How to Get Started with MQTT - Transcript

The Internet of Things or IOT is all about interconnecting devices and one lightweight, secure way for The Internet of Things or IOT is all about interconnecting devices and one lightweight, secure way for online devices to communicate is using MQTT.

MQTT is a publish/subscribe based messaging protocol that avoids direct

connections between devices by relaying data through a central server called the

broker. This is really desirable in IOT because it's easy to add new devices

without touching the existing infrastructure and since new devices

only need to communicate with the broker they don't actually need to be

compatible with the other clients. In this video I'll explain how the MQTT

model works, go over some advantages and disadvantages, and finally use it to

toggle a digital output.

To get an idea of how the publish/subscribe or pub/sub messaging

model works let's start with a real-life example. Let's say I have a switch

connected to one device and I want to use it to toggle the state of a light

attached to another separate device. Traditionally I would just connect them

directly but then if I added new devices with more lights and wanted them to be

triggered by the same source I would need more connections and the same goes

for adding more triggers. This gets very messy and difficult to maintain.

Especially with IOT you could have a whole building full of lights and

switches where each is a separate device. So instead of doing it this way I can

use the publish/subscribe system. This way I can have the device with the

switch publish the switch state under some topic say switch for example. Then I

just subscribe to that topic from another device and any time a message is

published with a matching topic the Broker will relay the new information over, and

this device can use that to toggle a light. Now if I go to add more lights or

more switches I just have one new connection for each client to the server

creating a nice hub-and-spoke model that is easy to maintain rather than the

spider web that we traditionally needed. What's more is that this only

demonstrates how one topic can work between devices. You can have as many

topics as you want for your application with each one having different clients

on either end. Being one to one, many to many, or any combination. The pub/sub

architecture is very flexible and scalable making it great for IOT

applications.

Now let's make this a reality. Here I have a groov EPIC, Edge

Programmable Industrial Controller that has a light hooked up to it. I can use

this screen to toggle the light but I'm not always going to be standing right

next to the device so I'm going to set up the groov EPIC to be an MQTT client

that is subscribed to a specific control topic. To toggle the light based on the

messages that come in on that topic.Then all I need to do is publish to that

topic from my laptop which is a separate client and I'll be able to control the

light from anywhere in the world. To control the light I'll be using Node-RED

since it has a basic MQTT client built-in and it's running on the groov EPIC

right out of the box.

To find out more about either groov EPIC or Node-RED check out opto22.com. Now, let's get that client

set up.To get to Node-RED I just go to my host name slash Node-RED. Then I just

proceed through security, login.

I can drag a subscription node in right away.

Double-click it to edit its settings.

For this example, I'll be using the public broker. And in my case MQTT.groov.com

Here you could use a broker running on a local device like your Mac, Linux, or

Windows PC, a Raspberry Pi, or cloud service like Amazon Web Services.

The default secure port for MQTT is 8883 but I'll just be using the

insecure broker 1883. Many brokers are encrypted or password protected but

this one is open to the public so we'll leave username and password blank. The

final setting screen shows three additional optional messages birth, death

and will. I'll come back and set these up later once we know everything else is

working. So now with the broker setup we're almost ready to subscribe. The next

thing to look at is the topic. The topic namespace is an unmanaged way to

identify your messages. I say unmanaged because there is no enforced format or

standard to follow. Each client determines the topic of that devices

messages which means you, as the programmer, have to be consistent and

specific with your topics. And you'll need to share it with anyone else that

interacts with your broker since they'll need to know what to subscribe to and

what to publish to. This is a lot of freedom but I'll keep things simple for

this example and just use workshop slash switch. Where switch is called a sub

topic of workshop. I could have workshop itself be a sub-topic of

something larger or I could just publish directly to the topic switch but that's

going to depend on your application.

Next, I can choose the quality of service which is a big part of MQTT. In short it

determines how much of a guarantee of delivery you want for your message.

0 quality of service means that the broker received the message at most once. But

might miss it since there is no confirmation made. Quality of service 1

means that the broker will receive the message at least once but may get

duplicates while it's waiting for confirmation. And finally, Quality of

Service 2 means that the broker will receive the message exactly once. The

highest quality of service sounds the most attractive and, for control purposes,

it is really the only option worth considering but it

does come at the cost of extra network traffic since the broker and client must

exchange confirmation packets. Particularly for quality level 2 since

both sides have to acknowledge the publication. Now with my subscribed

client all set up I'll need to use this switch message to toggle a light.

To do that I'll use the SNAP PAC write Node. These are used to interface with

the i/o modules installed on the groov EPIC chassis. This will just connect to

the local host device since Node-RED itself is running on the groov EPIC. And I'm

going to be writing to the digital output I have setup on the system

strategy called panel red LED. And that's all there is to setup.

If I wanted, just for testing, I could also add a debug node to double-check

the output here as well. Now with my Node-RED flow all set up

I'll hit deploy. And, now I just need to publish to this topic to test it out. To

do that I'll be using a separate client running on this laptop called MQTT FX.

Once it's loaded I just bring up my server list and I can go ahead and add

the broker. In this case it'll be the same public broker that I added to Node-RED

it's at MQTT.groov.com and I am using the insecure broker port 1883. And,

I'll not be putting in a user name or password since it's open to the public.

Now I'll just hit apply, close this window, and connect to the broker. Now all

I need to do is publish to that same workshop / switch topic and give it a

message. And, in this case I want to turn the switch on so I'll send the message

true. Now all I have to do is publish and the light comes on. Fantastic!

The connection is both sound and very fast and if I were to use my own broker

I can encrypt the data and securely password protect it - in this case the

publishing client is my laptop.This represents the switch. It publishes the

switch state up to the broker and then the broker relays that message down to

Node-RED on the groov EPIC where it makes the control decision to turn on

the light. So now that we know everything is working let's take a quick look at

some of the more advanced features that I skipped. Birth, death, and will messages,

retain topics, and finally wildcards. So if I go back to Node-RED

and bring open the broker properties I can see those three messages birth,

death and will. Like the node info tab says these messages are sent whenever

this device connects, disconnects and is forcibly disconnected from the broker.

For example I can have the birth message publish under the topic workshop slash

status and give it the payload: Online. Since that will be the status once this

message is sent. Because I want this to arrive at least once I'll give it the

quality of service level 1 and I'll turn the retain flag to true you'll see what

this does in just a moment. Similarly with the closed message or a death

message I can say "offline". Since when I successfully disconnect the workshop

status will be offline I'll give it the same quality of service and retain flag

now with these two messages setup I can just listen to this one workshop status

topic and I'll be able to track the state of this device as it comes on and

offline. But if it drops unexpectedly without sending a disconnect message I

want to know that this is handled by the will or last will and testament

message. This message gets sent by the broker if this client ever unexpectedly

disconnects. So, in this case I'm going to use that same workshop status tag and

give it the payload "disconnected". Since it is an abrupt disconnection that

causes this message to be sent. Now I'll save these new settings, deploy,

and head over to MQTT FX to subscribe to this message. What's great about the

retain flag for MQTT topics is that I can now subscribe to this status and

I'll get the message right away when I connect because it was set to retain it.

Now the broker will keep track of the latest copy of this tag to come in,

retain that data, and send it out to anyone that comes and subscribes to this

topic. While I can't use this feature to request the state of the device at any

time it does mean that new devices to join the system are aware of the health

of all the other clients that are already connected. So, now to test this

out. If I go and make a change to my Node-RED flow and deploy that should mean

that this client disconnected and reconnected. And if I go back to MQTT I

can see that the disconnect message went through and let me know that my device

was offline and then it came back up and I got another message. Finally I want to

see the last will and testament. So, to test that one out I'm going to try and

use a wild card. A wild card is essentially a way to select any matching

topic. For example if I wanted the status of every device, not just the

workshop, I could replace workshop with the single level of wild card "plus". And

what this does is plus will replace anything that matches a subtopic status.

So if I subscribe to that I'll be getting both a copy of the workshop

status and a copy of the developer status, which is another machine running

in the building.

The other thing I can do is subscribe to a workshop slash hashtag or the pound

character. This is a multi-leveled wildcard that will match all sub topics

of workshop including both switch and status. And if status had any sub topics

it would get those as well. It's also possible to get everything matching

every single topic by just using a single multi-level wildcard on its own.

But this gets a little bit loud so it's usually not recommended.

I'll unsubscribe from these topics and just use that single level wildcard plus

slash status. To test the last will and testament when I subscribe

I see the retained messages come in. Then when I unplug the ethernet cable on my

groov EPIC it disconnects from the internet and the broker lets all the

subscribed clients know that this device has dropped. This "disconnected" here is

that last will and testament message. Using this it's possible to monitor the

health of your various devices and even have a system in place to alert you when

something is down, attempt to reboot it, or shift that devices responsibilities

over to another system. And there you have it, there's a quick run through of

MQTT and its place in the Internet of Things. To learn more check out

opto22.com Thanks for watching.

you