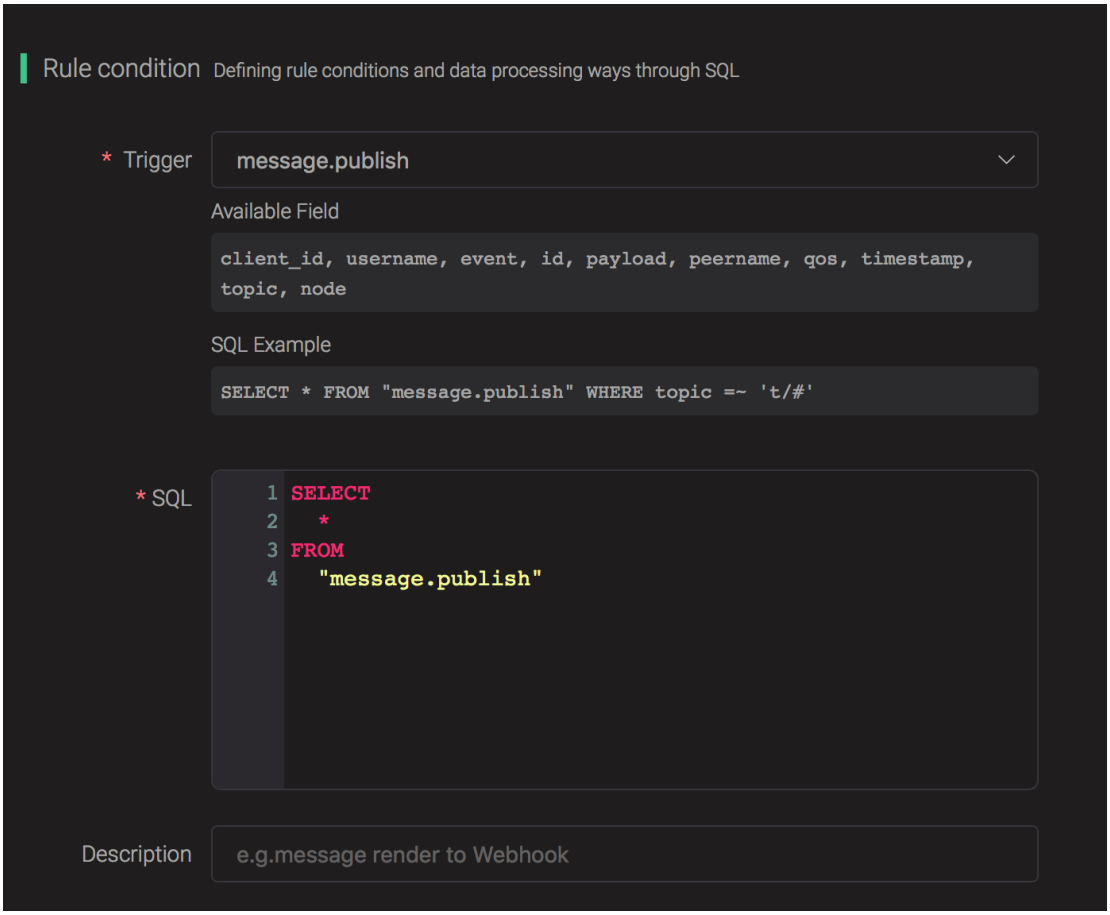
```
$ brew install mosquitto

$ vim /usr/local/etc/mosquitto/mosquitto.conf

port 2883

# start mosquitto
$ brew services start mosquitto
```

1. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

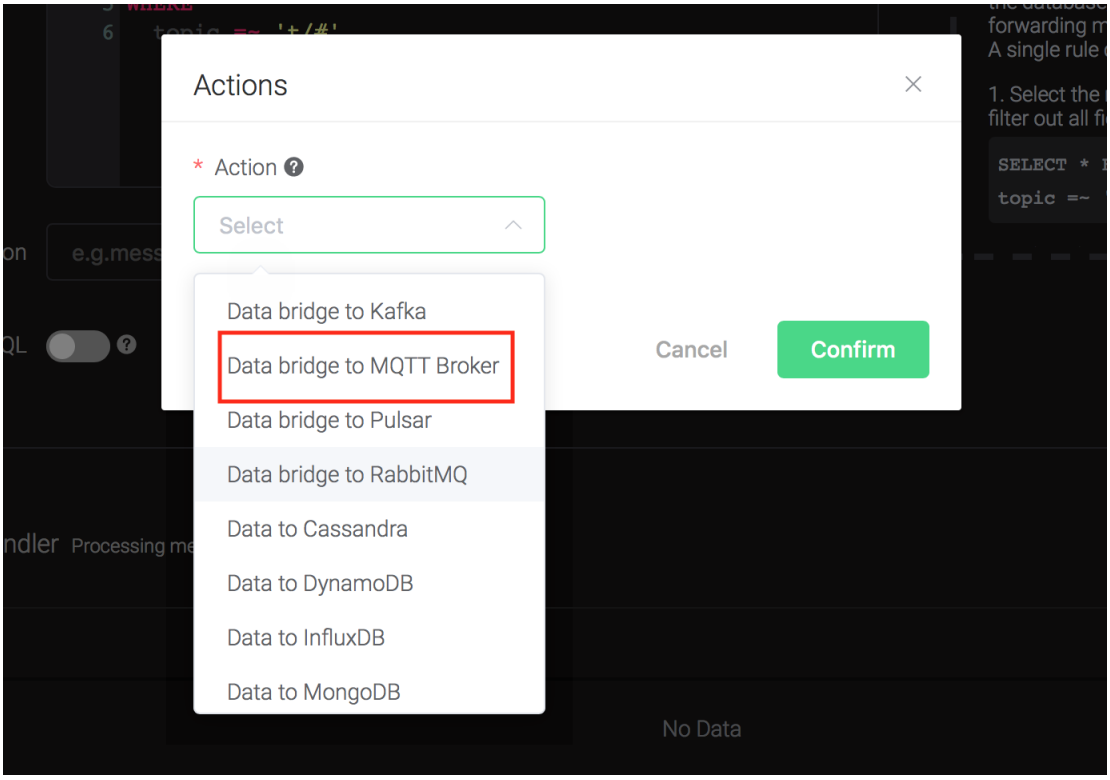
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```

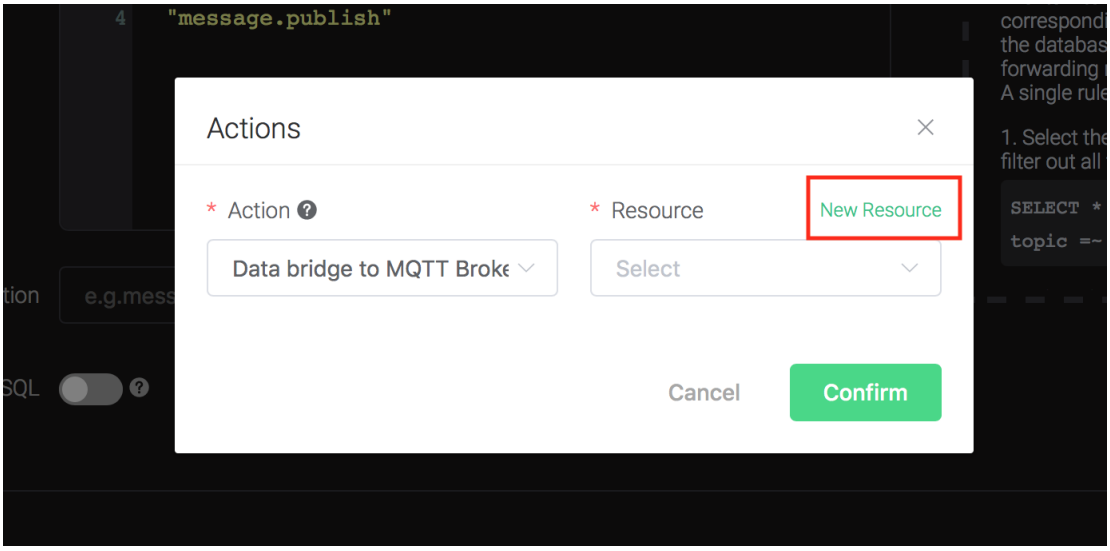


2. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data bridge to MQTT Broker" in the pop-up dialog window.

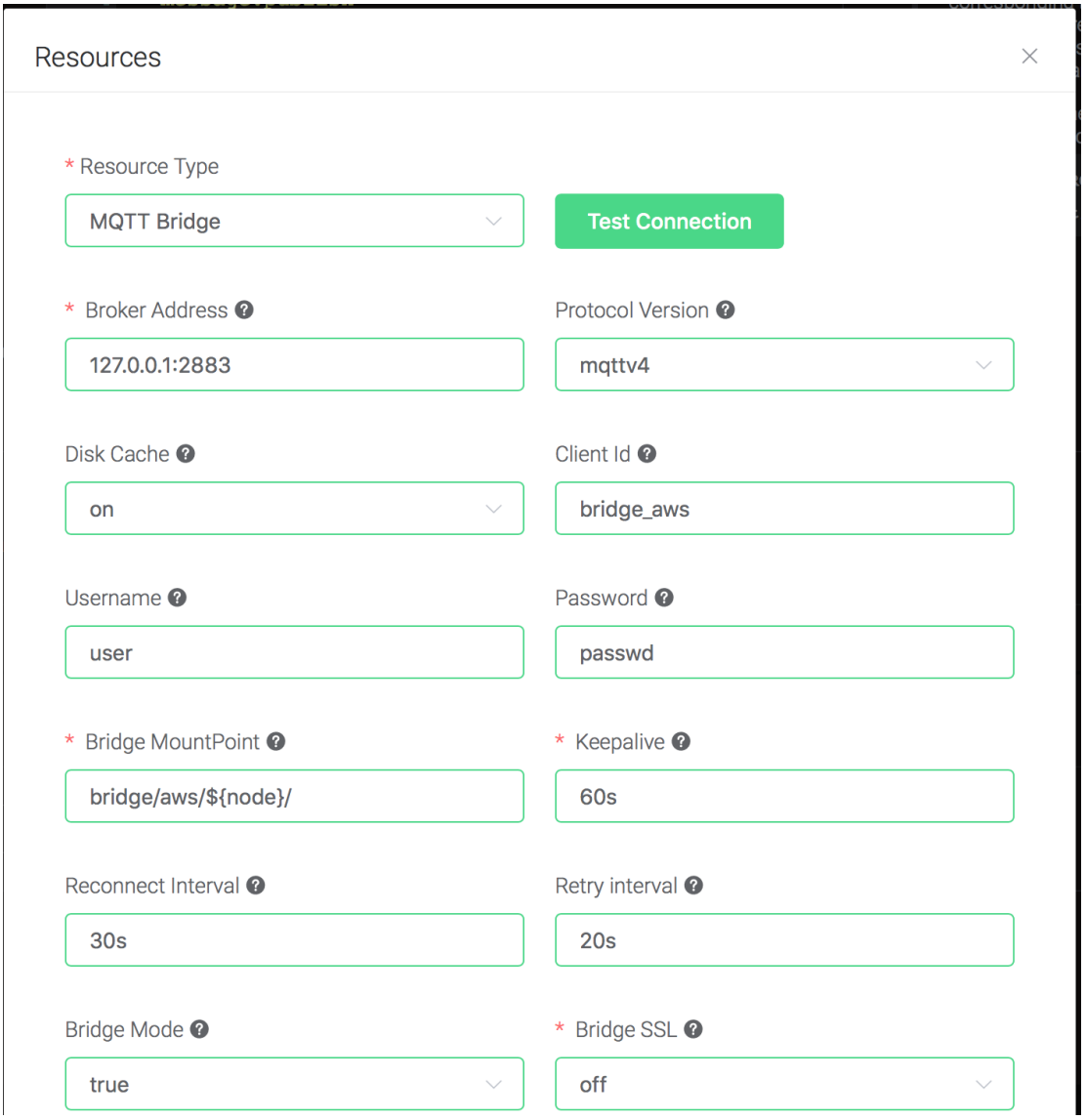


3. Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "MQTT Bridge":

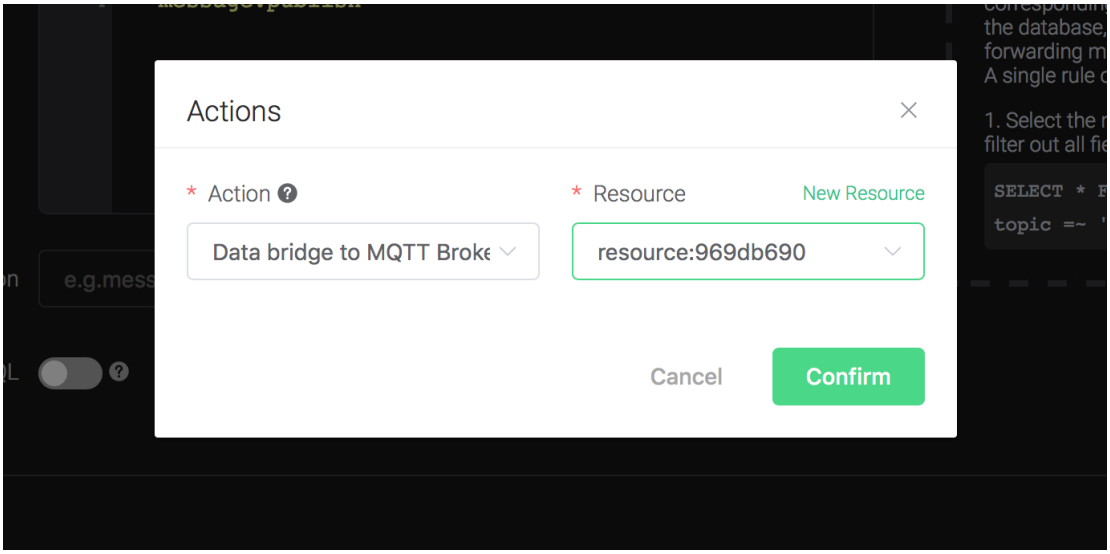


4. Configure the resource:

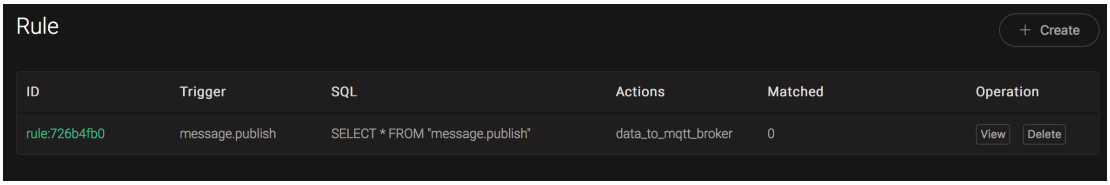
   Set "Broker Address" to the address of mosquitto, here is 127.0.0.1:2883, and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

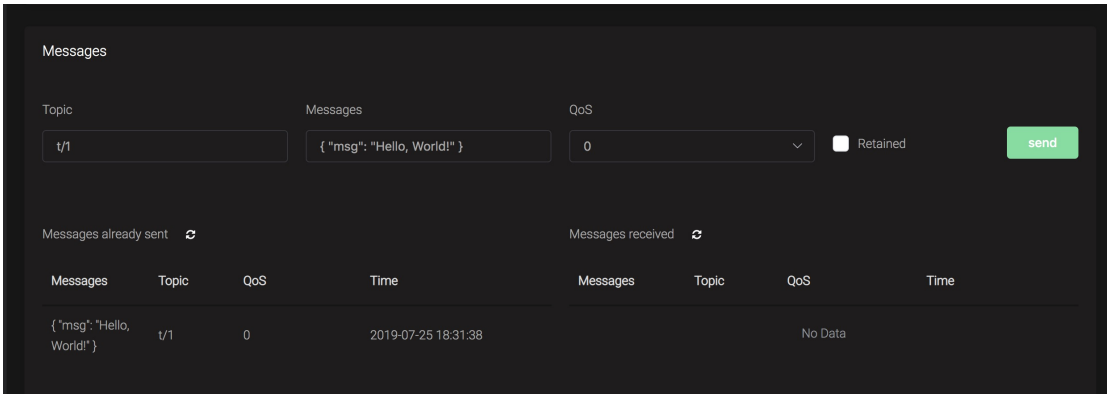
   Topic: "t/1"

   QoS: 0

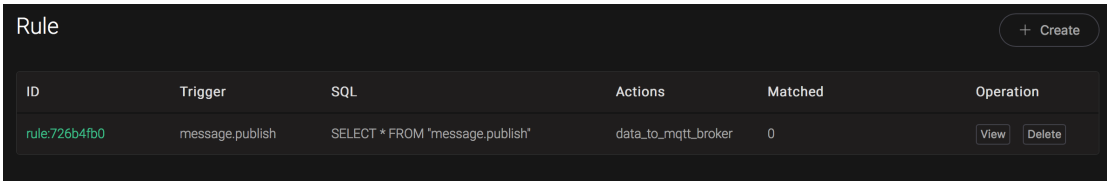
   Retained: false

   Payload: "Hello, World!"

   Then verify a message has been published to mosquitto:



   And from the rule list, verify that the "Matched" column has increased to 1:



8.15 Create EMQX Bridge Rules

0. Setup anther MQTT broker using emqx, taking Mac OSX for instance:

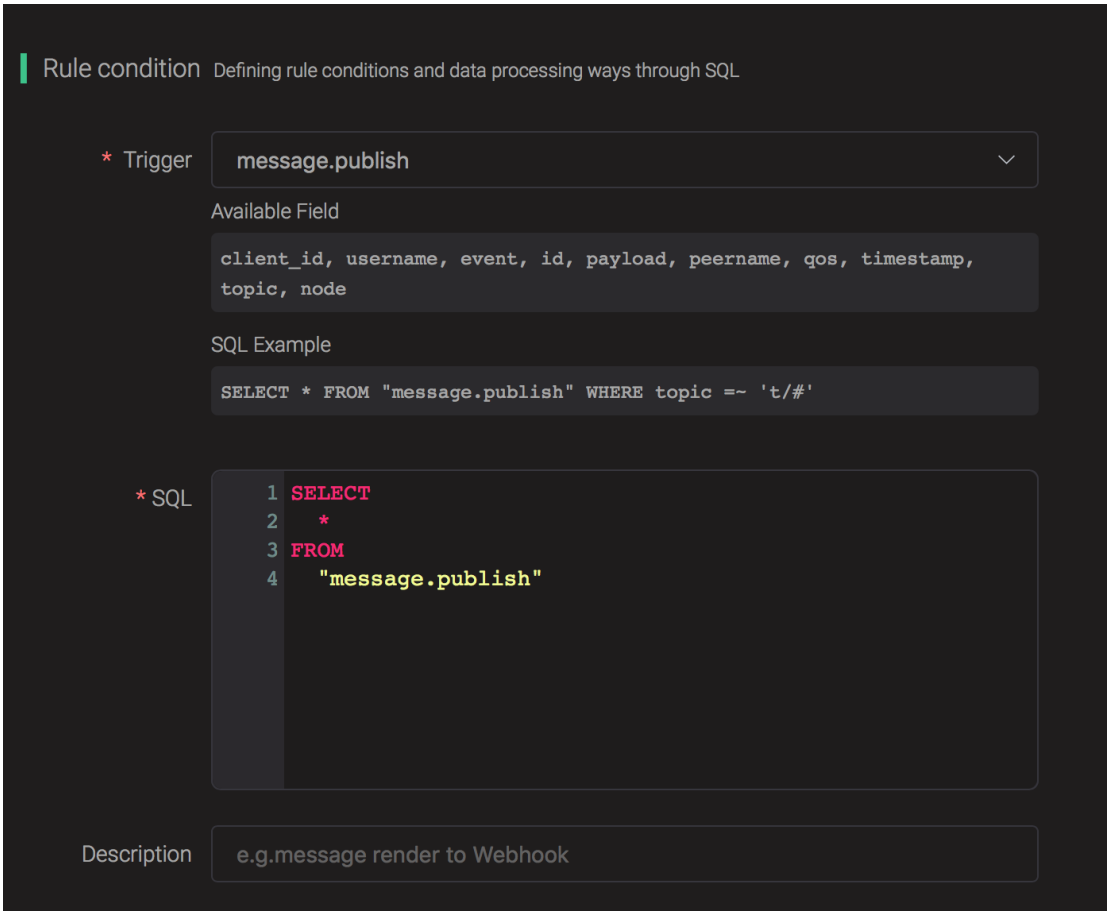
```
$ brew tap emqx/emqx/emqx

$ brew install emqx

# start emqx
$ emqx console
```

1. Create a rule:

   Go to emqx dashboard, select the "rule" tab on the menu to the left.

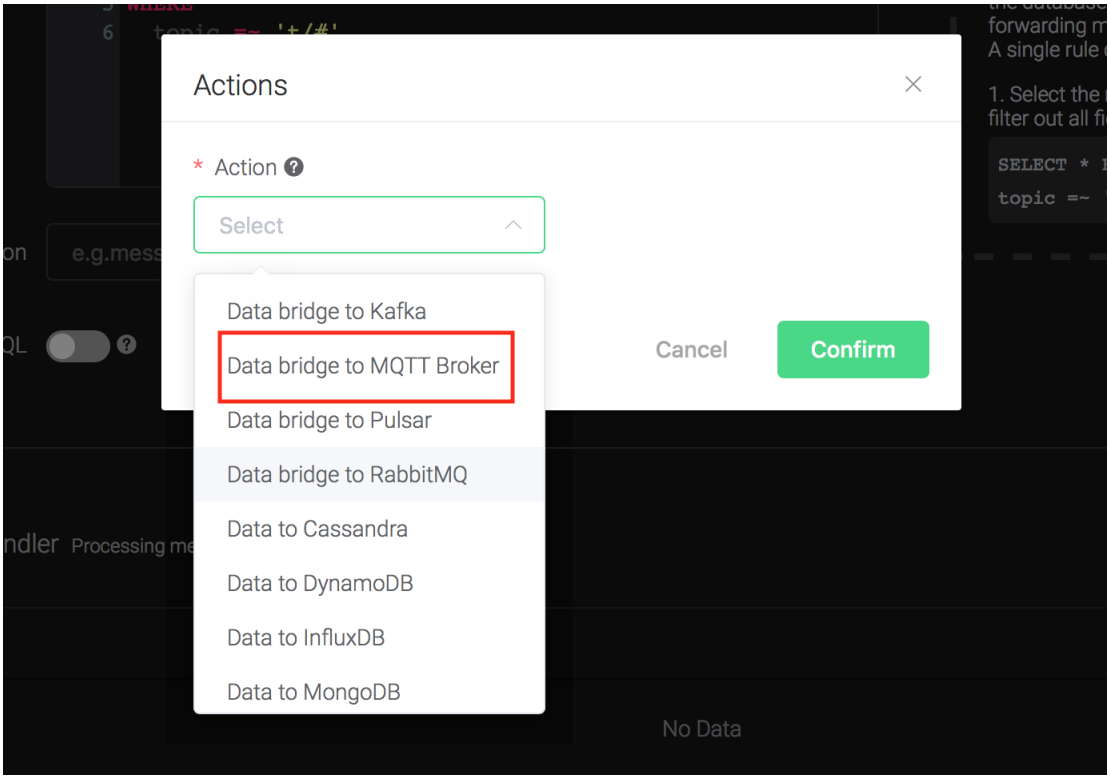
   Select "message.publish", then type in the following SQL:

```
SELECT
  *
FROM
  "message.publish"
```

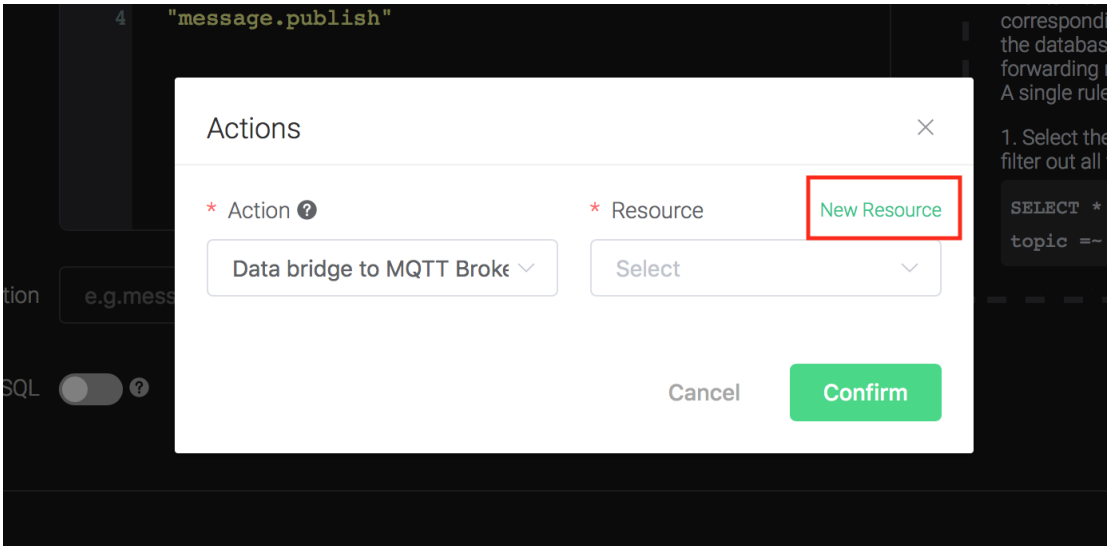


2. Bind an action:

   Click on the "+ Add" button under "Action Handler", and then select "Data bridge to MQTT Broker" in the pop-up dialog window.

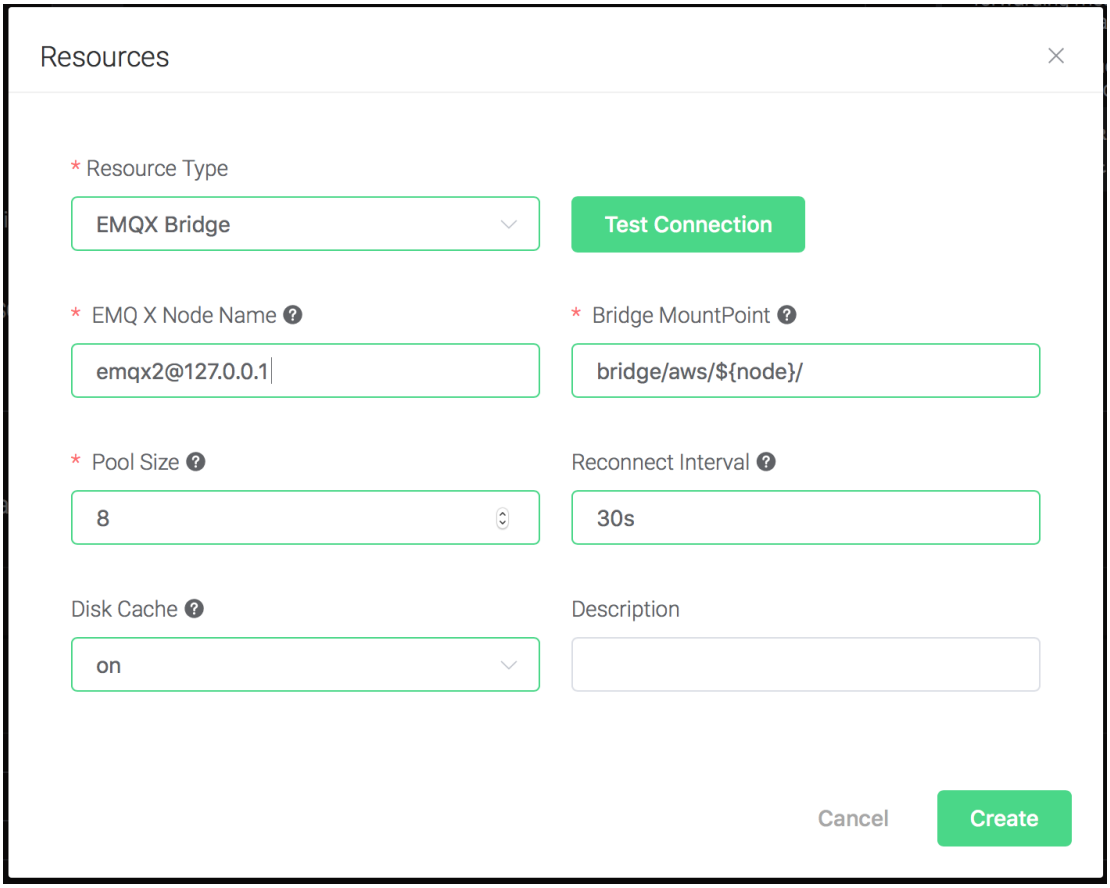


3. Bind a resource to the action. Since the dropdown list "Resource" is empty for now, we create a new resource by clicking on the "New Resource" to the top right, and then select "MQTT Bridge":

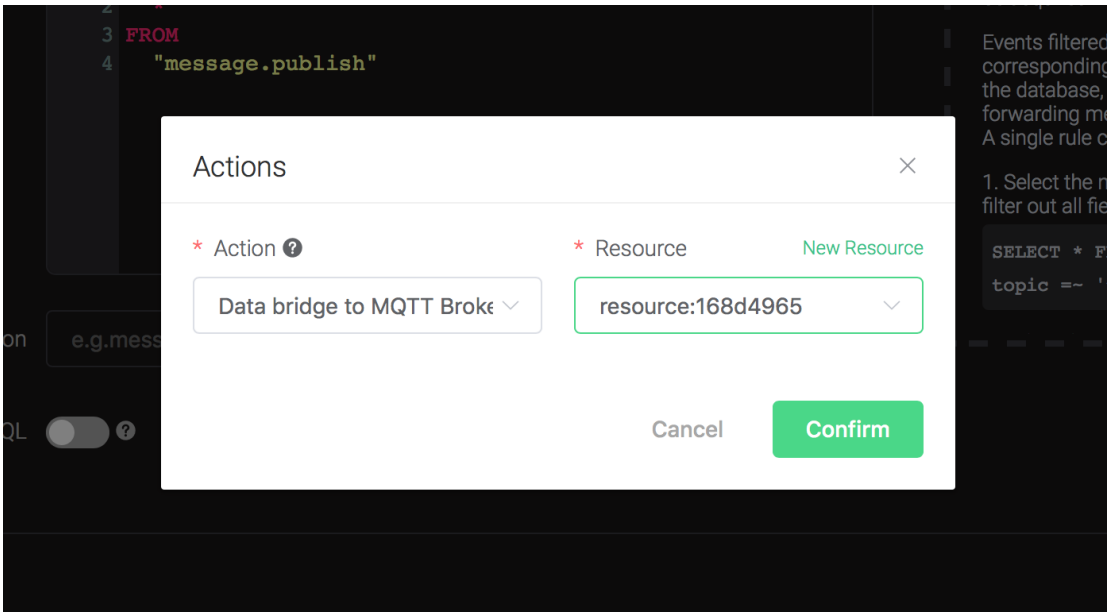


4. Configure the resource:

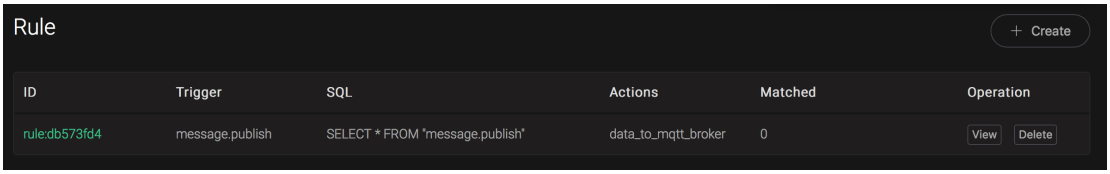
   Set "EMQ X Node Name" to the node name of the remote name, and keep all other configs as default, and click on the "Testing Connection" button to make sure the connection can be created successfully, and then click on the "Create" button.



6. Back to the "Actions" dialog, and then click on the "Confirm" button.



7. Back to the creating rule page, then click on "Create" button. The rule we created will be show in the rule list:



8. We have finished, testing the rule by sending an MQTT message to emqx:

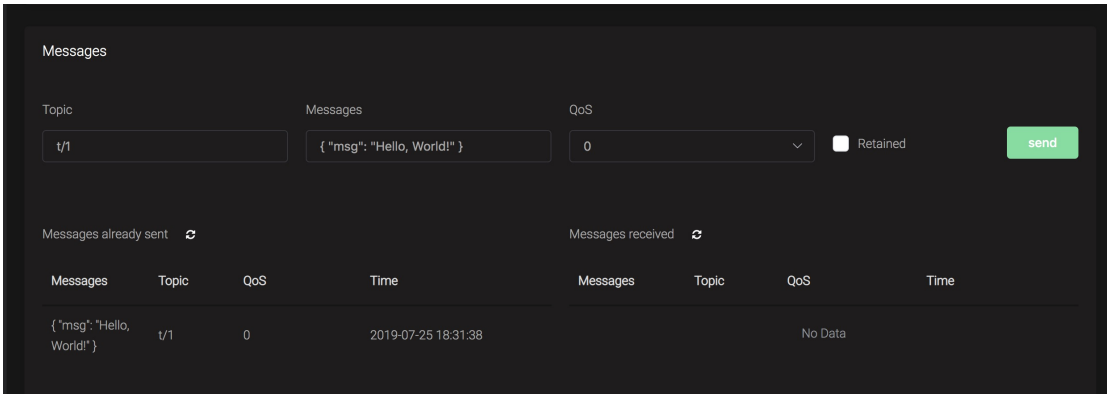
   Topic: "t/1"

   QoS: 0

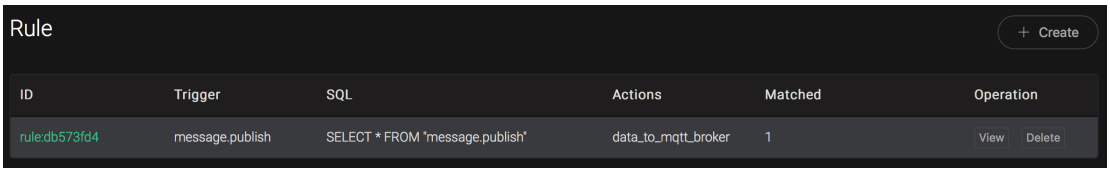
   Retained: false

   Payload: "Hello, World!"

   Then verify a message has been published to the other emqx:



   And from the rule list, verify that the "Matched" column has increased to 1:



8.16 Create Simple Rules using CLI

8.16.1 Create Inspect Rules

Create a rule for testing: print the content of the message and all the args of the action, when a MQTT message is sent to topic 't/a'.

- The filter SQL is: SELECT * FROM "message.publish" WHERE topic = 't/a';
- The action is: "print the content of the message and all the args of the action", the action we need is 'inspect'.

```
$ ./bin/emqx_ctl rules create \
  "SELECT * FROM \"message.publish\" WHERE topic = 't/a'" \
  '[{"name":"inspect", "params": {"a": 1}}]' \
  -d 'Rule for debug'

Rule rule:803de6db created
```

The CLI above created a rule with ID='Rule rule:803de6db'.

The first two args are mandatory:

- SQL: SELECT * FROM "message.publish" WHERE topic = 't/a'
- Action List: [{"name":"inspect", "params": {"a": 1}}]. Action List is of tye JSON Array. "name" is the name of the action, "params" is the parameters of the action. Note that the action `inspect` does not need a resource.

The last arg is an optional description of the rule: 'Rule for debug'.

If a MQTT message "hello" is sent to topic 't/a', the rule "Rule rule:803de6db" will be matched, and then action "inspect" will be triggered, the following info will be printed to the emqx console:

```
$ tail -f log/erlang.log.1

(emqx@127.0.0.1)1> [inspect]
    Selected Data: #{client_id => <<"shawn">>,event => 'message.publish',
                     flags => #{dup => false,retain => false},
                     id => <<"5898704A55D6AF4430000083D0002">>,
                     payload => <<"hello">>,
                     peername => <<"127.0.0.1:61770">>,qos => 1,
                     timestamp => 1558587875090,topic => <<"t/a">>,
                     username => undefined}
    Envs: #{event => 'message.publish',
            flags => #{dup => false,retain => false},
            from => <<"shawn">>,
            headers =>
                #{allow_publish => true,
                  peername => {{127,0,0,1},61770},
                  username => undefined},
            id => <<0,5,137,135,4,165,93,106,244,67,0,0,8,61,0,2>>,
            payload => <<"hello">>,qos => 1,
            timestamp => {1558,587875,89754},
            topic => <<"t/a">>}
    Action Init Params: #{<<"a">> => 1}
```

- `Selected Data` listed the fields that selected by the SQL.All available fields will be listed here, as we used `select *`.
- `Envs` is the environment varibles that can be used internally in the action.
- `Action Init Params` is the params we passed to the action.

8.16.2 Create WebHook Rule

Create a rule: Forward all the messages that send from client_id='Steven', to the Web Server at 'http://127.0.0.1:9910':

- The filter SQL: SELECT username as u, payload FROM "message.publish" where u='Steven';
- Actions: "Forward to 'http://127.0.0.1:9910'";
- Resource Type: web_hook;
- Resource: "The WebHook resource at 'http://127.0.0.1:9910'".

0. Create a simpile Web service using linux tool `nc`:

```
$ while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l 127.0.0.1␣
↪9910; done;
```

1. Create a resource of resource type "WebHook", and configure the url:

   1). List all available resource types, make sure 'web_hook' exists:

```
$ ./bin/emqx_ctl resource-types list

resource_type(name='web_hook', provider='emqx_web_hook', params=#{...}},␣
↪on_create={emqx_web_hook_actions,on_resource_create}, description=
↪'WebHook Resource')
...
```

   2). Create a new resource using resource type 'web_hook', configure "url"="http://127.0.0.1:9910":

```
$ ./bin/emqx_ctl resources create \
  'web_hook' \
  -c '{"url": "http://127.0.0.1:9910", "headers": {"token":
↪"axfw34y235wrq234t4ersgw4t"}, "method": "POST"}'

Resource resource:691c29ba created
```

   Above CLI created a resource with ID='resource:691c29ba', the first arg is mandatory - The resource type (web_hook). HTTP method is POST, and an HTTP Header is set: "token".

2. Create a rule, and bind action 'data_to_webserver' to it:

   1). List all available actions, make sure 'data_to_webserver' exists:

```
$ ./bin/emqx_ctl rule-actions list

action(name='data_to_webserver', app='emqx_web_hook', for='$any',␣
↪types=[web_hook], params=#{'$resource' => ...}, title ='Data to Web␣
↪Server', description='Forward Messages to Web Server')
...
```

   2). Create the rule, bind the action data_to_webserver, and bind resource resource:691c29ba to the action via the arg "$resource":

```
$ ./bin/emqx_ctl rules create \
 "SELECT username as u, payload FROM \"message.publish\" where u='Steven'
↪" \
 '[{"name":"data_to_webserver", "params": {"$resource":
↪"resource:691c29ba"}}]' \
 -d "Forward publish msgs from steven to webserver"

rule:26d84768
```

Above CLI is simlar to the first Inspect rule, with exception that the resource 'resource:691c29ba' is bound to 'data_to_webserver'. The binding is done by a special arg named '$resource'. What the action 'data_to_webserver' does is sending messages to the specified web server.

3. Now let's send a message "hello" to an arbitrary topic using username "Steven", this will trigger the rule we created above, and the Web Server will receive an message and return 200 OK:

```
$ while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l 127.0.0.1␣
↪9910; done;

POST / HTTP/1.1
content-type: application/json
content-length: 32
te:
host: 127.0.0.1:9910
connection: keep-alive
token: axfw34y235wrq234t4ersgw4t

{"payload":"hello","u":"Steven"}
```

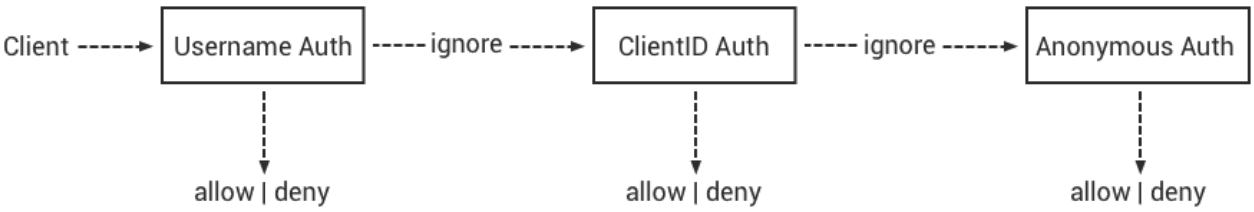
CHAPTER 9

AuthN/AuthZ

9.1 Design of MQTT Auth

EMQ X utilizes plugins to provide various Authentication mechanism. EMQ X supports username / password, ClientID and anonymous authentication. It also supports Auth integration with MySQL, PostgreSQL, Redis, MongoDB, HTTP, OpenLDAP and JWT.

Anonymous Auth is enabled by default. An Auth chain can be built of multiple Auth plugins:



9.2 Anonymous Auth

Anonymous Auth is the last validation of Auth chain. By default, it is enabled in file *etc/emqx.conf*. Recommended to disable it in production deployment:

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

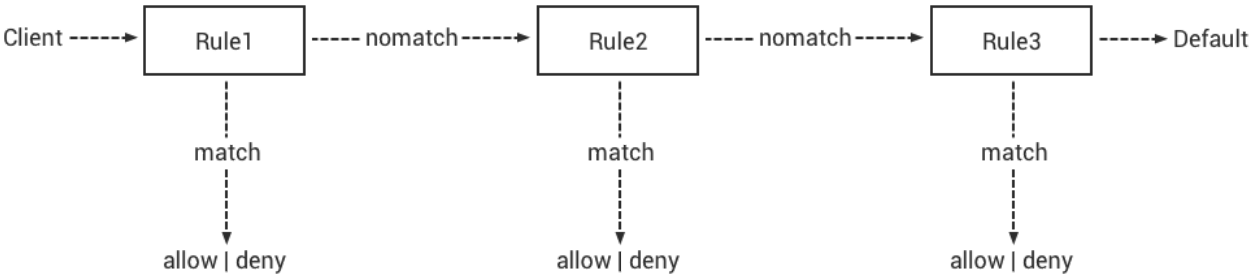
9.3 Access Control List (ACL)

EMQ X Server utilizes Access Control List (ACL) to realize the access control upon clients.

ACL defines:

```
Allow|Deny Whom Subscribe|Publish Topics
```

When MQTT clients subscribe to topics or publish messages, the EMQ X access control module tries to check against all rules in the list till a first match. Otherwise it fallbacks to default routine if no match found:



9.4 Default Access Control File

The default access control of EMQ X server is configured by the file acl.conf:

```
## ACL nomatch
acl_nomatch = allow

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

## Whether to enable ACL cache.
enable_acl_cache = on

## The maximum count of ACL entries can be cached for a client.
acl_cache_max_size = 32

## The time after which an ACL cache entry will be deleted
acl_cache_ttl = 1m

## The action when acl check reject current operation
acl_deny_action = ignore
```

ACL is defined in *etc/acl.conf*, and is loaded when EMQ X starts:

```
%% 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
{allow, all}.
```

If ACL is modified, it can be reloaded using CLI:

```
$ ./bin/emqx_ctl acl reload

reload acl_internal successfully
```

9.5 List of AuthN/ACL Plugins

EMQ X supports integrated authentications by using ClientId, Username, HTTP, OpenLDAP, MySQL, Redis, PostgreSQL, MongoDB and JWT. Multiple Auth plugins can be loaded simultaneously to build an authentication chain.

The config files of Auth plugins are located in ‘/etc/emqx/plugins/‘(RPM/DEB installation) or in ‘etc/plugins/‘(standalone installation)

Auth Plugin	Config file	Description
emqx_auth_clientid	emqx_auth_clientid.conf	ClientId AuthN Plugin
emqx_auth_username	emqx_auth_username.conf	Username/Password AuthN Plugin
emqx_auth_ldap	emqx_auth_ldap.conf	OpenLDAP AuthN/AuthZ Plugin
emqx_auth_http	emqx_auth_http.conf	HTTP AuthN/AuthZ
emqx_auth_mysql	emqx_auth_redis.conf	MySQL AuthN/AuthZ
emqx_auth_pgsql	emqx_auth_mysql.conf	PostgreSQL AuthN/AuthZ
emqx_auth_redis	emqx_auth_pgsql.conf	Redis AuthN/AuthZ
emqx_auth_mongo	emqx_auth_mongo.conf	MongoDB AuthN/AuthZ
emqx_auth_jwt	emqx_auth_jwt.conf	JWT AuthN/AuthZ

9.6 ClientID Auth Plugin

Configure the password hash in the *emqx_auth_clientid.conf*:

```
## Default usernames 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
##auth.client.4.clientid = client~!@#$%^&*()_+
##auth.client.4.password = passwd~!@#$%^&*()_+
## Password hash: plain | md5 | sha | sha256
auth.client.password_hash = sha256
```

Load ClientId Auth plugin:

```
./bin/emqx_ctl plugins load emqx_auth_clientid
```

After the plugin is loaded, there are two possible ways to add users:

1. Use the './bin/emqx_ctl' CLI tool to add clients:

```
$ ./bin/emqx_ctl clientid add <ClientId> <Password>
```

2. Use the HTTP API to add clients:

```
POST api/v3/auth_clientid
{
    "clientid": "clientid",
    "password": "password"
}
```

9.7 Username/Passwordd Auth Plugin

Configure the password hash in the *emqx_auth_username.conf*:

```
## Default usernames Examples:
##auth.user.1.username = admin
##auth.user.1.password = public
##auth.user.2.username = feng@emqtt.io
##auth.user.2.password = public
##auth.user.3.username = name~!@#$%^&*()_+
##auth.user.3.password = pwsswd~!@#$%^&*()_+
## Password hash: plain | md5 | sha | sha256
auth.user.password_hash = sha256
```

Load Username Auth plugin:

```
./bin/emqx_ctl plugins load emqx_auth_username
```

After the plugin is loaded, there are two possible ways to add users:

1. Use the './bin/emqx_ctl' CLI tool to add users:

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

2. Use the HTTP API to add users:

```
POST api/v3/auth_username
{
    "username": "username",
    "password": "password"
}
```

9.8 OpenLDAP Auth Plugin

Configure the OpenLDAP Auth Plugin in the *emqx_auth_ldap.conf*:

```
## OpenLDAP servers list
auth.ldap.servers = 127.0.0.1

## OpenLDAP server port
auth.ldap.port = 389

## OpenLDAP pool size
auth.ldap.pool = 8

## OpenLDAP Bind DN
auth.ldap.bind_dn = cn=root,dc=emqx,dc=io

## OpenLDAP Bind Password
auth.ldap.bind_password = public

## OpenLDAP query timeout
auth.ldap.timeout = 30s

## OpenLDAP Device DN
auth.ldap.device_dn = ou=device,dc=emqx,dc=io

## Specified ObjectClass
auth.ldap.match_objectclass = mqttUser

## attributetype for username
auth.ldap.username.attributetype = uid

## attributetype for password
auth.ldap.password.attributetype = userPassword

## Whether to enable SSL
auth.ldap.ssl = false
```

Load the OpenLDAP Auth plugin:

```
./bin/emqx_ctl plugins load emqx_auth_ldap
```

9.9 HTTP Auth/ACL Plugin

Configure the HTTP Auth/ACL Plugin in the *emqx_auth_http.conf*:

```
## Time-out time for the http request, 0 is never timeout
## auth.http.request.timeout = 0

## Connection time-out time, used during the initial request
## when the client is connecting to the server
```

()

()

```
## auth.http.request.connect_timout = 0

## Re-send http reuqest times
auth.http.request.retry_times = 3

## The interval for re-sending the http request
auth.http.request.retry_interval = 1s

## The 'Exponential Backoff' mechanism for re-sending request. The actually
## re-send time interval is `interval * backoff ^ times`
auth.http.request.retry_backoff = 2.0
```

Setup the Auth URL and parameters:

```
## 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
```

Setup the Super User URL and parameters:

```
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
```

Setup the ACL URL and parameters:

```
## '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
```

Design of HTTP Auth and ACL server API:

```
If Auth/ACL sucesses, API returns 200

If Auth/ACL fails, API return 4xx
```

Load HTTP Auth/ACL plugin:

```
./bin/emqx_ctl plugins load emqx_auth_http
```

9.10 MySQL Auth/ACL Plugin

Create MQTT users' ACL database, and configure the ACL and Auth queries in the *emqx_auth_mysql.conf*:

9.10.1 MQTT Auth User List

```
CREATE TABLE `mqtt_user` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(100) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `salt` varchar(40) 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;
```

: User can define the user list table and configure it in the `auth_query` statement.

9.10.2 MQTT Access Control List

```
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 #'),
    (4,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (5,1,'127.0.0.1',NULL,NULL,2,'#'),
    (6,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

9.10.3 MySQL Server Address

```
## 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

## MySQL query timeout
## auth.mysql.query_timeout = 5s
```

9.10.4 Configure MySQL Auth Query Statement

```
## Authentication query
##
## Variables:
##  - %u: username
##  - %c: clientid
##  - %C: common name of client TLS cert
##  - %d: subject of client TLS cert
##
auth.mysql.auth_query = select password from mqtt_user where username = '%u' limit 1
## auth.mysql.auth_query = select password_hash as password from mqtt_user where username = '%u' limit 1

## Password hash: plain | md5 | sha | sha256 | bcrypt
auth.mysql.password_hash = sha256

## sha256 with salt prefix
## auth.mysql.password_hash = salt,sha256

## bcrypt with salt only prefix
## auth.mysql.password_hash = salt,bcrypt

## sha256 with salt suffix
## auth.mysql.password_hash = sha256,salt

## pbkdf2 with macfun iterations dklen
## macfun: md4, md5, ripemd160, sha, sha224, sha256, sha384, sha512
## auth.mysql.password_hash = pbkdf2,sha256,1000,20

## Superuser query
##
## Variables:
##  - %u: username
##  - %c: clientid
##  - %C: common name of client TLS cert
##  - %d: subject of client TLS cert
```

()

()

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

9.10.5 Configure MySQL ACL Query Statement

```
## ACL query
##
## Variables:
##  - %a: ipaddr
##  - %u: username
##  - %c: clientid
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'
```

9.10.6 Load MySQL Auth Plugin

```
./bin/emqx_ctl plugins load emqx_auth_mysql
```

9.11 PostgreSQL Auth/ACL Plugin

Create MQTT users' ACL tables, and configure Auth, ACL queries in the *emqx_auth_pgsql.conf*:

9.11.1 PostgreSQL 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)
);
```

: User can define the user list table and configure it in the *auth_query* statement.

9.11.2 PostgreSQL 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 #'),
    (4,1,'127.0.0.1',NULL,NULL,2,'$SYS/#'),
    (5,1,'127.0.0.1',NULL,NULL,2,'#'),
    (6,1,NULL,'dashboard',NULL,1,'$SYS/#');
```

9.11.3 PostgreSQL Server Address

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

## PostgreSQL pool size
auth.pgsql.pool = 8

## PostgreSQL username
auth.pgsql.username = root

## PostgreSQL password
#auth.pgsql.password =

## PostgreSQL database
auth.pgsql.database = mqtt

## PostgreSQL database encoding
auth.pgsql.encoding = utf8

## Whether to enable SSL connection
auth.pgsql.ssl = false

## SSL keyfile
## auth.pgsql.ssl_opts.keyfile =

## SSL certfile
## auth.pgsql.ssl_opts.certfile =

## SSL cacertfile
## auth.pgsql.ssl_opts.cacertfile =
```

9.11.4 Configure PostgreSQL Auth Query Statement

```
## Authentication query
##
## Variables:
##  - %u: username
##  - %c: clientid
##  - %C: common name of client TLS cert
##  - %d: subject of client TLS cert
auth.pgsql.auth_query = select password from mqtt_user where username = '%u'␣
↪limit 1

## Password hash: plain | md5 | sha | sha256 | bcrypt
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

## bcrypt with salt prefix
## auth.pgsql.password_hash = salt,bcrypt

## pbkdf2 with macfun iterations dklen
## macfun: md4, md5, ripemd160, sha, sha224, sha256, sha384, sha512
## auth.pgsql.password_hash = pbkdf2,sha256,1000,20

## Superuser query
##
## Variables:
##  - %u: username
##  - %c: clientid
##  - %C: common name of client TLS cert
##  - %d: subject of client TLS cert
auth.pgsql.super_query = select is_superuser from mqtt_user where username = '
↪%u' limit 1
```

9.11.5 Configure PostgreSQL ACL Query Statement

```
## ACL query. Comment this query, the ACL will be disabled
##
## Variables:
##  - %a: ipaddress
##  - %u: username
##  - %c: clientid
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'
```

9.11.6 Load PostgreSQL Auth Plugin

```
./bin/emqx_ctl plugins load emqx_auth_pgsql
```

9.12 Redis/ACL Auth Plugin

Config file: *emqx_auth_redis.conf*:

9.12.1 Redis Server Address

```
## Redis Server cluster type: single | sentinel | cluster
auth.redis.type = single

## Redis server address
auth.redis.server = 127.0.0.1:6379

## Redis sentinel cluster name.
## auth.redis.sentinel = mymaster

## Redis pool size.
auth.redis.pool = 8

## Redis database no.
auth.redis.database = 0

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

## Redis query timeout
## auth.redis.query_timeout = 5s
```

9.12.2 Configure Auth Query Command

```
## Authentication query command
##
## Variables:
##  - %u: username
##  - %c: clientid
##  - %C: common name of client TLS cert
##  - %d: subject of client TLS cert
auth.redis.auth_cmd = HMGET mqtt_user:%u password

## Password hash: plain | md5 | sha | sha256 | bcrypt
auth.redis.password_hash = plain

## sha256 with salt prefix
```

()

()

```
## auth.redis.password_hash = salt,sha256

## sha256 with salt suffix
## auth.redis.password_hash = sha256,salt

## bcrypt with salt prefix
## auth.redis.password_hash = salt,bcrypt

## pbkdf2 with macfun iterations dklen
## macfun: md4, md5, ripemd160, sha, sha224, sha256, sha384, sha512
## auth.redis.password_hash = pbkdf2,sha256,1000,20

## Superuser query command
##
## Variables:
##  - %u: username
##  - %c: clientid
##  - %C: common name of client TLS cert
##  - %d: subject of client TLS cert
auth.redis.super_cmd = HGET mqtt_user:%u is_superuser
```

9.12.3 Configure ACL Query Command

```
## ACL Query Command
##
## Variables:
##  - %u: username
##  - %c: clientid
auth.redis.acl_cmd = HGETALL mqtt_acl:%u
```

9.12.4 Redis Authed Users Hash

By default, Hash is used to store Authed users:

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

9.12.5 Redis ACL Rules Hash

By default, Hash is used to store ACL rules:

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

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

9.12.6 Load Redis Auth Plugin

```
./bin/emqx_ctl plugins load emqx_auth_redis
```

9.13 MongoDB Auth/ACL Plugin

Configure MongoDB, MQTT users and ACL Collection in the *emqx_auth_mongo.conf*:

9.13.1 MongoDB Server

```
## MongoDB Topology Type: single | unknown | sharded | rs
auth.mongo.type = single

## The set name if type is rs
## auth.mongo.rs_set_name =

## MongoDB server list.
auth.mongo.server = 127.0.0.1:27017

## MongoDB pool size
auth.mongo.pool = 8

## MongoDB login user
## auth.mongo.login =

## MongoDB password
## auth.mongo.password =

## MongoDB AuthSource
## auth.mongo.auth_source = admin

## MongoDB database
auth.mongo.database = mqtt

## MongoDB query timeout
## auth.mongo.query_timeout = 5s

## Whether to enable SSL connection
## auth.mongo.ssl = false

## SSL keyfile
## auth.mongo.ssl_opts.keyfile =
```

()

()

```
## SSL certfile
## auth.mongo.ssl_opts.certfile =

## SSL cacertfile
## auth.mongo.ssl_opts.cacertfile =

## MongoDB write mode
## auth.mongo.w_mode =

## Mongo read mode
## auth.mongo.r_mode =

## MongoDB topology options
auth.mongo.topology.pool_size = 1
auth.mongo.topology.max_overflow = 0
## auth.mongo.topology.overflow_ttl = 1000
## auth.mongo.topology.overflow_check_period = 1000
## auth.mongo.topology.local_threshold_ms = 1000
## auth.mongo.topology.connect_timeout_ms = 20000
## auth.mongo.topology.socket_timeout_ms = 100
## auth.mongo.topology.server_selection_timeout_ms = 30000
## auth.mongo.topology.wait_queue_timeout_ms = 1000
## auth.mongo.topology.heartbeat_frequency_ms = 10000
## auth.mongo.topology.min_heartbeat_frequency_ms = 1000
```

9.13.2 Configure Auth Query Collection

```
## Authentication query
auth.mongo.auth_query.collection = mqtt_user

## password with salt prefix
## auth.mongo.auth_query.password_hash = salt,sha256
auth.mongo.auth_query.password_field = password

## Password hash: plain | md5 | sha | sha256 | bcrypt
auth.mongo.auth_query.password_hash = sha256

## sha256 with salt suffix
## auth.mongo.auth_query.password_hash = sha256,salt

## sha256 with salt prefix
## auth.mongo.auth_query.password_hash = salt,sha256

## bcrypt with salt prefix
## auth.mongo.auth_query.password_hash = salt,bcrypt

## pbkdf2 with macfun iterations dklen
## macfun: md4, md5, ripemd160, sha, sha224, sha256, sha384, sha512
## auth.mongo.auth_query.password_hash = pbkdf2,sha256,1000,20
```

()

()

```
## Authentication Selector
##
## Variables:
##  - %u: username
##  - %c: clientid
##  - %C: common name of client TLS cert
##  - %d: subject of client TLS cert
auth.mongo.auth_query.selector = username=%u

## Enable superuser query
auth.mongo.super_query = on

auth.mongo.super_query.collection = mqtt_user

auth.mongo.super_query.super_field = is_superuser

#auth.mongo.super_query.selector = username=%u, clientid=%c
auth.mongo.super_query.selector = username=%u
```

9.13.3 Configure ACL Query Collection

```
## Enable ACL query
auth.mongo.acl_query = on

auth.mongo.acl_query.collection = mqtt_acl

## ACL Selector
##
## Variables:
##  - %u: username
##  - %c: clientid
## auth.mongo.acl_query.selector.3 = clientid=$all

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

9.13.4 MongoDB Database

```
use mqtt
db.createCollection("mqtt_user")
db.createCollection("mqtt_acl")
db.mqtt_user.ensureIndex({"username":1})
```

: The DB name and Collection name are free of choice

9.13.5 Example of a MongoDB User Collection

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

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

9.13.6 Example of a MongoDB ACL Collection

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

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

9.13.7 Load Mognodb Auth Plugin

```
./bin/emqx_ctl plugins load emqx_auth_mongo
```

9.14 JWT Auth Plugin

9.14.1 Configure JWT Auth

```
## HMAC Hash Secret
auth.jwt.secret = emqxsecret

## From where the JWT string can be got
auth.jwt.from = password

## RSA or ECDSA public key file
## auth.jwt.pubkey = etc/certs/jwt_public_key.pem

## Enable to verify claims fields
```

()

()

```
auth.jwt.verify_claims = off

## The checklist of claims to validate
##
## Variables:
##  - %u: username
##  - %c: clientid
# auth.jwt.verify_claims.username = %u
```

9.14.2 Load JWT Auth Plugin

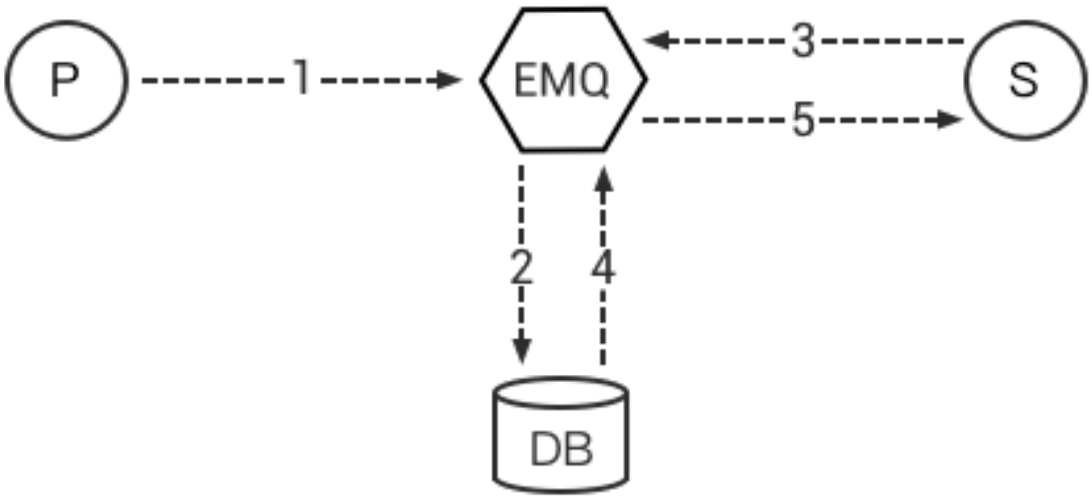
```
./bin/emqx_ctl plugins load emqx_auth_jwt
```

CHAPTER 10

Backends

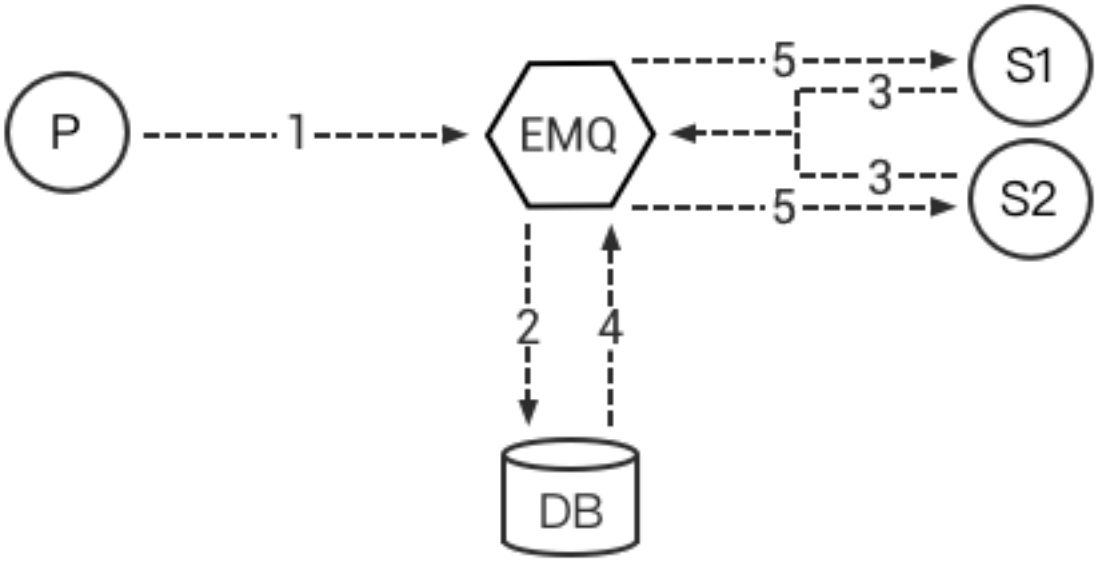
10.1 MQTT Message Persistence

10.1.1 One-to-one message Persistence



1. PUB publishes a message;
2. Backend records this message in DB;
3. SUB subscribes to a topic;
4. Backend retrieves the messages of this topic from DB;
5. Messages are sent to SUB;
6. Once the SUB acknowledged / received the message, backend removes the message from DB.

10.1.2 Many-to-many message Persistence



1. PUB publishes a message;
2. Backend records the message in DB;
3. SUB1 and SUB2 subscribe to a topic;
4. Backend retrieves the messages of this topic;
5. Messages are sent to SUB1 and SUB2;
6. Backend records the read position of SUB1 and SUB2, the next message's retrieval starts from this position.

10.1.3 Client Connection State

EMQ X supports retaining the client's connection state in Redis or DB.

10.1.4 Client Subscription by Broker

EMQ X Persistence supports subscription by broker. When a client goes online, the persistence module loads the subscriptions of the client from Redis or Databases.

10.1.5 List of Persistence Plugins

EMQ X allowes storing messages in Redis, MySQL, PostgreSQL, MongoDB, Cassandra, DynamoDB, InfluxDB, OpenTSDB and Timescale:

Persistence Plugins	Config File	Description
emqx_backend_redis	emqx_backend_redis.conf	Redis Message Persistence
emqx_backend_mysql	emqx_backend_mysql.conf	MySQL Message Persistence
emqx_backend_pgsql	emqx_backend_pgsql.conf	PostgreSQL Message Persistence
emqx_backend_mongo	emqx_backend_mongo.conf	MongoDB Message Persistence
emqx_backend_cassa	emqx_backend_cassa.conf	Cassandra Message Persistence
emqx_backend_dynamo	emqx_backend_dynamo.conf	DynamoDB Message Persistence
emqx_backend_influxdb	emqx_backend_influxdb.conf	InfluxDB Message Persistence
emqx_backend_opentsdb	emqx_backend_opentsdb.conf	OpenTSDB Message Persistence
emqx_backend_timescale	emqx_backend_timescale.conf	Timescale Message Persistence

10.2 Redis Backend

Config file: emqx_backend_redis.conf

10.2.1 Configure the Redis Server

Config Connection Pool of Multiple Redis Servers:

```
## Redis Server
backend.redis.pool1.server = 127.0.0.1:6379

## Redis Sentinel
## backend.redis.server = 127.0.0.1:26378

##Redis Sentinel Cluster name
## backend.redis.sentinel = mymaster

## Redis Pool Size
backend.redis.pool1.pool_size = 8

## Redis database
backend.redis.pool1.database = 1

## Redis subscribe channel
backend.redis.pool1.channel = mqtt_channel
```

10.2.2 Configure Persistence Hooks

```
## Expired after seconds, if =< 0 take the default value
backend.redis.msg.expired_after = 3600

## Client Connected Record
backend.redis.hook.client.connected.1    = {"action": {"function": "on_client_
↪connected"}, "pool": "pool1"}
```

()

()

```
## Subscribe Lookup Record
backend.redis.hook.client.connected.2    = {"action": {"function": "on_
↪subscribe_lookup"}, "pool": "pool1"}

## Client DisConnected Record
backend.redis.hook.client.disconnected.1 = {"action": {"function": "on_client_
↪disconnected"}, "pool": "pool1"}

## Lookup Unread Message for one QOS > 0
backend.redis.hook.session.subscribed.1  = {"topic": "queue/#", "action": {
↪"function": "on_message_fetch_for_queue"}, "pool": "pool1"}

## Lookup Unread Message for many QOS > 0
backend.redis.hook.session.subscribed.2  = {"topic": "pubsub/#", "action": {
↪"function": "on_message_fetch_for_pubsub"}, "pool": "pool1"}

## Lookup Retain Message
backend.redis.hook.session.subscribed.3  = {"action": {"function": "on_retain_
↪lookup"}, "pool": "pool1"}

## Store Publish Message  QOS > 0
backend.redis.hook.message.publish.1     = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1"}

## Store Retain Message
backend.redis.hook.message.publish.2     = {"topic": "#", "action": {"function
↪": "on_message_retain"}, "pool": "pool1"}

## Delete Retain Message
backend.redis.hook.message.publish.3     = {"topic": "#", "action": {"function
↪": "on_retain_delete"}, "pool": "pool1"}

## Store Ack for one
backend.redis.hook.message.acked.1       = {"topic": "queue/#", "action": {
↪"function": "on_message_acked_for_queue"}, "pool": "pool1"}

## Store Ack for many
backend.redis.hook.message.acked.2       = {"topic": "pubsub/#", "action": {
↪"function": "on_message_acked_for_pubsub"}, "pool": "pool1"}
```

10.2.3 Description of Persistence Hooks

hook	topic	action/function	Description
client.connected		on_client_connected	Store client connected state
client.connected		on_subscribe_lookup	Subscribe to topics
client.disconnected		on_client_disconnected	Store the client disconnected state
session.subscribed	queue/#	on_message_fetch_for_queue	Fetch one to one offline message
session.subscribed	pub-sub/#	on_message_fetch_for_pubsub	Fetch one to many offline message
session.subscribed	#	on_retain_lookup	Lookup retained message
message.publish	#	on_message_publish	Store the published messages
message.publish	#	on_message_retain	Store retained messages
message.publish	#	on_retain_delete	Delete retained messages
message.acked	queue/#	on_message_acked_for_queue	Process ACK of one to one messages
message.acked	pub-sub/#	on_message_acked_for_pubsub	Process ACK of one to many messages

10.2.4 Redis Command Line Parameters

hook	Parameter	Example (Fields separated exactly by one space)
client.connected	clientid	SET conn:${clientid} clientid
client.disconnected	clientid	SET disconn:${clientid} clientid
session.subscribed	clientid, topic, qos	HSET sub:${clientid} topic qos
session.unsubscribed	clientid, topic	SET unsub:${clientid} topic
message.publish	message, msgid, topic, payload, qos, clientid	RPUSH pub:${topic} msgid
message.acked	msgid, topic, clientid	HSET ack:${clientid} topic msgid
message.delivered	msgid, topic, clientid	HSET delivered:${clientid} topic msgid

10.2.5 Configure 'action' with Redis Commands

Redis backend supports raw 'commands' in 'action', e.g.:

```
## After a client connected to the EMQ X server, it executes a redis command␣
↪(multiple redis commands also supported)
backend.redis.hook.client.connected.3 = {"action": {"commands": ["SET conn:$
↪{clientid} clientid"]}, "pool": "pool1"}
```

10.2.6 Using Redis Hash for Devices' Connection State

mqtt:client Hash for devices' connection state:

```
hmset
key = mqtt:client:${clientid}
value = {state:int, online_at:timestamp, offline_at:timestamp}

hset
key = mqtt:node:${node}
field = ${clientid}
value = ${ts}
```

Lookup devices' connection state:

```
HGETALL "mqtt:client:${clientId}"
```

E.g.: Client with ClientId 'test' goes online:

```
HGETALL mqtt:client:test
1) "state"
2) "1"
3) "online_at"
4) "1481685802"
5) "offline_at"
6) "undefined"
```

Client with ClientId 'test' goes offline:

```
HGETALL mqtt:client:test
1) "state"
2) "0"
3) "online_at"
4) "1481685802"
5) "offline_at"
6) "1481685924"
```

10.2.7 Using Redis Hash for Retained Messages

mqtt:retain Hash for retained messages:

```
hmset
key = mqtt:retain:${topic}
value = {id: string, from: string, qos: int, topic: string, retain: int,␣
↪payload: string, ts: timestamp}
```

Lookup retained message:

```
HGETALL "mqtt:retain:${topic}"
```

Lookup retained messages with a topic of 'retain':

```
HGETALL mqtt:retain:topic
 1) "id"
 2) "6P9NLcJ65VXBbC22sYb4"
 3) "from"
 4) "test"
 5) "qos"
 6) "1"
 7) "topic"
 8) "topic"
 9) "retain"
10) "true"
11) "payload"
12) "Hello world!"
13) "ts"
14) "1481690659"
```

10.2.8 Using Redis Hash for messages

mqtt:msg Hash for MQTT messages:

```
hmset
key = mqtt:msg:${msgid}
value = {id: string, from: string, qos: int, topic: string, retain: int,␣
↪payload: string, ts: timestamp}

zadd
key = mqtt:msg:${topic}
field = 1
value = ${msgid}
```

10.2.9 Using Redis Set for Message Acknowledgements

mqtt:acked SET stores acknowledgements from the clients:

```
set
key = mqtt:acked:${clientid}:${topic}
value = ${msgid}
```

10.2.10 Using Redis Hash for Subscription

mqtt:sub Hash for Subscriptions:

```
hset
key = mqtt:sub:${clientid}
field = ${topic}
value = ${qos}
```

A client subscribes to a topic:

```
HSET mqtt:sub:${clientid} ${topic} ${qos}
```

A client with ClientId of 'test' subscribes to topic1 and topic2:

```
HSET "mqtt:sub:test" "topic1" 1
HSET "mqtt:sub:test" "topic2" 2
```

Lookup the subscribed topics of client with ClientId of 'test':

```
HGETALL mqtt:sub:test
1) "topic1"
2) "1"
3) "topic2"
4) "2"
```

10.2.11 Redis SUB/UNSUB Publish

When a device subscribes / unsubscribes to topics, EMQ X broker publish an event to the Redis:

```
PUBLISH
channel = "mqtt_channel"
message = {type: string , topic: string, clientid: string, qos: int}
\*type: [subscribe/unsubscribe]
```

client with ClientID 'test' subscribe to 'topic0':

```
PUBLISH "mqtt_channel" "{\"type\": \"subscribe\", \"topic\": \"topic0\", \
↪"clientid\": \"test\", \"qos\": \"0\"}"
```

Client with ClientId 'test' unsubscribes to 'test_topic0':

```
PUBLISH "mqtt_channel" "{\"type\": \"unsubscribe\", \"topic\": \"test_topic0\
↪", \"clientid\": \"test\"}"
```

10.2.12 Enable Redis Backend

```
./bin/emqx_ctl plugins load emqx_backend_redis
```

10.3 MySQL Backend

Config file: emqx_backend_mysql.conf

10.3.1 Configure MySQL Server

Connection pool of multiple MySQL servers is supported:

```
## Mysql Server
backend.mysql.pool1.server = 127.0.0.1:3306

## Mysql Pool Size
backend.mysql.pool1.pool_size = 8

## Mysql Username
backend.mysql.pool1.user = root

## Mysql Password
backend.mysql.pool1.password = public

## Mysql Database
backend.mysql.pool1.database = mqtt

## Max number of fetch offline messages. Without count limit if infinity
## backend.mysql.max_returned_count = 500

## Time Range. Without time limit if infinity
## d - day
## h - hour
## m - minute
## s - second
## backend.mysql.time_range = 2h
```

10.3.2 Configure MySQL Persistence Hooks

```
## Client Connected Record
backend.mysql.hook.client.connected.1    = {"action": {"function": "on_client_
↪connected"}, "pool": "pool1"}

## Subscribe Lookup Record
backend.mysql.hook.client.connected.2    = {"action": {"function": "on_
↪subscribe_lookup"}, "pool": "pool1"}

## Client DisConnected Record
backend.mysql.hook.client.disconnected.1 = {"action": {"function": "on_client_
↪disconnected"}, "pool": "pool1"}

## Lookup Unread Message QOS > 0
backend.mysql.hook.session.subscribed.1  = {"topic": "#", "action": {"function
↪": "on_message_fetch"}, "pool": "pool1"}

## Lookup Retain Message
backend.mysql.hook.session.subscribed.2  = {"topic": "#", "action": {"function
↪": "on_retain_lookup"}, "pool": "pool1"}

## Store Publish Message  QOS > 0
backend.mysql.hook.message.publish.1     = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1"}
```

()

()

```
## Store Retain Message
backend.mysql.hook.message.publish.2     = {"topic": "#", "action": {"function
↪": "on_message_retain"}, "pool": "pool1"}

## Delete Retain Message
backend.mysql.hook.message.publish.3     = {"topic": "#", "action": {"function
↪": "on_retain_delete"}, "pool": "pool1"}

## Store Ack
backend.mysql.hook.message.acked.1       = {"topic": "#", "action": {"function
↪": "on_message_acked"}, "pool": "pool1"}

## Get offline messages
##  "offline_opts": Get configuration for offline messages
##  max_returned_count: Maximum number of offline messages get at a time
##  time_range: Get only messages in the current time range
## backend.mysql.hook.session.subscribed.1  = {"topic": "#", "action": {
↪"function": "on_message_fetch"}, "offline_opts": {"max_returned_count": 500,
↪ "time_range": "2h"}, "pool": "pool1"}

## If you need to store Qos0 messages, you can enable the following␣
↪configuration
## Warning: When the following configuration is enabled, 'on_message_fetch'␣
↪needs to be disabled, otherwise qos1, qos2 messages will be stored twice
## backend.mysql.hook.message.publish.4     = {"topic": "#", "action": {
↪"function": "on_message_store"}, "pool": "pool1"}
```

10.3.3 Description of MySQL Persistence Hooks

hook	topic	action	Description
client.connected		on_client_connected	Store client connected state
client.connected		on_subscribe_lookup	Subscribed topics
client.disconnected		on_client_disconnected	Store client disconnected state
session.subscribed	#	on_message_fetch	Fetch offline messages
session.subscribed	#	on_retain_lookup	Lookup retained messages
message.publish	#	on_message_publish	Store published messages
message.publish	#	on_message_retain	Store retained messages
message.publish	#	on_retain_delete	Delete retained messages
message.acked	#	on_message_acked	Process ACK

10.3.4 SQL Parameters Description

hook	Parameters	Example (${name} represents available parameter)
client.connected	clientid	insert into conn(clientid) values(${clientid})
client.disconnected	clientid	insert into disconn(clientid) values(${clientid})
session.subscribed	clientid, topic, qos	insert into sub(topic, qos) values(${topic}, ${qos})
session.unsubscribed	clientid, topic	delete from sub where topic = ${topic}
message.publish	msgid, topic, payload, qos, clientid	insert into msg(msgid, topic) values(${msgid}, ${topic})
message.acked	msgid, topic, clientid	insert into ack(msgid, topic) values(${msgid}, ${topic})
message.delivered	msgid, topic, clientid	insert into delivered(msgid, topic) values(${msgid}, ${topic})

10.3.5 Configure 'action' with SQL

MySQL backend supports SQL in 'action':

```
## After a client is connected to the EMQ X server, it executes a SQL command␣
↪(multiple SQL commands also supported)
backend.mysql.hook.client.connected.3 = {"action": {"sql": ["insert into␣
↪conn(clientid) values(${clientid})"]}, "pool": "pool1"}
```

10.3.6 Create MySQL DB

```
create database mqtt;
```

10.3.7 Import MySQL DB & Table Schema

```
mysql -u root -p mqtt < etc/sql/emqx_backend_mysql.sql
```

: DB name is free of choice

10.3.8 MySQL Client Connection Table

mqtt_client stores client connection states:

```
DROP TABLE IF EXISTS `mqtt_client`;
CREATE TABLE `mqtt_client` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `clientid` varchar(64) DEFAULT NULL,
  `state` varchar(3) DEFAULT NULL,
  `node` varchar(100) DEFAULT NULL,
  `online_at` datetime DEFAULT NULL,
  `offline_at` datetime DEFAULT NULL,
  `created` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `mqtt_client_idx` (`clientid`),
  UNIQUE KEY `mqtt_client_key` (`clientid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Query the client connection state:

```
select * from mqtt_client where clientid = ${clientid};
```

If client 'test' is online:

```
select * from mqtt_client where clientid = "test";

+----+----------+-------+---------------+---------------------+--------------
↪-------+---------------------+
| id | clientid | state | node          | online_at           | offline_at  ⌟
↪       | created             |
+----+----------+-------+---------------+---------------------+--------------
↪-------+---------------------+
|  1 | test     | 1     | emqx@127.0.0.1 | 2016-11-15 09:40:40 | NULL        ⌟
↪       | 2016-12-24 09:40:22 |
+----+----------+-------+---------------+---------------------+--------------
↪-------+---------------------+
1 rows in set (0.00 sec)
```

If client 'test' is offline:

```
select * from mqtt_client where clientid = "test";

+----+----------+-------+---------------+---------------------+--------------
↪-------+---------------------+
| id | clientid | state | node          | online_at           | offline_at  ⌟
↪       | created             |
+----+----------+-------+---------------+---------------------+--------------
↪-------+---------------------+
|  1 | test     | 0     | emqx@127.0.0.1 | 2016-11-15 09:40:40 | 2016-11-15⌟
↪09:46:10 | 2016-12-24 09:40:22 |
+----+----------+-------+---------------+---------------------+--------------
↪-------+---------------------+
1 rows in set (0.00 sec)
```

10.3.9 MySQL Subscription Table

mqtt_sub table stores MQTT subscriptions of clients:

```
DROP TABLE IF EXISTS `mqtt_sub`;
CREATE TABLE `mqtt_sub` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `clientid` varchar(64) DEFAULT NULL,
  `topic` varchar(255) DEFAULT NULL,
  `qos` int(3) DEFAULT NULL,
  `created` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `mqtt_sub_idx` (`clientid`,`topic`(255),`qos`),
  UNIQUE KEY `mqtt_sub_key` (`clientid`,`topic`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

E.g., client 'test' subscribes to 'test_topic1' and 'test_topic2':

```
insert into mqtt_sub(clientid, topic, qos) values("test", "test_topic1", 1);
insert into mqtt_sub(clientid, topic, qos) values("test", "test_topic2", 2);
```

Query subscription of a client:

```
select * from mqtt_sub where clientid = ${clientid};
```

E.g., query the Subscription of client 'test':

```
select * from mqtt_sub where clientid = "test";

+----+--------------+-------------+------+---------------------+
| id | clientId     | topic       | qos  | created             |
+----+--------------+-------------+------+---------------------+
|  1 | test         | test_topic1 |    1 | 2016-12-24 17:09:05 |
|  2 | test         | test_topic2 |    2 | 2016-12-24 17:12:51 |
+----+--------------+-------------+------+---------------------+
2 rows in set (0.00 sec)
```

10.3.10 MySQL Message Table

mqtt_msg stores MQTT messages:

```
DROP TABLE IF EXISTS `mqtt_msg`;
CREATE TABLE `mqtt_msg` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `msgid` varchar(100) DEFAULT NULL,
  `topic` varchar(1024) NOT NULL,
  `sender` varchar(1024) DEFAULT NULL,
  `node` varchar(60) DEFAULT NULL,
  `qos` int(11) NOT NULL DEFAULT '0',
  `retain` tinyint(2) DEFAULT NULL,
  `payload` blob,
```

()

()

```
  `arrived` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Query messages published by a client:

```
select * from mqtt_msg where sender = ${clientid};
```

Query messages published by client 'test':

```
select * from mqtt_msg where sender = "test";

+----+----------------------------------+---------+--------+------+-----+-------
↪-+---------+---------------------+
| id | msgid                            | topic   | sender | node | qos |␣
↪retain | payload | arrived             |
+----+----------------------------------+---------+--------+------+-----+-------
↪-+---------+---------------------+
| 1  | 53F98F80F66017005000004A60003 | hello   | test   | NULL |   1 |      ␣
↪0 | hello   | 2016-12-24 17:25:12 |
| 2  | 53F98F9FE42AD7005000004A60004 | world   | test   | NULL |   1 |      ␣
↪0 | world   | 2016-12-24 17:25:45 |
+----+----------------------------------+---------+--------+------+-----+-------
↪-+---------+---------------------+
2 rows in set (0.00 sec)
```

10.3.11 MySQL Retained Message Table

mqtt_retain stores retained messages:

```
DROP TABLE IF EXISTS `mqtt_retain`;
CREATE TABLE `mqtt_retain` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `topic` varchar(200) DEFAULT NULL,
  `msgid` varchar(60) DEFAULT NULL,
  `sender` varchar(100) DEFAULT NULL,
  `node` varchar(100) DEFAULT NULL,
  `qos` int(2) DEFAULT NULL,
  `payload` blob,
  `arrived` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mqtt_retain_key` (`topic`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Query retained messages:

```
select * from mqtt_retain where topic = ${topic};
```

Query retained messages with topic 'retain':

```
select * from mqtt_retain where topic = "retain";

+----+---------+------------------------------+---------+------+------+-----
↪----+---------------------+
| id | topic     | msgid                          | sender   | node | qos   |␣
↪payload | arrived              |
+----+---------+------------------------------+---------+------+------+-----
↪----+---------------------+
|  1 | retain    | 53F33F7E4741E7007000004B70001 | test     | NULL |     1 | www␣
↪     | 2016-12-24 16:55:18 |
+----+---------+------------------------------+---------+------+------+-----
↪----+---------------------+
1 rows in set (0.00 sec)
```

10.3.12 MySQL Acknowledgement Table

mqtt_acked stores acknowledgements from the clients:

```
DROP TABLE IF EXISTS `mqtt_acked`;
CREATE TABLE `mqtt_acked` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `clientid` varchar(200) DEFAULT NULL,
  `topic` varchar(200) DEFAULT NULL,
  `mid` int(200) DEFAULT NULL,
  `created` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mqtt_acked_key` (`clientid`,`topic`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

10.3.13 Enable MySQL Backend

```
./bin/emqx_ctl plugins load emqx_backend_mysql
```

10.4 PostgreSQL Backend

Config file: emqx_backend_pgsql.conf

10.4.1 Configure PostgreSQL Server

Connection pool of multiple PostgreSQL servers is supported:

```
## Pgsql Server
backend.pgsql.pool1.server = 127.0.0.1:5432

## Pgsql Pool Size
```

()

()

```
backend.pgsql.pool1.pool_size = 8

## Pgsql Username
backend.pgsql.pool1.username = root

## Pgsql Password
backend.pgsql.pool1.password = public

## Pgsql Database
backend.pgsql.pool1.database = mqtt

## Pgsql Ssl
backend.pgsql.pool1.ssl = false

## Max number of fetch offline messages. Without count limit if infinity
## backend.pgsql.max_returned_count = 500

## Time Range. Without time limit if infinity
## d - day
## h - hour
## m - minute
## s - second
## backend.pgsql.time_range = 2h
```

10.4.2 Configure PostgreSQL Persistence Hooks

```
## Client Connected Record
backend.pgsql.hook.client.connected.1    = {"action": {"function": "on_client_
↪connected"}, "pool": "pool1"}

## Subscribe Lookup Record
backend.pgsql.hook.client.connected.2    = {"action": {"function": "on_
↪subscribe_lookup"}, "pool": "pool1"}

## Client DisConnected Record
backend.pgsql.hook.client.disconnected.1 = {"action": {"function": "on_client_
↪disconnected"}, "pool": "pool1"}

## Lookup Unread Message QOS > 0
backend.pgsql.hook.session.subscribed.1  = {"topic": "#", "action": {"function
↪": "on_message_fetch"}, "pool": "pool1"}

## Lookup Retain Message
backend.pgsql.hook.session.subscribed.2  = {"topic": "#", "action": {"function
↪": "on_retain_lookup"}, "pool": "pool1"}

## Store Publish Message  QOS > 0
backend.pgsql.hook.message.publish.1     = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1"}
```

()

()

```
## Store Retain Message
backend.pgsql.hook.message.publish.2     = {"topic": "#", "action": {"function
↪": "on_message_retain"}, "pool": "pool1"}

## Delete Retain Message
backend.pgsql.hook.message.publish.3     = {"topic": "#", "action": {"function
↪": "on_retain_delete"}, "pool": "pool1"}

## Store Ack
backend.pgsql.hook.message.acked.1       = {"topic": "#", "action": {"function
↪": "on_message_acked"}, "pool": "pool1"}

## Get offline messages
##  "offline_opts": Get configuration for offline messages
##  max_returned_count: Maximum number of offline messages get at a time
##  time_range: Get only messages in the current time range
## backend.pgsql.hook.session.subscribed.1  = {"topic": "#", "action": {
↪"function": "on_message_fetch"}, "offline_opts": {"max_returned_count": 500,
↪ "time_range": "2h"}, "pool": "pool1"}

## If you need to store Qos0 messages, you can enable the following␣
↪configuration
## Warning: When the following configuration is enabled, 'on_message_fetch'␣
↪needs to be disabled, otherwise qos1, qos2 messages will be stored twice
## backend.pgsql.hook.message.publish.4     = {"topic": "#", "action": {
↪"function": "on_message_store"}, "pool": "pool1"}
```

10.4.3 Description of PostgreSQL Persistence Hooks

hook	topic	action	Description
client.connected		on_client_connected	Store client connected state
client.connected		on_subscribe_lookup	Subscribed topics
client.disconnected		on_client_disconnected	Store client disconnected state
session.subscribed	#	on_message_fetch	Fetch offline messages
session.subscribed	#	on_retain_lookup	Lookup retained messages
message.publish	#	on_message_publish	Store published messages
message.publish	#	on_message_retain	Store retained messages
message.publish	#	on_retain_delete	Delete retained messages
message.acked	#	on_message_acked	Process ACK

10.4.4 SQL Parameters Description

hook	Parameters	Example (${name} represents available parameter)
client.connected	clientid	insert into conn(clientid) values(${clientid})
client.disconnected	clientid	insert into disconn(clientid) values(${clientid})
session.subscribed	clientid, topic, qos	insert into sub(topic, qos) values(${topic}, ${qos})
session.unsubscribed	clientid, topic	delete from sub where topic = ${topic}
message.publish	msgid, topic, payload, qos, clientid	insert into msg(msgid, topic) values(${msgid}, ${topic})
message.acked	msgid, topic, clientid	insert into ack(msgid, topic) values(${msgid}, ${topic})
message.delivered	msgid, topic, clientid	insert into delivered(msgid, topic) values(${msgid}, ${topic})

10.4.5 Configure 'action' with SQL

PostgreSQL backend supports SQL in 'action':

```
## After a client is connected to the EMQ X server, it executes a SQL command␣
↪(multiple command also supported)
backend.pgsql.hook.client.connected.3 = {"action": {"sql": ["insert into␣
↪conn(clientid) values(${clientid})"]}, "pool": "pool1"}
```

10.4.6 Create PostgreSQL DB

```
createdb mqtt -E UTF8 -e
```

10.4.7 Import PostgreSQL DB & Table Schema

```
\i etc/sql/emqx_backend_pgsql.sql
```

: DB name is free of choice

10.4.8 PostgreSQL Client Connection Table

mqtt_client stores client connection states:

```
CREATE TABLE mqtt_client(
  id SERIAL primary key,
  clientid character varying(100),
  state integer,
  node character varying(100),
  online_at timestamp,
  offline_at timestamp,
  created timestamp without time zone,
  UNIQUE (clientid)
);
```

Query a client's connection state:

```
select * from mqtt_client where clientid = ${clientid};
```

E.g., if client 'test' is online:

```
select * from mqtt_client where clientid = 'test';

 id | clientid | state | node             | online_at           | offline_at
↪       | created
----+----------+-------+------------------+---------------------+---------------
↪------+---------------------
  1 | test     | 1     | emqx@127.0.0.1 | 2016-11-15 09:40:40 | NULL
↪       | 2016-12-24 09:40:22
(1 rows)
```

Client 'test' is offline:

```
select * from mqtt_client where clientid = 'test';

 id | clientid | state | nod            | online_at           | offline_at
↪       | created
----+----------+-------+----------------+---------------------+---------------
↪------+---------------------
  1 | test     | 0     | emqx@127.0.0.1 | 2016-11-15 09:40:40 | 2016-11-15
↪09:46:10 | 2016-12-24 09:40:22
(1 rows)
```

10.4.9 PostgreSQL Subscription Table

mqtt_sub stores subscriptions of clients:

```
CREATE TABLE mqtt_sub(
  id SERIAL primary key,
  clientid character varying(100),
  topic character varying(200),
  qos integer,
  created timestamp without time zone,
  UNIQUE (clientid, topic)
);
```

E.g., client 'test' subscribes to topic 'test_topic1' and 'test_topic2':

```
insert into mqtt_sub(clientid, topic, qos) values('test', 'test_topic1', 1);
insert into mqtt_sub(clientid, topic, qos) values('test', 'test_topic2', 2);
```

Query subscription of a client:

```
select * from mqtt_sub where clientid = ${clientid};
```

Query subscription of client 'test':

```
select * from mqtt_sub where clientid = 'test';

 id | clientId     | topic       | qos  | created
----+--------------+-------------+------+---------------------
  1 | test         | test_topic1 |    1 | 2016-12-24 17:09:05
  2 | test         | test_topic2 |    2 | 2016-12-24 17:12:51
(2 rows)
```

10.4.10 PostgreSQL Message Table

mqtt_msg stores MQTT messages:

```
CREATE TABLE mqtt_msg (
  id SERIAL primary key,
  msgid character varying(60),
  sender character varying(100),
  topic character varying(200),
  qos integer,
  retain integer,
  payload text,
  arrived timestamp without time zone
);
```

Query messages published by a client:

```
select * from mqtt_msg where sender = ${clientid};
```

Query messages published by client 'test':

```
select * from mqtt_msg where sender = 'test';

 id | msgid                            | topic    | sender | node | qos | retain␣
↪| payload | arrived
----+----------------------------------+----------+--------+------+-----+--------
↪+---------+---------------------
 1  | 53F98F80F660170050000004A60003 | hello    | test   | NULL |   1 |      0␣
↪| hello   | 2016-12-24 17:25:12
 2  | 53F98F9FE42AD7005000004A60004 | world    | test   | NULL |   1 |      0␣
↪| world   | 2016-12-24 17:25:45
(2 rows)
```

10.4.11 PostgreSQL Retained Message Table

mqtt_retain stores retained messages:

```
CREATE TABLE mqtt_retain(
  id SERIAL primary key,
  topic character varying(200),
  msgid character varying(60),
  sender character varying(100),
  qos integer,
  payload text,
  arrived timestamp without time zone,
  UNIQUE (topic)
);
```

Query retained messages:

```
select * from mqtt_retain where topic = ${topic};
```

Query retained messages with topic 'retain':

```
select * from mqtt_retain where topic = 'retain';

 id | topic    | msgid                            | sender   | node | qos   |␣
↪payload | arrived
----+----------+----------------------------------+----------+------+-------+------
↪---+---------------------
  1 | retain   | 53F33F7E4741E7007000004B70001 | test     | NULL |     1 | www ␣
↪    | 2016-12-24 16:55:18
(1 rows)
```

10.4.12 PostgreSQL Acknowledgement Table

mqtt_acked stores acknowledgements from the clients:

```
CREATE TABLE mqtt_acked (
  id SERIAL primary key,
  clientid character varying(100),
  topic character varying(100),
  mid integer,
  created timestamp without time zone,
  UNIQUE (clientid, topic)
);
```

10.4.13 Enable PostgreSQL Backend

```
./bin/emqx_ctl plugins load emqx_backend_pgsql
```

10.5 MongoDB Backend

Config file: emqx_backend_mongo.conf

10.5.1 Configure MongoDB Server

Connection pool of multiple MongoDB servers is supported:

```
## MongoDB Server Pools
## Mongo Topology Type single|unknown|sharded|rs
backend.mongo.pool1.type = single

## If type rs, need config setname
## backend.mongo.pool1.rs_set_name = testrs

## Mongo Server 127.0.0.1:27017,127.0.0.2:27017...
backend.mongo.pool1.server = 127.0.0.1:27017

## MongoDB Pool Size
backend.mongo.pool1.c_pool_size = 8

## MongoDB Database
backend.mongo.pool1.database = mqtt

## Mongo User
## backend.mongo.pool1.login =  emqtt
## Mongo Password
## backend.mongo.pool1.password = emqtt

## MongoDB AuthSource
## Value: String
## Default: mqtt
## backend.mongo.pool1.auth_source = admin

## Whether to enable SSL connection.
##
## Value: true | false
## backend.mongo.pool1.ssl = false

## SSL keyfile.
##
## Value: File
## backend.mongo.pool1.keyfile =

## SSL certfile.
##
## Value: File
## backend.mongo.pool1.certfile =

## SSL cacertfile.
##
```

()

()

```
## Value: File
## backend.mongo.pool1.cacertfile =

# Value: unsafe | safe
## backend.mongo.pool1.w_mode = safe
## Value: master | slave_ok
## backend.mongo.pool1.r_mode = slave_ok

## Mongo Topology Options
## backend.mongo.topology.pool_size = 1
## backend.mongo.topology.max_overflow = 0
## backend.mongo.topology.overflow_ttl = 1000
## backend.mongo.topology.overflow_check_period = 1000
## backend.mongo.topology.local_threshold_ms = 1000
## backend.mongo.topology.connect_timeout_ms = 20000
## backend.mongo.topology.socket_timeout_ms = 100
## backend.mongo.topology.server_selection_timeout_ms = 30000
## backend.mongo.topology.wait_queue_timeout_ms = 1000
## backend.mongo.topology.heartbeat_frequency_ms = 10000
## backend.mongo.topology.min_heartbeat_frequency_ms = 1000
## Max number of fetch offline messages. Without count limit if infinity
## backend.mongo.max_returned_count = 500

## Time Range. Without time limit if infinity
## d - day
## h - hour
## m - minute
## s - second
## backend.mongo.time_range = 2h
```

10.5.2 Configure MongoDB Persistence Hooks

```
## Client Connected Record
backend.mongo.hook.client.connected.1    = {"action": {"function": "on_client_
↪connected"}, "pool": "pool1"}

## Subscribe Lookup Record
backend.mongo.hook.client.connected.2    = {"action": {"function": "on_
↪subscribe_lookup"}, "pool": "pool1"}

## Client DisConnected Record
backend.mongo.hook.client.disconnected.1 = {"action": {"function": "on_client_
↪disconnected"}, "pool": "pool1"}

## Lookup Unread Message QOS > 0
backend.mongo.hook.session.subscribed.1  = {"topic": "#", "action": {"function
↪": "on_message_fetch"}, "pool": "pool1"}

## Lookup Retain Message
backend.mongo.hook.session.subscribed.2  = {"topic": "#", "action": {"function
↪": "on_retain_lookup"}, "pool": "pool1"}
```

()

()

```
## Store Publish Message  QOS > 0, payload_format options mongo_json | plain_
↪text
backend.mongo.hook.message.publish.1     = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1", "payload_format": "mongo_json"}

## Store Retain Message, payload_format options mongo_json | plain_text
backend.mongo.hook.message.publish.2     = {"topic": "#", "action": {"function
↪": "on_message_retain"}, "pool": "pool1", "payload_format": "mongo_json"}

## Delete Retain Message
backend.mongo.hook.message.publish.3     = {"topic": "#", "action": {"function
↪": "on_retain_delete"}, "pool": "pool1"}

## Store Ack
backend.mongo.hook.message.acked.1       = {"topic": "#", "action": {"function
↪": "on_message_acked"}, "pool": "pool1"}

## Get offline messages
##  "offline_opts": Get configuration for offline messages
##  max_returned_count: Maximum number of offline messages get at a time
##  time_range: Get only messages in the current time range
## backend.mongo.hook.session.subscribed.1  = {"topic": "#", "action": {
↪"function": "on_message_fetch"}, "offline_opts": {"max_returned_count": 500,
↪ "time_range": "2h"}, "pool": "pool1"}

## If you need to store Qos0 messages, you can enable the following␣
↪configuration
## Warning: When the following configuration is enabled, 'on_message_fetch'␣
↪needs to be disabled, otherwise qos1, qos2 messages will be stored twice
## backend.mongo.hook.message.publish.4     = {"topic": "#", "action": {
↪"function": "on_message_store"}, "pool": "pool1"}
```

10.5.3 Description of MongoDB Persistence Hooks

hook	topic	action	Description
client.connected		on_client_connected	Store client connected state
client.connected		on_subscribe_lookup	Subscribed topics
client.disconnected		on_client_disconnected	Store client disconnected state
session.subscribed	#	on_message_fetch	Fetch offline messages
session.subscribed	#	on_retain_lookup	Lookup retained messages
message.publish	#	on_message_publish	Store published messages
message.publish	#	on_message_retain	Store retained messages
message.publish	#	on_retain_delete	Delete retained messages
message.acked	#	on_message_acked	Process ACK

10.5.4 Create MongoDB DB & Collections

```
use mqtt
db.createCollection("mqtt_client")
db.createCollection("mqtt_sub")
db.createCollection("mqtt_msg")
db.createCollection("mqtt_retain")
db.createCollection("mqtt_acked")

db.mqtt_client.ensureIndex({clientid:1, node:2})
db.mqtt_sub.ensureIndex({clientid:1})
db.mqtt_msg.ensureIndex({sender:1, topic:2})
db.mqtt_retain.ensureIndex({topic:1})
```

: DB name is free of choice

10.5.5 MongoDB MQTT Client Collection

mqtt_client stores MQTT clients' connection states:

```
{
    clientid: string,
    state: 0,1, //0 disconnected 1 connected
    node: string,
    online_at: timestamp,
    offline_at: timestamp
}
```

Query client's connection state:

```
db.mqtt_client.findOne({clientid: ${clientid}})
```

E.g., if client 'test' is online:

```
db.mqtt_client.findOne({clientid: "test"})

{
    "_id" : ObjectId("58646c9bdde89a9fb9f7fb73"),
    "clientid" : "test",
    "state" : 1,
    "node" : "emqx@127.0.0.1",
    "online_at" : 1482976411,
    "offline_at" : null
}
```

Client 'test' is offline:

```
db.mqtt_client.findOne({clientid: "test"})

{
    "_id" : ObjectId("58646c9bdde89a9fb9f7fb73"),
    "clientid" : "test",
    "state" : 0,
    "node" : "emq@127.0.0.1",
    "online_at" : 1482976411,
    "offline_at" : 1482976501
}
```

10.5.6 MongoDB Subscription Collection

mqtt_sub stores subscriptions of clients:

```
{
    clientid: string,
    topic: string,
    qos: 0,1,2
}
```

E.g., client 'test' subscribes to topic 'test_topic1' and 'test_topic2':

```
db.mqtt_sub.insert({clientid: "test", topic: "test_topic1", qos: 1})
db.mqtt_sub.insert({clientid: "test", topic: "test_topic2", qos: 2})
```

Query subscription of client 'test':

```
db.mqtt_sub.find({clientid: "test"})

{ "_id" : ObjectId("58646d90c65dff6ac9668ca1"), "clientid" : "test", "topic"␣
↪: "test_topic1", "qos" : 1 }
{ "_id" : ObjectId("58646d96c65dff6ac9668ca2"), "clientid" : "test", "topic"␣
↪: "test_topic2", "qos" : 2 }
```

10.5.7 MongoDB Message Collection

mqtt_msg stores MQTT messages:

```
{
    _id: int,
    topic: string,
    msgid: string,
    sender: string,
    qos: 0,1,2,
    retain: boolean (true, false),
    payload: string,
    arrived: timestamp
}
```

Query messages published by a client:

```
db.mqtt_msg.find({sender: ${clientid}})
```

Query messages published by client 'test':

```
db.mqtt_msg.find({sender: "test"})
{
    "_id" : 1,
    "topic" : "/World",
    "msgid" : "AAVEwm0la4RufgAABeIAAQ==",
    "sender" : "test",
    "qos" : 1,
    "retain" : 1,
    "payload" : "Hello world!",
    "arrived" : 1482976729
}
```

10.5.8 MongoDB Retained Message Collection

mqtt_retain stores retained messages:

```
{
    topic: string,
    msgid: string,
    sender: string,
    qos: 0,1,2,
    payload: string,
    arrived: timestamp
}
```

Query retained messages:

```
db.mqtt_retain.findOne({topic: ${topic}})
```

Query retained messages with topic 'retain':

```
db.mqtt_retain.findOne({topic: "/World"})
{
    "_id" : ObjectId("58646dd9dde89a9fb9f7fb75"),
    "topic" : "/World",
    "msgid" : "AAVEwm0la4RufgAABeIAAQ==",
    "sender" : "c1",
    "qos" : 1,
    "payload" : "Hello world!",
    "arrived" : 1482976729
}
```

10.5.9 MongoDB Acknowledgement Collection

mqtt_acked stores acknowledgements from the clients:

```
{
    clientid: string,
    topic: string,
    mongo_id: int
}
```

10.5.10 Enable MongoDB Backend

```
./bin/emqx_ctl plugins load emqx_backend_mongo
```

10.6 Cassandra Backend

Config file: etc/plugins/emqx_backend_cassa.conf

10.6.1 Configure Cassandra Cluster

Multi node Cassandra cluster is supported:

```
## Cassandra Node
backend.ecql.pool1.nodes = 127.0.0.1:9042

## Cassandra Pool Size
backend.ecql.pool1.size = 8

## Cassandra auto reconnect flag
backend.ecql.pool1.auto_reconnect = 1

## Cassandra Username
backend.ecql.pool1.username = cassandra

## Cassandra Password
backend.ecql.pool1.password = cassandra

## Cassandra Keyspace
backend.ecql.pool1.keyspace = mqtt

## Cassandra Logger type
backend.ecql.pool1.logger = info

## Max number of fetch offline messages. Without count limit if infinity
## backend.cassa.max_returned_count = 500

## Time Range. Without time limit if infinity
```

()

()

```
## d - day
## h - hour
## m - minute
## s - second
## backend.cassa.time_range = 2h
```

10.6.2 Configure Cassandra Persistence Hooks

```
## Client Connected Record
backend.cassa.hook.client.connected.1    = {"action": {"function": "on_client_
↪connected"}, "pool": "pool1"}

## Subscribe Lookup Record
backend.cassa.hook.client.connected.2    = {"action": {"function": "on_
↪subscription_lookup"}, "pool": "pool1"}

## Client DisConnected Record
backend.cassa.hook.client.disconnected.1 = {"action": {"function": "on_client_
↪disconnected"}, "pool": "pool1"}

## Lookup Unread Message QOS > 0
backend.cassa.hook.session.subscribed.1  = {"topic": "#", "action": {"function
↪": "on_message_fetch"}, "pool": "pool1"}

## Lookup Retain Message
backend.cassa.hook.session.subscribed.2  = {"action": {"function": "on_retain_
↪lookup"}, "pool": "pool1"}

## Store Publish Message  QOS > 0
backend.cassa.hook.message.publish.1     = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1"}

## Delete Acked Record
backend.cassa.hook.session.unsubscribed.1= {"topic": "#", action": {"cql": [
↪"delete from acked where client_id = ${clientid} and topic = ${topic}"]},
↪"pool": "pool1"}

## Store Retain Message
backend.cassa.hook.message.publish.2     = {"topic": "#", "action": {"function
↪": "on_message_retain"}, "pool": "pool1"}

## Delete Retain Message
backend.cassa.hook.message.publish.3     = {"topic": "#", "action": {"function
↪": "on_retain_delete"}, "pool": "pool1"}

## Store Ack
backend.cassa.hook.message.acked.1       = {"topic": "#", "action": {"function
↪": "on_message_acked"}, "pool": "pool1"}

## Get offline messages
```

()

()

```
##  "offline_opts": Get configuration for offline messages
##  max_returned_count: Maximum number of offline messages get at a time
##  time_range: Get only messages in the current time range
## backend.cassa.hook.session.subscribed.1  = {"topic": "#", "action": {
↪"function": "on_message_fetch"}, "offline_opts": {"max_returned_count": 500,
↪ "time_range": "2h"}, "pool": "pool1"}

## If you need to store Qos0 messages, you can enable the following␣
↪configuration
## Warning: When the following configuration is enabled, 'on_message_fetch'␣
↪needs to be disabled, otherwise qos1, qos2 messages will be stored twice
## backend.cassa.hook.message.publish.4     = {"topic": "#", "action": {
↪"function": "on_message_store"}, "pool": "pool1"}
```

10.6.3 Description of Cassandra Persistence Hooks

hook	topic	action	Description
client.connected		on_client_connected	Store client connected state
client.connected		on_subscribe_lookup	Subscribed topics
client.disconnected		on_client_disconnected	Store client disconnected state
session.subscribed	#	on_message_fetch	Fetch offline messages
session.subscribed	#	on_retain_lookup	Lookup retained messages
message.publish	#	on_message_publish	Store published messages
message.publish	#	on_message_retain	Store retained messages
message.publish	#	on_retain_delete	Delete retained messages
message.acked	#	on_message_acked	Process ACK

10.6.4 CQL Parameters Description

Customized CQL command parameters includes:

hook	Parameter	Example (${name} in CQL represents available parameter
client.connected	clientid	insert into conn(clientid) values(${clientid})
client.disconnected	clientid	insert into disconn(clientid) values(${clientid})
session.subscribed	clientid, topic, qos	insert into sub(topic, qos) values(${topic}, ${qos})
session.unsubscribed	clientid, topic	delete from sub where topic = ${topic}
message.publish	msgid, topic, payload, qos, clientid	insert into msg(msgid, topic) values(${msgid}, ${topic})
message.acked	msgid, topic, clientid	insert into ack(msgid, topic) values(${msgid}, ${topic})
message.delivered	msgid, topic, clientid	insert into delivered(msgid, topic) values(${msgid}, ${topic})

10.6.5 Configure 'action' with CQL

Cassandra backend supports CLQ in 'action':

```
## After a client is connected to the EMQ X server, it executes a CQL␣
↪command(multiple command also supported):
backend.cassa.hook.client.connected.3 = {"action": {"cql": ["insert into␣
↪conn(clientid) values(${clientid})"]}, "pool": "pool1"}
```

10.6.6 Initializing Cassandra

Create KeySpace:

```
CREATE KEYSPACE mqtt WITH REPLICATION = { 'class' : 'SimpleStrategy',
↪'replication_factor' : 1 };
USR mqtt;
```

Import Cassandra tables:

```
cqlsh -e "SOURCE 'emqx_backend_cassa.cql'"
```

: KeySpace is free of choice

10.6.7 Cassandra Client Connection Table

mqtt.client stores client connection states:

```
CREATE TABLE mqtt.client (
    client_id text,
    node text,
    state int,
    connected timestamp,
    disconnected timestamp,
    PRIMARY KEY(client_id)
);
```

Query a client's connection state:

```
select * from mqtt.client where clientid = ${clientid};
```

If client 'test' is online:

```
select * from mqtt.client where clientid = 'test';

 client_id | connected                       | disconnected   | node          ␣
↪| state
```

()

()

```
-----------+--------------------------------+--------------+---------------
↪+-------
      test | 2017-02-14 08:27:29.872000+0000 |          null | emqx@127.0.0.
↪1|     1
```

Client 'test' is offline:

```
select * from mqtt.client where clientid = 'test';

 client_id | connected                      | disconnected
↪ | node          | state
-----------+--------------------------------+-------------------------------
↪-+---------------+-------
      test | 2017-02-14 08:27:29.872000+0000 | 2017-02-14 08:27:35.
↪872000+0000 | emqx@127.0.0.1|     0
```

10.6.8 Cassandra Subscription Table

mqtt.sub stores subscriptions of clients:

```
CREATE TABLE mqtt.sub (
    client_id text,
    topic text,
    qos int,
    PRIMARY KEY(client_id, topic)
);
```

Client 'test' subscribes to topic 'test_topic1' and 'test_topic2':

```
insert into mqtt.sub(client_id, topic, qos) values('test', 'test_topic1', 1);
insert into mqtt.sub(client_id, topic, qos) values('test', 'test_topic2', 2);
```

Query subscriptions of a client:

```
select * from mqtt_sub where clientid = ${clientid};
```

Query subscriptions of client 'test':

```
select * from mqtt_sub where clientid = 'test';

 client_id | topic       | qos
-----------+-------------+-----
      test | test_topic1 |   1
      test | test_topic2 |   2
```

10.6.9 Cassandra Message Table

mqtt.msg stores MQTT messages:

```
CREATE TABLE mqtt.msg (
    topic text,
    msgid text,
    sender text,
    qos int,
    retain int,
    payload text,
    arrived timestamp,
    PRIMARY KEY(topic, msgid)
  ) WITH CLUSTERING ORDER BY (msgid DESC);
```

Query messages published by a client:

```
select * from mqtt_msg where sender = ${clientid};
```

Query messages published by client 'test':

```
select * from mqtt_msg where sender = 'test';

 topic | msgid                | arrived                         | payload
↪ | qos | retain | sender
-------+----------------------+---------------------------------+-------------
↪-+-----+--------+--------
 hello | 2PguFrHsrzEvIIBdctmb | 2017-02-14 09:07:13.785000+0000 | Hello world!
↪ |   1 |      0 |   test
 world | 2PguFrHsrzEvIIBdctmb | 2017-02-14 09:07:13.785000+0000 | Hello world!
↪ |   1 |      0 |   test
```

10.6.10 Cassandra Retained Message Table

mqtt.retain stores retained messages:

```
CREATE TABLE mqtt.retain (
    topic text,
    msgid text,
    PRIMARY KEY(topic)
);
```

Query retained messages:

```
select * from mqtt_retain where topic = ${topic};
```

Query retained messages with topic 'retain':

```
select * from mqtt_retain where topic = 'retain';

 topic  | msgid
--------+----------------------
 retain | 2PguFrHsrzEvIIBdctmb
```

10.6.11 Cassandra Acknowledgement Table

mqtt.acked stores acknowledgements from the clients:

```
CREATE TABLE mqtt.acked (
    client_id text,
    topic text,
    msgid text,
    PRIMARY KEY(client_id, topic)
  );
```

10.6.12 Enable Cassandra Backend

```
./bin/emqx_ctl plugins load emqx_backend_cassa
```

10.7 DynamoDB Backend

10.7.1 Configure DynamoDB Cluster

Config file: etc/plugins/emqx_backend_dynamo.conf

```
## DynamoDB Region
backend.dynamo.region = us-west-2

## DynamoDB Server
backend.dynamo.pool1.server = http://localhost:8000

## DynamoDB Pool Size
backend.dynamo.pool1.pool_size = 8

## AWS ACCESS KEY ID
backend.dynamo.pool1.aws_access_key_id = AKIAU5IM2XOC7AQWG7HK

## AWS SECRET ACCESS KEY
backend.dynamo.pool1.aws_secret_access_key = TZt7XoRi+vtCJYQ9YsAinh19jR1rngm/
↪hxZMWR2P

## DynamoDB Backend Hooks
backend.dynamo.hook.client.connected.1    = {"action": {"function": "on_
↪client_connected"}, "pool": "pool1"}
backend.dynamo.hook.session.created.1     = {"action": {"function": "on_
↪subscribe_lookup"}, "pool": "pool1"}
backend.dynamo.hook.client.disconnected.1 = {"action": {"function": "on_
↪client_disconnected"}, "pool": "pool1"}
backend.dynamo.hook.session.subscribed.1  = {"topic": "#", "action": {
↪"function": "on_message_fetch_for_queue"}, "pool": "pool1"}
backend.dynamo.hook.session.subscribed.2  = {"topic": "#", "action": {
↪"function": "on_retain_lookup"}, "pool": "pool1"}
```

()

()

```
backend.dynamo.hook.session.unsubscribed.1= {"topic": "#", "action": {
↪"function": "on_acked_delete"}, "pool": "pool1"}
backend.dynamo.hook.message.publish.1     = {"topic": "#", "action": {
↪"function": "on_message_publish"}, "pool": "pool1"}
backend.dynamo.hook.message.publish.2     = {"topic": "#", "action": {
↪"function": "on_message_retain"}, "pool": "pool1"}
backend.dynamo.hook.message.publish.3     = {"topic": "#", "action": {
↪"function": "on_retain_delete"}, "pool": "pool1"}
backend.dynamo.hook.message.acked.1       = {"topic": "#", "action": {
↪"function": "on_message_acked_for_queue"}, "pool": "pool1"}

# backend.dynamo.hook.message.publish.4   = {"topic": "#", "action": {
↪"function": "on_message_store"}, "pool": "pool1"}
```

10.7.2 Description of DynamoDB Persistence Hooks

hook	topic	action	Description
client.connected		on_client_connected	Store client connected state
client.connected		on_subscribe_lookup	Subscribed topics
client.disconnected		on_client_disconnected	Store client disconnected state
session.subscribed	#	on_message_fetch_for_queue	Fetch offline messages
session.subscribed	#	on_retain_lookup	Lookup retained messages
message.publish	#	on_message_publish	Store published messages
message.publish	#	on_message_retain	Store retained messages
message.publish	#	on_retain_delete	Delete retained messages
message.acked	#	on_message_acked_for_queue	Process ACK

10.7.3 Create DynamoDB DB

```
./test/dynamo_test.sh
```

: DB name is free of choice

10.7.4 DynamoDB Client Connection Table

mqtt_client stores client connection states:

```
{
    "TableName": "mqtt_client",
    "KeySchema": [
        { "AttributeName": "clientid", "KeyType": "HASH" }
    ],
```

()

()

```
    "AttributeDefinitions": [
        { "AttributeName": "clientid", "AttributeType": "S" }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

Query the client connection state:

```
aws dynamodb scan --table-name mqtt_client --region us-west-2  --endpoint-url␣
↪http://localhost:8000

{
    "Items": [
        {
            "offline_at": { "N": "0" },
            "node": { "S": "emqx@127.0.0.1" },
            "clientid": { "S": "mqttjs_384b9c73a9" },
            "connect_state": { "N": "1" },
            "online_at": { "N": "1562224940" }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

10.7.5 DynamoDB Subscription Table

mqtt_sub table stores MQTT subscriptions of clients:

```
{
    "TableName": "mqtt_sub",
    "KeySchema": [
        { "AttributeName": "clientid", "KeyType": "HASH" },
        { "AttributeName": "topic", "KeyType": "RANGE" }
    ],
    "AttributeDefinitions": [
        { "AttributeName": "clientid", "AttributeType": "S" },
        { "AttributeName": "topic", "AttributeType": "S" }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

Query topics subscribed by the client named "test-dynamo":

```
aws dynamodb scan --table-name mqtt_sub --region us-west-2  --endpoint-url␣
↪http://localhost:8000

{
    "Items": [{"qos": { "N": "2" }, "topic": { "S": "test-dynamo-sub" },
↪"clientid": { "S": "test-dynamo" }},
              {"qos": { "N": "2" }, "topic": { "S": "test-dynamo-sub-1"},
↪"clientid": { "S": "test-dynamo" }},
              {"qos": { "N": "2" }, "topic": { "S": "test-dynamo-sub-2"},
↪"clientid": { "S": "test-dynamo" }}],
    "Count": 3,
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

10.7.6 DynamoDB Message Table

mqtt_msg stores MQTT messages:

```
{
    "TableName": "mqtt_msg",
    "KeySchema": [
        { "AttributeName": "msgid", "KeyType": "HASH" }
    ],
    "AttributeDefinitions": [
        { "AttributeName": "msgid", "AttributeType": "S" }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

mqtt_topic_msg_map stores the mapping between topics and messages:

```
{
    "TableName": "mqtt_topic_msg_map",
    "KeySchema": [
        { "AttributeName": "topic", "KeyType": "HASH" }
    ],
    "AttributeDefinitions": [
        { "AttributeName": "topic", "AttributeType": "S" }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

Query *mqtt_msg* and *mqtt_topic_msg_map* after a client publishes a message to the "test" topic:

Query *mqtt_msg*:

```
 aws dynamodb scan --table-name mqtt_msg --region us-west-2  --endpoint-url␣
↪http://localhost:8000

{
    "Items": [
        {
             "arrived": { "N": "1562308553" },
             "qos": { "N": "1" },
             "sender": { "S": "mqttjs_231b962d5c" },
             "payload": { "S": "{ \"msg\": \"Hello, World!\" }"},
             "retain": { "N": "0" },
             "msgid": { "S": "Mjg4MTk1MDYwNTk0NjYwNzYzMTg4MDk3OTQ2MDU2Nzg1OTD
↪" },
             "topic": { "S": "test" }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

Query *mqtt_topic_msg_map*:

```
 aws dynamodb scan --table-name mqtt_topic_msg_map --region us-west-2  --
↪endpoint-url http://localhost:8000

{
    "Items": [
        {
             "topic": { "S": "test" },
             "MsgId": { "SS": [
↪"Mjg4MTk1MDYwNTk0NjYwNzYzMTg4MDk3OTQ2MDU2Nzg1OTD" ]}
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

10.7.7 DynamoDB Retained Message Table

mqtt_retain stores retained messages:

```
{
    "TableName": "mqtt_retain",
    "KeySchema": [
        { "AttributeName": "topic", "KeyType": "HASH" }
    ],
    "AttributeDefinitions": [
        { "AttributeName": "topic", "AttributeType": "S" }
    ],
```

()

()

```
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

Query *mqtt_retain* after a client publishes a message to the "test" topic:

```
{
    "Items": [
        {
            "arrived": { "N": "1562312113" },
            "qos": { "N": "1" },
            "sender": { "S": "mqttjs_d0513acfce" },
            "payload": { "S": "test" },
            "retain": { "N": "1" },
            "msgid": { "S": "Mjg4MTk1NzE3MTY4MjYxMjA5MDExMDg0NTk5ODgzMjAyNTH"␣
↪},
            "topic": { "S": "testtopic" }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

10.7.8 DynamoDB Acknowledgement Table

mqtt_acked stores acknowledgements from the clients:

```
{
    "TableName": "mqtt_acked",
    "KeySchema": [
        { "AttributeName": "topic", "KeyType": "HASH" },
        { "AttributeName": "clientid", "KeyType": "RANGE" }
    ],
    "AttributeDefinitions": [
        { "AttributeName": "topic", "AttributeType": "S" },
        { "AttributeName": "clientid", "AttributeType": "S" }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

Query *mqtt_acked* after a client publishes a message to the "test" topic:

```
{
    "Items": [
```

()

()

```
        {
            "topic": { "S": "test" },
            "msgid": { "S": "Mjg4MTk1MDYwNTk0NjYwNzYzMTg4MDk3OTQ2MDU2Nzg1OTD"␣
↪},
            "clientid": { "S": "mqttjs_861e582a70" }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

10.7.9 Enable DynamoDB Backend

```
./bin/emqx_ctl plugins load emqx_backend_dynamo
```

10.8 InfluxDB Backend

10.8.1 Configure InfluxDB Server

Config file: etc/plugins/emqx_backend_influxdb.conf:

```
## InfluxDB UDP Server
backend.influxdb.pool1.server = 127.0.0.1:8089

## InfluxDB Pool Size
backend.influxdb.pool1.pool_size = 5

## Wether to add timestamp automatically
backend.influxdb.pool1.set_timestamp = true

backend.influxdb.hook.message.publish.1 = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1"}
```

Parameters in hook rule:

Option	Description
topic	Configure which topics need to execute hooks
action	Configure specific action for hook, `function` is a built-in function provided as Backend for general functions
pool	Pool Name, used to connect multiple InfluxDB servers

Example:

```
## Store PUBLISH message whose topic is "sensor/#"
backend.influxdb.hook.message.publish.1 = {"topic": "sensor/#", "action": {
↪"function": "on_message_publish"}, "pool": "pool1"}

## Store PUBLISH message whose topic is "stat/#"
backend.influxdb.hook.message.publish.2 = {"topic": "stat/#", "action": {
↪"function": "on_message_publish"}, "pool": "pool1"}
```

10.8.2 Description of InfluxDB Persistence Hooks

hook	topic	action	Description
message.publish	#	on_message_publish	Store published messages

Since MQTT Message cannot be written directly to InfluxDB, InfluxDB Backend provides an *emqx_backend_influxdb.tmpl* template file to convert MQTT Message to DataPoint that can be written to InfluxDB.

Template file use Json format:

- `key` - MQTT Topic, Json String, support wildcard characters
- `value` - Template, Json Object, used to convert MQTT Message into `measurement,tag_key=tag_value,... field_key=field_value,... timestamp` and write to InfluxDB

You can define different templates for different topics or multiple templates for the same topic, likes:

```
{
    <Topic 1>: <Template 1>,
    <Topic 2>: <Template 2>
}
```

Template format:

```
{
    "measurement": <Measurement>,
    "tags": {
        <Tag Key>: <Tag Value>
    },
    "fields": {
        <Field Key>: <Field Value>
    },
    "timestamp": <Timestamp>
}
```

`measurement` and `fields` are required options, `tags` and `timestamp` are optional.

All values (such as `<Measurement>`) can be configured directly in the template as a fixed value that data types supported depending on the table you define. More realistically, of course, you can access the data in the MQTT message through the placeholder we provide.

Currently, we support placeholders as follows:

Placeholder	Description
$id	MQTT Message UUID, assigned by EMQ X
$clientid	Client ID used by the Client
$username	Username used by the Client
$peerhost	IP of Client
$qos	QoS of MQTT Message
$topic	Topic of MQTT Message
$payload	Payload of MQTT Message, must be valid Json data
$<Number>	It must be used with $paylaod to retrieve data from Json Array
$timestamp	The timestamp EMQ X sets when preparing to forward messages, precision: Nanoseconds

$payload and $<Number>:

You can directly use `$content` to obtain the complete message payload, you can use `["$payload", <Key>, ...]` to get the data inside the message payload.

For example `payload` is `{"data": {"temperature": 23.9}}`, you can via `["$payload", "data", "temperature"]` to get `23.9`.

In the case of array data type in Json, we introduced `$0` and `$<pos_integer>`, `$0` means to get all elements in the array, and `$<pos_integer>` means to get the <pos_integer>th element in the array.

A simple example, `["$payload", "$0", "temp"]` will get `[20, 21]` from `[{"temp": 20}, {"temp": 21}]`, and `["$payload", "$1", "temp"]` will only get `20`.

It is worth noting that when you use `$0`, we expect the number of data you get is same. Because we need to convert these arrays into multiple records and write it into InfluxDB, and when you have three pieces of data in one field and two in another, we won't know how to combine the data for you.

Example

data/templates directory provides a sample template (emqx_backend_influxdb_example.tmpl, please remove the "_example" suffix from the filename when using it formally) for the user's reference:

```
{
    "sample": {
        "measurement": "$topic",
        "tags": {
            "host": ["$payload", "data", "$0", "host"],
            "region": ["$payload", "data", "$0", "region"],
            "qos": "$qos",
            "clientid": "$clientid"
        },
        "fields": {
            "temperature": ["$payload", "data", "$0", "temp"]
        },
        "timestamp": "$timestamp"
```

()

()

```
    }
}
```

When an MQTT Message whose Topic is "sample" has the following Payload:

```
{
    "data": [
        {
            "temp": 1,
            "host": "serverA",
            "region": "hangzhou"
        },
        {
            "temp": 2,
            "host": "serverB",
            "region": "ningbo"
        }
    ]
}
```

Backend converts MQTT messages to:

```
[
    {
        "measurement": "sample",
        "tags": {
            "clientid": "mqttjs_ebcc36079a",
            "host": "serverA",
            "qos": "0",
            "region": "hangzhou",
        },
        "fields": {
            "temperature": "1"
        },
        "timestamp": "1560743513626681000"
    },
    {
        "measurement": "sample",
        "tags": {
            "clientid": "mqttjs_ebcc36079a",
            "host": "serverB",
            "qos": "0",
            "region": "ningbo",
        },
        "fields": {
            "temperature": "2"
        },
        "timestamp": "1560743513626681000"
    }
]
```

The data was finally encoded and written to InfluxDB as follows:

```
"sample,clientid=mqttjs_6990f0e886,host=serverA,qos=0,region=hangzhou␣
↪temperature=\"1\" 1560745505429670000\nsample,clientid=mqttjs_6990f0e886,
↪host=serverB,qos=0,region=ningbo temperature=\"2\" 1560745505429670000\n"
```

10.8.3 Enable InfluxDB Backend

```
./bin/emqx_ctl plugins load emqx_backend_influxdb
```

10.9 OpenTSDB Backend

10.9.1 Configure OpenTSDB Server

Config file: etc/plugins/emqx_backend_opentsdb.conf:

```
## OpenTSDB Server
backend.opentsdb.pool1.server = 127.0.0.1:4242

## OpenTSDB Pool Size
backend.opentsdb.pool1.pool_size = 8

## Whether to return summary info
backend.opentsdb.pool1.summary = true

## Whether to return detailed info
##
## Value: true | false
backend.opentsdb.pool1.details = false

## Synchronous write or not
##
## Value: true | false
backend.opentsdb.pool1.sync = false

## Synchronous write timeout in milliseconds
##
## Value: Duration
##
## Default: 0
backend.opentsdb.pool1.sync_timeout = 0

## Max batch size
##
## Value: Number >= 0
## Default: 20
backend.opentsdb.pool1.max_batch_size = 20

## Store PUBLISH Messages
backend.opentsdb.hook.message.publish.1 = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1"}
```

()

Parameters in hook rule:

Option	Description
topic	Configure which topics need to execute hooks
action	Configure specific action for hook, `function` is a built-in function provided as Backend for general functions
pool	Pool Name, used to connect multiple OpenTSDB servers

Example:

```
## Store PUBLISH message whose topic is "sensor/#"
backend.influxdb.hook.message.publish.1 = {"topic": "sensor/#", "action": {
↪"function": "on_message_publish"}, "pool": "pool1"}

## Store PUBLISH message whose topic is "stat/#"
backend.influxdb.hook.message.publish.2 = {"topic": "stat/#", "action": {
↪"function": "on_message_publish"}, "pool": "pool1"}
```

10.9.2 Description of OpenTSDB Persistence Hooks

hook	topic	action	Description
message.publish	#	on_message_publish	Store published messages

Since MQTT Message cannot be written directly to OpenTSDB, OpenTSDB Backend provides an *emqx_backend_opentsdb.tmpl* template file to convert MQTT Message to DataPoint that can be written to OpenTSDB.

Template file use Json format:

- `key` - MQTT Topic, Json String, support wildcard characters
- `value` - Template, Json Object, used to convert MQTT Message into `measurement,tag_key=tag_value,... field_key=field_value,... timestamp` and write to InfluxDB

You can define different templates for different topics or multiple templates for the same topic, likes:

```
{
    <Topic 1>: <Template 1>,
    <Topic 2>: <Template 2>
}
```

The template format is as follows:

```
{
    "measurement": <Measurement>,
    "tags": {
        <Tag Key>: <Tag Value>
    },
    "value": <Value>,
    "timestamp": <Timestamp>
}
```

`measurement` and `value` are required options, `tags` and `timestamp` are optional.

All values (such as `<Measurement>`) can be configured directly in the template as a fixed value that data types supported depending on the table you define. More realistically, of course, you can access the data in the MQTT message through the placeholder we provide.

Currently, we support placeholders as follows:

Placeholder	Description
$id	MQTT Message UUID, assigned by EMQ X
$clientid	Client ID used by the Client
$username	Username used by the Client
$peerhost	IP of Client
$qos	QoS of MQTT Message
$topic	Topic of MQTT Message
$payload	Payload of MQTT Message, must be valid Json data
$<Number>	It must be used with $paylaod to retrieve data from Json Array
$timestamp	The timestamp EMQ X sets when preparing to forward messages, precision: Nanoseconds

$payload and $<Number>:

You can directly use `$content` to obtain the complete message payload, you can use `["$payload", <Key>, ...]` to get the data inside the message payload.

For example `payload` is `{"data": {"temperature": 23.9}}`, you can via `["$payload", "data", "temperature"]` to get `23.9`.

In the case of array data type in Json, we introduced `$0` and `$<pos_integer>`, `$0` means to get all elements in the array, and `$<pos_integer>` means to get the <pos_integer>th element in the array.

A simple example, `["$payload", "$0", "temp"]` will get `[20, 21]` from `[{"temp": 20}, {"temp": 21}]`, and `["$payload", "$1", "temp"]` will only get `20`.

It is worth noting that when you use `$0`, we expect the number of data you get is same. Because we need to convert these arrays into multiple records and write it into OpenTSDB, and when you have three pieces of data in one field and two in another, we won't know how to combine the data for you.

Example

data/templates directory provides a sample template (emqx_backend_opentsdb_example.tmpl, please remove the "_example" suffix from the filename when using it formally) for the user's reference:

```
{
    "sample": {
        "measurement": "$topic",
        "tags": {
            "host": ["$payload", "data", "$0", "host"],
            "region": ["$payload", "data", "$0", "region"],
            "qos": "$qos",
            "clientid": "$clientid"
        },
        "value": ["$payload", "data", "$0", "temp"],
        "timestamp": "$timestamp"
    }
}
```

When an MQTT Message whose Topic is "sample" has the following Payload:

```
{
    "data": [
        {
            "temp": 1,
            "host": "serverA",
            "region": "hangzhou"
        },
        {
            "temp": 2,
            "host": "serverB",
            "region": "ningbo"
        }
    ]
}
```

Backend converts MQTT messages into the following data and writes it to OpenTSDB:

```
[
    {
        "measurement": "sample",
        "tags": {
            "clientid": "mqttjs_ebcc36079a",
            "host": "serverA",
            "qos": "0",
            "region": "hangzhou",
        },
        "value": "1",
        "timestamp": "1560743513626681000"
    },
    {
        "measurement": "sample",
        "tags": {
            "clientid": "mqttjs_ebcc36079a",
            "host": "serverB",
            "qos": "0",
            "region": "ningbo",
```

()

()

```
        },
        "value": "2",
        "timestamp": "1560743513626681000"
    }
]
```

10.9.3 Enable OpenTSDB Backend

```
./bin/emqx_ctl plugins load emqx_backend_opentsdb
```

10.10 Timescale Backend

10.10.1 Configure Timescale Server

Config file: etc/plugins/emqx_backend_timescale.conf:

```
## Timescale Server
backend.timescale.pool1.server = 127.0.0.1:5432
## Timescale Pool Size
backend.timescale.pool1.pool_size = 8
## Timescale Username
backend.timescale.pool1.username = postgres
## Timescale Password
backend.timescale.pool1.password = password
## Timescale Database
backend.timescale.pool1.database = tutorial
## Timescale SSL
backend.timescale.pool1.ssl = false

## SSL keyfile.
##
## Value: File
## backend.timescale.pool1.keyfile =

## SSL certfile.
##
## Value: File
## backend.timescale.pool1.certfile =

## SSL cacertfile.
##
## Value: File
## backend.timescale.pool1.cacertfile =

## Store Publish Message
backend.timescale.hook.message.publish.1 = {"topic": "#", "action": {"function
↪": "on_message_publish"}, "pool": "pool1"}
```

Parameters in hook rule:

Option	Description
topic	Configure which topics need to execute hooks
action	Configure specific action for hook, `function` is a built-in function provided as Backend for general functions
pool	Pool Name, used to connect multiple Timescale servers

Example:

```
## Store PUBLISH message whose topic is "sensor/#"
backend.influxdb.hook.message.publish.1 = {"topic": "sensor/#", "action": {
↪"function": "on_message_publish"}, "pool": "pool1"}

## Store PUBLISH message whose topic is "stat/#"
backend.influxdb.hook.message.publish.2 = {"topic": "stat/#", "action": {
↪"function": "on_message_publish"}, "pool": "pool1"}
```

10.10.2 Description of Timescale Persistence Hooks

hook	topic	action	Description
message.publish	#	on_message_publish	Store published messages

Timescale Backend provides the template file named `emqx_backend_timescale.tmpl`, which is used to extract data from MQTT messages with different topics for writing to Timescale.

Template file use Json format:

- `key` - MQTT Topic, Json String, support wildcard characters
- `value` - Template, Json Object, used to convert MQTT Message into `measurement,tag_key=tag_value,... field_key=field_value,... timestamp` and write to InfluxDB

You can define different templates for different topics or multiple templates for the same topic, likes:

```
{
    <Topic 1>: <Template 1>,
    <Topic 2>: <Template 2>
}
```

The template format is as follows:

```
{
    "name": <Name of template>,
    "sql": <SQL INSERT INTO>,
    "param_keys": <Param Keys>
}
```

`name`, `sql` and `param_keys` are required options.

`name` can be any string, just make sure there are no duplicates.

`sql` is SQL INSERT INTO statement for Timescale, like `insert into sensor_data(time, location, temperature, humidity) values (NOW(), $1, $2, $3)`.

`param_keys` is a array, its first element corresponds to `$1` appearing in `sql` and so on.

Any element in an array can be a fixed value, and the data type it supports depends on the table you define. More realistically, of course, you can access the data in the MQTT message through the placeholder we provide.

Currently, we support placeholders as follows:

Place-holder	Description
$id	MQTT Message UUID, assigned by EMQ X
$clientid	Client ID used by the Client
$username	Username used by the Client
$peerhost	IP of Client
$qos	QoS of MQTT Message
$topic	Topic of MQTT Message
$payload	Payload of MQTT Message, must be valid Json data
$<Number>	It must be used with $paylaod to retrieve data from Json Array
$timestamp	The timestamp EMQ X sets when preparing to forward messages, precision: Nanoseconds

$payload and $<Number>:

You can directly use `$content` to obtain the complete message payload, you can use `["$payload", <Key>, ...]` to get the data inside the message payload.

For example `payload` is `{"data": {"temperature": 23.9}}`, you can via `["$payload", "data", "temperature"]` to get `23.9`.

In the case of array data type in Json, we introduced `$0` and `$<pos_integer>`, `$0` means to get all elements in the array, and `$<pos_integer>` means to get the <pos_integer>th element in the array.

A simple example, `["$payload", "$0", "temp"]` will get `[20, 21]` from `[{"temp": 20}, {"temp": 21}]`, and `["$payload", "$1", "temp"]` will only get `20`.

It is worth noting that when you use `$0`, we expect the number of data you get is same. Because we need to convert these arrays into multiple records and write it into Timescale, and when you have three pieces of data in one field and two in another, we won't know how to combine the data for you.

Example

data/templates directory provides a sample template (emqx_backend_timescale_example.tmpl, please remove the "_example" suffix from the filename when using it formally) for the user's reference:

```
{
    "sensor_data": {
        "name": "insert_sensor_data",
        "sql": "insert into sensor_data(time, location, temperature,␣
↪humidity) values (NOW(), $1, $2, $3)",
        "param_keys": [
            ["$payload", "data", "$0", "location"],
            ["$payload", "data", "$0", "temperature"],
            ["$payload", "data", "$0", "humidity"]
        ]
    },
    "sensor_data2/#": {
        "name": "insert_sensor_data2",
        "sql": "insert into sensor_data(time, location, temperature,␣
↪humidity) values (NOW(), $1, $2, $3)",
        "param_keys": [
            ["$payload", "location"],
            ["$payload", "temperature"],
            ["$payload", "humidity"]
        ]
    },
    "easy_data": {
        "name": "insert_easy_data",
        "sql": "insert into easy_data(time, data) values (NOW(), $1)",
        "param_keys": [
            "$payload"
        ]
    }
}
```

When an MQTT Message whose Topic is "sensor_data" has the following Payload:

```
{
    "data":[
        {
            "location":"bedroom",
            "temperature":21.3,
            "humidity":40.3
        },
        {
            "location":"bathroom",
            "temperature":22.3,
            "humidity":61.8
        },
        {
            "location":"kitchen",
            "temperature":29.5,
            "humidity":58.7
        }
    ]
}
```

`["$payload", "data", "$0", "location"]` will extract Payload from MQTT Message first.

If the format of Payload is json, backend continue to extract `data` from Payload.

And the value of `data` is an array, we use `$0` to gets all elements in the array.

`["$payload", "data", "$0", "location"]` will help us get `["bedroom", "bathroom", "kitchen"]` finally.

Accordingly if you replace `$0` with `$1`, you get only `["bedroom"]`.

So in this scene, we will get the following SQL statement:

```
insert into sensor_data(time, location, temperature, humidity) values (NOW(),
↪'bedroom', 21.3, 40.3)
insert into sensor_data(time, location, temperature, humidity) values (NOW(),
↪'bathroom', 22.3, 61.8)
insert into sensor_data(time, location, temperature, humidity) values (NOW(),
↪'kitchen', 29.5, 58.7)
```

Eventually Timescale Backend executes these SQL statements to write data to Timescale.

CHAPTER 11

Bridges

EMQ X can bridge and forward messages to Kafka, RabbitMQ or other EMQ X nodes. Meanwhile, mosquitto and rsm can be bridged to EMQ X using common MQTT connection.

11.1 List of Bridge Plugins

Bridge Plugin	Config File	Description
emqx_bridge_kafka	emqx_bridge_kafka.conf	Kafka Bridge
emqx_bridge_rabbit	emqx_bridge_rabbit.conf	RabbitMQ Bridge
emqx_bridge_pulsar	emqx_bridge_pulsar.conf	Pulsar Bridge
emqx_bridge_mqtt	emqx_bridge_mqtt.conf	MQTT Broker Bridge

11.2 Kafka Bridge

EMQ X bridges and forwards MQTT messages to Kafka cluster:

Config file for Kafka bridge plugin: etc/plugins/emqx_bridge_kafka.conf

11.2.1 Configure Kafka Cluster

```
## Kafka Server
## bridge.kafka.servers = 127.0.0.1:9092,127.0.0.2:9092,127.0.0.3:9092
bridge.kafka.servers = 127.0.0.1:9092

## Kafka Parition Strategy. option value: per_partition | per_broker
bridge.kafka.connection_strategy = per_partition

bridge.kafka.min_metadata_refresh_interval = 5S

## Produce writes type. option value: sync | async
bridge.kafka.produce = sync

bridge.kafka.produce.sync_timeout = 3S

## Base directory for replayq to store messages on disk.
## If this config entry if missing or set to undefined,
## replayq works in a mem-only manner.
## i.e. messages are not queued on disk -- in such case,
## the send or send_sync API callers are responsible for
## possible message loss in case of application,
## network or kafka disturbances. For instance,
## in the wolff:send API caller may trap_exit then
## react on parition-producer worker pid's 'EXIT'
## message to issue a retry after restarting the producer.
## bridge.kafka.replayq_dir = /tmp/emqx_bridge_kafka/

## default=10MB, replayq segment size.
## bridge.kafka.producer.replayq_seg_bytes = 10MB

## producer required_acks. option value all_isr | leader_only | none.
bridge.kafka.producer.required_acks = none

## default=10000. Timeout leader wait for replicas before reply to producer.
## bridge.kafka.producer.ack_timeout = 10S

## default number of message sets sent on wire before block waiting for acks
## bridge.kafka.producer.max_batch_bytes = 1024KB

## by default, send max 1 MB of data in one batch (message set)
## bridge.kafka.producer.min_batch_bytes = 0

## Number of batches to be sent ahead without receiving ack for the last␣
↪request.
## Must be 0 if messages must be delivered in strict order.
## bridge.kafka.producer.max_send_ahead = 0

## by default, no compression
# bridge.kafka.producer.compression = no_compression

# bridge.kafka.encode_payload_type = base64
```

()

()

```
# bridge.kafka.sock.buffer = 32KB
# bridge.kafka.sock.recbuf = 32KB
bridge.kafka.sock.sndbuf = 1MB
# bridge.kafka.sock.read_packets = 20
```

11.2.2 Configure Kafka Bridge Hooks

```
## Bridge Kafka Hooks
## ${topic}: the kafka topics to which the messages will be published.
## ${filter}: the mqtt topic (may contain wildcard) on which the action will␣
↪be performed .

bridge.kafka.hook.client.connected.1     = {"topic": "client_connected"}
bridge.kafka.hook.client.disconnected.1  = {"topic": "client_disconnected"}
bridge.kafka.hook.session.subscribed.1   = {"filter": "#",  "topic": "session_
↪subscribed"}
bridge.kafka.hook.session.unsubscribed.1 = {"filter": "#",  "topic": "session_
↪unsubscribed"}
bridge.kafka.hook.message.publish.1      = {"filter": "#",  "topic": "message_
↪publish"}
bridge.kafka.hook.message.delivered.1    = {"filter": "#",  "topic": "message_
↪delivered"}
bridge.kafka.hook.message.acked.1        = {"filter": "#",  "topic": "message_
↪acked"}
```

11.2.3 Description of Kafka Bridge Hooks

Event	Description
bridge.kafka.hook.client.connected.1	Client connected
bridge.kafka.hook.client.disconnected.1	Client disconnected
bridge.kafka.hook.session.subscribed.1	Topics subscribed
bridge.kafka.hook.session.unsubscribed.1	Topics unsubscribed
bridge.kafka.hook.message.publish.1	Messages published
bridge.kafka.hook.message.delivered.1	Messages delivered
bridge.kafka.hook.message.acked.1	Messages acknowledged

11.2.4 Forward Client Connected / Disconnected Events to Kafka

Client goes online, EMQ X forwards 'client_connected' event message to Kafka:

```
topic = "client_connected",
value = {
          "client_id": ${clientid},
```

()

()

```
          "node": ${node},
          "ts": ${ts}
        }
```

Client goes offline, EMQ X forwards 'client_disconnected' event message to Kafka:

```
topic = "client_disconnected",
value = {
        "client_id": ${clientid},
        "reason": ${reason},
        "node": ${node},
        "ts": ${ts}
        }
```

11.2.5 Forward Subscription Event to Kafka

```
topic = session_subscribed

value = {
          "client_id": ${clientid},
          "topic": ${topic},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.2.6 Forward Unsubscription Event to Kafka

```
topic = session_unsubscribed

value = {
          "client_id": ${clientid},
          "topic": ${topic},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.2.7 Forward MQTT Messages to Kafka

```
topic = message_publish

value = {
          "client_id": ${clientid},
          "username": ${username},
          "topic": ${topic},
```

()

()

```
          "payload": ${payload},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.2.8 Forwarding MQTT Message Deliver Event to Kafka

```
topic = message_delivered

value = {"client_id": ${clientid},
          "username": ${username},
          "from": ${fromClientId},
          "topic": ${topic},
          "payload": ${payload},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.2.9 Forwarding MQTT Message Ack Event to Kafka

```
topic = message_acked

value = {
          "client_id": ${clientid},
          "username": ${username},
          "from": ${fromClientId},
          "topic": ${topic},
          "payload": ${payload},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.2.10 Examples of Kafka Message Consumption

Kafka consumes MQTT clients connected / disconnected event messages:

```
sh kafka-console-consumer.sh --zookeeper localhost:2181 --topic client_
↪connected --from-beginning

sh kafka-console-consumer.sh --zookeeper localhost:2181 --topic client_
↪disconnected --from-beginning
```

Kafka consumes MQTT subscription messages:

```
sh kafka-console-consumer.sh --zookeeper localhost:2181 --topic session_
↪subscribed --from-beginning

sh kafka-console-consumer.sh --zookeeper localhost:2181 --topic session_
↪unsubscribed --from-beginning
```

Kafka consumes MQTT published messages:

```
sh kafka-console-consumer.sh --zookeeper localhost:2181 --topic message_
↪publish --from-beginning
```

Kafka consumes MQTT message Deliver and Ack event messages:

```
sh kafka-console-consumer.sh --zookeeper localhost:2181 --topic message_
↪delivered --from-beginning

sh kafka-console-consumer.sh --zookeeper localhost:2181 --topic message_acked␣
↪--from-beginning
```

: the payload is base64 encoded

11.2.11 Enable Kafka Bridge

```
./bin/emqx_ctl plugins load emqx_bridge_kafka
```

11.3 RabbitMQ Bridge

EMQ X bridges and forwards MQTT messages to RabbitMQ cluster:

Config file of RabbitMQ bridge plugin: etc/plugins/emqx_bridge_rabbit.conf

11.3.1 Configure RabbitMQ Cluster

```
## Rabbit Brokers Server
bridge.rabbit.1.server = 127.0.0.1:5672

## Rabbit Brokers pool_size
bridge.rabbit.1.pool_size = 4

## Rabbit Brokers username
```

()

()

```
bridge.rabbit.1.username = guest

## Rabbit Brokers password
bridge.rabbit.1.password = guest

## Rabbit Brokers virtual_host
bridge.rabbit.1.virtual_host = /

## Rabbit Brokers heartbeat
bridge.rabbit.1.heartbeat = 30

# bridge.rabbit.2.server = 127.0.0.1:5672

# bridge.rabbit.2.pool_size = 8

# bridge.rabbit.2.username = guest

# bridge.rabbit.2.password = guest

# bridge.rabbit.2.virtual_host = /

# bridge.rabbit.2.heartbeat = 30
```

11.3.2 Configure RabbitMQ Bridge Hooks

```
## Bridge Hooks
bridge.rabbit.hook.client.subscribe.1 = {"action": "on_client_subscribe",
 ↪"rabbit": 1, "exchange": "direct:emq.subscription"}

bridge.rabbit.hook.client.unsubscribe.1 = {"action": "on_client_unsubscribe",
 ↪"rabbit": 1, "exchange": "direct:emq.unsubscription"}

bridge.rabbit.hook.message.publish.1 = {"topic": "$SYS/#", "action": "on_
 ↪message_publish", "rabbit": 1, "exchange": "topic:emq.$sys"}

bridge.rabbit.hook.message.publish.2 = {"topic": "#", "action": "on_message_
 ↪publish", "rabbit": 1, "exchange": "topic:emq.pub"}

bridge.rabbit.hook.message.acked.1 = {"topic": "#", "action": "on_message_
 ↪acked", "rabbit": 1, "exchange": "topic:emq.acked"}
```

11.3.3 Forward Subscription Event to RabbitMQ

```
routing_key = subscribe
exchange = emq.subscription
headers = [{<<"x-emq-client-id">>, binary, ClientId}]
payload = jsx:encode([{Topic, proplists:get_value(qos, Opts)} || {Topic, Opts}
 ↪ <- TopicTable])
```

11.3.4 Forward Unsubscription Event to RabbitMQ

```
routing_key = unsubscribe
exchange = emq.unsubscription
headers = [{<<"x-emq-client-id">>, binary, ClientId}]
payload = jsx:encode([Topic || {Topic, _Opts} <- TopicTable]),
```

11.3.5 Forward MQTT Messages to RabbitMQ

```
routing_key = binary:replace(binary:replace(Topic, <<"/">>, <<".">>,␣
↪[global]),<<"+">>, <<"*">>, [global])
exchange = emq.$sys | emq.pub
headers = [{<<"x-emq-publish-qos">>, byte, Qos},
           {<<"x-emq-client-id">>, binary, pub_from(From)},
           {<<"x-emq-publish-msgid">>, binary, emqx_base62:encode(Id)}]
payload = Payload
```

11.3.6 Forward MQTT Message Ack Event to RabbitMQ

```
routing_key = puback
exchange = emq.acked
headers = [{<<"x-emq-msg-acked">>, binary, ClientId}],
payload = emqx_base62:encode(Id)
```

11.3.7 Example of RabbitMQ Subscription Message Consumption

Sample code of Rabbit message Consumption in Python:

```
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost
↪'))
channel = connection.channel()

channel.exchange_declare(exchange='direct:emq.subscription', exchange_type=
↪'direct')

result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue

channel.queue_bind(exchange='direct:emq.subscription', queue=queue_name,␣
↪routing_key= 'subscribe')

def callback(ch, method, properties, body):
    print(" [x] %r:%r" % (method.routing_key, body))
```

()

()

```
channel.basic_consume(callback, queue=queue_name, no_ack=True)

channel.start_consuming()
```

Sample of RabbitMQ client coding in other programming languages:

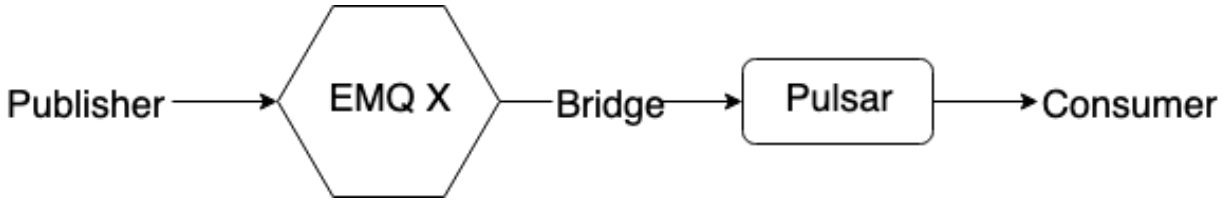
```
https://github.com/rabbitmq/rabbitmq-tutorials
```

11.3.8 Enable RabbitMQ Bridge

```
./bin/emqx_ctl plugins load emqx_bridge_rabbit
```

11.4 Pulsar Bridge

EMQ X bridges and forwards MQTT messages to Pulsar cluster:



Config file for Pulsar bridge plugin: etc/plugins/emqx_bridge_pulsar.conf

11.4.1 Configure Pulsar Cluster

```
## Pulsar Server
bridge.pulsar.servers = 127.0.0.1:6650

## Pick a partition producer and sync/async
bridge.pulsar.produce = sync

## bridge.pulsar.produce.sync_timeout = 3s

## bridge.pulsar.producer.batch_size = 1000

## by default, no compression
## bridge.pulsar.producer.compression = no_compression

## bridge.pulsar.encode_payload_type = base64

## bridge.pulsar.sock.buffer = 32KB
## bridge.pulsar.sock.recbuf = 32KB
bridge.pulsar.sock.sndbuf = 1MB
## bridge.pulsar.sock.read_packets = 20
```

11.4.2 Configure Pulsar Bridge Hooks

```
## Bridge Pulsar Hooks
## ${topic}: the pulsar topics to which the messages will be published.
## ${filter}: the mqtt topic (may contain wildcard) on which the action will␣
↪be performed .

## Client Connected Record Hook
bridge.pulsar.hook.client.connected.1     = {"topic": "client_connected"}

## Client Disconnected Record Hook
bridge.pulsar.hook.client.disconnected.1  = {"topic": "client_disconnected"}

## Session Subscribed Record Hook
bridge.pulsar.hook.session.subscribed.1   = {"filter": "#",  "topic":
↪"session_subscribed"}

## Session Unsubscribed Record Hook
bridge.pulsar.hook.session.unsubscribed.1 = {"filter": "#",  "topic":
↪"session_unsubscribed"}

## Message Publish Record Hook
bridge.pulsar.hook.message.publish.1      = {"filter": "#",  "topic":
↪"message_publish"}

## Message Delivered Record Hook
bridge.pulsar.hook.message.delivered.1    = {"filter": "#",  "topic":
↪"message_delivered"}

## Message Acked Record Hook
bridge.pulsar.hook.message.acked.1        = {"filter": "#",  "topic":
↪"message_acked"}

## More Configures
## partitioner strategy:
## Option:  random | roundrobin | first_key_dispatch
## Example: bridge.pulsar.hook.message.publish.1 = {"filter":"#", "topic":
↪"message_publish", "strategy":"random"}

## key:
## Option: ${clientid} | ${username}
## Example: bridge.pulsar.hook.message.publish.1 = {"filter":"#", "topic":
↪"message_publish", "key":"${clientid}"}

## format:
## Option: json | json
## Example: bridge.pulsar.hook.message.publish.1 = {"filter":"#", "topic":
↪"message_publish", "format":"json"}
```

11.4.3 Description of Pulsar Bridge Hooks

Event	Description
bridge.pulsar.hook.client.connected.1	Client connected
bridge.pulsar.hook.client.disconnected.1	Client disconnected
bridge.pulsar.hook.session.subscribed.1	Topics subscribed
bridge.pulsar.hook.session.unsubscribed.1	Topics unsubscribed
bridge.pulsar.hook.message.publish.1	Messages published
bridge.pulsar.hook.message.delivered.1	Messages delivered
bridge.pulsar.hook.message.acked.1	Messages acknowledged

11.4.4 Forward Client Connected / Disconnected Events to Pulsar

Client goes online, EMQ X forwards 'client_connected' event message to Pulsar:

```
topic = "client_connected",
value = {
          "client_id": ${clientid},
          "username": ${username},
          "node": ${node},
          "ts": ${ts}
        }
```

Client goes offline, EMQ X forwards 'client_disconnected' event message to Pulsar:

```
topic = "client_disconnected",
value = {
          "client_id": ${clientid},
          "username": ${username},
          "reason": ${reason},
          "node": ${node},
          "ts": ${ts}
        }
```

11.4.5 Forward Subscription Event to Pulsar

```
topic = session_subscribed

value = {
          "client_id": ${clientid},
          "topic": ${topic},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.4.6 Forward Unsubscription Event to Pulsar

```
topic = session_unsubscribed

value = {
          "client_id": ${clientid},
          "topic": ${topic},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.4.7 Forward MQTT Messages to Pulsar

```
topic = message_publish

value = {
          "client_id": ${clientid},
          "username": ${username},
          "topic": ${topic},
          "payload": ${payload},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.4.8 Forwarding MQTT Message Deliver Event to Pulsar

```
topic = message_delivered

value = {"client_id": ${clientid},
          "username": ${username},
          "from": ${fromClientId},
          "topic": ${topic},
          "payload": ${payload},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.4.9 Forwarding MQTT Message Ack Event to Pulsar

```
topic = message_acked

value = {
          "client_id": ${clientid},
```

()

()

```
          "username": ${username},
          "from": ${fromClientId},
          "topic": ${topic},
          "payload": ${payload},
          "qos": ${qos},
          "node": ${node},
          "ts": ${timestamp}
        }
```

11.4.10 Examples of Pulsar Message Consumption

Pulsar consumes MQTT clients connected / disconnected event messages:

```
sh pulsar-client consume client_connected  -s "client_connected" -n 1000

sh pulsar-client consume client_disconnected  -s "client_disconnected" -n 1000
```

Pulsar consumes MQTT subscription messages:

```
sh pulsar-client consume session_subscribed  -s "session_subscribed" -n 1000

sh pulsar-client consume session_unsubscribed  -s "session_unsubscribed" -n␣
↪1000
```

Pulsar consumes MQTT published messages:

```
sh pulsar-client consume message_publish  -s "message_publish" -n 1000
```

Pulsar consumes MQTT message Deliver and Ack event messages:

```
sh pulsar-client consume message_delivered  -s "message_delivered" -n 1000

sh pulsar-client consume message_acked  -s "message_acked" -n 1000
```

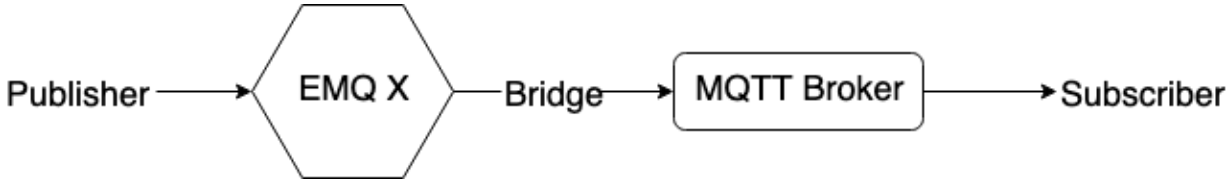
: the payload is base64 encoded default

11.4.11 Enable Pulsar Bridge

```
./bin/emqx_ctl plugins load emqx_bridge_pulsar
```

11.5 MQTT Bridge

EMQ X bridges and forwards MQTT messages to MQTT Broker:



Config file for MQTT bridge plugin: etc/plugins/emqx_bridge_mqtt.conf

11.5.1 Configure MQTT Bridge

```
## Bridge address: node name for local bridge, host:port for remote
bridge.mqtt.aws.address = 127.0.0.1:1883

## Protocol version of the bridge: mqttv3 | mqttv4 | mqttv5
bridge.mqtt.aws.proto_ver = mqttv4

## Whether to enable bridge mode for mqtt bridge
bridge.mqtt.aws.bridge_mode = true

## The ClientId of a remote bridge
bridge.mqtt.aws.client_id = bridge_aws

## The Clean start flag of a remote bridge
## NOTE: Some IoT platforms require clean_start must be set to 'true'
bridge.mqtt.aws.clean_start = true

## The username for a remote bridge
bridge.mqtt.aws.username = user

## The password for a remote bridge
bridge.mqtt.aws.password = passwd

## Bribge to remote server via SSL
bridge.mqtt.aws.ssl = off

## PEM-encoded CA certificates of the bridge
bridge.mqtt.aws.cacertfile = etc/certs/cacert.pem

## Client SSL Certfile of the bridge
bridge.mqtt.aws.certfile = etc/certs/client-cert.pem

## Client SSL Keyfile of the bridge
bridge.mqtt.aws.keyfile = etc/certs/client-key.pem

## SSL Ciphers used by the bridge
bridge.mqtt.aws.ciphers = ECDHE-ECDSA-AES256-GCM-SHA384,ECDHE-RSA-AES256-GCM-
↪SHA384

## Ciphers for TLS PSK
## Note that 'bridge.${BridgeName}.ciphers' and 'bridge.${BridgeName}.psk_
↪ciphers' cannot be configured at the same time.
```

()

()

```
##
## See 'https://tools.ietf.org/html/rfc4279#section-2'.
bridge.mqtt.aws.psk_ciphers = PSK-AES128-CBC-SHA,PSK-AES256-CBC-SHA,PSK-3DES-
↪EDE-CBC-SHA,PSK-RC4-SHA

## Ping interval of a down bridge.
bridge.mqtt.aws.keepalive = 60s

## TLS versions used by the bridge.
bridge.mqtt.aws.tls_versions = tlsv1.2,tlsv1.1,tlsv1
```

11.5.2 Configure Topics MQTT Bridge Forwards and Subscribes

```
## Mountpoint of the bridge
bridge.mqtt.aws.mountpoint = bridge/aws/${node}/

## Forward message topics
bridge.mqtt.aws.forwards = topic1/#,topic2/#

## Subscriptions of the bridge topic
bridge.mqtt.aws.subscription.1.topic = cmd/topic1

## Subscriptions of the bridge qos
bridge.mqtt.aws.subscription.1.qos = 1

## Subscriptions of the bridge topic
bridge.mqtt.aws.subscription.2.topic = cmd/topic2

## Subscriptions of the bridge qos
bridge.mqtt.aws.subscription.2.qos = 1
```

11.5.3 Description of Topics MQTT Bridge Forwards and Subscribes

Mountpoint: Mountpoint is used to prefix of topic when forwarding a message, this option must be used with `forwards`. Forwards the message whose topic is "sensor1/hello", its topic will change to "bridge/aws/emqx1@192.168.1.1/sensor1/hello" when it reaches the remote node.

Forwards: Messages forwarded to `forwards` specified by local EMQ X are forwarded to the remote MQTT Broker.

Subscription: Local EMQ X synchronizes messages from a remote MQTT Broker to local by subscribing to the topic of the remote MQTT Broker.

11.5.4 Enable MQTT Bridge

```
./bin/emqx_ctl plugins load emqx_bridge_mqtt
```

11.5.5 Bridge CLI Command

```
$ cd emqx && ./bin/emqx_ctl bridges
bridges list                                        # List bridges
bridges start <Name>                                # Start a bridge
bridges stop <Name>                                 # Stop a bridge
bridges forwards <Name>                             # Show a bridge forward topic
bridges add-forward <Name> <Topic>                  # Add bridge forward topic
bridges del-forward <Name> <Topic>                  # Delete bridge forward topic
bridges subscriptions <Name>                        # Show a bridge subscriptions␣
↪topic
bridges add-subscription <Name> <Topic> <Qos>    # Add bridge subscriptions␣
↪topic
```

11.5.6 List Status of All Bridges

```
$ ./bin/emqx_ctl bridges list
name: emqx     status: Stopped
```

11.5.7 Start Specified Bridge

```
$ ./bin/emqx_ctl bridges start emqx
Start bridge successfully.
```

11.5.8 Stop Specified Bridge

```
$ ./bin/emqx_ctl bridges stop emqx
Stop bridge successfully.
```

11.5.9 List Forwarded Topic of Specified Bridge

```
$ ./bin/emqx_ctl bridges forwards emqx
topic:   topic1/#
topic:   topic2/#
```

11.5.10 Add Forwarded Topic for Specified Bridge

```
$ ./bin/emqx_ctl bridges add-forwards emqx topic3/#
Add-forward topic successfully.
```

11.5.11 Delete Forwarded Topic for Specified Bridge

```
$ ./bin/emqx_ctl bridges del-forwards emqx topic3/#
Del-forward topic successfully.
```

11.5.12 List Subscriptions of Specified Bridge

```
$ ./bin/emqx_ctl bridges subscriptions emqx
topic: cmd/topic1, qos: 1
topic: cmd/topic2, qos: 1
```

11.5.13 Add Subscriptions for Specified Bridge

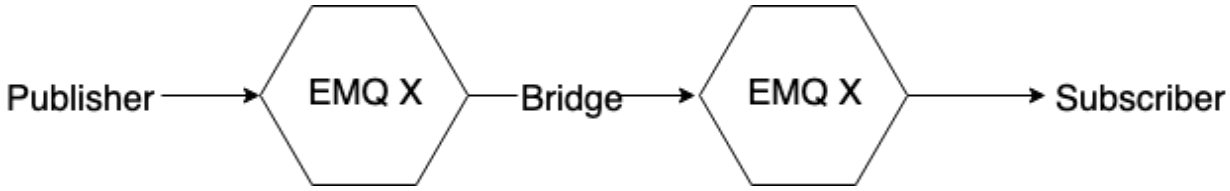
```
$ ./bin/emqx_ctl bridges add-subscription emqx cmd/topic3 1
Add-subscription topic successfully.
```

11.5.14 Delete Subscriptions of Specified Bridge

```
$ ./bin/emqx_ctl bridges del-subscription emqx cmd/topic3
Del-subscription topic successfully.
```

11.6 RPC Bridge

EMQ X bridges and forwards MQTT messages to remote EMQ X:



Config file for RPC bridge plugin: etc/plugins/emqx_bridge_mqtt.conf

11.6.1 Configure Broker Address for RPC Bridge

```
bridge.mqtt.emqx.address = emqx2@192.168.1.2
```

11.6.2 Configure Topics RPC Bridge Forwards and Subscribes

```
## Mountpoint of the bridge
bridge.mqtt.emqx.mountpoint = bridge/emqx1/${node}/

## Forward message topics
bridge.mqtt.emqx.forwards = topic1/#,topic2/#
```

Mountpoint: Mountpoint is used to prefix of topic when forwarding a message, this option must be used with `forwards`. Forwards the message whose topic is "sensor1/hello", its topic will change to "bridge/aws/emqx1@192.168.1.1/sensor1/hello" when it reaches the remote node.

Forwards: Messages forwarded to `forwards` specified by local EMQ X are forwarded to the remote EMQ X.

11.6.3 Bridge CLI Command

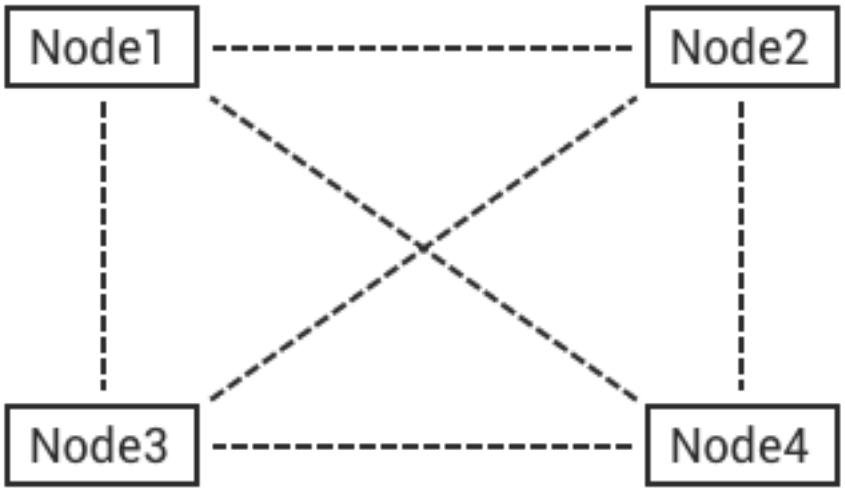
CLI of RPC bridge is used in the same way as the MQTT bridge.

CHAPTER 12

Clustering

12.1 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.



12.1.1 Node

An erlang runtime system called 'node' is identified by a unique name like email address. 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']
```

12.1.2 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}]}
```

12.1.3 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>
```

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

12.1.4 Distribution Protocol

Erlang nodes can be connected via different distributed protocols including TCPv4, TCPv6 and TLS.

```
## Specify the erlang distributed protocol.
##
## Value: Enum
##  - inet_tcp: the default; handles TCP streams with IPv4 addressing.
##  - inet6_tcp: handles TCP with IPv6 addressing.
##  - inet_tls: using TLS for Erlang Distribution.
##
## vm.args: -proto_dist inet_tcp
node.proto_dist = inet_tcp

## Specify SSL Options in the file if using SSL for Erlang Distribution.
##
## Value: File
##
## vm.args: -ssl_dist_optfile <File>
## node.ssl_dist_optfile = {{ platform_etc_dir }}/ssl_dist.conf
```

12.2 Cluster Design

The cluster architecture of *EMQ X* broker is based on distributed 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 have a client subscribed to 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 to the Topic.

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

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

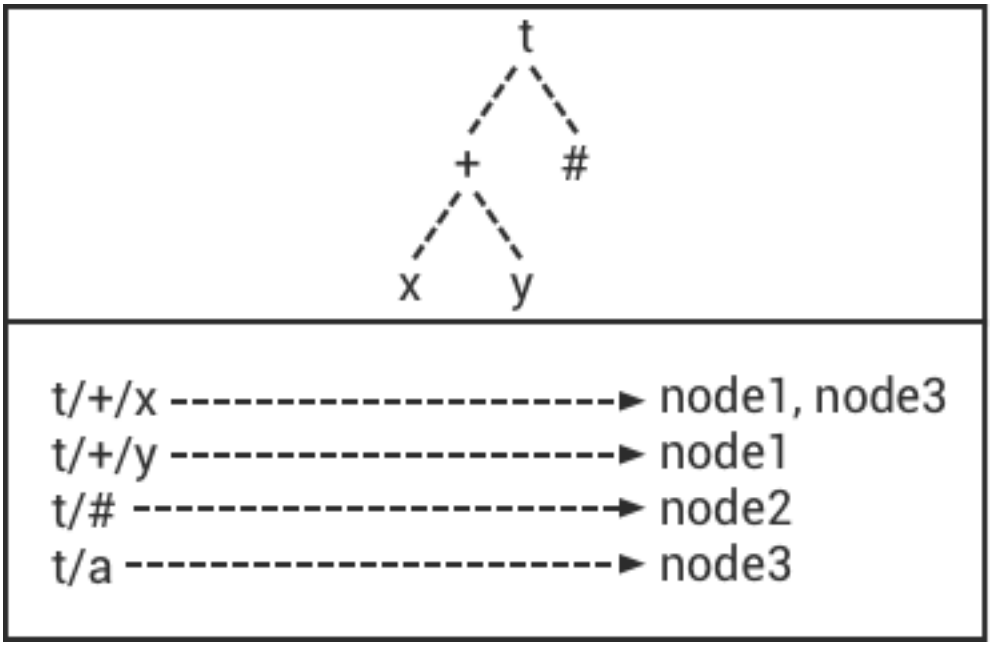
12.2.1 Topic Trie and Route Table

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

Suppose that we create following subscriptions:

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

The topic trie and route table in the cluster will be:

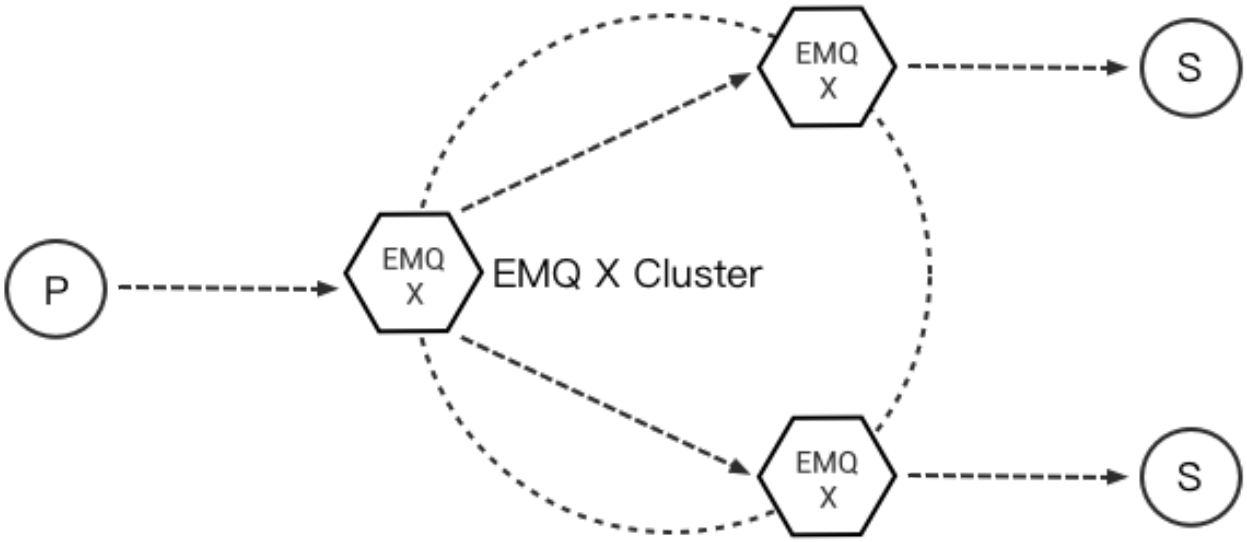


12.2.2 Message Route and Deliver

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]
```



12.3 Cluster Setup

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

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

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

12.3.1 emqx@s1.emqx.io config

etc/emqx.conf:

```
node.name = emqx@s1.emqx.io

or

node.name = emqx@192.168.0.10
```

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

12.3.2 emqx@s2.emqx.io config

etc/emqx.conf:

```
node.name = emqx@s2.emqx.io

or

node.name = emqx@192.168.0.20
```

12.3.3 Join the cluster

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

```
$ ./bin/emqx_ctl cluster join emqx@s1.emqx.io

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

Or 'cluster join' on emqx@s1.emqx.io:

```
$ ./bin/emqx_ctl cluster join emqx@s2.emqx.io

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

Query the cluster status:

```
$ ./bin/emqx_ctl cluster status

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

12.3.4 Leave the cluster

Two ways to leave the cluster:

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

emqx@s2.emqx.io node tries to leave the cluster:

```
$ ./bin/emqx_ctl cluster leave
```

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

```
$ ./bin/emqx_ctl cluster remove emqx@s2.emqx.io
```

12.4 Node Discovery and Autocluster

EMQ X 3.0 supports node discovery and autocluster with various strategies:

Strategy	Description
static	Autocluster by static node list
mcast	Autocluster by UDP Multicast
dns	Autocluster by DNS A Record
etcd	Autocluster using etcd
k8s	Autocluster on Kubernetes

12.4.1 Autocluster by static node list

```
cluster.discovery = static

##--------------------------------------------------------------------
## Cluster with static node list

cluster.static.seeds = emq1@127.0.0.1,ekka2@127.0.0.1
```

12.4.2 Autocluster by IP Multicast

```
cluster.discovery = mcast

##--------------------------------------------------------------------
## Cluster with multicast

cluster.mcast.addr = 239.192.0.1

cluster.mcast.ports = 4369,4370

cluster.mcast.iface = 0.0.0.0

cluster.mcast.ttl = 255

cluster.mcast.loop = on
```

12.4.3 Autocluster by DNS A Record

```
cluster.discovery = dns

##--------------------------------------------------------------------
## Cluster with DNS

cluster.dns.name = localhost

cluster.dns.app  = ekka
```

12.4.4 Autocluster using etcd

```
cluster.discovery = etcd

##--------------------------------------------------------------------
## Cluster with Etcd

cluster.etcd.server = http://127.0.0.1:2379

cluster.etcd.prefix = emqcl

cluster.etcd.node_ttl = 1m
```

12.4.5 Autocluster on Kubernetes

```
cluster.discovery = k8s

##--------------------------------------------------------------------
## Cluster with k8s

cluster.k8s.apiserver = http://10.110.111.204:8080

cluster.k8s.service_name = ekka

## Address Type: ip | dns
cluster.k8s.address_type = ip

## The Erlang application name
cluster.k8s.app_name = ekka
```

12.5 Network Partition and Autoheal

Enable autoheal of Network Partition:

```
cluster.autoheal = on
```

When network partition occurs, the following steps are performed to heal the cluster if autoheal is enabled:

1. Node reports the partitions to a leader node which has the oldest guid.
2. Leader node create a global netsplit view and choose one node in the majority as coordinator.
3. Leader node requests the coordinator to autoheal the network partition.
4. Coordinator node reboots all the nodes in the minority side.

12.6 Node down and Autoclean

A down node will be removed from the cluster if autoclean is enabled:

```
cluster.autoclean = 5m
```

12.7 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:

_static/images/10_3.png

12.8 The Firewall

If the nodes need to go through a Firewall, TCP port *4369* must be allowed for *epmd*, as well as a sequential range of TCP ports for communication between the distributed nodes.

That range of ports for erlang distribution is configured in *etc/emqx.conf*, defaults to *6369-7369*:

```
## Distributed node port range
node.dist_listen_min = 6369
node.dist_listen_max = 7369
...
```

So by default, make sure TCP ports *4369* and *6369-7369* are allowed by your Firewall roles.

12.9 Consistent Hash and DHT

Consistent Hash and DHT are popular in the design of NoSQL databases. Cluster of *EMQ X* 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.

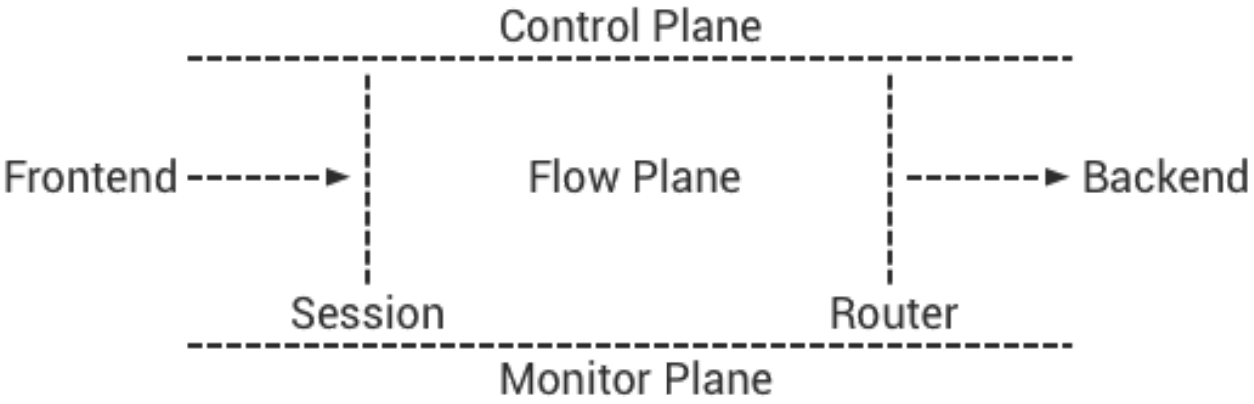
CHAPTER 13

Design

13.1 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:



13.1.1 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.

13.1.2 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.

13.2 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
emqx_connection	TCP Client
emqx_ws_connection	WebSocket Client
emqx_protocol	MQTT Protocol Handler
emqx_frame	MQTT Frame Parser

13.3 Session Layer

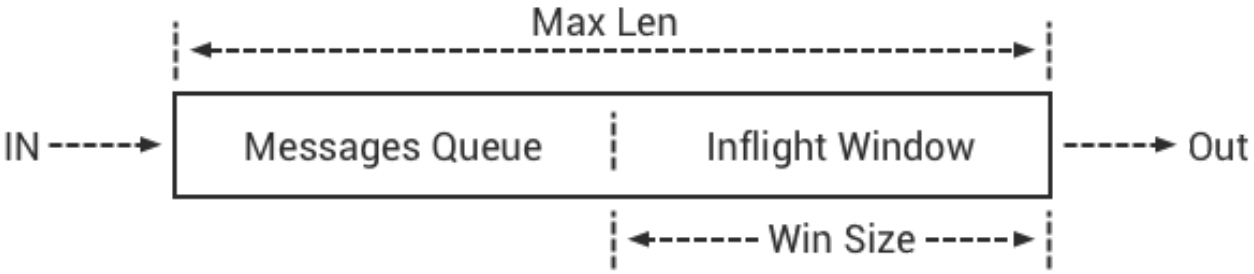
The session layer processes MQTT packets received from client and delivers PUBLISH packets to client.

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.

13.3.1 MQueue and Inflight Window

Concept of Message Queue and Inflight Window:



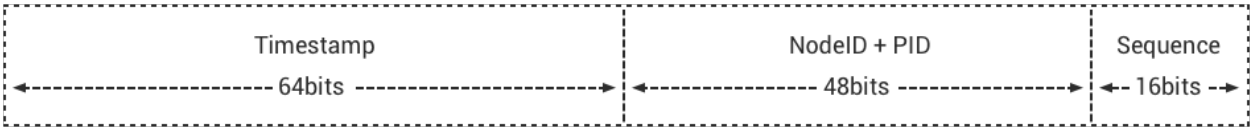
1. Inflight Window to store the messages delivered and await for PUBACK.
2. Enqueue messages when the inflight window is full.
3. If the queue is full, drop qos0 messages if store_qos0 is true, otherwise drop the oldest one.

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.

13.3.2 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:

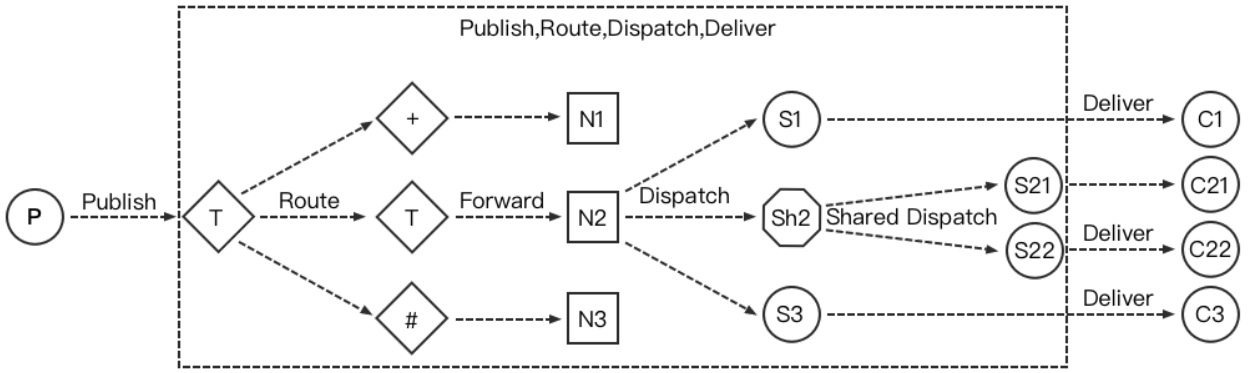


1. Timestamp: erlang:system_time if Erlang >= R18, otherwise os:timestamp
2. NodeId: encode node() to 2 bytes integer
3. Pid: encode pid to 4 bytes integer
4. Sequence: 2 bytes sequence in one process

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

13.4 PubSub Layer

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

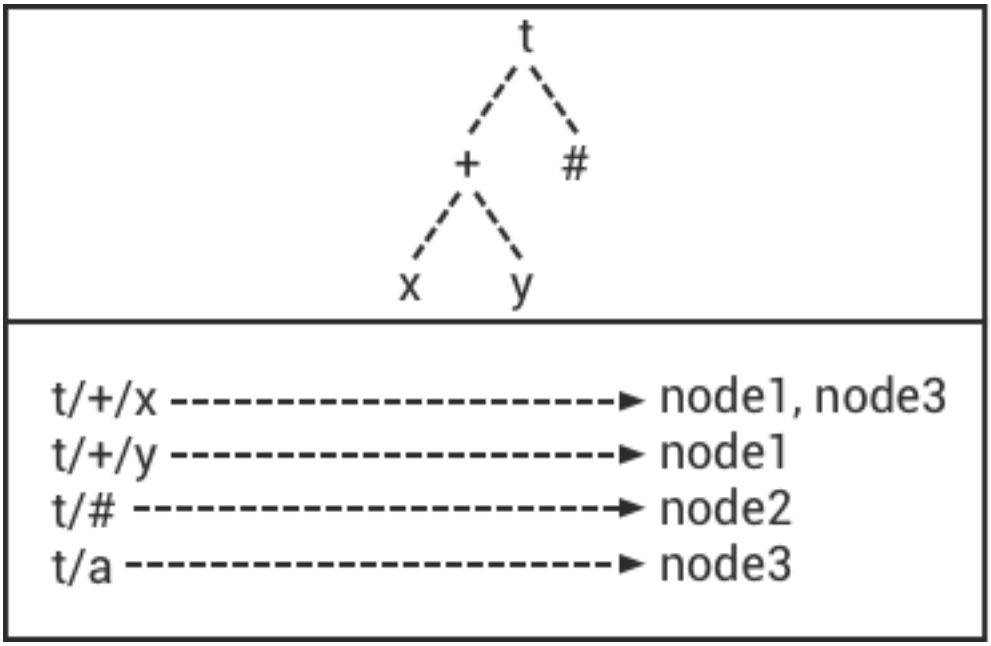


MQTT messages will be dispatched to the subscriber's session, which finally delivers the messages to client.

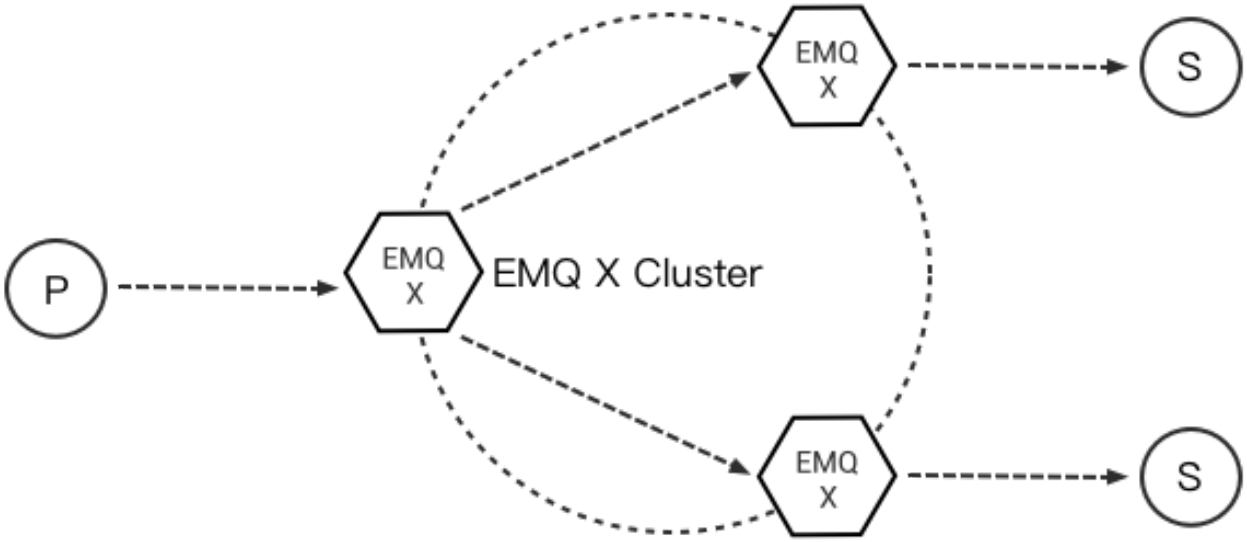
13.5 Routing Layer

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:



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.

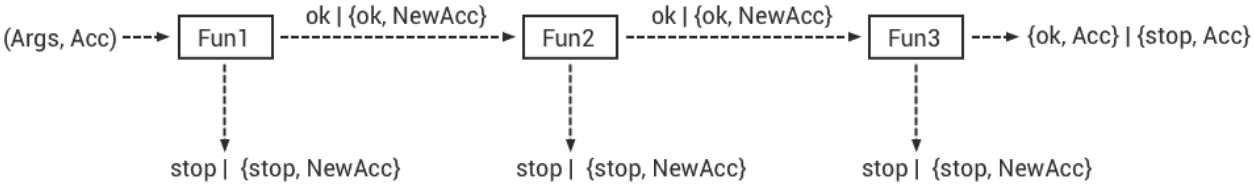
13.6 Hooks Design

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* 3.0 broker:

Hook	Description
client.authenticate	Run when client is trying to connect to the broker
client.check_acl	Run when client is trying to publish or subscribe to a topic
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.deliver	Run when a MQTT message is delivering to target client
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:



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 depends on the types of hook. Checkout the emq_plugin_template project to see the hook examples in detail.

13.6.1 Hook Implementation

The hook APIs are defined in the `emqx` module:

```
-spec(hook(emqx_hooks:hookpoint(), emqx_hooks:action()) -> ok | {error,␣
↪already_exists}).
hook(HookPoint, Action) ->
    emqx_hooks:add(HookPoint, Action).

-spec(hook(emqx_hooks:hookpoint(), emqx_hooks:action(), emqx_hooks:filter() |␣
↪integer())
    -> ok | {error, already_exists}).
hook(HookPoint, Action, Priority) when is_integer(Priority) ->
    emqx_hooks:add(HookPoint, Action, Priority);
hook(HookPoint, Action, Filter) when is_function(Filter); is_tuple(Filter) ->
    emqx_hooks:add(HookPoint, Action, Filter);
hook(HookPoint, Action, InitArgs) when is_list(InitArgs) ->
    emqx_hooks:add(HookPoint, Action, InitArgs).

-spec(hook(emqx_hooks:hookpoint(), emqx_hooks:action(), emqx_hooks:filter(),␣
↪integer())
    -> ok | {error, already_exists}).
hook(HookPoint, Action, Filter, Priority) ->
    emqx_hooks:add(HookPoint, Action, Filter, Priority).

-spec(unhook(emqx_hooks:hookpoint(), emqx_hooks:action()) -> ok).
unhook(HookPoint, Action) ->
    emqx_hooks:del(HookPoint, Action).

-spec(run_hook(emqx_hooks:hookpoint(), list(any())) -> ok | stop).
run_hook(HookPoint, Args) ->
    emqx_hooks:run(HookPoint, Args).

-spec(run_fold_hook(emqx_hooks:hookpoint(), list(any()), any()) -> any()).
run_fold_hook(HookPoint, Args, Acc) ->
    emqx_hooks:run_fold(HookPoint, Args, Acc).
```

13.6.2 Hook Usage

The emq_plugin_template project provides the examples for hook usage:

```
-module(emqx_plugin_template).

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

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

load(Env) ->
    emqx:hook('message.publish', fun ?MODULE:on_message_publish/2, [Env]),
    emqx:hook('message.deliver', fun ?MODULE:on_message_deliver/3, [Env]),
    emqx:hook('message.acked', fun ?MODULE:on_message_acked/3, [Env]).

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

()

()

```
on_message_deliver(Credentials, Message, _Env) ->
    io:format("deliver to client ~s: ~s~n", [Credentials, emqx_
↪message:format(Message)]),
    {ok, Message}.

on_message_acked(Credentials, Message, _Env) ->
    io:format("client ~s acked: ~s~n", [Credentials, emqx_
↪message:format(Message)]),
    {ok, Message}.

unload() ->
    emqx:unhook('message.publish', fun ?MODULE:on_message_publish/2),
    emqx:unhook('message.acked', fun ?MODULE:on_message_acked/3),
    emqx:unhook('message.deliver', fun ?MODULE:on_message_deliver/3).
```

13.7 Authentication and ACL

The *EMQ* broker supports extensible Authentication/ACL by hooking to hook-points `client.authenticate` and `client.check_acl`:

13.7.1 Write Authentication Hook CallBacks

To register a callback function to `client.authenticate`:

```
emqx:hook('client.authenticate', fun ?MODULE:on_client_authenticate/1, []).
```

The callbacks must have an argument that receives the `Credentials`, and returns an updated Credentials:

```
on_client_authenticate(Credentials = #{password := Password}) ->
    {ok, Credentials#{result => success}}.
```

The `Credentials` is a map that contains AUTH related info:

```
#{
  client_id => ClientId,      %% The client id
  username  => Username,      %% The username
  peername  => Peername,      %% The peer IP Address and Port
  password  => Password,      %% The password (Optional)
  result    => Result         %% The authentication result, must be set to␣
↪``success`` if OK,
                              %% or ``bad_username_or_password`` or ``not_
↪authorized`` if failed.
}
```

13.7.2 Write ACL Hook Callbacks

To register a callback function to `client.authenticate`:

```
emqx:hook('client.check_acl', fun ?MODULE:on_client_check_acl/4, []).
```

The callbacks must have arguments that receives the `Credentials`, `AccessType`, `Topic`, `ACLResult`, and then returns a new ACLResult:

```
on_client_check_acl(#{client_id := ClientId}, AccessType, Topic, ACLResult) ->
    {ok, allow}.
```

AccessType can be one of `publish` and `subscribe`. Topic is the MQTT topic. The ACLResult is either `allow` or `deny`.

The module `emqx_mod_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}.
```

The Authentication/ACL plugins implemented by emqx organization:

Plugin	Authentication
emq_auth_username	Username and Password
emq_auth_clientid	ClientID and Password
emq_auth_ldap	LDAP
emq_auth_http	HTTP API
emq_auth_mysql	MySQL
emq_auth_pgsql	PostgreSQL
emq_auth_redis	Redis
emq_auth_mongo	MongoDB
emq_auth_jwt	JWT

13.8 Plugin Design

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

13.8.1 emqx_plugins Module

The plugin mechanism is implemented by emqx_plugins module:

```
-module(emqx_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()}.
```

13.8.2 Load a Plugin

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

```
./bin/emqx_ctl plugins load emq_auth_redis

./bin/emqx_ctl plugins unload emq_auth_redis
```

13.8.3 Plugin Template

http://github.com/emqx/emq_plugin_template

13.9 Mnesia/ETS Tables

Table	Type	Description
emqx_conn	ets	Connection Table
emqx_metrics	ets	Metrics Table
emqx_session	ets	Session Table
emqx_hooks	ets	Hooks Table
emqx_subscriber	ets	Subscriber Table
emqx_subscription	ets	Subscription Table
emqx_admin	mnesia	The Dashboard admin users Table
emqx_retainer	mnesia	Retained Message Table
emqx_shared_subscription	mnesia	Shared Subscription Table
emqx_session_registry	mnesia	Global Session Registry Table
emqx_alarm_history	mnesia	Alarms History
emqx_alarm	mnesia	Alarms
emqx_banned	mnesia	Built-In Banned Table
emqx_route	mnesia	Global Route Table
emqx_trie	mnesia	Trie Table
emqx_trie_node	mnesia	Trie Node Table
mqtt_app	mnesia	App table

CHAPTER 14

Commands

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

14.1 status

Show running status of the broker:

```
$ ./bin/emqx_ctl status

Node 'emqx@127.0.0.1' is started
emqx v3.1.0 is running
```

14.2 mgmt

Manages the `apps` of the broker.

mgmt list	List the Apps
mgmt insert <AppId> <Name>	Add App for REST API
mgmt update <AppId> <status>	Update App for REST API
mgmt lookup <AppId>	Query App details of an AppId
mgmt delete <AppId>	Delete Apps of an AppId

14.2.1 mgmt list

List the `apps`:

```
$ ./bin/emqx_ctl mgmt list
app_id: 901abdba8eb8c, secret:␣
↪MjgzMzQ5MjM1MzUzMTc4MjgyMjE3NzU4ODcwMDg0NjQ4OTG, name: hello, desc: ,␣
↪status: true, expired: undefined
```

14.2.2 mgmt insert <AppId> <Name>

Add an app for REST API:

```
$ ./bin/emqx_ctl mgmt insert dbcb6e023370b world
AppSecret: MjgzMzQ5MjYyMTY3ODk4MjA5NzMwODExODMxMDM1NDk0NDA
```

14.2.3 mgmt update <AppId> <status>

mgmt update <AppId> <status>:

```
$ ./bin/emqx_ctl mgmt update dbcb6e023370b stop
update successfully.
```

14.2.4 mgmt lookup <AppId>

Query details of an app:

```
$ ./bin/emqx_ctl mgmt lookup dbcb6e023370b
app_id: dbcb6e023370b
secret: MjgzMzQ5MjYyMTY3ODk4MjA5NzMwODExODMxMDM1NDk0NDA
name: world
desc: Application user
status: stop
expired: undefined
```

14.2.5 mgmt delete <AppId>

Delete an app:

```
$ ./bin/emqx_ctl mgmt delete dbcb6e023370b
ok
```

14.3 broker

Query basic information, statistics and metrics of the broker.

broker	Show version, description, uptime of the broker
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/emqx_ctl broker

sysdescr  : EMQ X Broker
version   : v3.1.0
uptime    : 25 seconds
datetime  : 2019-04-29 10:42:10
```

14.3.1 broker stats

Query statistics of MQTT Connections, Sessions, Topics, Subscriptions and Routes:

```
$ ./bin/emqx_ctl broker stats

actions/max                   : 2
connections/count             : 1
connections/max               : 1
resources/max                 : 0
retained/count                : 2
retained/max                  : 2
routes/count                  : 0
routes/max                    : 0
rules/max                     : 0
sessions/count                : 0
sessions/max                  : 0
sessions/persistent/count     : 0
sessions/persistent/max       : 0
suboptions/max                : 0
subscribers/count             : 0
subscribers/max               : 1
subscriptions/count           : 1
subscriptions/max             : 0
subscriptions/shared/count    : 0
subscriptions/shared/max      : 0
topics/count                  : 0
topics/max                    : 0
```

14.3.2 broker metrics

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

```
$ ./bin/emqx_ctl broker metrics

```

()

()

```
bytes/received          : 0
bytes/sent              : 0
messages/dropped        : 0
messages/expired        : 0
messages/forward        : 0
messages/qos0/received  : 0
messages/qos0/sent      : 0
messages/qos1/received  : 0
messages/qos1/sent      : 0
messages/qos2/dropped   : 0
messages/qos2/expired   : 0
messages/qos2/received  : 0
messages/qos2/sent      : 0
messages/received       : 0
messages/retained       : 3
messages/sent           : 0
packets/auth            : 0
packets/connack         : 0
packets/connect         : 0
packets/disconnect/recei: 0
packets/disconnect/sent : 0
packets/pingreq         : 0
packets/pingresp        : 0
packets/puback/missed   : 0
packets/puback/received : 0
packets/puback/sent     : 0
packets/pubcomp/missed  : 0
packets/pubcomp/received: 0
packets/pubcomp/sent    : 0
packets/publish/received: 0
packets/publish/sent    : 0
packets/pubrec/missed   : 0
packets/pubrec/received : 0
packets/pubrec/sent     : 0
packets/pubrel/missed   : 0
packets/pubrel/received : 0
packets/pubrel/sent     : 0
packets/received        : 0
packets/sent            : 0
packets/suback          : 0
packets/subscribe       : 0
packets/unsuback        : 0
packets/unsubscribe     : 0
```

14.4 cluster

Cluster two or more *EMQ X* brokers:

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

Suppose we have two *EMQ X* nodes on localhost and we want to cluster them together:

Folder	Node	MQTT Port
emqx1	emqx1@127.0.0.1	1883
emqx2	emqx2@127.0.0.1	2883

Start emqx1 and emqx2:

```
$ cd emqx1 && ./bin/emqx start

$ cd emqx2 && ./bin/emqx start
```

Under emqx2 folder:

```
$ ./bin/emqx_ctl cluster join emqx1@127.0.0.1

Join the cluster successfully.
Cluster status: [{running_nodes,['emqx1@127.0.0.1','emqx2@127.0.0.1']}]
```

Query cluster status:

```
$ ./bin/emqx_ctl cluster status

Cluster status: [{running_nodes,['emqx2@127.0.0.1','emqx1@127.0.0.1']}]
```

Message route between nodes:

```
# Subscribe topic 'x' on emqx1 node
$ mosquitto_sub -t x -q 1 -p 1883

# Publish to topic 'x' on emqx2 node
$ mosquitto_pub -t x -q 1 -p 2883 -m hello
```

emqx2 leaves the cluster:

```
$ cd emqx2 && ./bin/emqx_ctl cluster leave
```

Or remove emqx2 from the cluster on emqx1 node:

```
$ cd emqx1 && ./bin/emqx_ctl cluster force-leave emqx2@127.0.0.1
```

14.5 acl

reload acl.conf:

```
$ ./bin/emqx_ctl acl reload
```

14.6 clients

Query MQTT clients connected to the broker:

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

14.6.1 clients list

Query all MQTT clients connected to the broker:

```
$ ./bin/emqx_ctl clients list

Connection(mosqsub/43832-airlee.lo, clean_start=true, username=test,␣
↪peername=127.0.0.1:64896, connected_at=1452929113)
Connection(mosqsub/44011-airlee.lo, clean_start=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

14.6.2 clients show <ClientId>

Show a specific MQTT Client:

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

Connection(mosqsub/43832-airlee.lo, clean_sess=true, username=test,␣
↪peername=127.0.0.1:64896, connected_at=1452929113)
```

14.6.3 clients kick <ClientId>

Kick out a MQTT Client:

```
$ ./bin/emqx_ctl clients kick "clientid"
```

14.7 sessions

Query all MQTT sessions. The broker will create a session for each MQTT client.

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

14.7.1 sessions list

Query all sessions:

```
$ ./bin/emqx_ctl sessions list

Session(clientid, clean_start=true, expiry_interval=0, subscriptions_count=0,␣
↪max_inflight=32, inflight=0, mqueue_len=0, mqueue_dropped=0, awaiting_rel=0,
↪ deliver_msg=0, enqueue_msg=0, created_at=1553760799)
Session(mosqsub/44101-airlee.lo, clean_start=true, expiry_interval=0,␣
↪subscriptions_count=0, max_inflight=32, inflight=0, mqueue_len=0, mqueue_
↪dropped=0, awaiting_rel=0, deliver_msg=0, enqueue_msg=0, created_
↪at=1553760314)
```

Properties of the Session:

clean_start	Clean sessions
expiry_interval	Session expiration interval
subscriptions_count	Current subscription
max_inflight	Max inflight window
inflight	Inflight window
mqueue_len	Current cached messages.
mqueue_dropped	Session dropped messages
awaiting_rel	QoS2 msg waiting client to send PUBREL
deliver_msg	Deliver messages count
enqueue_msg	Number of cached messages
created_at	Session create timestamp

14.7.2 sessions show <ClientId>

Show a session:

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

Session(clientid, clean_start=true, expiry_interval=0, subscriptions_count=0,␣
↪max_inflight=32, inflight=0, mqueue_len=0, mqueue_dropped=0, awaiting_rel=0,
↪ deliver_msg=0, enqueue_msg=0, created_at=1553760799)
```

14.8 routes

Show routing table of the broker.

routes list	List all Routes
routes show <Topic>	Show a Route

14.8.1 routes list

List all routes:

```
$ ./bin/emqx_ctl routes list

t2/# -> emqx2@127.0.0.1
t/+/x -> emqx2@127.0.0.1,emqx@127.0.0.1
```

14.8.2 routes show <Topic>

Show a route:

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

t/+/x -> emqx2@127.0.0.1,emqx@127.0.0.1
```

14.9 subscriptions

Query the subscription table of the broker:

subscriptions list	Query all subscriptions
subscriptions show <ClientId>	Show the a client subscriptions
subscriptions add <ClientId> <Topic> <QoS>	Manually add a subscription
subscriptions del <ClientId> <Topic>	Manually delete a subscription

14.9.1 subscriptions list

Query all subscriptions:

```
$ ./bin/emqx_ctl subscriptions list

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

14.9.2 subscriptions show <ClientId>

Show the subscriptions of an MQTT client:

```
$ ./bin/emqx_ctl subscriptions show 'mosqsub/90475-airlee.lo'

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

14.9.3 subscriptions add <ClientId> <Topic> <QoS>

Manually add a subscription:

```
$ ./bin/emqx_ctl subscriptions add 'mosqsub/90475-airlee.lo' '/world' 1

ok
```

14.9.4 subscriptions del <ClientId> <Topic>

Manually delete a subscription:

```
$ ./bin/emqx_ctl subscriptions del 'mosqsub/90475-airlee.lo' '/world'

ok
```

14.10 plugins

List, loadunloadreload plugins of *EMQ X* broker.

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

: When modifying the configuration file of a plugin, you need to execute the `reload` command if it needs to take effect immediately. Because the `unload/load` command does not compile new configuration files

14.10.1 plugins list

List all plugins:

```
$ ./bin/emqx_ctl plugins list

Plugin(emqx_auth_clientid, version=v3.1.0, description=EMQ X Authentication␣
↪with ClientId/Password, active=false)
Plugin(emqx_auth_http, version=v3.1.0, description=EMQ X Authentication/ACL␣
↪with HTTP API, active=false)
Plugin(emqx_auth_jwt, version=v3.1.0, description=EMQ X Authentication with␣
↪JWT, active=false)
Plugin(emqx_auth_ldap, version=v3.1.0, description=EMQ X Authentication/ACL␣
↪with LDAP, active=false)
Plugin(emqx_auth_mongo, version=v3.1.0, description=EMQ X Authentication/ACL␣
↪with MongoDB, active=false)
Plugin(emqx_auth_mysql, version=v3.1.0, description=EMQ X Authentication/ACL␣
↪with MySQL, active=false)
Plugin(emqx_auth_pgsql, version=v3.1.0, description=EMQ X Authentication/ACL␣
↪with PostgreSQL, active=false)
Plugin(emqx_auth_redis, version=v3.1.0, description=EMQ X Authentication/ACL␣
↪with Redis, active=false)
Plugin(emqx_auth_username, version=v3.1.0, description=EMQ X Authentication␣
↪with Username and Password, active=false)
Plugin(emqx_coap, version=v3.1.0, description=EMQ X CoAP Gateway,␣
↪active=false)
Plugin(emqx_dashboard, version=v3.1.0, description=EMQ X Web Dashboard,␣
↪active=true)
Plugin(emqx_delayed_publish, version=v3.1.0, description=EMQ X Delayed␣
↪Publish, active=false)
Plugin(emqx_lua_hook, version=v3.1.0, description=EMQ X Lua Hooks,␣
↪active=false)
Plugin(emqx_lwm2m, version=v3.1.0, description=EMQ X LwM2M Gateway,␣
↪active=false)
Plugin(emqx_management, version=v3.1.0, description=EMQ X Management API and␣
↪CLI, active=true)
Plugin(emqx_plugin_template, version=v3.1.0, description=EMQ X Plugin␣
↪Template, active=false)
Plugin(emqx_psk_file, version=v3.1.0, description=EMQX PSK Plugin from File,␣
↪active=false)
Plugin(emqx_recon, version=v3.1.0, description=EMQ X Recon Plugin,␣
↪active=true)
Plugin(emqx_reloader, version=v3.1.0, description=EMQ X Reloader Plugin,␣
↪active=false)
Plugin(emqx_retainer, version=v3.1.0, description=EMQ X Retainer, active=true)
Plugin(emqx_rule_engine, version=v3.1.0, description=EMQ X Rule Engine,␣
↪active=true)
Plugin(emqx_sn, version=v3.1.0, description=EMQ X MQTT SN Plugin,␣
↪active=false)
Plugin(emqx_statsd, version=v3.1.0, description=Statsd for EMQ X,␣
↪active=false)
Plugin(emqx_stomp, version=v3.1.0, description=EMQ X Stomp Protocol Plugin,␣
↪active=false)
```

()

()

```
Plugin(emqx_web_hook, version=v3.1.0, description=EMQ X Webhook Plugin,␣
↪active=false)
```

Properties of a plugin:

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

14.10.2 plugins load <Plugin>

Load a plugin:

```
$ ./bin/emqx_ctl plugins load emqx_lua_hook

Start apps: [emqx_lua_hook]
Plugin emqx_lua_hook loaded successfully.
```

14.10.3 plugins unload <Plugin>

Unload a plugin:

```
$ ./bin/emqx_ctl plugins unload emqx_lua_hook

Plugin emqx_lua_hook unloaded successfully.
```

14.10.4 plugins reload <Plugin>

Reload a plugin:

```
$ ./bin/emqx_ctl plugins reload emqx_lua_hook

Plugin emqx_lua_hook reloaded successfully.
```

14.11 bridges

Bridges command is used to create bridges between multiple *EMQ X* nodes:

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

bridges list	List bridges
bridges start <Name>	Start a bridge
bridges stop <Name>	Stop a bridge
bridges forwards <Name>	Show a bridge forward topic
bridges add-forward <Name> <Topic>	Add bridge forward topic
bridges del-forward <Name> <Topic>	Delete bridge forward topic
bridges subscriptions <Name>	Show a bridge subscriptions topic
bridges add-subscription <Name> <Topic> <QoS>	Add bridge subscriptions topic
bridges del-subscription <Name> <Topic>	Delete bridge subscriptions topic

The configuration items for bridges are in the emqx/emqx.config file.

14.11.1 bridges list

List all bridge status:

```
$ ./bin/emqx_ctl bridges list

name: emqx     status: Stopped
```

14.11.2 bridges start <Name>

Start a bridge

```
$ ./bin/emqx_ctl bridges start emqx

Start bridge successfully.
```

14.11.3 bridges stop <Name>

Stop a bridge:

```
$ ./bin/emqx_ctl bridges stop emqx

Stop bridge successfully.
```

14.11.4 bridges forwards <Name>

Show a bridge forward topic:

```
$ ./bin/emqx_ctl bridges forwards emqx

topic:   sensor/#
```

14.11.5 bridges add-forward <Name> <Topic>

Add bridge forward topic:

```
$ ./bin/emqx_ctl bridges add-forward emqx device_status/#

Add-forward topic successfully.
```

14.11.6 bridges del-forward <Name> <Topic>

Delete bridge forward topic:

```
$ ./bin/emqx_ctl bridges del-forward emqx device_status/#

Del-forward topic successfully.
```

14.11.7 bridges subscriptions <Name>

Show a bridge subscriptions topic:

```
$ ./bin/emqx_ctl bridges subscriptions emqx

topic: cmd/topic, qos: 1
```

14.11.8 bridges add-subscription <Name> <Topic> <QoS>

Add bridge subscriptions topic:

```
$ ./bin/emqx_ctl bridges add-subscription emqx cmd/topic 1

Add-subscription topic successfully.
```

14.11.9 bridges del-subscription <Name> <Topic>

Delete bridge subscriptions topic:

```
$ ./bin/emqx_ctl bridges del-subscription emqx cmd/topic

Del-subscription topic successfully.
```

14.12 vm

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

vm	Query all
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 ports	Query VM ports

14.12.1 vm all

Query all VM information, including load, memory, number of Erlang processes:

```
cpu/load1               : 4.22
cpu/load5               : 3.29
cpu/load15              : 3.16
memory/total            : 99995208
memory/processes        : 38998248
memory/processes_used   : 38938520
memory/system           : 60996960
memory/atom             : 1189073
memory/atom_used        : 1173808
memory/binary           : 100336
memory/code             : 25439961
memory/ets              : 7161128
process/limit           : 2097152
process/count           : 315
io/max_fds              : 10240
io/active_fds           : 0
ports/count             : 18
ports/limit             : 1048576
```

14.12.2 vm load

Query load:

```
$ ./bin/emqx_ctl vm load

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

14.12.3 vm memory

Query memory:

```
$ ./bin/emqx_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
```

14.12.4 vm process

Query number of erlang processes:

```
$ ./bin/emqx_ctl vm process

process/limit            : 2097152
process/count            : 314
```

14.12.5 vm io

Query max, active file descriptors of IO:

```
$ ./bin/emqx_ctl vm io

io/max_fds               : 10240
io/active_fds            : 0
```

14.12.6 vm ports

Query VM ports:

```
$ ./bin/emqx_ctl vm ports

ports/count            : 18
ports/limit            : 1048576
```

14.13 mnesia

Query the mnesia database system status.

14.14 log

The log command is used to set the log level. Visit the Documentation of logger for more information

log set-level <Level>	Set the primary log level and all Handlers log levels
log primary-level	Show the main log level
log primary-lelvel <Level>	Set the primary log level
log handlers list	Show all currently useing Hanlders
log handlers set-level <HandlerId> <Level>	Set the log level for the specified Hanlder

14.14.1 log set-level <Level>

Set the primary log level and all Handlers log levels:

```
$ ./bin/emqx_ctl log set-level debug

debug
```

14.14.2 log primary-level

Show the main log level:

```
$ ./bin/emqx_ctl log primary-level

debug
```

14.14.3 log primary-level <Level>

Set the primary log level:

```
$ ./bin/emqx_ctl log primary-level info

info
```

14.14.4 log handlers list

Show all logger handlers:

```
$ ./bin/emqx_ctl log handlers list

LogHandler(id=emqx_logger_handler, level=debug, destination=unknown)
LogHandler(id=file, level=debug, destination=log/emqx.log)
LogHandler(id=default, level=debug, destination=console)
```

14.14.5 log handlers set-level <HandlerId> <Level>

Set the log level for a specified handler:

```
$ ./bin/emqx_ctl log handlers set-level emqx_logger_handler error

error
```

14.15 trace

The trace command is used to trace a client or a topic and redirect related log messages to a file.

trace list	Query all open traces
trace start client <ClientId> <File> [<Level>]	Start Client trace
trace stop client <ClientId>	Stop Client trace
trace start topic <Topic> <File> [<Level>]	Start Topic trace
trace stop topic <Topic>	Stop Topic trace

: Before using trace, you need to set the primary logger level to a value low enough. To improve system performance, the default primary log level is error.

14.15.1 trace start client <ClientId> <File> [<Level>]

Start Client trace:

```
$ ./bin/emqx_ctl log primary-level debug

debug

$ ./bin/emqx_ctl trace start client clientid log/clientid_trace.log

trace client clientid successfully

$ ./bin/emqx_ctl trace start client clientid2 log/clientid2_trace.log error

trace client_id clientid2 successfully
```

14.15.2 trace stop client <ClientId>

Stop Client trace:

```
$ ./bin/emqx_ctl trace stop client clientid

stop tracing client_id clientid successfully
```

14.15.3 trace start topic <Topic> <File> [<Level>]

Start Topic trace:

```
$ ./bin/emqx_ctl log primary-level debug

debug

$ ./bin/emqx_ctl trace start topic topic log/topic_trace.log

trace topic topic successfully

$ ./bin/emqx_ctl trace start topic topic2 log/topic2_trace.log error

trace topic topic2 successfully
```

14.15.4 trace stop topic <Topic>

Stop Topic trace:

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

stop tracing topic topic successfully
```

14.15.5 trace list

Query all open traces:

```
$ ./bin/emqx_ctl trace list

Trace(client_id=clientid2, level=error, destination="log/clientid2_trace.log")
Trace(topic=topic2, level=error, destination="log/topic2_trace.log")
```

14.16 listeners

The listeners command is used to query open TCP service listeners.

listeners	Show all the TCP listeners
listeners stop <Proto> <Port>	Stop listener port

14.16.1 listeners list

Show all the TCP listeners:

```
$ ./bin/emqx_ctl listeners

listener on mqtt:ssl:8883
  acceptors       : 16
  max_conns       : 102400
  current_conn    : 0
  shutdown_count  : []
listener on mqtt:tcp:0.0.0.0:1883
  acceptors       : 8
  max_conns       : 1024000
  current_conn    : 0
  shutdown_count  : []
listener on mqtt:tcp:127.0.0.1:11883
  acceptors       : 4
  max_conns       : 1024000
  current_conn    : 2
  shutdown_count  : []
listener on http:dashboard:18083
  acceptors       : 2
  max_conns       : 512
  current_conn    : 0
  shutdown_count  : []
listener on http:management:8080
  acceptors       : 2
  max_conns       : 512
  current_conn    : 0
  shutdown_count  : []
listener on mqtt:ws:8083
  acceptors       : 2
  max_conns       : 102400
  current_conn    : 0
  shutdown_count  : []
listener on mqtt:wss:8084
  acceptors       : 2
  max_conns       : 16
  current_conn    : 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	

14.16.2 listeners stop <Proto> <Port>

Stop listener port:

```
$ ./bin/emqx_ctl listeners stop mqtt:tcp 0.0.0.0:1883

Stop mqtt:tcp listener on 0.0.0.0:1883 successfully.
```

14.17 Rule Engine

14.18 rules

rules list	List all rules
rules show <RuleId>	Show a rule
rules create <name> <hook> <sql> <actions> [-d [<descr>]]	Create a rule
rules delete <RuleId>	Delete a rule

14.18.1 rules create

Create a new rule:

```
## create a simple rule for testing, printing all messages sent to topic 't/a'
$ ./bin/emqx_ctl rules create \
  'test1' \
  'message.publish' \
  'select * from "t/a"' \
  '[{"name":"built_in:inspect_action", "params": {"a": 1}}]' \
  -d 'Rule for debug'

Rule test1:1556242324634254201 created
```

: A rule is identified by a server-side-generated ID. So run 'rules create' multiple times using the same name will create multiple rules with the same name but different IDs.

14.18.2 rules list

List all rules:

```
$ ./bin/emqx_ctl rules list

rule(id='test1:1556242324634254201', name='test1', for='message.publish',␣
↪rawsql='select * from "t/a"', actions=[{"name":"built_in:inspect_action",
↪"params":{"a":1}}], enabled='true', description='Rule for debug')
```

14.18.3 rules show

Query a rule:

```
## Query a rule by whose ID is 'test1:1556242324634254201'
$ ./bin/emqx_ctl rules show 'test1:1556242324634254201'

rule(id='test1:1556242324634254201', name='test1', for='message.publish',␣
↪rawsql='select * from "t/a"', actions=[{"name":"built_in:inspect_action",
↪"params":{"a":1}}], enabled='true', description='Rule for debug')
```

14.18.4 rules delete

Delete a rule:

```
## Delete a rule whose ID is 'test1:1556242324634254201'
$ ./bin/emqx_ctl rules delete 'test1:1556242324634254201'

ok
```

14.19 rule-actions

rule-actions list [-t <type>]] [-k [<hook>]]	List all actions
rule-actions show <ActionId>	Show a rule action

: Actions could be built-in actions, or provided by emqx plugins, but cannot be added/deleted dynamically via CLI/API.

14.19.1 rule-actions show

Query actions:

```
## Query the action named 'built_in:inspect_action'
$ ./bin/emqx_ctl rule-actions show 'built_in:inspect_action'

action(name='built_in:inspect_action', app='emqx_rule_engine', for='$any',␣
↪type='built_in', params=#{}, description='Inspect the details of action␣
↪params for debug purpose')
```

14.19.2 rule-actions list

List actions by hook or resource-type:

```
## List all the actions
$ ./bin/emqx_ctl rule-actions list

action(name='built_in:republish_action', app='emqx_rule_engine', for='message.
↪publish', type='built_in', params=#{target_topic => #{description => <<
↪"Repubilsh the message to which topic">>,format => topic,required => true,
↪title => <<"To Which Topic">>,type => string}}, description='Republish a␣
↪MQTT message to a another topic')
action(name='web_hook:event_action', app='emqx_web_hook', for='$events', type=
↪'web_hook', params=#{'$resource' => #{description => <<"Bind a resource to␣
↪this action">>,required => true,title => <<"Resource ID">>,type => string},
↪template => #{description => <<"The payload template to be filled with␣
↪variables before sending messages">>,required => false,schema => #{},title␣
↪=> <<"Payload Template">>,type => object}}, description='Forward Events to␣
↪Web Server')
action(name='web_hook:publish_action', app='emqx_web_hook', for='message.
↪publish', type='web_hook', params=#{'$resource' => #{description => <<"Bind␣
↪a resource to this action">>,required => true,title => <<"Resource ID">>,
↪type => string}}, description='Forward Messages to Web Server')
action(name='built_in:inspect_action', app='emqx_rule_engine', for='$any',␣
↪type='built_in', params=#{}, description='Inspect the details of action␣
↪params for debug purpose')

## List all the hooks whose resource type is web_hook
$ ./bin/emqx_ctl rule-actions list -t web_hook

action(name='web_hook:event_action', app='emqx_web_hook', for='$events', type=
↪'web_hook', params=#{'$resource' => #{description => <<"Bind a resource to␣
↪this action">>,required => true,title => <<"Resource ID">>,type => string},
↪template => #{description => <<"The payload template to be filled with␣
↪variables before sending messages">>,required => false,schema => #{},title␣
↪=> <<"Payload Template">>,type => object}}, description='Forward Events to␣
↪Web Server')
action(name='web_hook:publish_action', app='emqx_web_hook', for='message.
↪publish', type='web_hook', params=#{'$resource' => #{description => <<"Bind␣
↪a resource to this action">>,required => true,title => <<"Resource ID">>,
↪type => string}}, description='Forward Messages to Web Server')

## List all the hooks whose hook type matched to 'client.connected'
$ ./bin/emqx_ctl rule-actions list -k 'client.connected'

action(name='built_in:inspect_action', app='emqx_rule_engine', for='$any',␣
↪type='built_in', params=#{}, description='Inspect the details of action␣
↪params for debug purpose')
```

14.20 resources

emqx_ctl resources create <name> <type> [-c [<config>]] [-d [<descr>]]	Create a resource
resources list [-t <ResourceType>]	List all resources
resources show <ResourceId>	Show a resource
resources delete <ResourceId>	Delete a resource

14.20.1 resources create

Create a new resource:

```
$ ./bin/emqx_ctl resources create 'webhook1' 'web_hook' -c '{"url": "http://
↪host-name/chats"}' -d 'forward msgs to host-name/chats'

Resource web_hook:webhook1 created
```

14.20.2 resources list

List all the resources:

```
$ ./bin/emqx_ctl resources list

resource(id='web_hook:webhook1', name='webhook1', type='web_hook', config=#{<<
↪"url">> => <<"http://host-name/chats">>}, attrs=undefined, description=
↪'forward msgs to host-name/chats')
```

14.20.3 resources list by type

List resources by resource-type:

```
$ ./bin/emqx_ctl resources list --type 'debug_resource_type'

resource(id='web_hook:webhook1', name='webhook1', type='web_hook', config=#{<<
↪"url">> => <<"http://host-name/chats">>}, attrs=undefined, description=
↪'forward msgs to host-name/chats')
```

14.20.4 resources show

Query resources:

```
$ ./bin/emqx_ctl resources show 'web_hook:webhook1'

resource(id='web_hook:webhook1', name='webhook1', type='web_hook', config=#{<<
↪"url">> => <<"http://host-name/chats">>}, attrs=undefined, description=
↪'forward msgs to host-name/chats')
```

14.20.5 resources delete

Delete resources:

```
$ ./bin/emqx_ctl resources delete 'web_hook:webhook1'

ok
```

14.21 resource-types

resource-types list	List all resource-types
resource-types show <Type>	Show a resource-type

: Resource types could be built-in resource types, or provided by emqx plugins, but cannot be added/deleted dynamically via CLI/API.

14.21.1 resource-types list

List all the resource types:

```
./bin/emqx_ctl resource-types list

resource_type(name='built_in', provider='emqx_rule_engine', params=#{}, on_
↪create={emqx_rule_actions,on_resource_create}, description='The built in␣
↪resource type for debug purpose')
resource_type(name='web_hook', provider='emqx_web_hook', params=#{headers => #
↪{default => #{},description => <<"Request Header">>,schema => #{},title => <
↪<"Request Header">>,type => object},method => #{default => <<"POST">>,
↪description => <<"Request Method">>,enum => [<<"PUT">>,<<"POST">>],title =>
↪<<"Request Method">>,type => string},url => #{description => <<"Request URL
↪">>,format => url,required => true,title => <<"Request URL">>,type =>␣
↪string}}, on_create={emqx_web_hook_actions,on_resource_create}, description=
↪'WebHook Resource')
```

14.21.2 resource-types show

Query a resource type by name:

```
$ ./bin/emqx_ctl resource-types show built_in

resource_type(name='built_in', provider='emqx_rule_engine', params=#{}, on_
↪create={emqx_rule_actions,on_resource_create}, description='The built in␣
↪resource type for debug purpose')
```

14.22 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)

See Documentation for recon for more information.

14.22.1 recon memory

recon_alloc:memory/2:

```
$ ./bin/emqx_ctl recon memory

usage/current       : 0.810331960305788
usage/max           : 0.7992495929358717
used/current        : 84922296
used/max            : 122519208
allocated/current   : 104345600
allocated/max       : 153292800
unused/current      : 19631520
unused/max          : 30773592
```

14.22.2 recon allocated

recon_alloc:memory(allocated_types, current/max):

```
$ ./bin/emqx_ctl recon allocated

binary_alloc/current: 425984
driver_alloc/current: 425984
eheap_alloc/current : 4063232
ets_alloc/current   : 3833856
fix_alloc/current   : 1474560
ll_alloc/current    : 90439680
sl_alloc/current    : 163840
std_alloc/current   : 2260992
temp_alloc/current  : 655360
binary_alloc/max    : 4907008
driver_alloc/max    : 425984
eheap_alloc/max     : 25538560
ets_alloc/max       : 5931008
fix_alloc/max       : 1736704
ll_alloc/max        : 90439680
sl_alloc/max        : 20348928
```

()

()

```
std_alloc/max       : 2260992
temp_alloc/max      : 1703936
```

14.22.3 recon bin_leak

recon:bin_leak(100):

```
$ ./bin/emqx_ctl recon bin_leak

{<10623.1352.0>,-3,
 [cowboy_clock,
  {current_function,{gen_server,loop,7}},
  {initial_call,{proc_lib,init_p,5}}]}
{<10623.3865.0>,0,
 [{current_function,{recon_lib,proc_attrs,2}},
  {initial_call,{erlang,apply,2}}]}
{<10623.3863.0>,0,
 [{current_function,{dist_util,con_loop,2}},
  {initial_call,{inet_tcp_dist,do_accept,7}}]}
  ...
```

14.22.4 recon node_stats

recon:node_stats(10, 1000):

```
$ ./bin/emqx_ctl recon node_stats

{[{process_count,302},
  {run_queue,0},
  {memory_total,88925536},
  {memory_procs,27999296},
  {memory_atoms,1182843},
  {memory_bin,24536},
  {memory_ets,7163216}],
 [{bytes_in,62},
  {bytes_out,458},
  {gc_count,4},
  {gc_words_reclaimed,3803},
  {reductions,3036},
  {scheduler_usage,[{1,9.473889959272245e-4},
                    {2,5.085983030767205e-5},
                    {3,5.3851477624711046e-5},
                    {4,7.579021269127057e-5},
                    {5,0.0},
                    {6,0.0},
                    {7,0.0},
                    {8,0.0}]}]}
...
```

14.22.5 recon remote_load Mod

recon:remote_load(Mod):

```
$ ./bin/emqx_ctl recon remote_load
```

14.23 retainer

retainer info	show retainer messages count
retainer topics	show all retainer topic
retainer clean	Clear all retainer messages

14.23.1 retainer info

Show count of retained messages:

```
$ ./bin/emqx_ctl retainer info

retained/total: 3
```

14.23.2 retainer topics

Show retained topics:

```
$ ./bin/emqx_ctl retainer topics

$SYS/brokers/emqx@127.0.0.1/version
$SYS/brokers/emqx@127.0.0.1/sysdescr
$SYS/brokers
```

14.23.3 retainer clean

Clean all retained messages:

```
$ ./bin/emqx_ctl retainer clean

Cleaned 3 retained messages
```

14.24 admins

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

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

14.24.1 admins add <Username> <Password> <Tags>

Create admin account:

```
$ ./bin/emqx_ctl admins add root public test

ok
```

14.24.2 admins passwd <Username> <Password>

Reset admin account:

```
$ ./bin/emqx_ctl admins passwd root private

ok
```

14.24.3 admins del <Username>

Delete admin account:

```
$ ./bin/emqx_ctl admins del root

ok
```

CHAPTER 15

Plugins

Through module registration and hooks (Hooks) mechanism user can develop plugin to customize authentication and service functions for EMQ X.

The official plug-ins provided by EMQ X include:

Plugin	Configuration file	Description
emqx_dashboard	etc/plugins/emqx_dashbord.conf	Web dashboard Plugin (Default)
emqx_management	etc/plugins/emqx_management.conf	HTTP API and CLI Management Plugin
emqx_psk_file	etc/plugins/emqx_psk_file.conf	PSK support
emqx_web_hook	etc/plugins/emqx_web_hook.conf	Web Hook Plugin
emqx_lua_hook	etc/plugins/emqx_lua_hook.conf	Lua Hook Plugin
emqx_retainer	etc/plugins/emqx_retainer.conf	Retain Message storage module
emqx_rule_engine	etc/plugins/emqx_rule_engine.conf	Rule engine
emqx_bridge_mqtt	etc/plugins/emqx_bridge_mqtt.conf	MQTT Message Bridge Plugin
emqx_delayed_publish	etc/plugins/emqx_delayed_publish.conf	Delayed publish support
emqx_coap	etc/plugins/emqx_coap.conf	CoAP protocol support
emqx_lwm2m	etc/plugins/emqx_lwm2m.conf	LwM2M protocol support

1 –

Plugin	Configuration file	Description
emqx_sn	etc/plugins/emqx_sn.conf	MQTT-SN protocol support
emqx_stomp	etc/plugins/emqx_stomp.conf	Stomp protocol support
emqx_recon	etc/plugins/emqx_recon.conf	Recon performance debugging
emqx_reloader	etc/plugins/emqx_reloader.conf	Hot load plugin
emqx_plugin_template	etc/plugins/emqx_plugin_template.conf	plugin develop template

There are four ways to load plugins:

1. Default loading
2. Start and stop plugin on command line
3. Start and stop plugin on Dashboard
4. Start and stop plugin by calling management API

Default loading

If a plugin needs to start with the broker, add this plugin in `data/loaded_plugins`. For example, the plugins that are loaded by default are:

```
emqx_management.
emqx_rule_engine.
emqx_recon.
emqx_retainer.
emqx_dashboard.
```

Start and stop plugin on command line

When the EMQ X is running, plugin list can be displayed and plugins can be loaded/unloaded using CLI command:

```
## Display a list of all available plugins
./bin/emqx_ctl plugins list

## Load a plugin
./bin/emqx_ctl plugins load emqx_auth_username

## Unload a plugin
./bin/emqx_ctl plugins unload emqx_auth_username

## Reload a plugin
./bin/emqx_ctl plugins reload emqx_auth_username
```

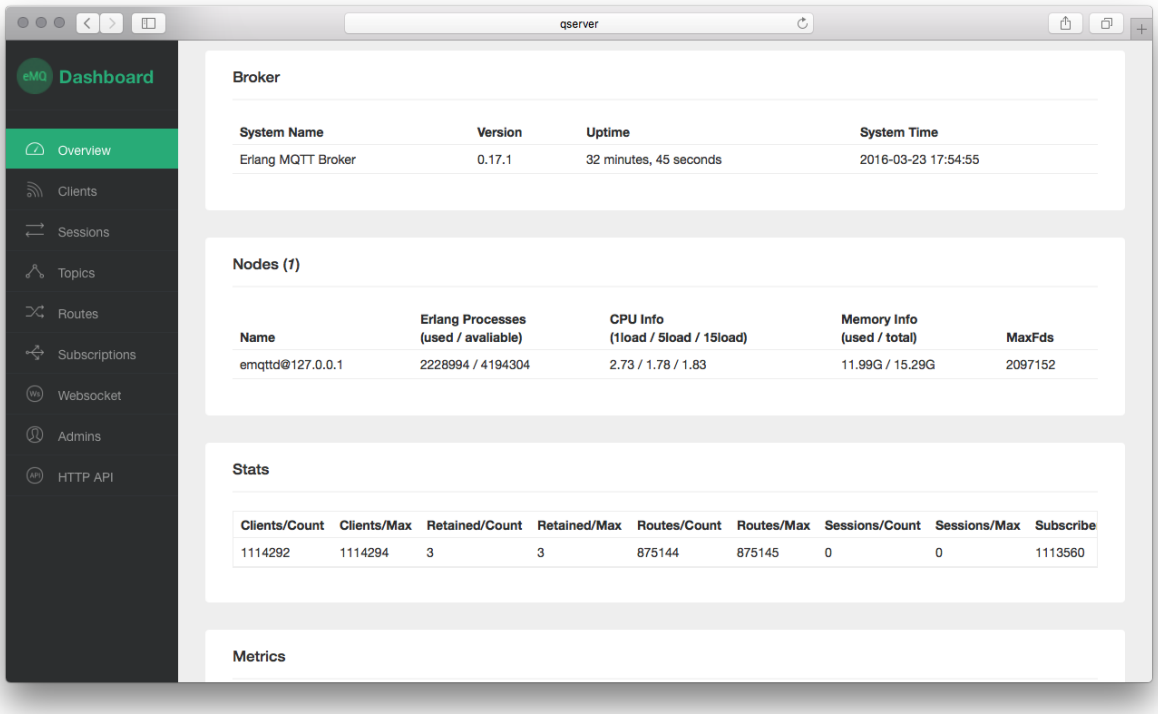
Start and stop plugin on Dashboard

If Dashboard plugin is started (by default), the plugins can be also managed on the dashboard. the managing page can be found under `http://localhost:18083/plugins`.

15.1 Dashboard Plugin

emqx_dashboard is the web management console for the EMQ X broker, which is enabled by default. When EMQ X starts successfully, user can access it by visiting `http://localhost:18083` with the default username/password: admin/public.

> The basic information, statistics, and load status of the EMQ X broker, as well as the current client list (Connections), Sessions, Routing Table (Topics), and Subscriptions can be queried through dashboard.



In addition, dashboard provides a set of REST APIs for front-end calls. See `Dashboard -> HTTP API` for details.

15.1.1 Dashboard plugin settings

etc/plugins/emqx_dashboard.conf:

```
## Dashboard default username/password
dashboard.default_user.login = admin
dashboard.default_user.password = public
```

()

()

```
## Dashboard HTTP service Port Configuration
dashboard.listener.http = 18083
dashboard.listener.http.acceptors = 2
dashboard.listener.http.max_clients = 512

## Dashboard HTTPS service Port Configuration
## dashboard.listener.https = 18084
## dashboard.listener.https.acceptors = 2
## dashboard.listener.https.max_clients = 512
## dashboard.listener.https.handshake_timeout = 15s
## 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
```

15.2 HTTP API and CLI Management Plugin

emqx_management is the HTTP API and CLI management plugin of the *EMQ X* brokerwhich is enabled by default. When *EMQ X* starts successfully, users can query the current client list and other operations via the HTTP API and CLI provided by this plugin. For details see *REST API* and *Commands*.

15.2.1 HTTP API and CLI Management Configuration

etc/plugins/emqx_management.conf:

```
## Max Row Limit
management.max_row_limit = 10000

## Default Application Secret
# management.application.default_secret = public

## Management HTTP Service Port Configuration
management.listener.http = 8080
management.listener.http.acceptors = 2
management.listener.http.max_clients = 512
management.listener.http.backlog = 512
management.listener.http.send_timeout = 15s
management.listener.http.send_timeout_close = on

## Management HTTPS Service Port Configuration
## management.listener.https = 8081
## management.listener.https.acceptors = 2
## management.listener.https.max_clients = 512
## management.listener.https.backlog = 512
## management.listener.https.send_timeout = 15s
## management.listener.https.send_timeout_close = on
## management.listener.https.certfile = etc/certs/cert.pem
```

()

()

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

15.3 PSK Authentication Plugin

emqx_psk_file mainly provides PSK support that aimes to implement connection authentication through PSK when the client establishes a TLS/DTLS connection.

15.3.1 PSK Authentication Plugin Configuration

etc/plugins/emqx_psk_file.conf:

```
psk.file.path = etc/psk.txt
```

15.4 WebHook Plugin

emqx_web_hook can send all EMQ X events and messages to the specified HTTP server.

15.4.1 WebHook plugin configuration

etc/plugins/emqx_web_hook.conf:

```
## Callback Web Server Address
web.hook.api.url = http://127.0.0.1:8080

## Encode message payload field
## Value: undefined | base64 | base62
## Default: undefined (Do not encode)
## web.hook.encode_payload = base64

## Message and event configuration
web.hook.rule.client.connected.1     = {"action": "on_client_connected"}
web.hook.rule.client.disconnected.1  = {"action": "on_client_disconnected"}
web.hook.rule.client.subscribe.1     = {"action": "on_client_subscribe"}
web.hook.rule.client.unsubscribe.1   = {"action": "on_client_unsubscribe"}
web.hook.rule.session.created.1      = {"action": "on_session_created"}
web.hook.rule.session.subscribed.1   = {"action": "on_session_subscribed"}
web.hook.rule.session.unsubscribed.1 = {"action": "on_session_unsubscribed"}
web.hook.rule.session.terminated.1   = {"action": "on_session_terminated"}
web.hook.rule.message.publish.1      = {"action": "on_message_publish"}
web.hook.rule.message.deliver.1      = {"action": "on_message_deliver"}
web.hook.rule.message.acked.1        = {"action": "on_message_acked"}
```

15.5 Lua Plugin

emqx_lua_hook sends all events and messages to the specified Lua function. See its README for specific use.

15.6 Retainer Plugin

emqx_retainer is set to start by default and provides Retained type message support for EMQ X. It stores the Retained messages for all topics in the cluster's database and posts the message when the client subscribes to the topic

15.6.1 Retainer Plugin Configuration

etc/plugins/emqx_retainer.conf:

```
## retained Message storage method
##  - ram: memory only
##  - disc: memory and disk
##  - disc_only: disk only
retainer.storage_type = ram

## Maximum number of storage (0 means unrestricted)
retainer.max_retained_messages = 0

## Maximum storage size for single message
retainer.max_payload_size = 1MB

## Expiration time, 0 means never expired
## Unit:  h hour; m minute; s second.For example, 60m means 60 minutes.
retainer.expiry_interval = 0
```

15.7 MQTT Message Bridge Plugin

The concept of **Bridge** is that EMQ X forwards messages of some of its topics to another MQTT Broker in some way.

Difference between **Bridge** and **cluster** is that bridge does not replicate topic trees and routing tables, a bridge only forwards MQTT messages based on Bridge rules.

Currently the Bridge methods supported by EMQ X are as follows:

- RPC bridge: RPC Bridge only supports message forwarding and does not support subscribing to the topic of remote nodes to synchronize data.
- MQTT Bridge: MQTT Bridge supports both forwarding and data synchronization through subscription topic

In EMQ X, bridge is configured by modifying `etc/plugins/emqx_bridge_mqtt.conf`. EMQ X distinguishes between different bridges based on different names. E.g:

```
## Bridge address: node name for local bridge, host:port for remote.
bridge.mqtt.aws.address = 127.0.0.1:1883
```

This configuration declares a bridge named `aws` and specifies that it is bridged to the MQTT server of `127.0.0.1:1883` by MQTT mode.

In case of creating multiple bridges, it is convenient to replicate all configuration items of the first bridge, and modify the bridge name and other configuration items if necessary (such as bridge.mqtt.$name.address, where $name refers to the name of bridge)

15.7.1 MQTT Bridge Plugin Configuration

etc/plugins/emqx_bridge_mqtt.conf

```
## Bridge Address: Use node name (nodename@host) for rpc Bridge, and␣
↪host:port for mqtt connection
bridge.mqtt.aws.address = emqx2@192.168.1.2

## Forwarding topics of the message
bridge.mqtt.aws.forwards = sensor1/#,sensor2/#

## bridged mountpoint
bridge.mqtt.aws.mountpoint = bridge/emqx2/${node}/

## Bridge Address: Use node name for rpc Bridge, use host:port for mqtt␣
↪connection
bridge.mqtt.aws.address = 192.168.1.2:1883

## Bridged Protocol Version
## Enumeration value: mqttv3 | mqttv4 | mqttv5
bridge.mqtt.aws.proto_ver = mqttv4

## mqtt client's client_id
bridge.mqtt.aws.client_id = bridge_emq

## mqtt client's clean_start field
## Note: Some MQTT Brokers need to set the clean_start value as `true`
bridge.mqtt.aws.clean_start = true

##  mqtt client's username field
bridge.mqtt.aws.username = user

## mqtt client's password field
bridge.mqtt.aws.password = passwd

## Whether the mqtt client uses ssl to connect to a remote serve or not
bridge.mqtt.aws.ssl = off
```

()

()

```
## CA Certificate of Client SSL Connection (PEM format)
bridge.mqtt.aws.cacertfile = etc/certs/cacert.pem

## SSL certificate of Client SSL connection
bridge.mqtt.aws.certfile = etc/certs/client-cert.pem

## Key file of Client SSL connection
bridge.mqtt.aws.keyfile = etc/certs/client-key.pem

## SSL encryption
bridge.mqtt.aws.ciphers = ECDHE-ECDSA-AES256-GCM-SHA384,ECDHE-RSA-AES256-GCM-
↪SHA384

## TTLS PSK password
## Note 'listener.ssl.external.ciphers' and 'listener.ssl.external.psk_ciphers
↪' cannot be configured at the same time
##
## See 'https://tools.ietf.org/html/rfc4279#section-2'.
## bridge.mqtt.aws.psk_ciphers = PSK-AES128-CBC-SHA,PSK-AES256-CBC-SHA,PSK-
↪3DES-EDE-CBC-SHA,PSK-RC4-SHA

## Client's heartbeat interval
bridge.mqtt.aws.keepalive = 60s

## Supported TLS version
bridge.mqtt.aws.tls_versions = tlsv1.2,tlsv1.1,tlsv1

## Forwarding topics of the message
bridge.mqtt.aws.forwards = sensor1/#,sensor2/#

## Bridged mountpoint
bridge.mqtt.aws.mountpoint = bridge/emqx2/${node}/

## Subscription topic for Bridge
bridge.mqtt.aws.subscription.1.topic = cmd/topic1

## Subscription qos for Bridge
bridge.mqtt.aws.subscription.1.qos = 1

## Subscription topic for Bridge
bridge.mqtt.aws.subscription.2.topic = cmd/topic2

## Subscription qos for Bridge
bridge.mqtt.aws.subscription.2.qos = 1

## Bridge reconnection interval
## Default: 30s
bridge.mqtt.aws.reconnect_interval = 30s

## QoS1 message retransmission interval
bridge.mqtt.aws.retry_interval = 20s
```

()

()

```
## Inflight Size.
bridge.mqtt.aws.max_inflight_batches = 32

## emqx_bridge internal number of messages used for batch
bridge.mqtt.aws.queue.batch_count_limit = 32

##  emqx_bridge internal number of message bytes used for batch
bridge.mqtt.aws.queue.batch_bytes_limit = 1000MB

## The path for placing replayq queue. If the item is not specified in the␣
↪configuration, then replayq will run in `mem-only` mode and messages will␣
↪not be cached on disk.
bridge.mqtt.aws.queue.replayq_dir = data/emqx_emqx2_bridge/

## Replayq data segment size
bridge.mqtt.aws.queue.replayq_seg_bytes = 10MB
```

15.8 Delayed Publish Plugin

emqx_delayed_publish provides the function to delay publishing messages. When the client posts a message to EMQ X using the special topic prefix `$delayed/<seconds>/`, EMQ X will publish this message after <seconds> seconds.

15.9 CoAP Protocol Plugin

emqx_coap provides support for the CoAP protocol (RFC 7252)

15.9.1 CoAP protocol Plugin Configuration

etc/plugins/emqx_coap.conf:

```
coap.port = 5683

coap.keepalive = 120s

coap.enable_stats = off
```

DTLS can be enabled if the following two configuration items are set:

```
## Listen port for DTLS
coap.dtls.port = 5684

coap.dtls.keyfile = {{ platform_etc_dir }}/certs/key.pem
coap.dtls.certfile = {{ platform_etc_dir }}/certs/cert.pem

## DTLS options
```

()

()

```
## coap.dtls.verify = verify_peer
## coap.dtls.cacertfile = {{ platform_etc_dir }}/certs/cacert.pem
## coap.dtls.fail_if_no_peer_cert = false
```

15.9.2 Test the CoAP Plugin

A CoAP client is necessary to test CoAP plugin. In following example the libcoap is used.

```
yum install libcoap

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

15.10 LwM2M Protocol Plugin

emqx_lwm2m provides support for the LwM2M protocol.

15.10.1 LwM2M plugin configuration

etc/plugins/emqx_lwm2m.conf:

```
## LwM2M listening port
lwm2m.port = 5683

## Lifetime Limit
lwm2m.lifetime_min = 1s
lwm2m.lifetime_max = 86400s

## `time window` length under Q Mode Mode, in seconds.
## Messages that exceed the window will be cached
#lwm2m.qmode_time_window = 22

## Whether LwM2M is deployed after coaproxy
#lwm2m.lb = coaproxy

## Actively observe all objects after the device goes online
#lwm2m.auto_observe = off

## the subscribed topic from EMQ X after client register succeeded
## Placeholder:
##    '%e': Endpoint Name
##    '%a': IP Address
lwm2m.topics.command = lwm2m/%e/dn/#

## client response message to EMQ X topic
```

()

()

```
lwm2m.topics.response = lwm2m/%e/up/resp

## client notify message to EMQ X topic
lwm2m.topics.notify = lwm2m/%e/up/notify

## client register message to EMQ X topic
lwm2m.topics.register = lwm2m/%e/up/resp

# client update message to EMQ X topic
lwm2m.topics.update = lwm2m/%e/up/resp

# xml file location defined by object
lwm2m.xml_dir =  etc/lwm2m_xml
```

DTLS support can be enabled with the following configuration:

```
# DTLS Certificate Configuration
lwm2m.certfile = etc/certs/cert.pem
lwm2m.keyfile = etc/certs/key.pem
```

15.11 MQTT-SN Protocol Plugin

emqx_sn provides support for the MQTT-SN protocol

15.11.1 MQTT-SN protocol plugin configuration

etc/plugins/emqx_sn.conf:

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

15.12 Stomp Protocol Plugin

emqx_stomp provides support for the Stomp protocol. Clients connect to EMQ X through Stomp 1.0/1.1/1.2 protocol, publish and subscribe to MQTT message.

15.12.1 Stomp plugin configuration

:	Stomp protocol port: 61613

etc/plugins/emqx_stomp.conf:

```
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
```

15.13 Recon Performance Debugging Plugin

emqx_recon integrates the recon performance tuning library to view status information about the current system, for example:

```
./bin/emqx_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)
```

15.13.1 Recon Plugin Configuration

etc/plugins/emqx_recon.conf:

```
%% Garbage Collection: 10 minutes
recon.gc_interval = 600
```

15.14 Reloader Hot Reload Plugin

emqx_reloader is used for code hot-upgrade during impelementation and debugging. After loading this plug-in, EMQ X updates the codes automatically according to the configuration interval.

A CLI command is also provided to force a module to reload:

```
./bin/emqx_ctl reload <Module>
```

: This plugin is not recommended for production environments.

15.14.1 Reloader Plugin Configuration

etc/plugins/emqx_reloader.conf:

```
reloader.interval = 60

reloader.logfile = log/reloader.log
```

15.15 Plugin Development Template

emqx_plugin_template is an EMQ X plugin template and provides no functionality by itself.

When developers need to customize a plugin, they can view this plugin's code and structure to deliver a standard EMQ X plugin faster. The plugin is actually a normal `Erlang Application` with the configuration file: `etc/${PluginName}.config`.

15.16 EMQ X R3.2 Plugin Development

15.16.1 Create a Plugin Project

For creating a new plugin project please refer to the emqx_plugin_template . .. NOTE:: The tag `-emqx_plugin(?MODULE).` must be added to the `<plugin name>_app.erl` file to indicate that this is a plugin for EMQ X.

15.16.2 Create an Authentication/Access Control Module

A demo of authentication module - emqx_auth_demo.erl

```
-module(emqx_auth_demo).

-export([ init/1
        , check/2
        , description/0
        ]).

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

check(_Credentials = #{client_id := ClientId, username := Username, password
↪:= Password}, _State) ->
```

()

()

```
    io:format("Auth Demo: clientId=~p, username=~p, password=~p~n", [ClientId,
↪ Username, Password]),
    ok.

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

A demo of access control module - emqx_acl_demo.erl

```
-module(emqx_acl_demo).

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

%% ACL callbacks
-export([ init/1
        , check_acl/5
        , reload_acl/1
        , description/0
        ]).

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

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

reload_acl(_State) ->
    ok.

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

Registration of authentication, access control module - emqx_plugin_template_app.erl

```
ok = emqx:hook('client.authenticate', fun emqx_auth_demo:check/2, []),
ok = emqx:hook('client.check_acl', fun emqx_acl_demo:check_acl/5, []).
```

15.16.3 Hooks

Events of client's online and offline, topic subscription, message sending and receiving can be handled through hooks.

emqx_plugin_template.erl:

```
%% Called when the plugin application start
load(Env) ->
    emqx:hook('client.authenticate', fun ?MODULE:on_client_authenticate/2,␣
↪[Env]),
    emqx:hook('client.check_acl', fun ?MODULE:on_client_check_acl/5, [Env]),
    emqx:hook('client.connected', fun ?MODULE:on_client_connected/4, [Env]),
    emqx:hook('client.disconnected', fun ?MODULE:on_client_disconnected/3,␣
↪[Env]),
```

()

()

```
    emqx:hook('client.subscribe', fun ?MODULE:on_client_subscribe/3, [Env]),
    emqx:hook('client.unsubscribe', fun ?MODULE:on_client_unsubscribe/3,␣
↪[Env]),
    emqx:hook('session.created', fun ?MODULE:on_session_created/3, [Env]),
    emqx:hook('session.resumed', fun ?MODULE:on_session_resumed/3, [Env]),
    emqx:hook('session.subscribed', fun ?MODULE:on_session_subscribed/4,␣
↪[Env]),
    emqx:hook('session.unsubscribed', fun ?MODULE:on_session_unsubscribed/4,␣
↪[Env]),
    emqx:hook('session.terminated', fun ?MODULE:on_session_terminated/3,␣
↪[Env]),
    emqx:hook('message.publish', fun ?MODULE:on_message_publish/2, [Env]),
    emqx:hook('message.deliver', fun ?MODULE:on_message_deliver/3, [Env]),
    emqx:hook('message.acked', fun ?MODULE:on_message_acked/3, [Env]),
    emqx:hook('message.dropped', fun ?MODULE:on_message_dropped/3, [Env]).
```

Available hooks description:

Hooks	Description
client.authenticate	connection authentication
client.check_acl	ACL validation
client.connected	client online
client.disconnected	client disconnected
client.subscribe	subscribe topic by client
client.unsubscribe	unsubscribe topic by client
session.created	session created
session.resumed	session resumed
session.subscribed	session after topic subscribed
session.unsubscribed	session after topic unsubscribed
session.terminated	session terminated
message.publish	MQTT message publish
message.deliver	MQTT message deliver
message.acked	MQTT message acknowledged
message.dropped	MQTT message dropped

15.16.4 Register CLI Command

Demo module for extending command line - emqx_cli_demo.erl

```
-module(emqx_cli_demo).

-export([cmd/1]).

cmd(["arg1", "arg2"]) ->
    emqx_cli:print("ok");
```

()

()

```
cmd(_) ->
    emqx_cli:usage([{"cmd arg1 arg2", "cmd demo"}]).
```

Register command line module - emqx_plugin_template_app.erl

```
ok = emqx_ctl:register_command(cmd, {emqx_cli_demo, cmd}, []),
```

After the plugin is loadeda new CLI command is added to `./bin/emqx_ctl`

```
./bin/emqx_ctl cmd arg1 arg2
```

15.16.5 Plugin Configuration File

The plugin comes with a configuration file placed in `etc/${plugin_name}.conf|config`. EMQ X supports two plugin configuration formats:

1. Erlang native configuration file format - `${plugin_name}.config`:

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

2. sysctl's `k = v` universal forma - `${plugin_name}.conf`:

   ```
   plugin_name.key = value
   ```

: `k = v` format configuration requires the plugin developer to create a `priv/plugin_name.schema` mapping file.

15.16.6 Compile and Release Plugin

1. clone emqx-rel project:

```
git clone https://github.com/emqx/emqx-rel.git
```

2. Add dependency in rebar.config:

```
{deps,
   [ {plugin_name, {git, "url_of_plugin", {tag, "tag_of_plugin"}}}
   , ....
   ....
   ]
}
```

3. The relx paragraph in rebar.config is added:

```
{relx,
    [...
    , ...
    , {release, {emqx, git_describe},
       [
         {plugin_name, load},
       ]
      }
    ]
}
```

CHAPTER 16

REST API

The REST API allows you to query MQTT clients, sessions, subscriptions, and routes. You can also query and monitor the metrics and statistics of the broker.

Nodes	
List all Nodes in the Cluster	GET api/v2/management/nodes
Retrieve a Node's Info	GET api/v2/management/nodes/{node_name}
List all Nodes'statistics in the Cluster	GET api/v2/monitoring/nodes
Retrieve a node's statistics	GET api/v2/monitoring/nodes/{node_name}
Clients	
List all Clients on a Node	GET api/v2/nodes/{node_name}/clients
Retrieve a Client on a Node	GET api/v2/nodes/{node_name}/clients/{client_id}
Retrieve a Client in the Cluster	GET api/v2/clients/{client_id}
Disconnect a Client	DELETE api/v2/clients/{clientid}
Sessions	
List all Sessions on a Node	GET api/v2/node/{node_name}/sessions
Retrieve a Session on a Node	GET api/v2/nodes/{node_name}/sessions/{client_id}
Retrieve a Session in the Cluster	GET api/v2/sessions/{client_id}
Subscriptions	
List all Subscriptions of a Node	GET api/v2/nodes/{node_name}/subscriptions
List Subscriptions of a Client on a node	GET api/v2/nodes/{node_name}subscriptions/{cliet_id}
List Subscriptions of a Client	GET api/v2/subscriptions/{cliet_id}
Routes	

1 –

List all Routes in the Cluster	GET api/v2/routes
Retrieve a Route in the Cluster	GET api/v2/routes/{topic}
Publish/Subscribe/Unsubscribe	
Publish MQTT Message	POST api/v2/mqtt/publish
Subscribe	POST api/v2/mqtt/subscribe
Unsubscribe	POST api/v2/mqtt/unsubscribe
Plugins	
List all Plugins of a Node	GET /api/v2/nodes/{node_name}/plugins/
Start/Stop a Plugin on a node	PUT /api/v2/nodes/{node_name}/plugins/{plugin_name}
Listeners	
List all Listeners	GET api/v2/monitoring/listeners
List listeners of a Node	GET api/v2/monitoring/listeners/{node_name}
Metrics	
Get Metrics of all Nodes	GET api/v2/monitoring/metrics/
Get Metrics of a Node	GET api/v2/monitoring/metrics/{node_name}
Statistics	
Get Statistics of all Nodes	GET api/v2/monitoring/stats
Get Statistics of a Node	GET api/v2/monitoring/stats/{node_name}

16.1 Base URL

All REST APIs in the documentation have the following base URL:

```
http(s)://host:8080/api/v2/
```

16.2 Basic Authentication

The HTTP requests to the REST API are protected with HTTP Basic authentication, For example:

```
curl -v --basic -u <user>:<passwd> -k http://localhost:8080/api/v2/nodes/
↪emqx@127.0.0.1/clients
```

16.3 Nodes

16.3.1 List all Nodes in the Cluster

Definition:

```
GET api/v2/management/nodes
```

Example Request:

```
GET api/v2/management/nodes
```

Response:

```
{
    "code": 0,
    "result":
    [
        {
            "name": "emqx@127.0.0.1",
            "version": "2.1.1",
            "sysdescr": "EMQ X",
            "uptime": "1 hours, 17 minutes, 1 seconds",
            "datetime": "2017-04-14 14 (tel:2017041414):11:38",
            "otp_release": "R19/8.3",
            "node_status": "Running"
        }
    ]
}
```

16.3.2 Retrieve a Node's Info

Definition:

```
GET api/v2/management/nodes/{node_name}
```

Example Request:

```
GET api/v2/management/nodes/emqx@127.0.0.1
```

Response:

```
{
    "code": 0,
    "result":
    {
        "version": "2.1.1",
        "sysdescr": "EMQ X",
        "uptime": "1 hours, 17 minutes, 18 seconds",
        "datetime": "2017-04-14 14 (tel:2017041414):11:55",
        "otp_release": "R19/8.3",
        "node_status": "Running"
    }
}
```

16.3.3 List all Nodes'statistics in the Cluster

Definition:

```
GET api/v2/monitoring/nodes
```

Example Request:

```
GET api/v2/monitoring/nodes
```

Response:

```
{
    "code": 0,
    "result":
    [
        {
            "name": "emqx@127.0.0.1",
            "otp_release": "R19/8.3",
            "memory_total": "69.19M",
            "memory_used": "49.28M",
            "process_available": 262144,
            "process_used": 303,
            "max_fds": 256,
            "clients": 1,
            "node_status": "Running",
            "load1": "1.93",
            "load5": "1.93",
            "load15": "1.89"
        }
    ]
}
```

16.3.4 Retrieve a node's statistics

Definition:

```
GET api/v2/monitoring/nodes/{node_name}
```

Example Request:

```
GET api/v2/monitoring/nodes/emqx@127.0.0.1
```

Response:

```
{
    "code": 0,
    "result":
    {
        "name": "emqx@127.0.0.1",
        "otp_release": "R19/8.3",
```

()

()

```
        "memory_total": "69.19M",
        "memory_used": "49.24M",
        "process_available": 262144,
        "process_used": 303,
        "max_fds": 256,
        "clients": 1,
        "node_status": "Running",
        "load1": "2.21",
        "load5": "2.00",
        "load15": "1.92"
    }
}
```

16.4 Clients

16.4.1 List all Clients on a Node

Definition:

```
GET api/v2/nodes/{node_name}/clients
```

Request parameter:

```
curr_page={page_no}&page_size={page_size}
```

Example Request:

```
GET api/v2/nodes/emqx@127.0.0.1/clients?curr_page=1&page_size=20
```

Response:

```
{
    "code": 0,
    "result":
    {
        "current_page": 1,
        "page_size": 20,
        "total_num": 1,
        "total_page": 1,
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "username": "undefined",
                "ipaddress": "127.0.0.1",
                "port": 49639,
                "clean_sess": true,
                "proto_ver": 4,
```

()

()

```
                "keepalive": 60,
                "connected_at": "2017-04-14 12:50:15"
            }
        ]
    }
}
```

16.4.2 Retrieve a Client on a Node

Definition:

```
GET api/v2/nodes/{node_name}/clients/{client_id}
```

Example Request:

```
GET api/v2/nodes/emqx@127.0.0.1/clients/C_1492145414740
```

Response:

```
{
    "code": 0,
    "result":
    {
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "username": "undefined",
                "ipaddress": "127.0.0.1",
                "port": 50953,
                "clean_sess": true,
                "proto_ver": 4,
                "keepalive": 60,
                "connected_at": "2017-04-14 13:35:15"
            }
        ]
    }
}
```

16.4.3 Retrieve a Client in the Cluster

Definition:

```
GET api/v2/clients/{client_id}
```

Example Request:

```
GET api/v2/clients/C_1492145414740
```

Response:

```
{
    "code": 0,
    "result":
    {
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "username": "undefined",
                "ipaddress": "127.0.0.1",
                "port": 50953,
                "clean_sess": true,
                "proto_ver": 4,
                "keepalive": 60,
                "connected_at": "2017-04-14 13:35:15"
            }
        ]
    }
}
```

16.4.4 Disconnect a Client

Difinition:

```
DELETE api/v2/clients/{clientid}
```

Expample Requst:

```
DELETE api/v2/clients/C_1492145414740
```

Response:

```
{
    "code":0,
    "result":[]
}
```

16.5 Sessions

16.5.1 List all Sessions on a Node

Definition:

```
GET api/v2/node/{node_name}/sessions
```

Request parameters:

```
curr_page={page_no}&page_size={page_size}
```

Example Request:

```
GET api/v2/nodes/emqx@127.0.0.1/sessions?curr_page=1&page_size=20
```

Response:


```
{
    "code": 0,
    "result":
    {
        "current_page": 1,
        "page_size": 20,
        "total_num": 1,
        "total_page": 1,
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "clean_sess": true,
                "max_inflight": "undefined",
                "inflight_queue": "undefined",
                "message_queue": "undefined",
                "message_dropped": "undefined",
                "awaiting_rel": "undefined",
                "awaiting_ack": "undefined",
                "awaiting_comp": "undefined",
                "created_at": "2017-04-14 13:35:15"
            }
        ]
    }
}
```


16.5.2 Retrieve a Session on a Node

Definition:

```
GET api/v2/nodes/{node_name}/sessions/{client_id}
```

Example Request:

```
GET api/v2/nodes/emqx@127.0.0.1/sessions/C_1492145414740
```

Response:

```
{
    "code": 0,
    "result":
    {
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "clean_sess": true,
                "max_inflight": "undefined",
                "inflight_queue": "undefined",
                "message_queue": "undefined",
                "message_dropped": "undefined",
                "awaiting_rel": "undefined",
                "awaiting_ack": "undefined",
                "awaiting_comp": "undefined",
                "created_at": "2017-04-14 13:35:15"
            }
        ]
    }
}
```

16.5.3 Retrieve Sessions of a client in the Cluster

Definition:

```
GET api/v2/sessions/{client_id}
```

Example Request:

```
GET api/v2/sessions/C_1492145414740
```

Response:

```
{
    "code": 0,
    "result":
    {
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "clean_sess": true,
                "max_inflight": "undefined",
                "inflight_queue": "undefined",
                "message_queue": "undefined",
                "message_dropped": "undefined",
                "awaiting_rel": "undefined",
                "awaiting_ack": "undefined",
                "awaiting_comp": "undefined",
```

()

()

```
                "created_at": "2017-04-14 13:35:15"
            }
        ]
    }
}
```

16.6 Subscriptions

16.6.1 List all Subscriptions of a Node

Definition:

```
GET api/v2/nodes/{node_name}/subscriptions
```

Request parameters:

```
curr_page={page_no}&page_size={page_size}
```

Example Request:

```
GET api/v2/nodes/emqx@127.0.0.1/subscriptions?curr_page=1&page_size=20
```

Response:

```
{
    "code": 0,
    "result":
    {
        "current_page": 1,
        "page_size": 20,
        "total_num": 1,
        "total_page": 1,
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "topic": "$client/C_1492145414740",
                "qos": 1
            }
        ]
    }
}
```

16.6.2 List Subscriptions of a client on a node

Definition:

```
GET api/v2/nodes/{node_name}/subscriptions/{clientid}
```

Example Request:

```
GET api/v2/nodes/emqx@127.0.0.1/subscriptions/C_1492145414740
```

Response:

```
{
    "code": 0,
    "result":
    {
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "topic": "$client/C_1492145414740",
                "qos": 1
            }
        ]
    }
}
```

16.6.3 List Subscriptions of a Client

Definition:

```
GET api/v2/subscriptions/{cliet_id}
```

Example Request:

```
GET api/v2/subscriptions/C_1492145414740
```

Response:

```
{
    "code": 0,
    "result":
    {
        "objects":
        [
            {
                "client_id": "C_1492145414740",
                "topic": "$client/C_1492145414740",
                "qos": 1
            }
        ]
    }
}
```

16.7 Routes

16.7.1 List all Routes in the Cluster

Definition:

```
GET api/v2/routes
```

Request parameters:

```
curr_page={page_no}&page_size={page_size}
```

Example request:

```
GET api/v2/routes
```

Response:

```
{
    "code": 0,
    "result":
    {
        "current_page": 1,
        "page_size": 20,
        "total_num": 1,
        "total_page": 1,
        "objects":
        [
            {
                "topic": "$client/C_1492145414740",
                "node": "emqx@127.0.0.1"
            }
        ]
    }
}
```

16.7.2 Retrieve Route Information of a Topic in the Cluster

Definition:

```
GET api/v2/routes/{topic}
```

Example Request:

```
GET api/v2/routes/test_topic
```

Response:

```
{
    "code": 0,
    "result":
    {
        "objects":
        [
            {
                "topic": "test_topic",
                "node": "emqx@127.0.0.1"
            }
        ]
    }
}
```

16.8 Publish/Subscribe/Unsubscribe

16.8.1 Publish MQTT Message

Definition:

```
POST api/v2/mqtt/publish
```

Request parameters:

```
{
    "topic"          : "test",
    "payload"        : "hello",
    "qos"            : 1,
    "retain"         : false,
    "client_id"      : "C_1492145414740"
}
```

: "topic" is mandatory and other parameters are optional. by default "payload":"", "qos":0, "retain":false, "client_id":"http".

Example request:

```
POST api/v2/mqtt/publish
```

Response:

```
{
    "code": 0,
    "result": []
}
```

16.8.2 Subscribe

Definition:

```
POST api/v2/mqtt/subscribe
```

Request parameters:

```
{
    "topic"          : "test",
    "qos"            : 1,
    "client_id"      : "C_1492145414740"
}
```

Example request:

```
POST api/v2/mqtt/subscribe
```

Response:

```
{
    "code"   : 0,
    "result": []
}
```

16.8.3 Unsubscribe

Definition:

```
POST api/v2/mqtt/unsubscribe
```

Request parameters:

```
{
    "topic"     : "test",
    "client_id": "C_1492145414740"
}
```

Example request:

```
POST api/v2/mqtt/unsubscribe
```

Response:

```
{
    "code": 0,
    "result": []
}
```

16.9 Plugins

16.9.1 List all Plugins of a Node

Definition:

```
   GET /api/v2/nodes/{node_name}/plugins/

Example request::

    GET api/v2/nodes/emqx@127.0.0.1/plugins
```

Response:


```
{
    "code": 0,
    "result":
    [
        {
            "name": "emqx_auth_clientid",
            "version": "2.1.1",
            "description": "EMQ X Authentication with ClientId/Password",
            "active": false
        },
        {
            "name": "emqx_auth_eems",
            "version": "1.0",
            "description": "EMQ X Authentication/ACL with eems",
            "active": false
        },
        {
            "name": "emqx_auth_http",
            "version": "2.1.1",
            "description": "EMQ X Authentication/ACL with HTTP API",
            "active": false
        },
        {
            "name": "emqx_auth_ldap",
            "version": "2.1.1",
            "description": "EMQ X Authentication/ACL with LDAP",
            "active": false
        },
        {
            "name": "emqx_auth_mongo",
            "version": "2.1.1",
            "description": "EMQ X Authentication/ACL with MongoDB",
            "active": false
        },
        {
            "name": "emqx_auth_mysql",
            "version": "2.1.1",
            "description": "EMQ X Authentication/ACL with MySQL",
```


()

()


```
            "active": false
        },
        {
            "name": "emqx_auth_pgsql",
            "version": "2.1.1",
            "description": "EMQ X Authentication/ACL with PostgreSQL",
            "active": false
        },
        {
            "name": "emqx_auth_redis",
            "version": "2.1.1",
            "description": "EMQ X Authentication/ACL with Redis",
            "active": false
        },
        {
            "name": "emqx_auth_username",
            "version": "2.1.1",
            "description": "EMQ X Authentication with Username/Password",
            "active": false
        },
        {
            "name": "emqx_backend_cassa",
            "version": "2.1.1",
            "description": "EMQ X Cassandra Backend",
            "active": false
        },
        {
            "name": "emqx_backend_mongo",
            "version": "2.1.1",
            "description": "EMQ X Mongodb Backend",
            "active": false
        },
        {
            "name": "emqx_backend_mysql",
            "version": "2.1.0",
            "description": "EMQ X MySQL Backend",
            "active": false
        },
        {
            "name": "emqx_backend_pgsql",
            "version": "2.1.1",
            "description": "EMQ X PostgreSQL Backend",
            "active": false
        },
        {
            "name": "emqx_backend_redis",
            "version": "2.1.1",
            "description": "EMQ X Redis Backend",
            "active": false
        },
        {
            "name": "emqx_bridge_kafka",
```


()

()


```
            "version": "2.1.1",
            "description": "EMQ X Kafka Bridge",
            "active": false
        },
        {
            "name": "emqx_bridge_rabbit",
            "version": "2.1.1",
            "description": "EMQ X Bridge RabbitMQ",
            "active": false
        },
        {
            "name": "emqx_dashboard",
            "version": "2.1.1",
            "description": "EMQ X Dashboard",
            "active": true
        },
        {
            "name": "emqx_modules",
            "version": "2.1.1",
            "description": "EMQ X Modules",
            "active": true
        },
        {
            "name": "emqx_recon",
            "version": "2.1.1",
            "description": "Recon Plugin",
            "active": true
        },
        {
            "name": "emqx_reloader",
            "version": "2.1.1",
            "description": "Reloader Plugin",
            "active": false
        },
        {
            "name": "emqx_retainer",
            "version": "2.1.1",
            "description": "EMQ X Retainer",
            "active": true
        }
    ]
}
```


16.9.2 Start/Stop a Plugin on a node

Definition:

```
PUT /api/v2/nodes/{node_name}/plugins/{plugin_name}
```

Request parameters:

```
{
    "active": true/false,
}
```

Example request:

```
PUT api/v2/nodes/emqx@127.0.0.1/plugins/emqx_recon
```

Response:

```
{
    "code": 0,
    "result": []
}
```

16.10 Listeners

16.10.1 List all Listeners

Definition:

```
GET api/v2/monitoring/listeners
```

Response:

```
{
    "code": 0,
    "result":
    {
        "emqx@127.0.0.1":
        [
            {
                "protocol": "mqtt:tcp",
                "listen": "127.0.0.1:11883",
                "acceptors": 16,
                "max_clients": 102400,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:tcp",
                "listen": "0.0.0.0:1883",
                "acceptors": 16,
                "max_clients": 102400,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:ws",
```

()

()


```
                "listen": "8083",
                "acceptors": 4,
                "max_clients": 64,
                "current_clients": 1,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:ssl",
                "listen": "8883",
                "acceptors": 16,
                "max_clients": 102400,
                "current_clients": 0,
                "shutdown_count": []
            },
            {
                "protocol": "mqtt:wss",
                "listen": "8084",
                "acceptors": 4,
                "max_clients": 64,
                "current_clients": 0,
                "shutdown_count": []
            }
        ]
    }
}
```


16.10.2 List listeners of a Node

Definition:

```
GET api/v2/monitoring/listeners/{node_name}
```

Example Request:

```
GET api/v2/monitoring/listeners/emqx@127.0.0.1
```

Response:


```
{
    "code": 0,
    "result":
    [
        {
            "protocol": "mqtt:wss",
            "listen": "8084",
            "acceptors": 4,
            "max_clients": 64,
            "current_clients": 0,
            "shutdown_count": []
        },
```


()

()


```
        {
            "protocol": "mqtt:ssl",
            "listen": "8883",
            "acceptors": 16,
            "max_clients": 102400,
            "current_clients": 0,
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:ws",
            "listen": "8083",
            "acceptors": 4,
            "max_clients": 64,
            "current_clients": 1,
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:tcp",
            "listen": "0.0.0.0:1883",
            "acceptors": 16,
            "max_clients": 102400,
            "current_clients": 0,
            "shutdown_count": []
        },
        {
            "protocol": "mqtt:tcp",
            "listen": "127.0.0.1:11883",
            "acceptors": 16,
            "max_clients": 102400,
            "current_clients": 0,
            "shutdown_count": []
        }
    ]
}
```


16.11 Metrics

16.11.1 Get Metrics of all Nodes

Definition:

```
GET api/v2/monitoring/metrics/
```

Response:


```
{
    "code": 0,
    "result": {
        "emqx@127.0.0.1":
```


()

()


```
        {
            "packets/disconnect":0,
            "messages/dropped":0,
            "messages/qos2/received":0,
            "packets/suback":0,
            "packets/pubcomp/received":0,
            "packets/unsuback":0,
            "packets/pingresp":0,
            "packets/puback/missed":0,
            "packets/pingreq":0,
            "messages/retained":3,
            "packets/sent":0,
            "messages/qos2/dropped":0,
            "packets/unsubscribe":0,
            "packets/pubrec/missed":0,
            "packets/connack":0,
            "messages/received":0,
            "packets/pubrec/sent":0,
            "packets/publish/received":0,
            "packets/pubcomp/sent":0,
            "bytes/received":0,
            "packets/connect":0,
            "packets/puback/received":0,
            "messages/sent":0,
            "packets/publish/sent":0,
            "bytes/sent":0,
            "packets/pubrel/missed":0,
            "packets/puback/sent":0,
            "messages/qos0/received":0,
            "packets/subscribe":0,
            "packets/pubrel/sent":0,
            "messages/forward":0,
            "messages/qos2/sent":0,
            "packets/received":0,
            "packets/pubrel/received":0,
            "messages/qos1/received":0,
            "messages/qos1/sent":0,
            "packets/pubrec/received":0,
            "packets/pubcomp/missed":0,
            "messages/qos0/sent":0
        }
    }
}
```


16.11.2 Get Metrics of a Node

Definition:

```
GET api/v2/monitoring/metrics/{node_name}
```

Example Request:

```
GET api/v2/monitoring/metrics/emqx@127.0.0.1
```

Response:


```
{
    "code": 0,
    "result": {
        "packets/disconnect":0,
        "messages/dropped":0,
        "messages/qos2/received":0,
        "packets/suback":0,
        "packets/pubcomp/received":0,
        "packets/unsuback":0,
        "packets/pingresp":0,
        "packets/puback/missed":0,
        "packets/pingreq":0,
        "messages/retained":3,
        "packets/sent":0,
        "messages/qos2/dropped":0,
        "packets/unsubscribe":0,
        "packets/pubrec/missed":0,
        "packets/connack":0,
        "messages/received":0,
        "packets/pubrec/sent":0,
        "packets/publish/received":0,
        "packets/pubcomp/sent":0,
        "bytes/received":0,
        "packets/connect":0,
        "packets/puback/received":0,
        "messages/sent":0,
        "packets/publish/sent":0,
        "bytes/sent":0,
        "packets/pubrel/missed":0,
        "packets/puback/sent":0,
        "messages/qos0/received":0,
        "packets/subscribe":0,
        "packets/pubrel/sent":0,
        "messages/forward":0,
        "messages/qos2/sent":0,
        "packets/received":0,
        "packets/pubrel/received":0,
        "messages/qos1/received":0,
        "messages/qos1/sent":0,
        "packets/pubrec/received":0,
        "packets/pubcomp/missed":0,
        "messages/qos0/sent":0
    }
}
```

16.12 Statistics

16.12.1 Get Statistics of all Nodes

Definition:

```
GET api/v2/monitoring/stats
```

Example Request:

```
GET api/v2/monitoring/stats
```

Response:


```
{
    "code": 0,
    "result": {
        "emqx@127.0.0.1":
        {
            "clients/count":0,
            "clients/max":0,
            "retained/count":0,
            "retained/max":0,
            "routes/count":0,
            "routes/max":0,
            "sessions/count":0,
            "sessions/max":0,
            "subscribers/count":0,
            "subscribers/max":0,
            "subscriptions/count":0,
            "subscriptions/max":0,
            "topics/count":0,
            "topics/max":0
        }
    }
}
```


16.12.2 Get Statistics of a Node

Definition:

```
GET api/v2/monitoring/stats/{node_name}
```

Example Request:

```
GET api/v2/monitoring/stats/emqx@127.0.0.1
```

Response:

```
{
    "code": 0,
    "result": {
        "clients/count":0,
        "clients/max":0,
        "retained/count":0,
        "retained/max":0,
        "routes/count":0,
        "routes/max":0,
        "sessions/count":0,
        "sessions/max":0,
        "subscribers/count":0,
        "subscribers/max":0,
        "subscriptions/count":0,
        "subscriptions/max":0,
        "topics/count":0,
        "topics/max":0
    }
}
```


16.13 Error Code

Code	Comment
0	Success
101	badrpc
102	Unknown error
103	Username or password error
104	empty username or password
105	user does not exist
106	admin can not be deleted
107	missing request parameter
108	request parameter type error
109	request parameter is not a json
110	plugin has been loaded
111	plugin has been unloaded
112	user is not online

CHAPTER 17

Rate Limit

EMQ X Enterprise Edition supports rate limit in multiple aspects to ensure stability of the system.

17.1 Max Cocurrent Connections

MQTT TCP or SSL lister, set the maximum concurrent connections:

```
## Maximum number of concurrent MQTT/TCP connections.
##
## Value: Number
listener.tcp.<name>.max_clients = 102400

## Maximum number of concurrent MQTT/SSL connections.
##
## Value: Number
listener.ssl.<name>.max_clients = 102400
```

17.2 Max Connection Rate

MQTT TCP or SSL listener, set the maximum allowed connecting speed. It is set to 1000 connenctions per seconde by default:

```
## Maximum external connections per second.
##
## Value: Number
listener.tcp.<name>.max_conn_rate = 1000
```

()

()

```
## Maximum MQTT/SSL connections per second.
##
## Value: Number
listener.ssl.<name>.max_conn_rate = 1000
```

17.3 Traffic Rate Limit

MQTT TCP or SSL listener, set rate limit for a single connection:

```
## Rate limit for the external MQTT/TCP connections. Format is 'rate,burst'.
##
## Value: rate,burst
## Unit: Bps
## listener.tcp.<name>.rate_limit = 1024,4096

## Rate limit for the external MQTT/SSL connections.
##
## Value: rate,burst
## Unit: Bps
## listener.ssl.<name>.rate_limit = 1024,4096
```

17.4 Publish Rate Limit

MQTT TCP or SSL listenern, set message publishing rate limit for a single connection:

```
## Maximum publish rate of MQTT messages.
##
## Value: Number,Seconds
## Default: 10 messages per minute
## listener.tcp.<name>.max_publish_rate = 10,60

## Maximum publish rate of MQTT messages.
##
## See: listener.tcp.<name>.max_publish_rate
##
## Value: Number,Seconds
## Default: 10 messages per minute
## listener.ssl.external.max_publish_rate = 10,60
```

CHAPTER 18

Tuning Guide

EMQ X R2 scaled to 1.3 million concurrent MQTT connections on a 8 Core/32GB CentOS server.

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

18.1 Linux Kernel Tuning

The system-wide limit on max opened file handles:

```
# 2 millions 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
```

18.1.1 /etc/sysctl.conf

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

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

18.1.2 /etc/security/limits.conf

Persist the maximum number of opened file handles for users in /etc/security/limits.conf:

```
emqx       soft    nofile      1048576
emqx       hard    nofile      1048576
```

Note: Under ubuntu, '/etc/systemd/system.conf' has to to be modified:

```
DefaultLimitNOFILE=1048576
```

18.2 TCP 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'
```

TCP Socket read/write buffer:

```
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'
```

TCP 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
```

TIME-WAIT Bucket Pool, Recycling and Reuse:

```
net.ipv4.tcp_max_tw_buckets=1048576

# Note: Enabling following option is not recommended. It could cause␣
↪connection reset under NAT.
# net.ipv4.tcp_tw_recycle = 1
# net.ipv4.tcp_tw_reuse = 1
```

Timeout for FIN-WAIT-2 Sockets:

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

18.3 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
```

18.4 EMQ X Broker

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

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

18.5 Client Machine

Tune the client machine to benchmark emqttd broker:

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

18.5.1 mqtt-jmeter

Test tool for concurrent connections: https://github.com/emqtt/mqtt-jmeter

CHAPTER 19

MQTT Protocol

19.1 MQTT - Light Weight Pub/Sub Messaging Protocol for IoT

19.1.1 Introduction

MQTT is a light weight client server publish/subscribe messaging transport protocol. It is ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things context where a small code footprint is required and/or network bandwidth is at a premium.

MQTT Web site: http://mqtt.org

MQTT V3.1.1 standard: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

19.1.2 Features

1. Open messaging transport protocol, easy to implement.
2. Use of publish/subscribe message pattern, which supports many-to-many communication.
3. Based on TCP/IP network connection.
4. 1 byte Fixed header, 2bytes KeepAlive packet. Compact packet structure.
5. Support QoS, reliable transmission.

19.1.3 Applications

MQTT protocol is widely used in Internet of Things, Mobile Internet, smart hardware, automobile, energy and other industry.

1. IoT M2M communication, IoT big data collection
2. Mobile message push, Web message push
3. Mobile instant messenger
4. Smart hardware, smart home, smart phone
5. Automobile communication, electric vehicle charging station/pole
6. Smart city, telemedicine, distance education
7. Energy industry

19.2 MQTT Topic-based Message Routing

MQTT protocol uses topic to route message. Topic is a hierarchical structured string, like:

```
chat/room/1

sensor/10/temperature

sensor/+/temperature

$SYS/broker/metrics/packets/received

$SYS/broker/metrics/#
```

A forward slash (/) is used to separate levels within a topic tree and provide a hierarchical structure to the topic space. The number sigh (#) is a wildcard for multi-level in a topic and the plus sign (+) is a wildcard for single-level:

```
'+': a/+ matches a/x, a/y

'#': a/# matches a/x, a/b/c/d
```

Publisher and subscriber communicate using topic routing mechanism. E.g., mosquitto CLI message pub/sub:

```
mosquitto_sub -t a/b/+ -q 1

mosquitto_pub -t a/b/c -m hello -q 1
```

: Wildcards are only allowed when subscribing, they are not allowed when publishing.

19.3 MQTT V3.1.1 Protocol Packet

19.3.1 MQTT Packet Format

Fixed header
Variable header
Payload

19.3.2 Fixed Header

Bit	7	6	5	4	3	2	1	0
byte1	MQTT Packet type				Flags			
byte2...	Remaining Length							

19.3.3 Packet Type

Type Name	Value	Description
CONNECT	1	Client request to connect to Server
CONNACK	2	Connect acknowledgement
PUBLISH	3	Publish message
PUBACK	4	Publish acknowledgement
PUBREC	5	Publish received (assured delivery part 1)
PUBREL	6	Publish release (assured delivery part 2)
PUBCOMP	7	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client subscribe request
SUBACK	9	Subscribe acknowledgement
UNSUBSCRIBE	10	Unsubscribe request
UNSUBACK	11	Unsubscribe acknowledgement
PINGREQ	12	PING request
PINGRESP	13	PING response
DISCONNECT	14	Client is disconnecting

19.3.4 PUBLISH

A PUBLISH Control Packet is sent from a client to a server or from a server to a client to transport an application message. PUBACK is used to acknowledge a QoS1 packet and PUBREC/PUBREL/PUBCOMP are used to accomplish a QoS2 message delivery.

19.3.5 PINGREQ/PINGRESP

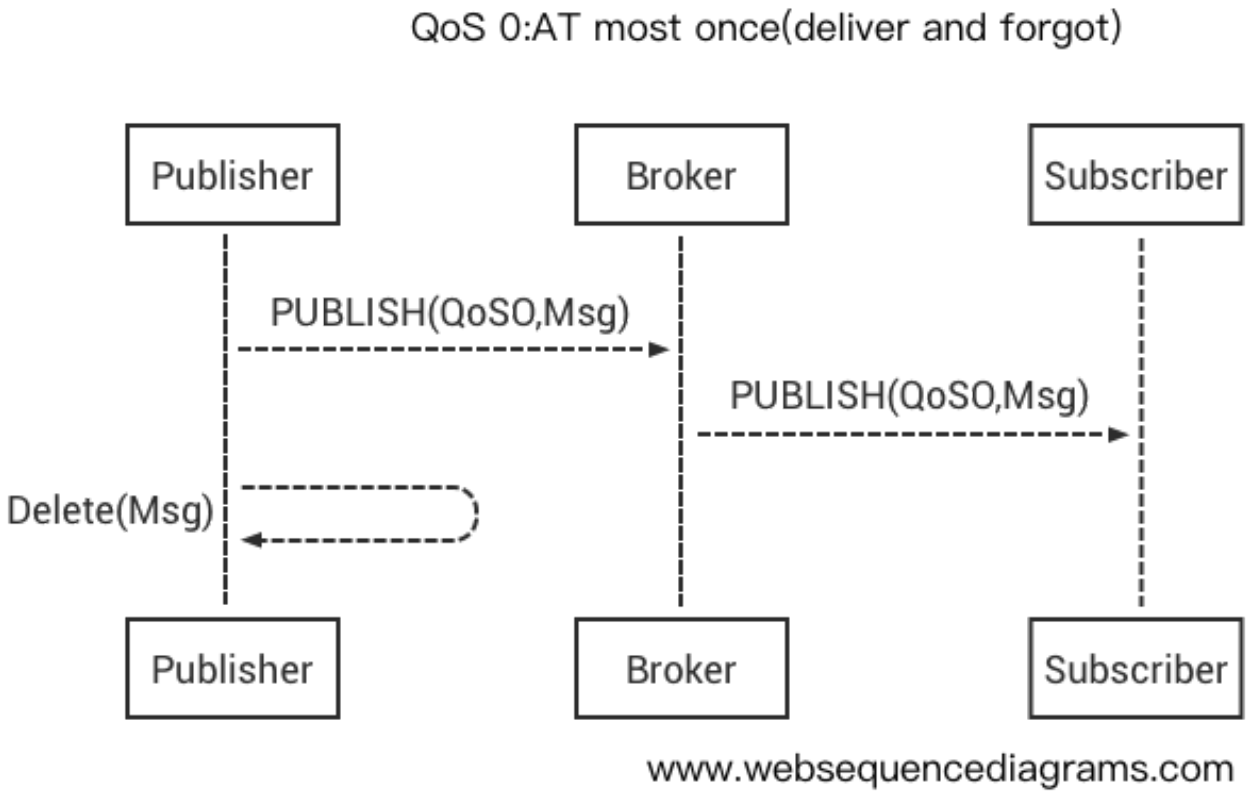
PINGREQ can be sent from a client to server in a KeepAlive interval in absence of any other control packets. The server responses with a PINGRESP packet. PINGREQ and PINGRESP each have a length of 2 bytes.

19.4 MQTT Message QoS

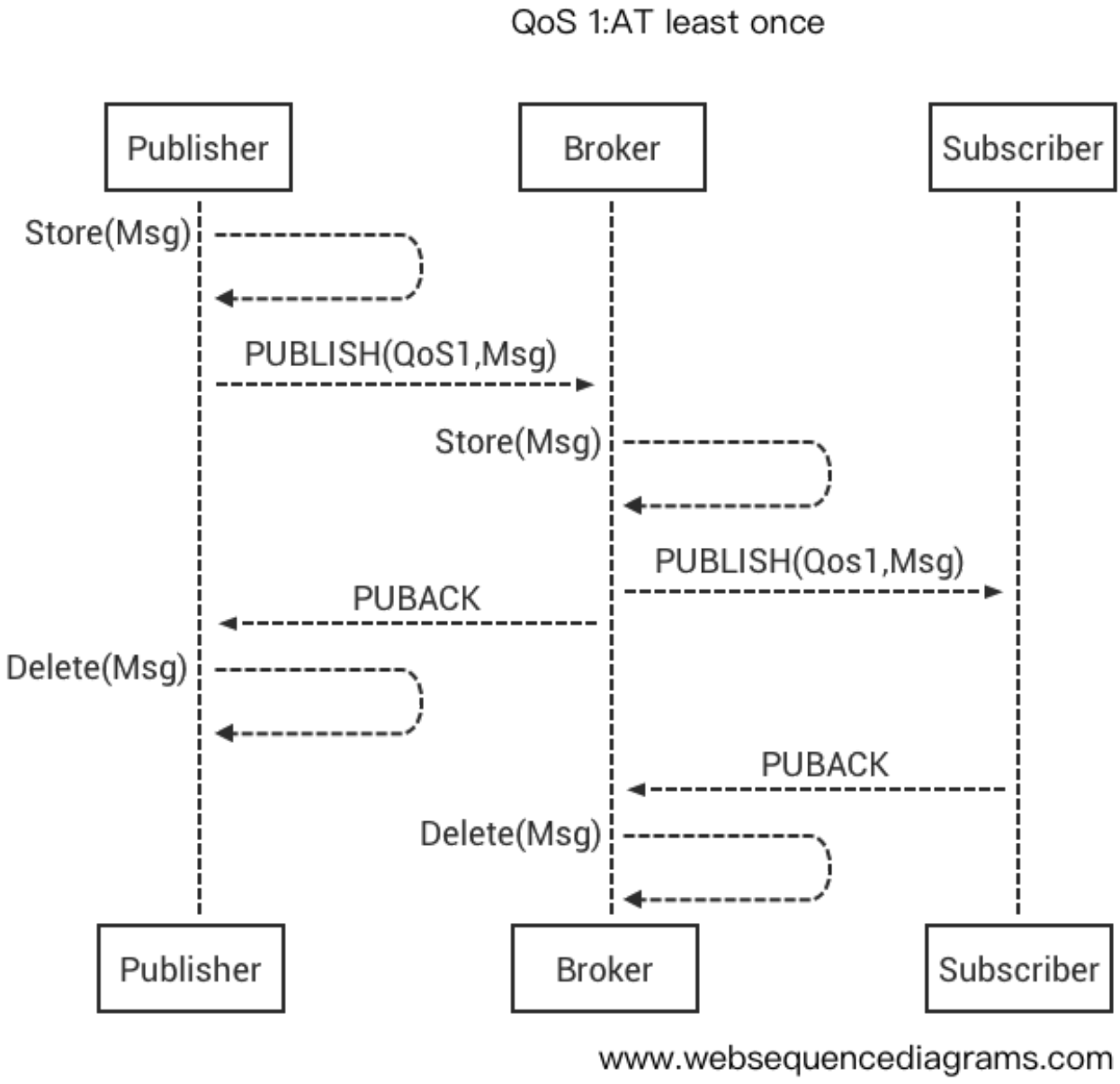
MQTT Message QoS is not end to end, but between the client and the server. The QoS level of a message being received, depends on both the message QoS and the topic QoS.

Published QoS	Topic QoS	Received QoS
0	0	0
0	1	0
0	2	0
1	0	0
1	1	1
1	2	1
2	0	0
2	1	1
2	2	2

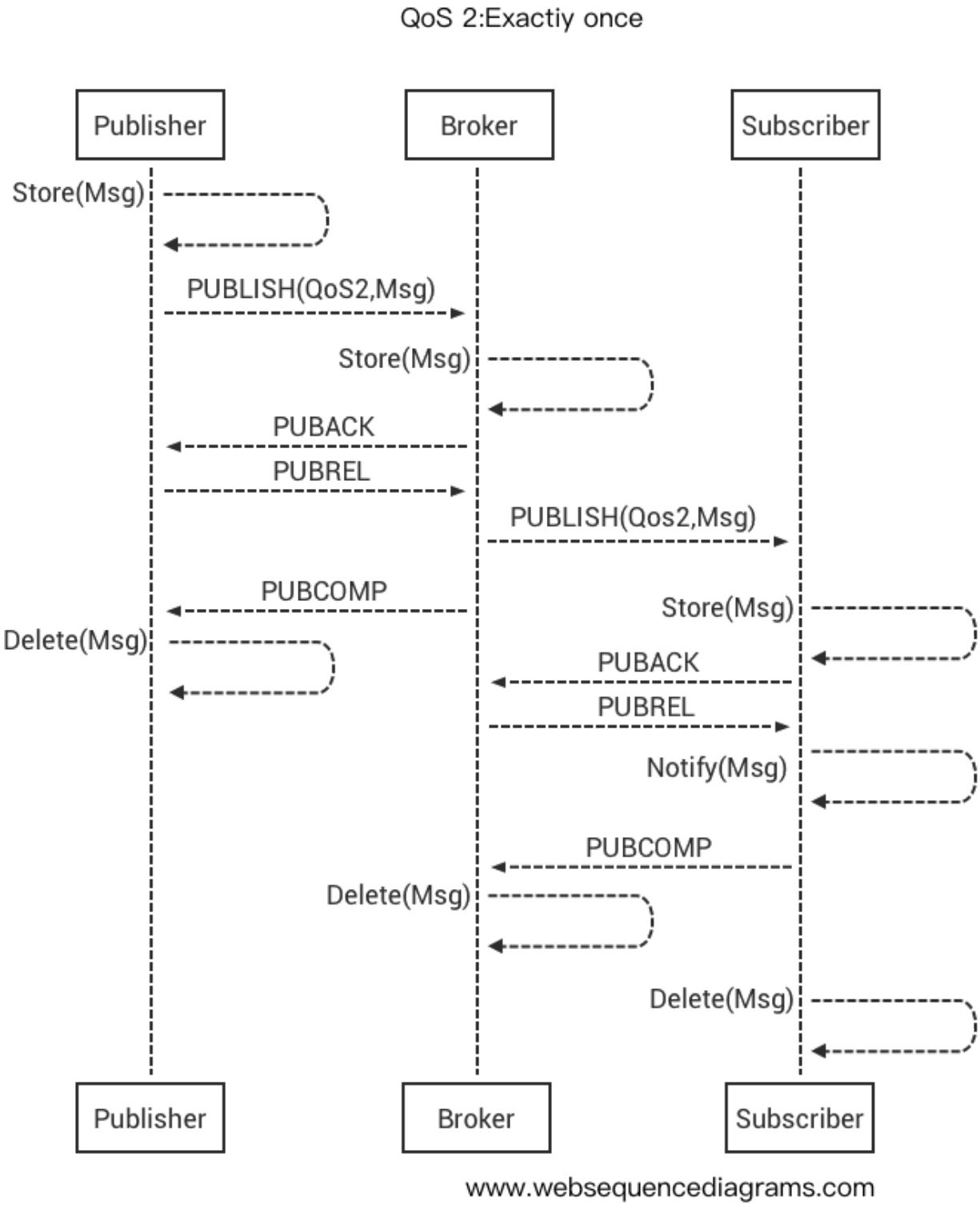
19.4.1 Qos0 Message Publish & Subscribe



19.4.2 Qos1 Message Publish & Subscribe



19.4.3 Qos2 Message Publish & Subscribe



19.5 MQTT Session (Clean Session Flag)

When a MQTT client sends CONNECT request to a server, it can use 'Clean Session' flag to set the session state.

'Clean Session' is 0 indicating a persistent session. When a client is disconnected the session retains and offline messages are also retained, until the session times out.

'Clean Session' is 1 indicating a transient session. If a client is disconnected, the session is destroyed.

19.6 MQTT CONNECT Keep Alive

When MQTT client sends CONNECT packet to server, it uses KEEP Alive bytes to indicate the KeepAlive interval.

In the absence of sending any other control packet, the client must send a PINGREQ packet in ther KeepAlive interval and the server responses with a PINGRESP packet.

If the server doesn't receive any packet from a client within 1.5 * KeepAlive time interval, it close the connection to the client.

: By default EMQ X uses 2.5 * KeepAlive interval.

19.7 MQTT Last Will

A client can declare a Will Message with a Topic and Payload, inside its MQTT CONNECT message sending to the server.

If the MQTT client goes offline abnormally (without sending a DISCONNECT), the server published the Will Message of this client.

19.8 MQTT Retained Message

When a MQTT client sends PUBLISH, it can set the RETAIN flag to indicate a retained message. A retained message is stored by broker and will be sent to clients who subscribe this topic later. A topic has only one retained message which implies new retained message will take place of the old one.

E.g.: A mosquitto client sent a retained message to topic 'a/b/c':

```
mosquitto_pub -r -q 1 -t a/b/c -m 'hello'
```

Later, a client sbuscribes to topic 'a/b/c', and it will receive:

```
$ mosquitto_sub -t a/b/c -q 1
hello
```

Two ways to clean a retained message:

1. Client sends an empty message using the same topic of the retained message.:

```
mosquitto_pub -r -q 1 -t a/b/c -m ''
```

2. The server set a timeout interval for retained message.

19.9 MQTT WebSocket Connection

Besides TCP, MQTT Protocol supports WebSocket as transport layer. A client can connect to server and publish/subscribe through a WebSocket browser.

When using MQTT WebSocket protocol, binary mode must be used and header of sub-protocol must be carried:

```
Sec-WebSocket-Protocol: mqttv3.1 (or mqttv3.1.)1
```

19.10 MQTT Client Library

19.10.1 emqtt Client Library

emqtt project: https://github.com/emqtt

emqttc	Erlang MQTT Client Library
CocoaMQTT	Swift MQTT Client Library
QMQTT	QT Framework MQTT Client Library

19.10.2 Eclipse Paho Client Library

Paho's Website: http://www.eclipse.org/paho/

19.10.3 mqtt.org Client Library

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

19.11 MQTT v.s. XMPP

MQTT is designed to be light weight and easy to use. It is suitable for the mobile Internet and the Internet of Things. While XMPP is a product of the PC era.

1. MQTT uses a one-byte fixed header and two-byte KeepAlive packet, its packet has a size and simple to en/decode. While XMPP is encapsulated in XML, it is large in size and complicated in interaction.

2. MQTT uses topic for routing, it is more flexible than XMPP's peer to peer routing based on JID.
3. MQTT protocol doesn't define a payload format, thus it carries different higher level protocol with ease. While the XMPP uses XML for payload, it must encapsulate binary in Base64 format.
4. MQTT supports message acknowledgement and QoS mechanism, which is absent in XMPP, thus MQTT is more reliable.