EMQTT 分布物联网消息队列

2019年03月

**修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订日期 | 修订人员 | 修订纪要 |
| V1.0 | 2019-03-25 | 徐强辉 | 初始版本 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**文档目录**

[EMQTT 分布物联网消息队列 1](#_Toc5809107)

[1. MQTT协议 5](#_Toc5809108)

[1.1. MQTT轻量发布订阅消息协议 5](#_Toc5809109)

[1.1.1. 概览 5](#_Toc5809110)

[1.1.2. 特点 5](#_Toc5809111)

[1.1.3. 应用 5](#_Toc5809112)

[1.2. MQTT基于主题(Topic)消息路由 5](#_Toc5809113)

[1.3. MQTT V3.1.1协议报文 6](#_Toc5809114)

[1.3.1. 报文结构 6](#_Toc5809115)

[1.3.2. 固定报头 6](#_Toc5809116)

[1.3.3. 报文类型 7](#_Toc5809117)

[1.3.4. PUBLISH发布消息 7](#_Toc5809118)

[1.3.5. PINGREQ/PINGRESP心跳 7](#_Toc5809119)

[1.4. MQTT消息QoS 7](#_Toc5809120)

[1.4.1. Qos0消息发布订阅 8](#_Toc5809121)

[1.4.2. Qos1消息发布订阅 9](#_Toc5809122)

[1.4.3. Qos2消息发布订阅 10](#_Toc5809123)

[1.5. MQTT会话(Clean Session) 10](#_Toc5809124)

[1.6. MQTT连接保活心跳 11](#_Toc5809125)

[1.7. MQTT遗愿消息(Last Will) 11](#_Toc5809126)

[1.8. MQTT保留消息(Retained Message) 11](#_Toc5809127)

[1.9. MQTT WebSocket连接 12](#_Toc5809128)

[1.10. MQTT协议客户端库 12](#_Toc5809129)

[1.10.1. emqtt客户端库 12](#_Toc5809130)

[1.10.2. Eclipse Paho客户端库 13](#_Toc5809131)

[1.10.3. mqtt.org官网客户端库 13](#_Toc5809132)

[1.11. MQTT与XMPP协议对比 13](#_Toc5809133)

[2. MQTT-SN 协议 14](#_Toc5809134)

[2.1. MQTT-SN 和 MQTT 的区别 14](#_Toc5809135)

[2.2. EMQ-SN 网关插件 14](#_Toc5809136)

[3. MQTT安装部署 15](#_Toc5809137)

[3.1. 安装启动 15](#_Toc5809138)

[3.1.1. 下载版本 v2.3.11 15](#_Toc5809139)

[3.1.2. 虚拟机准备 15](#_Toc5809140)

[3.1.3. 解压安装包 16](#_Toc5809141)

[3.1.4. 配置参数 16](#_Toc5809142)

[3.1.4.1. Erlang虚拟机参数 16](#_Toc5809143)

[3.1.4.2. EMQ消息服务器参数 16](#_Toc5809144)

[3.1.4.3. linux操作系统配置 16](#_Toc5809145)

[3.1.4.4. 其他配置 18](#_Toc5809146)

[3.1.5. 启动emq集群 18](#_Toc5809147)

[3.1.5.1. 启动单节点 18](#_Toc5809148)

[3.1.5.2. 加入emq集群 18](#_Toc5809149)

[3.1.5.3. 查看集群状态 18](#_Toc5809150)

[3.1.5.4. web控制台 19](#_Toc5809151)

[4. 安装Haproxy 19](#_Toc5809152)

[5. 压测工具 19](#_Toc5809153)

[5.1. 依赖安装 19](#_Toc5809154)

[5.1.1. erlang依赖 19](#_Toc5809155)

[5.1.2. 安装erlang 20](#_Toc5809156)

[5.1.2.1. 下载解压 20](#_Toc5809157)

[5.1.2.2. 配置 20](#_Toc5809158)

[5.1.2.3. 安装 20](#_Toc5809159)

[5.1.2.4. 配置profile 20](#_Toc5809160)

[5.2. 安装测试工具 20](#_Toc5809161)

[5.2.1. 从github上下载源码 20](#_Toc5809162)

[5.2.2. 发布测试指令 21](#_Toc5809163)

[5.2.3. 订阅测试指令 22](#_Toc5809164)

[5.2.4. 管理控制后台 23](#_Toc5809165)

# MQTT协议

## MQTT轻量发布订阅消息协议

### 概览

MQTT是一个轻量的发布订阅模式消息传输协议，专门针对低带宽和不稳定网络环境的物联网应用设计。

MQTT官网: http://mqtt.org

MQTT V3.1.1协议规范: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

### 特点

1. 开放消息协议，简单易实现
2. 发布订阅模式，一对多消息发布
3. 基于TCP/IP网络连接
4. 1字节固定报头，2字节心跳报文，报文结构紧凑
5. 消息QoS支持，可靠传输保证

### 应用

MQTT协议广泛应用于物联网、移动互联网、智能硬件、车联网、电力能源等领域。

1. 物联网M2M通信，物联网大数据采集
2. Android消息推送，WEB消息推送
3. 移动即时消息，例如Facebook Messenger
4. 智能硬件、智能家具、智能电器
5. 车联网通信，电动车站桩采集
6. 智慧城市、远程医疗、远程教育
7. 电力、石油与能源等行业市场

## MQTT基于主题(Topic)消息路由

**MQTT协议基于主题(Topic)进行消息路由，主题(Topic)类似URL路径，例如:**

chat/room/1

sensor/10/temperature

sensor/+/temperature

$SYS/broker/metrics/packets/received

$SYS/broker/metrics/#

**主题(Topic)通过’/’分割层级，支持’+’, ‘#’通配符:**

'+': 表示通配一个层级，例如a/+，匹配a/x, a/y

'#': 表示通配多个层级，例如a/#，匹配a/x, a/b/c/d

**订阅者与发布者之间通过主题路由消息进行通信，例如采用mosquitto命令行发布订阅消息:**

mosquitto\_sub -t a/b/+ -q 1

mosquitto\_pub -t a/b/c -m hello -q 1

**备注: 订阅者可以订阅含通配符主题，但发布者不允许向含通配符主题发布消息。**

## MQTT V3.1.1协议报文

### 报文结构

|  |
| --- |
| 固定报头(Fixed header) |
| 可变报头(Variable header) |
| 报文有效载荷(Payload) |

### 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| byte1 | Mqtt packet type | | | | Flags | | | |
| byte2 | Remaining Length | | | | | | | |

### 报文类型

|  |  |  |
| --- | --- | --- |
| 类型名称 | 类型值 | 报文说明 |
| CONNECT | 1 | 发起连接 |
| CONNACK | 2 | 连接回执 |
| PUBLISH | 3 | 发布消息 |
| PUBACK | 4 | 发布回执 |
| PUBREC | 5 | QoS2消息回执 |
| PUBREL | 6 | QoS2消息释放 |
| PUBCOMP | 7 | QoS2消息完成 |
| SUBSCRIBE | 8 | 订阅主题 |
| SUBACK | 9 | 订阅回执 |
| UNSUBSCRIBE | 10 | 取消订阅 |
| UNSUBACK | 11 | 取消订阅回执 |
| PINGREQ | 12 | PING请求 |
| PINGRESP | 13 | PING响应 |
| DISCONNECT | 14 | 断开连接 |

### PUBLISH发布消息

PUBLISH报文承载客户端与服务器间双向的发布消息。

PUBACK报文用于接收端确认QoS1报文，

PUBREC/PUBREL/PUBCOMP报文用于QoS2消息流程。

### PINGREQ/PINGRESP心跳

客户端在无报文发送时，按保活周期(KeepAlive)定时向服务端发送PINGREQ心跳报文，服务端响应PINGRESP报文。

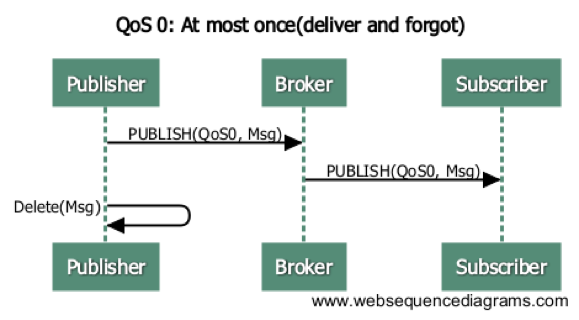
PINGREQ/PINGRESP报文均2个字节。

## MQTT消息QoS

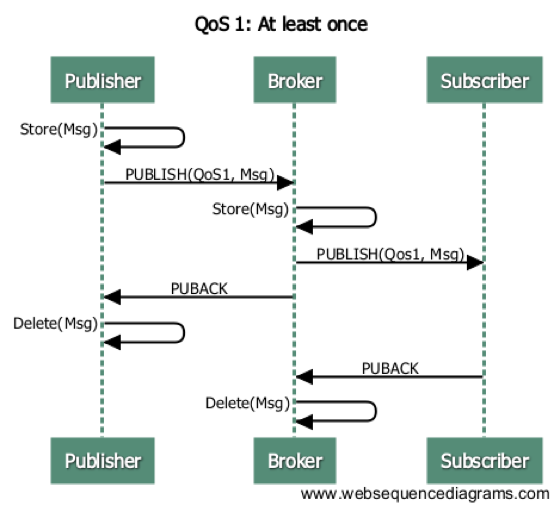
MQTT发布消息QoS保证不是端到端的，是客户端与服务器之间的。订阅者收到MQTT消息的QoS级别，最终取决于发布消息的QoS和主题订阅的QoS。

|  |  |  |
| --- | --- | --- |
| 发布消息的QoS | 主题订阅的QoS | 接收消息的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 |

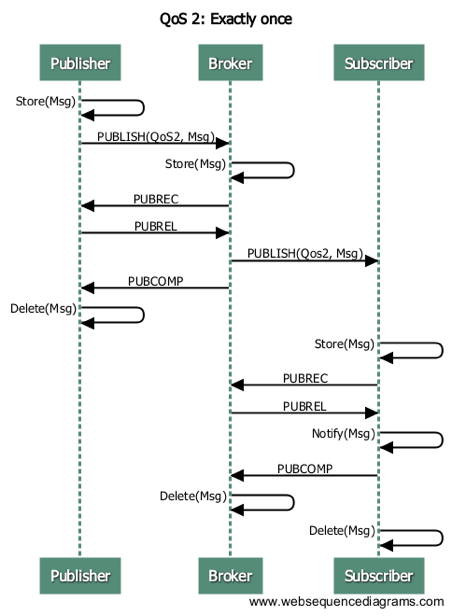
### Qos0消息发布订阅



### Qos1消息发布订阅



### Qos2消息发布订阅



## MQTT会话(Clean Session)

MQTT客户端向服务器发起CONNECT请求时，可以通过’Clean Session’标志设置会话。

‘Clean Session’设置为0，表示创建一个持久会话，在客户端断开连接时，会话仍然保持并保存离线消息，直到会话超时注销。

‘Clean Session’设置为1，表示创建一个新的临时会话，在客户端断开时，会话自动销毁。

## MQTT连接保活心跳

MQTT客户端向服务器发起CONNECT请求时，通过KeepAlive参数设置保活周期。

客户端在无报文发送时，按KeepAlive周期定时发送2字节的PINGREQ心跳报文，服务端收到PINGREQ报文后，回复2字节的PINGRESP报文。

服务端在1.5个心跳周期内，既没有收到客户端发布订阅报文，也没有收到PINGREQ心跳报文时，主动心跳超时断开客户端TCP连接。

注意: emqttd消息服务器默认按最长2.5心跳周期超时设计

## MQTT遗愿消息(Last Will)

MQTT客户端向服务器端CONNECT请求时，可以设置是否发送遗愿消息(Will Message)标志，和遗愿消息主题(Topic)与内容(Payload)。

MQTT客户端异常下线时(客户端断开前未向服务器发送DISCONNECT消息)，MQTT消息服务器会发布遗愿消息。

## MQTT保留消息(Retained Message)

MQTT客户端向服务器发布(PUBLISH)消息时，可以设置保留消息(Retained Message)标志。保留消息(Retained Message)会驻留在消息服务器，后来的订阅者订阅主题时仍可以接收该消息。

例如mosquitto命令行发布一条保留消息到主题’a/b/c’:

mosquitto\_pub -r -q 1 -t a/b/c -m 'hello'

之后连接上来的MQTT客户端订阅主题’a/b/c’时候，仍可收到该消息:

$ mosquitto\_sub -t a/b/c -q 1hello

保留消息(Retained Message)有两种清除方式:

1. 客户端向有保留消息的主题发布一个空消息:

mosquitto\_pub -r -q 1 -t a/b/c -m ''

2. 消息服务器设置保留消息的超期时间

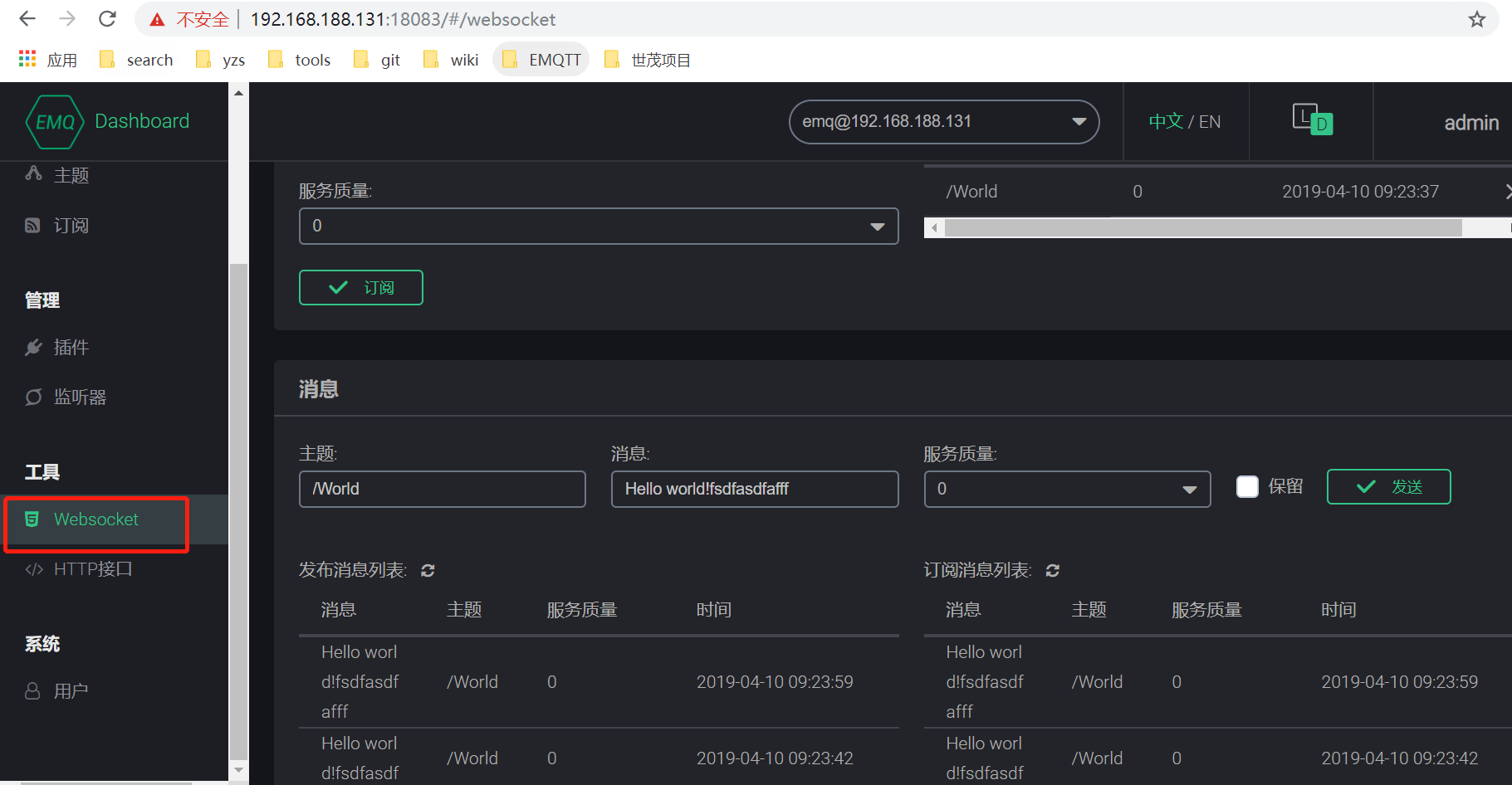
## MQTT WebSocket连接

MQTT协议除支持TCP传输层外，还支持WebSocket作为传输层。通过WebSocket浏览器可以直连MQTT消息服务器，发布订阅模式与其他MQTT客户端通信。

MQTT协议的WebSocket连接，必须采用binary模式，并携带子协议Header:

Sec-WebSocket-Protocol: mqttv3.1 或 mqttv3.1.1

可以用管理控制台websocket 工具测试.



## MQTT协议客户端库

### emqtt客户端库

emqtt项目组: <https://github.com/emqtt>

|  |  |  |
| --- | --- | --- |
| emqttc | https://github.com/emqtt/emqttc | Erlang MQTT客户端库 |
| CocoaMQTT | https://github.com/emqtt/CocoaMQTT | Swift语言MQTT客户端库 |
| QMQTT | https://github.com/emqtt/qmqtt | QT框架MQTT客户端库 |

### Eclipse Paho客户端库

Paho官网: <http://www.eclipse.org/paho/>

Spring-boot maven依赖

<!--mqtt-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-integration</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.integration</groupId>

<artifactId>spring-integration-stream</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.integration</groupId>

<artifactId>spring-integration-mqtt</artifactId>

</dependency>

### mqtt.org官网客户端库

mqtt.org: <https://github.com/mqtt/mqtt.github.io/wiki/libraries>

## MQTT与XMPP协议对比

**MQTT协议设计简单轻量、路由灵活，将在移动互联网物联网消息领域，全面取代PC时代的XMPP协议:**

1. MQTT协议一个字节固定报头，两个字节心跳报文，报文体积小编解码容易。XMPP协议基于繁重的XML，报文体积大且交互繁琐。
2. MQTT协议基于主题(Topic)发布订阅模式消息路由，相比XMPP基于JID的点对点消息路由更为灵活。
3. MQTT协议未定义报文内容格式，可以承载JSON、二进制等不同类型报文。XMPP协议采用XML承载报文，二进制必须Base64编码等处理。
4. MQTT协议支持消息收发确认和QoS保证，XMPP主协议并未定义类似机制。MQTT协议有更好的消息可靠性保证。

# MQTT-SN 协议

MQTT-SN 协议是 MQTT 的直系亲属，它使用 UDP 进行通信，标准的端口是1884。MQTT-SN 的主要目的是为了适应受限的设备和网络，比如一些传感器，只有很小的内存和 CPU，TCP 对于这些设备来说非常奢侈。还有一些网络，比如 ZIGBEE，报文的长度在300字节以下，无法承载太大的数据包。所以 MQTT-SN 的数据包更小巧。

MQTT-SN 的官方标准下载地址:

<http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf>

## MQTT-SN 和 MQTT 的区别

MQTT-SN 的命令和 MQTT 大部分都相同，比如都有 Will, 都有 Connect/Subscribe/Publish 命令.

MQTT-SN 最大的不同是，Topic 使用 TopicId 来代替，而 TopicId 是一个16比特的数字。每一个数字对应一个 Topic, 设备和云端需要使用 REGISTER 命令映射 TopicId 和 Topic 的对应关系。

MQTT-SN 可以随时更改 Will 的内容, 甚至可以取消. 而 MQTT 只允许在 CONNECT 时设定 Will 的内容, 而且不允许更改.

MQTT-SN 的网络中有网关这种设备，它负责把 MQTT-SN 转换成 MQTT，和云端的 MQTT Broker 通信. MQTT-SN 的协议支持自动发现网关的功能。

MQTT-SN 还支持设备的睡眠功能，如果设备进入睡眠状态，无法接收 UDP 数据，网关将把下行的 PUBLISH 消息缓存起来，直到设备苏醒后再传送。

## EMQ-SN 网关插件

EMQ-SN 是 EMQ 的一个网关插件，实现了 MQTT-SN 的大部分功能，它相当于一个在云端的 MQTT-SN 网关，直接和 EMQ Broker 相连。

配置参数

File: etc/plugins/emq\_sn.conf:

mqtt.sn.port = 1884

mqtt.sn.advertise\_duration = 900

mqtt.sn.gateway\_id = 1

mqtt.sn.username = mqtt\_sn\_user

mqtt.sn.password = abc

启动emq-sn

MQTT-SN 客户端库

<https://github.com/eclipse/paho.mqtt-sn.embedded-c/>

<https://github.com/ty4tw/MQTT-SN>

<https://github.com/njh/mqtt-sn-tools>

<https://github.com/arobenko/mqtt-sn>

# MQTT配置

## EMQ 2.0配置文件

**EMQ 2.0 消息服务器通过 etc/ 目录下配置文件进行设置，主要配置文件包括:**

|  |  |
| --- | --- |
| 配置文件 | 说明 |
| etc/emq.conf | EMQ 2.0 消息服务器配置文件 |
| etc/acl.conf | EMQ 2.0 默认ACL规则配置文件 |
| etc/plugins/\*.conf | EMQ 2.0 各类插件配置文件 |

## EMQ 2.2 环境变量

|  |  |
| --- | --- |
| 参数名 | 描述 |
| node.name | Erlang 节点名称，例如: emq@127.0.0.1 |
| node.cookie | Erlang 分布式节点通信 Cookie |
| node.max\_ports | Erlang 虚拟机最大允许打开文件 Socket 数 |
| listener.tcp.external | MQTT/TCP 监听端口，默认: 1883 |
| listener.ssl.external | MQTT/SSL 监听端口，默认: 8883 |
| listener.ws.external | MQTT/WebSocket 监听端口，默认: 8083 |
| listener.wss.external | MQTT/WebSocket/SSL 监听端口，默认: 8084 |

## EMQ 集群设置

### 集群名称

## Cluster name

cluster.name = emqcl

### 自动发现策略

EMQ R2.3 版本支持多种策略的节点自动发现与集群:

|  |  |
| --- | --- |
| 策略 | 说明 |
| manual | 手工命令创建集群 |
| static | 静态节点列表自动集群 |
| mcast | UDP 组播方式自动集群 |
| dns | DNS A 记录自动集群 |
| etcd | 通过 etcd 自动集群 |
| k8s | Kubernetes 服务自动集群 |

#### manual 手动创建集群

默认配置为手动创建集群，节点通过 ./bin/emqttd\_ctl join <Node> 命令加入:

cluster.discovery = manual

#### 基于 static 节点列表自动集群

配置固定的节点列表，自动发现并创建集群:

cluster.discovery = static

##--------------------------------------------------------------------

## Cluster with static node list

cluster.static.seeds = [emq1@127.0.0.1,ekka2@127.0.0.1](mailto:emq1@127.0.0.1,ekka2@127.0.0.1)

#### 基于 mcast 组播自动集群

基于 UDP 组播自动发现并创建集群:

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

#### 基于 DNS A 记录自动集群

基于 DNS A 记录自动发现并创建集群:

cluster.discovery = dns

##--------------------------------------------------------------------

## Cluster with DNS

cluster.dns.name = localhost

cluster.dns.app = ekka

#### 基于 etcd 自动集群

基于 `etcd`\_ 自动发现并创建集群:

cluster.discovery = etcd

##--------------------------------------------------------------------

## Cluster with Etcd

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

cluster.etcd.prefix = emqcl

cluster.etcd.node\_ttl = 1m

#### 基于 Kubernetes 自动集群

`Kubernetes`\_ 下自动发现并创建集群:

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

### 启用集群自愈

## Cluster Autoheal: on | off

cluster.autoheal = on

### 节点自动清除

自动清除宕机节点:

## Clean down node of the cluster

cluster.autoclean = 5m

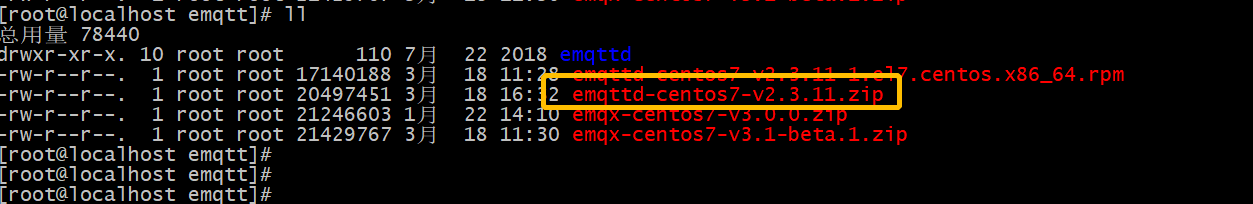
# MQTT安装部署

我这里是选择 安装包安装, 还可以 docker, rpm 等安装方式.

## 安装启动

### 下载版本 v2.3.11

下载地址: <http://www.emqtt.com/downloads>



### 虚拟机准备

|  |  |
| --- | --- |
| emq虚拟机 | 192.168.188.131, 192.168.188.132 |
| haproxy虚拟机 | 192.168.188.133 |

### 解压安装包

131, 132两台虚拟机分别解压安装包到对应安装目录.

unzip emqttd-centos7-v2.3.11.zip

### 配置参数

#### Erlang虚拟机参数

##vim emqttd/etc/emq.conf

#节点名称

node.name = emq@192.168.188.131

#Erlang 虚拟机允许的最大进程数，EMQ 一个连接会消耗2个Erlang进程,

#参数值 > 最大允许连接数 \* 2

node.process\_limit = 2097152

#Erlang 虚拟机允许的最大 Port 数量，EMQ 一个连接消耗1个 Port.

#Erlang 的 Port 非 TCP 端口，可以理解为文件句柄。

#参数值 > 最大允许连接数

node.max\_ports = 1048576

#### EMQ消息服务器参数

##vim emqttd/etc/emq.conf

listener.tcp.external = 0.0.0.0:1883

#设置 TCP 监听器的 Acceptor 池大小

listener.tcp.external.acceptors = 64

#emq最大允许连接数

listener.tcp.external.max\_clients = 1000000

#### linux操作系统配置

1. vim /etc/sysctl.conf

#系统全局允许分配的最大文件句柄数

fs.file-max=2097152

fs.nr\_open=2097152

#并发连接 backlog 设置

net.core.somaxconn=32768

net.ipv4.tcp\_max\_syn\_backlog=16384

net.core.netdev\_max\_backlog=16384

#可用知名端口范围

net.ipv4.ip\_local\_port\_range='1000 65535'

#TCP Socket 读写 Buffer 设置

net.core.rmem\_default=262144

net.core.wmem\_default=262144

net.core.rmem\_max=16777216

net.core.wmem\_max=16777216

net.core.optmem\_max=16777216

net.ipv4.tcp\_rmem='1024 4096 16777216'

net.ipv4.tcp\_wmem='1024 4096 16777216'

#TCP 连接追踪设置

net.nf\_conntrack\_max=1000000

net.netfilter.nf\_conntrack\_max=1000000

net.netfilter.nf\_conntrack\_tcp\_timeout\_time\_wait=30

#TIME-WAIT Socket 最大数量、回收与重用设置

net.ipv4.tcp\_max\_tw\_buckets=1048576

# 注意: 不建议开启該设置，NAT模式下可能引起连接RST

# net.ipv4.tcp\_tw\_recycle = 1

# net.ipv4.tcp\_tw\_reuse = 1

#FIN-WAIT-2 Socket 超时设置

net.ipv4.tcp\_fin\_timeout = 15

2. vim /etc/systemd/system.conf

DefaultLimitNOFILE=1048576

3. vim /etc/security/limits.conf

#持久化设置允许用户/进程打开文件句柄数

\* soft nofile 1048576

\* hard nofile 1048576

#### 其他配置

echo 2097152 > /proc/sys/fs/nr\_open

#允许当前会话/进程打开文件句柄数

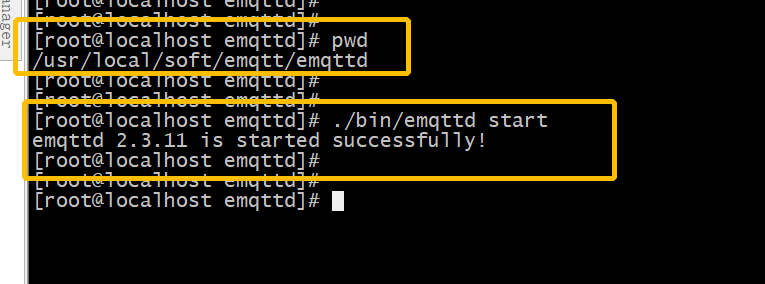
ulimit -n 1048576

### 启动emq集群

131,132两台虚拟机修改好配置

#### 启动单节点

./bin/emqttd start



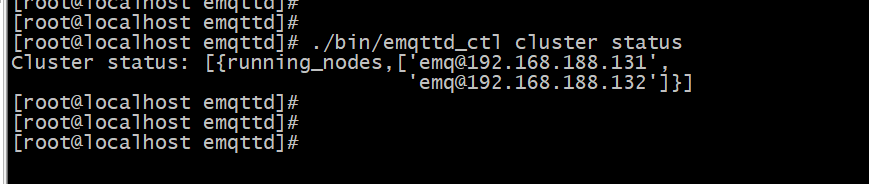
#### 加入emq集群

在132节点 执行. (也可以在131节点执行)

./bin/emqttd\_ctl cluster join [emq@192.168.188.131](mailto:emq@192.168.188.131)

#### 查看集群状态

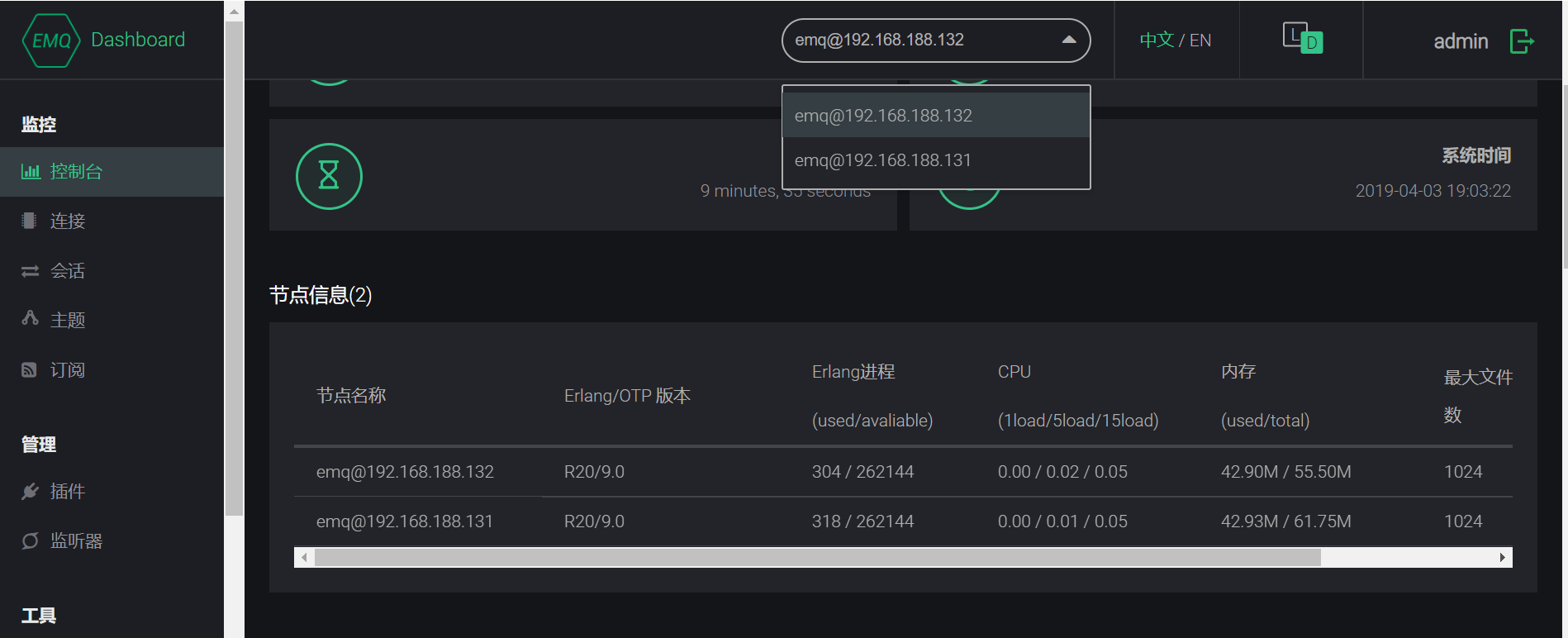
./bin/emqttd\_ctl cluster status



#### web控制台

默认web控制台端口为18083 账号 admin /public

<http://192.168.188.131:18083/#/>



# 安装Haproxy

根据官网私有云部署架构推荐, 使用haproxy 做 负载均衡.

在192.168.188.133安装好 haproxy 并做好对应配置.

监听tcp端口为 1883

# 压测工具

## 依赖安装

### erlang依赖

yum -y install make gcc gcc-c++ kernel-devel m4 ncurses-devel openssl-devel unixODBC-devel

### 安装erlang

#### 下载解压

wget [http://erlang.org/download/](http://erlang.org/download/otp_src_21.0.tar.gz)otp\_src\_20.0.tar.gz

tar -xvzf otp\_src\_21.0.tar.gz

#### 配置

进入 解压缩目录

./configure --prefix=/usr/local/erlang --with-ssl -enable-threads -enable-smmp-support -enable-kernel-poll --enable-hipe --without-javac

#### 安装

make && make install

#### 配置profile

vim /etc/profile

export ERLANG\_HOME=/usr/local/erlang

export PATH=$PATH:$JAVA\_HOME/bin:$ERLANG\_HOME/bin

使配置生效

source /etc/profile

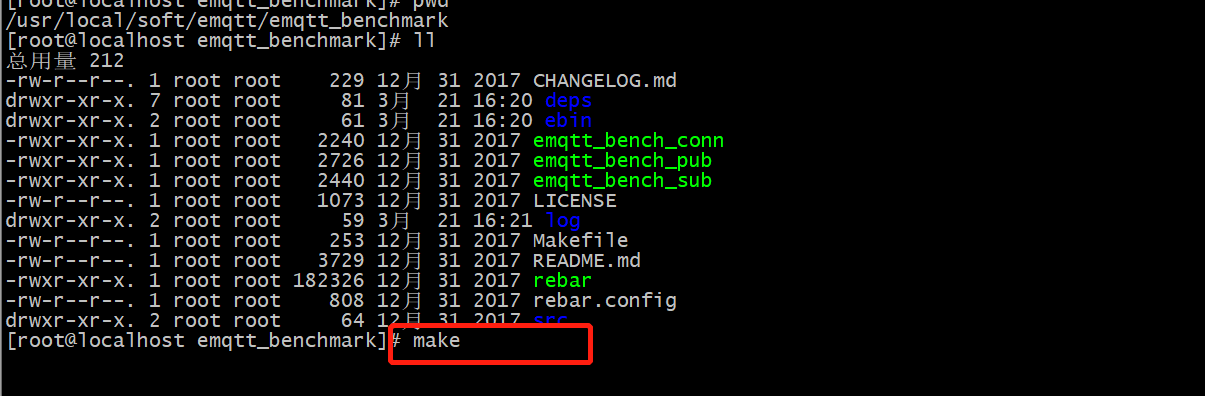
## 安装测试工具

### 从github上下载源码

<https://github.com/emqtt/emqtt_benchmark/tree/benchmark>

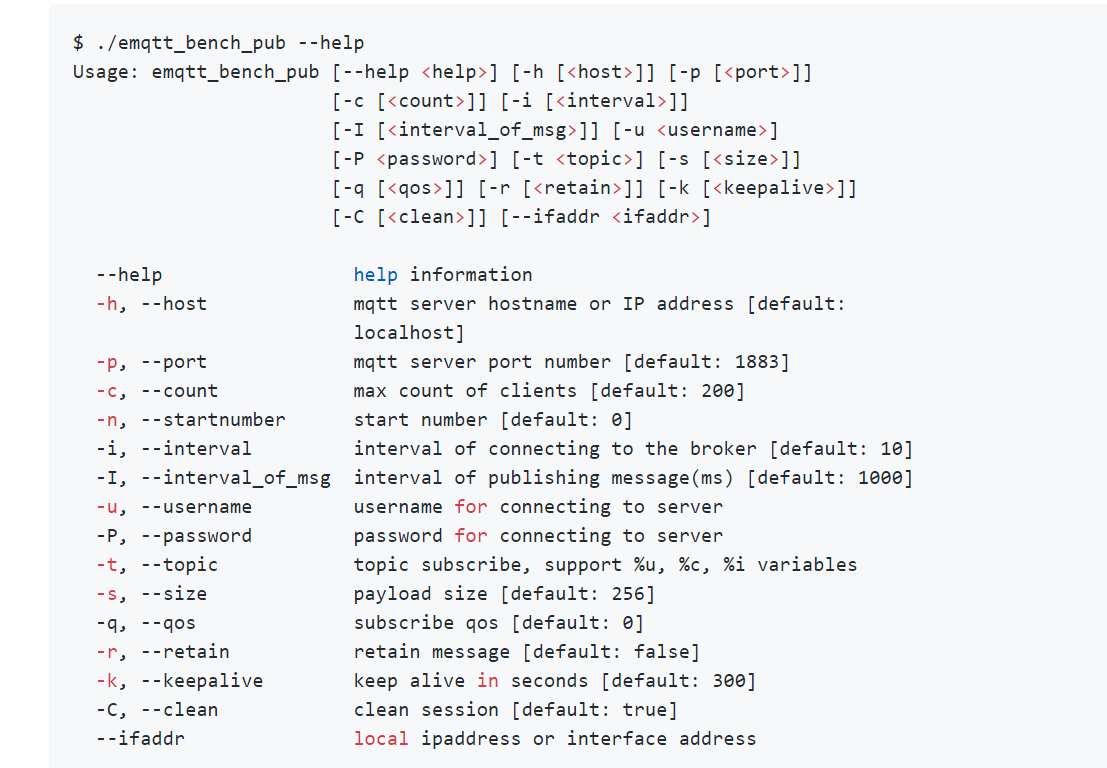
解压到对应目录 进入对应目录

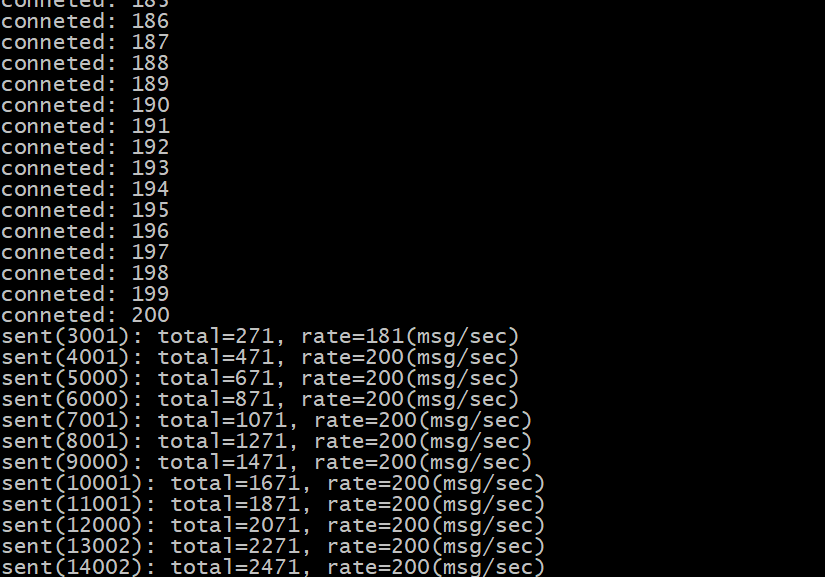
执行 make (这里对erlang版本有要求, 20.0版本的才能用)



make执行完之后

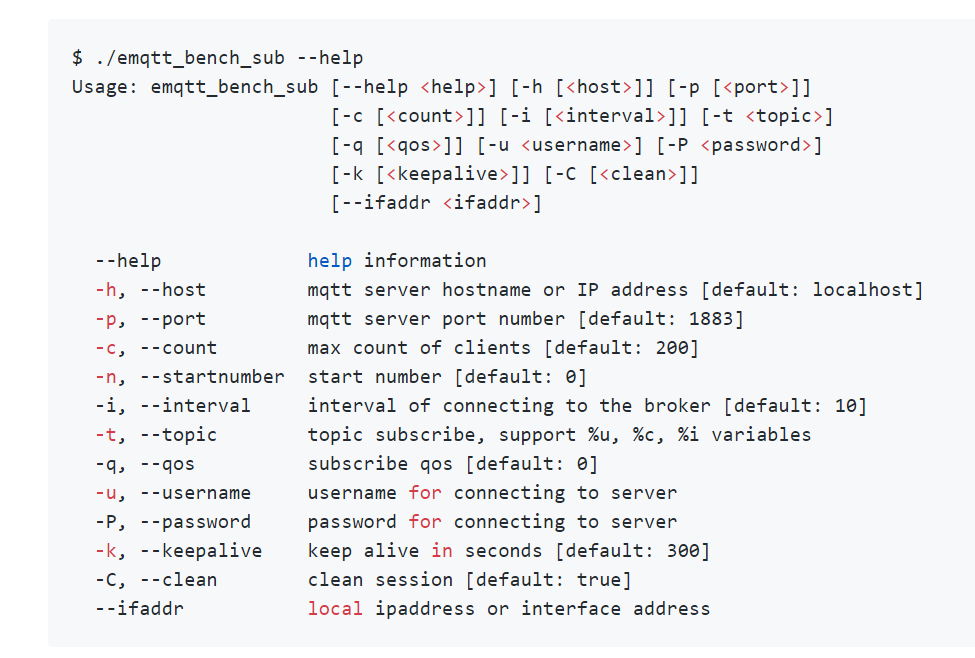
### 发布测试指令

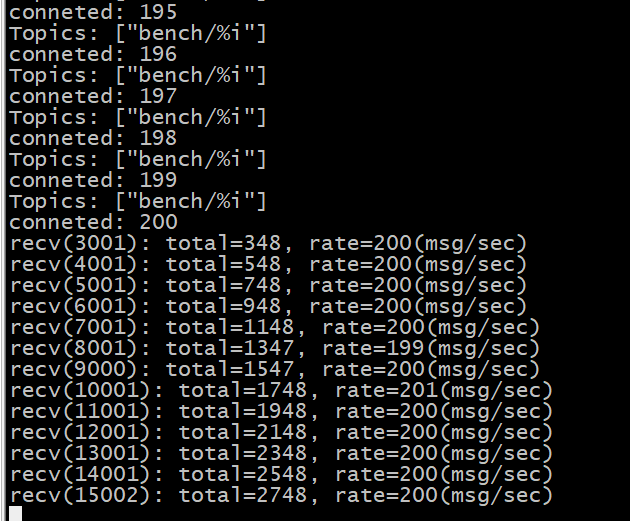
./emqtt\_bench\_pub -h 192.168.188.133 -c 200 -i 10 -t bench/%i



### 订阅测试指令

./emqtt\_bench\_sub –h 192.168.188.133 –c 200 –i 10 –t bench/%i





### 管理控制后台

Haproxy balance 配置为source 根据请求源的ip 所有请求都路由到 131节点.

